Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-03

What is This Course About?

- Calendar description:

  > Introduction to logic and proof techniques for practical reasoning: propositional logic, predicate logic, structural induction; rigorous proofs in discrete mathematics and programming.

- **Discrete Mathematics** is
  - the math of data— **whether complex or big**
  - the math of reasoning— **logic**
  - the math of some kinds of AI— **machine reasoning**
  - the math of **specifying software**
- **Logical Reasoning** is used for
  - **exploring the theoretical limits of computability**
  - **proving sophisaticated algorithms correct**
  - **justifying software designs**
  - **proving software implementations correct**

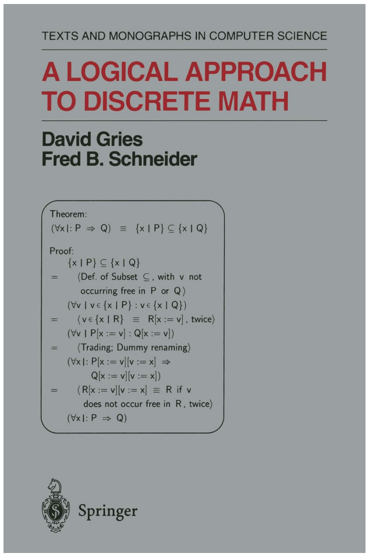
Goals and Rough Outline

- Understand the mechanics of mathematical expressions and proof
  — starting in a familiar area: **Reasoning about integers**
- Develop skill in **propositional calculus**
  - "**propositional**": statements that can be true or false, not numbers
  - "**calculus**": **formalised** reasoning, **calculation** — $\mathbb{B}, \neg, \wedge, \vee, \Rightarrow, \ldots$
- Develop skill in **predicate calculus**
  - "predicate": statement about some subjects. — $\forall, \exists$
- Develop skill in using **basic theories of "data mathematics"**
  - Sets, Functions, Relations, Sequences, Trees, Graphs, . . .
- Develop skill in **correctness reasoning about (imperative) programs**
- . . . *skill development takes time and effort* . . .

All along:

- Encounter computer support for logical reasoning, mechanised discrete mathematics
- Introduction to mechanised software correctness tools
  — **Formal Methods**: increasingly important in industry

Textbook: “LADM”



“This is a rather extraordinary book, and deserves to be read by everyone involved in computer science and — perhaps more importantly — software engineering. I recommend it highly [...]. If the book is taken seriously, the rigor that it unfolds and the clarity of its concepts could have a significant impact on the way in which software is conceived and developed.”

— Peter G. Neumann
(Founder of ACM SIGSOFT)

The Importance of Proof in CS

ACM’s Computer Science Curricula recognize proofs as one of several areas of mathematics that are integral to a wide variety of sub-fields of computer science:

> *...an ability to create and understand a proof — either a formal symbolic proof or a less formal but still mathematically rigorous argument — is important in virtually every area of computer science, including (to name just a few) formal specification, verification, databases, and cryptography.*
>
> ACM/IEEE: Computer Science Curricula 2013, p. 79

“Mathematically rigorous” — “if I really needed to formalise it, I could.”

- **Rigorous** (informal) proofs (e.g. in LADM) strive to “make the eventual formalisation effort minimal”.
- There is value to **readable proofs**, no matter whether formal or informal.
- There is value to **formal, machine-checkable proofs**, especially in the software context, where the world of mathematics is not watching.

Strive for readable formal proofs!

2023 COMPSCI 1DM3 Final 1(a) — Calculational Proof Presentation

Lemma: $(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$

Proof:

$(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$
$\equiv$ ⟨ “Material implication” ⟩
$\neg(\neg q \wedge (\neg p \vee q)) \vee \neg p$
$\equiv$ ⟨ “De Morgan” ⟩
$\neg\neg q \vee (\neg\neg p \wedge \neg q) \vee \neg p$
$\equiv$ ⟨ “Double negation” ⟩
$q \vee (p \wedge \neg q) \vee \neg p$
$\equiv$ ⟨ “Distributivity of $\vee$ over $\wedge$” ⟩
$(q \vee p \vee \neg p) \wedge (q \vee \neg q \vee \neg p)$
$\equiv$ ⟨ “Excluded middle” ⟩
$(q \vee \mathsf{true}) \wedge (\mathsf{true} \vee \neg p)$
$\equiv$ ⟨ “Zero of $\vee$” ⟩
$\mathsf{true} \wedge \mathsf{true}$
$\equiv$ ⟨ “Idempotency of $\wedge$” ⟩
true

Lemma “F1(a)”: $(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$

Proof:

$(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$
$\equiv$ ⟨ “Material implication” ⟩
$(\neg q \wedge (\neg p \vee q)) \Rightarrow \neg p$
$\equiv$ ⟨ “Absorption” ⟩
$(\neg q \wedge \neg p) \Rightarrow \neg p$
$\equiv$ ⟨ “De Morgan” ⟩
$\neg(q \vee p) \Rightarrow \neg p$
$\equiv$ ⟨ “Contrapositive” ⟩
$p \Rightarrow q \vee p$
$\equiv$ ⟨ “Weakening” ⟩
true

2023 COMPSCI 1DM3 Final 1(b) — Calculational Proof Presentation

Lemma "F1(b)": $(\exists x \bullet P \Rightarrow Q) \equiv (\forall x \bullet P) \Rightarrow (\exists x \bullet Q)$

Proof:

$(\exists x \bullet P \Rightarrow Q)$

$\equiv \langle$ "Material implication" $\rangle$

$(\exists x \bullet \neg P \vee Q)$

$\equiv \langle$ "Distributivity of $\exists$ over $\vee$" $\rangle$

$(\exists x \bullet \neg P) \vee (\exists x \bullet Q)$

$\equiv \langle$ "Generalised De Morgan" $\rangle$

$\neg (\forall x \bullet P) \vee (\exists x \bullet Q)$

$\equiv \langle$ "Material implication" $\rangle$

$(\forall x \bullet P) \Rightarrow (\exists x \bullet Q)$

First Tool: CalcCheck

- CalcCheck: A proof checker for the textbook logic
- CalcCheck analyses textbook-style presentations of proofs
- $\text{CalcCheck}_{\text{Web}}$: A notebook-style web-app interface to CalcCheck
- **You can check your proofs before handing them in!**
- **Will be used in exams!**

  — initially with proof checking turned off...

  ... but syntax checking left on
- **Will be used in exams**

  — as far as possible...

  You need to be able to do both:
  - Write formalisations and proofs using CalcCheck
  - Write formalisations and proofs **by hand on paper**

(Firefox and Chrome can be expected to work with $\text{CalcCheck}_{\text{Web}}$.
Safari, Edge, IE not necessarily.)

From the LADM Instructor's Manual

Emphasis on skill acquisition:

- "a course taught from this text will give students a solid understanding of what constitutes a proof and a skill in developing, presenting, and reading proof."
- "We believe that teaching a skill in formal manipulation makes learning the other material easier."
- "Logic as a tool is so important to later work in computer science and mathematics that students must understand the use of logic and be sure in that understanding."
- "One benefit of our new approach to teaching logic, we believe is that students become more effective in communicating and thinking in other scientific and engineering disciplines."
- "Frequent but shorter homeworks ensure that students get practice"

Consciously departing from existing mechanised logics:

- "Our equational logic is a "People Logic", instead of a "Machine Logic"."

- CalcCheck mechanises this "People Logic"

CalcCheck: A Recognisable Version of the Textbook Proof Language

(11.5) $S = \{x \mid x \in S : x\}$.
According to axiom Extensionality (11.4), it suffices to prove that $v \in S \equiv v \in \{x \mid x \in S : x\}$, for arbitrary v. We have,

$$
\begin{array}{ll}
& v \in \{x \mid x \in S : x\} \\
= & \langle \text{Definition of membership (11.3)} \rangle \\
& (\exists x \mid x \in S : v = x) \\
= & \langle \text{Trading (9.19), twice} \rangle \\
& (\exists x \mid x = v : x \in S) \\
= & \langle \text{One-point rule (8.14)} \rangle \\
& v \in S
\end{array}
$$

```
Theorem (11.5):  S = { x ❙ x ∈ S • x }
Proof:
  Using “Set extensionality” (11.4):
    For any `v`:
        v ∈ { x ❙ x ∈ S • x }
      =    ⟨ “Set membership” (11.3) ⟩
        (∃ x ❙ x ∈ S • v = x)
      =    ⟨ “Trading for ∃” (9.19) ⟩
        (∃ x ❙ x = v • x ∈ S)
      =    ⟨ “One-point rule for ∃” (8.14), substitution ⟩
        v ∈ S
```

Note:

1. The calculation part is transliterated into Unicode plain text (only minimal notation changes).
2. The prose top-level of the proof is formalised into Using and For any structures in the spirit of LADM

From the LADM Instructor's Manual: "Some Hints on Mechanics"

- "We have been successful (in a class of 70 students) with occasionally writing a few problems on the board and walking around the class as the students work on them."
  - COMPSCI&SFWRENG 2DM3: ≈240 students in 2016, 360 in 2020
  - COMPSCI 2LC3: Over 180 students in 2021; over 200 in 2023
  - Tutorials normally have 20–40 students and use this approach, with students working on their computers
    — this still worked with online course delivery
- "Frequent short homework assignments are much more effective than longer but less frequent ones. Handing out a short problem set that is due the next lecture forces the students to practice the material immediately, instead of waiting a week or two."
  - Since 2018, giving homework up to three times per week
  - Only feasible due to online submission and autograding
  - **Clear improvement in course results**

From the LADM Instructor's Manual: "Some Hints on Mechanics" (ctd.)

- "There is no substitute for practice accompanied by ample and timely feedback"
  - Most "timely feedback" is provided by interaction with $\text{CalcCheck}_{\text{Web}}$
  - Autograding for homework and assignments produces some additional feedback
  - CalcCheck is intentionally a proof checker, not a proof assistant
  - Providing ample TA office hours (and now a "Course Help" channel) helps students overcome roadblocks.
- "We tell the students that they are all capable of mastering the material (for they are)."
  - …and CalcCheck homework makes more of them actually master the material.

Organisation

- Schedule
- Grading
- Exams
- Avenue
- Course Page: `http://www.cas.mcmaster.ca/~kahl/CS2LC3/2024/`

— check in case of Avenue and MSTeams outage!

— See the Outline (on course page and on Avenue)

— Read the Outline!

Schedule

	Mon	Tue	Wed	Thu	Fri
10:30–11:20			T1–4		
11:30–12:20		**Lecture**	T1–4	**Lecture**	**Lecture**
12:30– –14:20				T5	Office hour
14:30– –16:20					T7
16:30– –18:20			T6		

- **Lectures: attend!, take notes!**
- **2-hour Tutorials — starting tomorrow, Wednesday, September 4**
  – discuss student approaches to "Exercise" questions.
- **TA office hours**: *TBA*
- **Studying and Homework:** — **Reading the textbook**
  — **Writing proofs in CALCCHECK$_{\text{Web}}$**

Grading

- **Homework**, from one lecture to the next — in total: **10%**
  - (Not Thursday to Friday)
  - Homework will be more frequent in the first part of the term
  - The weakest 2 or 3 homeworks are dropped (see outline)
  - MSAFs for homework are not processed
- **Roughly-biweekly assignments** — in total: **10%**
  - Assignments will be less frequent in the first part of the term
  - The weakest 1 or 2 assignments are dropped (see outline)
  - MSAFs for assignments are not processed
- **2 Midterm Tests**, closed book, **on CALCCHECK$_{\text{Web}}$ / on paper**, **each:**
  - **10%** if not better than your final
  - **20%** if better than your final

  — in total at least: **20%**
  — in total up to: **40%**
- **Final** (closed book, 2.5 hours, **on CALCCHECK$_{\text{Web}}$ / on paper**) **40% to 60%**

= **100%**

Exams

- Exercise questions, assignment questions, and the questions on midterm tests, and on the final —

  — will be somewhat similar...

- All tests and exams are **closed-book**.
  – The main difference to open-book lies in how you prepare...
  – **Knowledge is important:**
  Without the right knowledge, you would not even know what to look up where!

- **You need to be able and prepared to do both:**
  - Write formalisations and proofs using CALCCHECK
  - Write formalisations and proofs by hand on paper

- **Know your stuff!**

  — ... and not only in the exams ...

  — ... and not only for this term ...

  — ... similar to learning a new language

The Language of Logical Reasoning

The mathematical foundations of Computing Science involve **language skills and knowledge**:

- **Vocabulary:** Commonly known concepts and technical terms
- **Syntax/Grammar:** How to produce complex statements and arguments
- **Semantics:** How to relate complex statements with their meaning
- **Pragmatics:** How people actually use the features of the language

> Conscious and fluent use of the
> **language of logical reasoning**
> is the foundation for
> **precise specification and rigorous argumentation**
> in **Computer Science and Software Engineering**.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-03

Part 2: Expressions and Calculations

H1 Starting Point

The Answer

$$
\begin{array}{ll}
& 7 \cdot 8 \\
= & \langle \text{Fact } `8 = 7 + 1` \rangle \\
& 7 \cdot (7 + 1) \\
= & \langle \text{Fact } `7 = 10 - 3` \rangle \\
& (10 - 3) \cdot (7 + 1) \\
= & \langle \text{“Distributivity of} \cdot \text{over} + \text{”} \rangle \\
& (10 - 3) \cdot 7 + (10 - 3) \cdot 1 \\
= & \langle \text{“Distributivity of} \cdot \text{over} - \text{”} \rangle \\
& 10 \cdot 7 - 3 \cdot 7 + 10 \cdot 1 - 3 \cdot 1 \\
= & \langle \text{“Identity of} \cdot \text{” — twice} \rangle \\
& 10 \cdot 7 - 3 \cdot 7 + 10 - 3 \\
= & \langle \text{Fact } `3 \cdot 7 = 21` \rangle \\
& 10 \cdot 7 - 21 + 10 - 3 \\
= & \langle \text{Fact } `10 \cdot 7 = 70` \rangle \\
& 70 - 21 + 10 - 3 \\
= & \langle \text{Fact } `10 - 3 = 7` \rangle \\
& 70 - 21 + 7 \\
= & \langle \text{Fact } `21 + 7 = 28` \rangle \\
& 70 - 28 \\
= & \langle \text{Fact } `70 - 28 = 42` \rangle \\
& 42
\end{array}
$$

Calculational Proof Format

$$
\begin{array}{ll}
& E_0 \\
= & \langle \text{Explanation of why } E_0 = E_1 \rangle \\
& E_1 \\
= & \langle \text{Explanation of why } E_1 = E_2 \rangle \\
& E_2 \\
= & \langle \text{Explanation of why } E_2 = E_3 \rangle \\
& E_3
\end{array}
$$

This is a proof for:

$$E_0 = E_3$$

Calculational Proof Format

$$
\begin{array}{ll}
& E_0 \\
= & \langle \text{Explanation of why } E_0 = E_1 \rangle \\
& E_1 \\
= & \langle \text{Explanation of why } E_1 = E_2 \rangle \\
& E_2 \\
= & \langle \text{Explanation of why } E_2 = E_3 \rangle \\
& E_3
\end{array}
$$

The calculational presentation as such is conjunctional: This reads as:

$$E_0 = E_1 \quad \wedge \quad E_1 = E_2 \quad \wedge \quad E_2 = E_3$$

Because $=$ is **transitive**, this justifies:

$$E_0 = E_3$$

Syntax of Conventional Mathematical Expressions

LADM 1.1, p. 7

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression

- If E is an expression, then (E) is an expression

- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.

  For example, the negation symbol $-$ is used as a unary prefix operator, so -5 is an expression.

- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

  For example, the symbols $+$ and $\cdot$ are binary infix operators, so $1+2$ and $(-5)\cdot(3+x)$ are expressions.

Syntax of Conventional Mathematical Expressions

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression
- If E is an expression, then (E) is an expression
- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.
- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

The intention of this is that each expression is **at least one** of the following alternatives:

- **either some constant**
- **or some variable**
- **or some simpler expression** in parentheses
- **or** the application of **some unary prefix operator** to **some simpler expression**
- **or** the application of **some binary infix operator** to **two simpler expressions**

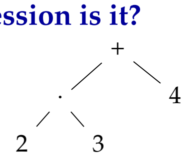
Why is this an expression?

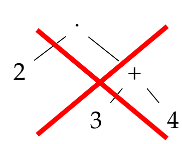
$$2 \cdot 3 + 4$$

- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

- **or** the application of **some binary infix operator** to **two simpler expressions**

Which expression is it?





Why?

$\Longrightarrow$ The multiplication operator $\cdot$ has **higher precedence** than the addition operator $+$.

Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- . (function application)
- unary prefix operators $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot$ $/$ $\div$ mod gcd
- $+$ $-$ $\cup$ $\cap$ $\times$ $\circ$ $\bullet$
- $\downarrow$ $\uparrow$
- $\#$
- $\triangleleft$ $\triangleright$ $\hat{}$
- $=$ $<$ $>$ $\in$ $\subset$ $\subseteq$ $\supset$ $\supseteq$ $|$ (conjunctional)
- $\vee$ $\wedge$
- $\Rightarrow$ $\Leftarrow$
- $\equiv$ **(lowest precedence)**

All non-associative binary infix operators associate to the left, except $**$, $\triangleleft$, $\Rightarrow$, $\rightarrow$, which associate to the right.

Why are these expressions? Which expressions are these?

1. $n - k - 1$

2. $5 - 6 + 7$

3. $a + b - c$

The operators + and − **associate to the left**, also mutually.

Associativity versus Association

- If we write $a + b + c$, there appears to be no need to discuss whether we mean $(a + b) + c$ or $a + (b + c)$, because they evaluate to the same values:

$$(a + b) + c = a + (b + c)$$ **"+" is associative**

- If we write $a - b - c$, we mean $(a - b) - c$:

"−" associates to the left $9 - (5 - 2) \neq (9 - 5) - 2$

- If we write a^{b^c}, we mean $a^{(b^c)}$:

exponentiation associates to the right $2^{(3^2)} \neq (2^3)^2$

- If we write $a ** b ** c$, we mean $a ** (b ** c)$:

"$**$" associates to the right

- If we write $a \Rightarrow b \Rightarrow c$, we mean $a \Rightarrow (b \Rightarrow c)$:

"$\Rightarrow$" associates to the right $(\mathit{false} \Rightarrow (\mathit{true} \Rightarrow \mathit{false})) \neq ((\mathit{false} \Rightarrow \mathit{true}) \Rightarrow \mathit{false})$

An Equational Theory of Integers — Axioms (LADM Ch. 15)

(15.1) **Axiom, Associativity:** $(a + b) + c = a + (b + c)$
$(a \cdot b) \cdot c = a \cdot (b \cdot c)$

(15.2) **Axiom, Symmetry:** $a + b = b + a$
$a \cdot b = b \cdot a$

(15.3) **Axiom, Additive identity:** $0 + a = a$
$a + 0 = a$

(15.4) **Axiom, Multiplicative identity:** $1 \cdot a = a$
$a \cdot 1 = a$

(15.5) **Axiom, Distributivity:** $a \cdot (b + c) = a \cdot b + a \cdot c$
$(b + c) \cdot a = b \cdot a + c \cdot a$

(15.13) **Axiom, Unary minus:** $a + (-a) = 0$

(15.14) **Axiom, Subtraction:** $a - b = a + (-b)$

An Equational Theory of Integers — Axioms (CalcCheck)

Ex1.2 Starting Point

Declaration: $\mathbb{Z}$: Type
Declaration: _+_ : $\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
Declaration: _·_ : $\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
Axiom (15.1) (15.1*a*) "Associativity of +": $(a + b) + c = a + (b + c)$
Axiom (15.1) (15.1*b*) "Associativity of ·": $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
Axiom (15.2) (15.2*a*) "Symmetry of +": $a + b = b + a$
Axiom (15.2) (15.2*b*) "Symmetry of ·": $a \cdot b = b \cdot a$
Axiom (15.3) "Additive identity" "Identity of +": $0 + a = a$
Axiom (15.4) "Multiplicative identity" "Identity of ·": $1 \cdot a = a$
Axiom (15.5) "Distributivity of · over +": $a \cdot (b + c) = a \cdot b + a \cdot c$
Axiom (15.9) "Zero of ·": $a \cdot 0 = 0$
Declaration: −_ : $\mathbb{Z} \to \mathbb{Z}$
Declaration: _−_ : $\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
Axiom (15.13) "Unary minus": $a + (-a) = 0$
Axiom (15.14) "Subtraction": $a - b = a + (-b)$

Calculational Proofs of Theorems — (15.17) $-(-a) = a$

(15.3) **Identity of +** $0 + a = a$	(15.13) **Unary minus** $a + (-a) = 0$

LADM:

Theorem (15.17): $-(-a) = a$

Proof:

$-(-a)$
= ⟨ Identity of + (15.3) ⟩
$0 + -(-a)$
= ⟨ Unary minus (15.13) ⟩
$a + (-a) + -(-a)$
= ⟨ Unary minus (15.13) ⟩
$a + 0$
= ⟨ Identity of + (15.3) ⟩
a

CalcCheck:

Theorem (15.17) "Self-inverse of unary minus":
$-(-a) = a$

Proof:

$-(-a)$
= ⟨ "Identity of +" ⟩
$0 + -(-a)$
= ⟨ "Unary minus" ⟩
$a + (-a) + -(-a)$
= ⟨ "Unary minus" ⟩
$a + 0$
= ⟨ "Identity of +" ⟩
a

in Ex1.2

Get Started with CalcCheck Now!

H1 Starting Point

$$
\begin{array}{l}
\quad 7 \cdot 8 \\
= \langle \text{Fact } `8 = 7 + 1` \rangle \\
\quad 7 \cdot (7 + 1) \\
= \langle \text{Fact } `7 = 10 - 3` \rangle \\
\quad (10 - 3) \cdot (7 + 1) \\
= \langle \text{“Distributivity of } \cdot \text{ over } +\text{”} \rangle \\
\quad (10 - 3) \cdot 7 + (10 - 3) \cdot 1 \\
= \langle \text{“Distributivity of } \cdot \text{ over } -\text{”} \rangle \\
\quad 10 \cdot 7 - 3 \cdot 7 + 10 \cdot 1 - 3 \cdot 1 \\
= \langle \text{“Identity of } \cdot \text{” — twice} \rangle \\
\quad 10 \cdot 7 - 3 \cdot 7 + 10 - 3 \\
= \langle \text{Fact } `3 \cdot 7 = 21` \rangle \\
\quad 10 \cdot 7 - 21 + 10 - 3 \\
= \langle \text{Fact } `10 \cdot 7 = 70` \rangle \\
\quad 70 - 21 + 10 - 3 \\
= \langle \text{Fact } `10 - 3 = 7` \rangle \\
\quad 70 - 21 + 7 \\
= \langle \text{Fact } `21 + 7 = 28` \rangle \\
\quad 70 - 28 \\
= \langle \text{Fact } `70 - 28 = 42` \rangle \\
\quad 42
\end{array}
$$

- **Tutorials start tomorrow, Wednesday, Sept. 4!**
- Work through Homework 1 before your tutorial!
- Get started working on Exercises 1.*
- Go to your tutorial to continue working on Ex1 — bring your laptop!
- Submit H1 by 23:59 on Friday, Sept. 6

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-05

Expressions and Substitution — LADM Chapter 1

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-05

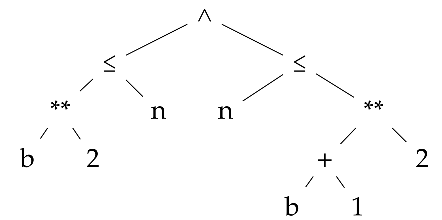
Part 1: Syntax of Mathematical Expressions (ctd.)

Term Tree Presentation of Mathematical Expression

(Using linear notation $x ** y$ for exponentiation x^y)

$$b^2 \le n \le (b+1)^2$$

$$b^2 \quad \le \quad n \quad \wedge \quad n \quad \le \quad (b+1)^2$$



We write strings, but we think trees.

All the rules we have for implicit parentheses only serve to encode the tree structure.

(These term trees are the essence of the abstract syntax trees (ASTs) used centrally in compilers.)

Recall: Syntax of Conventional Mathematical Expressions

Textbook 1.1, p. 7

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression
- If E is an expression, then (E) is an expression
- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.

  For example, the negation symbol $-$ is used as a unary prefix operator, so -5 is an expression.
- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

  For example, the symbols $+$ and $\cdot$ are binary infix operators, so $1+2$ and $(-5)\cdot(3+x)$ are expressions.

Recall: Syntax of Conventional Mathematical Expressions

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression
- If E is an expression, then (E) is an expression
- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.
- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

The intention of this is that each expression is **at least one** of the following alternatives:

- **either some constant**
- **or some variable**
- **or some simpler expression** in parentheses
- **or** the application of **some unary prefix operator** to **some simpler expression**
- **or** the application of **some binary infix operator** to **two simpler expressions**

Precedences and Association — *We write strings, but we think trees*

All the rules we have for implicit parentheses only serve to encode the tree structure.

(We use underscores to denote operator argument positions.
So _⊗_ is a binary infix operator, and ⊟_ is a unary prefix operator.)

_⊗_ **has higher precedence than** _⊙_	means	$a \otimes b \odot c = (a \otimes b) \odot c$ $a \odot b \otimes c = a \odot (b \otimes c)$
_⊗_ **has higher precedence than** ⊟_	means	$\boxminus a \otimes b = \boxminus (a \otimes b)$
⊟_ **has higher precedence than** _⊗_	means	$\boxminus a \otimes b = (\boxminus a) \otimes b$
_⊗_ **associates to the left**	means	$a \otimes b \otimes c = (a \otimes b) \otimes c$
_⊗_ **associates to the right**	means	$a \otimes b \otimes c = a \otimes (b \otimes c)$
_⊗_ **mutually associates to the left** with (same prec.) _⊙_	means	$a \otimes b \odot c = (a \otimes b) \odot c$
_⊗_ **mutually associates to the right** with (same prec.) _⊙_	means	$a \otimes b \odot c = a \otimes (b \odot c)$

Associativity versus Association

- If we write $a + b + c$, there appears to be no need to discuss whether we mean $(a + b) + c$ or $a + (b + c)$, because they evaluate to the same values:

  $(a + b) + c = a + (b + c)$ **"+" is associative**

- If we write $a - b - c$, we mean $(a - b) - c$:

  "−" associates to the left $9 - (5 - 2) \neq (9 - 5) - 2$

- If we write a^{b^c}, we mean $a^{(b^c)}$:

  exponentiation associates to the right $2^{(3^2)} \neq (2^3)^2$

- If we write $a ** b ** c$, we mean $a ** (b ** c)$:

  "" associates to the right**

- If we write $a \Rightarrow b \Rightarrow c$, we mean $a \Rightarrow (b \Rightarrow c)$:

  "⇒" associates to the right $\mathsf{F} \Rightarrow (\mathsf{T} \Rightarrow \mathsf{F}) \neq (\mathsf{F} \Rightarrow \mathsf{T}) \Rightarrow \mathsf{F}$

Conjunctional Operators

Chains can involve different conjunctional operators:

$1 < i \leq j < 5 = k$
≡ ⟨ "Reflexivity of =" `x = x` — conjunctional operators ⟩
$1 < i \quad \wedge \quad i \leq j \quad \wedge \quad j < 5 \quad \wedge \quad 5 = k$
≡ ⟨ "Reflexivity of =" — ∧ has lower precedence ⟩
$(1 < i) \quad \wedge \quad (i \leq j) \quad \wedge \quad (j < 5) \quad \wedge \quad (5 = k)$

Remember this!!!

$x < 5 \in S \subseteq T$
≡ ⟨ "Reflexivity of =" — conjunctional operators ⟩
$x < 5 \quad \wedge \quad 5 \in S \quad \wedge \quad S \subseteq T$
≡ ⟨ "Reflexivity of =" — ∧ has lower precedence ⟩
$(x < 5) \quad \wedge \quad (5 \in S) \quad \wedge \quad (S \subseteq T)$

Mathematical Expressions, Terms, Formulae …

"Expression" is not the only word used for this kind of concept.

Related terminology:

- Both "term" and "expression" are frequently used names for the same kind of concept.
- The textbook's "expression" subsumes both "term" and "formula" of conventional first-order predicate logic.

Remember:

- Expressions are **understood** as tree-structures
  — "*abstract syntax*"
- Expressions are **written** as strings
  — "*concrete syntax*"
- Parentheses, precedences, and association rules **only serve to disambiguate the encoding of trees in strings.**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-05

Part 2: Substitution

Plan for Part 2

- **Substitution as such:** Replaces variables with expressions in expressions, e.g.,

$$
\begin{aligned}
& (x + 2 \cdot y)[x, y := 3 \cdot a, b + 5] \\
= \ & \langle \text{ Substitution } \rangle \\
& 3 \cdot a + 2 \cdot (b + 5)
\end{aligned}
$$

- **Applying substitution instances of theorems** and making the substitution explicit:

$$
\begin{aligned}
& 2 \cdot y \ + \ -(2 \cdot y) \\
= \ & \langle \text{ “Unary minus” `} a \ + \ -a \ = \ 0\text{` with `} a \ := \ 2 \cdot y \text{` } \rangle \\
& 0
\end{aligned}
$$

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$$E[x := R] \qquad \text{or} \qquad E^x_R$$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 1:

$$
\begin{aligned}
& (x+y)[x := z+2] \\
= & \langle \text{ Substitution — performing substitution } \rangle \\
& ((z+2)+y) \\
= & \langle \text{ “Reflexivity of =” — removing unnecessary parentheses } \rangle \\
& z+2+y
\end{aligned}
$$

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$$E[x := R]$$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 2:

$$
\begin{aligned}
& (x \cdot y)[x := z+2] \\
= & \langle \text{ Substitution } \rangle \\
& ((z+2) \cdot y) \\
= & \langle \text{ “Reflexivity of =” — removing unnecessary parentheses } \rangle \\
& (z+2) \cdot y
\end{aligned}
$$

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$$E[x := R]$$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 3:

$$
\begin{aligned}
& (0+a)[a := -(-a)] \\
= & \langle \text{ Substitution } \rangle \\
& (0+(-(-a))) \\
= & \langle \text{ “Reflexivity of =” — removing (some) unnecessary parenth. } \rangle \\
& 0 \;+\; -(-a)
\end{aligned}
$$

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$$E[x := R]$$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 4:

$$
\begin{array}{ll}
 & x + y[x := z + 2] \\
= & \langle \text{“Reflexivity of =” — adding parentheses for clarity} \rangle \\
 & x + (y[x := z + 2]) \\
= & \langle \text{Substitution} \rangle \\
 & x + (y) \\
= & \langle \text{“Reflexivity of =” — removing unnecessary parentheses} \rangle \\
 & x + y
\end{array}
$$

Note: Substitution $[x := R]$ is a **highest precedence** postfix operator

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$$E[x := R] \quad \text{or} \quad E^x_R$$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Examples:

Expression	Result	Unnecessary parentheses removed
$x[x := z + 2]$	$(z + 2)$	$z + 2$
$(x + y)[x := z + 2]$	$((z + 2) + y)$	$z + 2 + y$
$(x \cdot y)[x := z + 2]$	$((z + 2) \cdot y)$	$(z + 2) \cdot y$
$x + y[x := z + 2]$	$x + y$	$x + y$

Note: Substitution $[x := R]$ is a **highest precedence** postfix operator

Sequential Substitution

$$
\begin{array}{ll}
 & (x + y)[x := y - 3][y := z + 2] \\
= & \langle \text{“Reflexivity of =” — adding parentheses for clarity} \rangle \\
 & ((x + y)[x := y - 3])[y := z + 2] \\
= & \langle \text{Substitution — performing inner substitution} \rangle \\
 & (((y - 3) + y))[y := z + 2] \\
= & \langle \text{Substitution — performing outer substitution} \rangle \\
 & ((((z + 2) - 3) + (z + 2))) \\
= & \langle \text{“Reflexivity of =” — removing unnecessary parentheses} \rangle \\
 & z + 2 - 3 + z + 2
\end{array}
$$

Simultaneous Textual Substitution

If R is a **list** $R_1, \ldots, R_n$ of expressions
and x is a **list** $x_1, \ldots, x_n$ of **distinct variables**, we write:

$$E[x := R]$$

to denote the **simultaneous** replacement of the variables of x
by the corresponding expressions of R,
each expression being enclosed in parentheses.

Example:

$$\begin{aligned} & (x + y)[x, y := y - 3, z + 2] \\ = \; & \langle \text{ Substitution — performing substitution } \rangle \\ & ((y - 3) + (z + 2)) \\ = \; & \langle \text{ “Reflexivity of =” — removing unnecessary parentheses } \rangle \\ & y - 3 + z + 2 \end{aligned}$$

Simultaneous Textual Substitution

If R is a **list** $R_1, \ldots, R_n$ of expressions
and x is a **list** $x_1, \ldots, x_n$ of **distinct** variables, we write:

$$E[x := R]$$

to denote the **simultaneous** replacement of the variables of x
by the corresponding expressions of R,
each expression being enclosed in parentheses.

Examples:

Expression	Result	Unnecessary parentheses removed
$x[x, y := y - 3, z + 2]$	$(y - 3)$	$y - 3$
$(y + x)[x, y := y - 3, z + 2]$	$((z + 2) + (y - 3))$	$z + 2 + y - 3$
$(x + y)[x, y := y - 3, z + 2]$	$((y - 3) + (z + 2))$	$y - 3 + z + 2$
$x + y[x, y := y - 3, z + 2]$	$x + (z + 2)$	$x + z + 2$

Simultaneous Substitution:

$$\begin{aligned} & (x + y)[x, y := y - 3, z + 2] \\ = \; & \langle \text{ Substitution — performing substitution } \rangle \\ & ((y - 3) + (z + 2)) \\ = \; & \langle \text{ “Reflexivity of =” — removing unnecessary parentheses } \rangle \\ & y - 3 + z + 2 \end{aligned}$$

Sequential Substitution:

$$\begin{aligned} & (x + y)[x := y - 3][y := z + 2] \\ = \; & \langle \text{ “Reflexivity of =” — adding parentheses for clarity } \rangle \\ & ((x + y)[x := y - 3])[y := z + 2] \\ = \; & \langle \text{ Substitution — performing inner substitution } \rangle \\ & (((y - 3) + y))[y := z + 2] \\ = \; & \langle \text{ Substitution — performing outer substitution } \rangle \\ & ((((z + 2) - 3) + (z + 2))) \\ = \; & \langle \text{ “Reflexivity of =” — removing unnecessary parentheses } \rangle \\ & z + 2 - 3 + z + 2 \end{aligned}$$

Recall: An Equational Theory of Integers — Axioms (LADM Ch. 15)

(15.1) **Axiom, Associativity:** $(a + b) + c = a + (b + c)$
$(a \cdot b) \cdot c = a \cdot (b \cdot c)$

(15.2) **Axiom, Symmetry:** $a + b = b + a$
$a \cdot b = b \cdot a$

(15.3) **Axiom, Additive identity:** $0 + a = a$
$a + 0 = a$

(15.4) **Axiom, Multiplicative identity:** $1 \cdot a = a$
$a \cdot 1 = a$

(15.5) **Axiom, Distributivity:** $a \cdot (b + c) = a \cdot b + a \cdot c$
$(b + c) \cdot a = b \cdot a + c \cdot a$

(15.13) **Axiom, Unary minus:** $a + (-a) = 0$

(15.14) **Axiom, Subtraction:** $a - b = a + (-b)$

Calculational Proofs of Theorems — (15.17) $-(-a) = a$

(15.3) **Identity of +** $0 + a = a$	(15.13) **Unary minus** $a + (-a) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$
= ⟨ Identity of + (15.3) ⟩
$0 + -(-a)$
= ⟨ Unary minus (15.13) ⟩
$a + (-a) + -(-a)$
= ⟨ Unary minus (15.13) ⟩
$a + 0$
= ⟨ Identity of + (15.3) ⟩
a

Three different variables named "a"!

Calculational Proofs of Theorems — (15.17) — Renamed Theorem Variables

(15.3x) **Identity of +** $0 + x = x$	(15.13y) **Unary minus** $y + (-y) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$
= ⟨ Identity of + (15.3x) ⟩
$0 + -(-a)$
= ⟨ Unary minus (15.13y) ⟩
$a + (-a) + -(-a)$
= ⟨ Unary minus (15.13y) ⟩
$a + 0$
= ⟨ Identity of + (15.3x) ⟩
a

Three different variables "x", "y", "a".

Details of Applying Theorems — (15.17) with Explicit Substitutions I

(15.3x) **Identity of +** $0 + x = x$	(15.13y) **Unary minus** $y + (-y) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$
$=$ ⟨ Identity of + (15.3x) with $x := -(-a)$ ⟩ $\quad (0 + x = x)[x := -(-a)] \quad = \quad (0 + -(-a) = -(-a))$
$0 + -(-a)$
$=$ ⟨ Unary minus (15.13y) with $y := a$ ⟩ $\quad (y + (-y) = 0)[y := a] \quad = \quad (a + (-a) = 0)$
$a + (-a) + -(-a)$
$=$ ⟨ Unary minus (15.13y) with $y := -a$ ⟩ $\quad (y + (-y) = 0)[y := -a] \quad = \quad (-a + (-(-a)) = 0)$
$a + 0$
$=$ ⟨ Identity of + (15.3x) with $x := a$ ⟩ $\quad (0 + x = x)[x := a)] \quad = \quad (0 + a = a$
a

Details of Applying Theorems — (15.17) with Explicit Substitutions II

(15.3) **Identity of +** $0 + a = a$	(15.13) **Unary minus** $a + (-a) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$
$=$ ⟨ Identity of + (15.3) with $a := -(-a)$ ⟩
$0 + -(-a)$
$=$ ⟨ Unary minus (15.13) with $a := a$ ⟩
$a + (-a) + -(-a)$
$=$ ⟨ Unary minus (15.13) with $a := -a$ ⟩
$a + 0$
$=$ ⟨ Identity of + (15.3) with $a := a$ ⟩
a

Three different variables named "a"!

Specifying Substitutions for Theorem Application in CalcCheck

Theorem (15.19) "Distributivity of unary minus over +": $-(a + b) = (-a) + (-b)$
Proof:

$-(a + b)$
$=$ ⟨ (15.20) with `$a := a + b$` ⟩
$(-1) \cdot (a + b)$
$=$ ⟨ "Distributivity of · over +" with `$a, b, c := -1, a, b$` ⟩
$(-1) \cdot a + (-1) \cdot b$
$=$ ⟨ (15.20) with `$a := b$` ⟩
$(-1) \cdot a + -b$
$=$ ⟨ (15.20) with `$a := a$` ⟩
$(-a) + (-b)$

Theorem (15.20):
$-a = (-1) \cdot a$

- Backquotes enclose math embedded in English. (Markdown convention)
- Substitution notation as in LADM: $\textit{variables} := \textit{expressions}$
- ":=" reads "becomes" or "is/are replaced with"
- ":=" is entered by typing "\:=" or "\becomes"!
- The variable list has the same length as the expression list.
- No variable occurs twice in the variable list.
- $\text{CalcCheck}_{\text{Web}}$ notebooks "with rigid matching" **require** all theorem variables to be substituted.
  "Rigid matching" means: The theorems you specify need to match without substitution.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-06

Applying Equations, Boolean Expressions, Propositional Calculus

---

Plan for Today

- **Anatomy of calculation** based on **Substitution** (LADM 1.3–1.5):
  - **Inference rule Substitution:** Justifies applying instances of theorems:

$$
\begin{array}{ll}
 & 2 \cdot y \ + \ -(2 \cdot y) \\
= & \langle \text{“Unary minus” } a \ + \ -a \ = \ 0 \text{ with ‘} a \ := \ 2 \cdot y \text{’} \rangle \\
 & 0
\end{array}
$$

  - **Inference rule Leibniz:** Justifies applying (instances of) **equational** theorems deeper inside expressions:

$$
\begin{array}{ll}
 & 2 \cdot x + 3 \cdot (y \ - \ 5 \cdot (4 \cdot x + 7)) \\
= & \langle \text{“Subtraction” } a \ - \ b \ = \ a \ + \ -b \text{ with ‘} a, b \ := \ y, 5 \cdot (4 \cdot x + 7) \text{’} \rangle \\
 & 2 \cdot x + 3 \cdot (y \ + \ -(5 \cdot (4 \cdot x + 7)))
\end{array}
$$

- LADM Chapter 2: Boolean Expressions
  - Meaning of Boolean Operators
  - Equality versus Equivalence
  - Satisfiability and Validity

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-06

Part 1: Foundations of Applying Equations in Context

What is an Inference Rule?

$$\frac{\text{premise}_1 \quad \ldots \quad \text{premise}_n}{\text{conclusion}}$$

- **If all the premises are theorems,**
  then the conclusion is a theorem.
- A theorem is a "proved truth"
  — either an axiom,
  — or the result of an inference rule application.
- *Inference rules are the building blocks of proofs.*
- The premises are also called hypotheses.
- The conclusion and each premise all have to be Boolean.
- **Axioms** are inference rules with zero premises

Inference Rule: Substitution

(1.1) **Substitution:** $\dfrac{E}{E[x := R]}$ "If E is a theorem, then $E[x := R]$ is a theorem as well"

Example:

If $a + 0 = a$ is a theorem, "Identity of +"

then $3 \cdot q + 0 = 3 \cdot q$ is also a theorem. "Identity of +" with '$a := 3 \cdot q$'

$$\frac{a + 0 = a}{(a + 0 = a)[a := 3 \cdot q]} \qquad \frac{a + 0 = a}{3 \cdot q + 0 = 3 \cdot q}$$

Inference Rule Scheme: Substitution

(1.1) **Substitution:** $\dfrac{E}{E[x := R]}$ "If E is a theorem, then $E[x := R]$ is a theorem as well"

Really an **inference rule scheme**:
works for **every (well-typed) combination** of

- expression E,
- variable x, and
- expression R.

Example:

If $a + 0 = a$ is a theorem,
then $3 \cdot q + 0 = 3 \cdot q$ is also a theorem.

$$\frac{a + 0 = a}{3 \cdot q + 0 = 3 \cdot q}$$

- expression E is $a + 0 = a$
- the variable x substituted into is a
- the substituted expression R is $3 \cdot q$

Inference Rule Scheme: Substitution — Also for Simultaneous Substitution

(1.1) **Substitution:** $\dfrac{E}{E[x := R]}$

Really an **inference rule scheme**:
works for **every (well-typed) combination** of

- expression E,
- variable **list** x, and
- **corresponding** expression **list** R.

Example:

If $a + b \;=\; b + a$ is a theorem,
then $2 \cdot y + 3 \;=\; 3 + 2 \cdot y$ is also a theorem.

$$\frac{a + b \;=\; b + a}{2 \cdot y + 3 \;=\; 3 + 2 \cdot y}$$

- expression E is $a + b \;=\; b + a$
- variable list x is a, b
- corresponding expression list R is $2 \cdot y, 3$

Logical Definition of Equality

Two **axioms** (i.e., postulated as theorems):

- (1.2) **Reflexivity of =:** $x = x$
- (1.3) **Symmetry of =:** $(x = y) = (y = x)$

Two **inference rule schemes**:

- (1.4) **Transitivity of =:** $\dfrac{X = Y \qquad Y = Z}{X = Z}$
- (1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$

— the rule of "replacing equals for equals"

Using Leibniz' Rule in (15.21)

Given: (15.20) $-\,a = (-\,1) \cdot a$

$$\frac{X = Y}{E[z := X] = E[z := Y]}$$

Proving (15.21) $(-\,a) \cdot b = a \cdot (-\,b)$**:**

$$\begin{array}{ll} & (-\,a) \cdot b \\ = & \langle \text{(15.20) — via Leibniz (1.5) with } E \text{ chosen as } z \cdot b \rangle \\ & ((-\,1) \cdot a) \cdot b \\ = & \langle \text{Associativity (15.1) and Symmetry (15.2) of } \cdot \rangle \\ & a \cdot ((-\,1) \cdot b) \\ = & \langle \text{(15.20)} \rangle \\ & a \cdot (-\,b) \end{array}$$

Using Leibniz together with Substitution in (15.21)

Given: (15.20) $-\,a = (-\,1)\cdot a$

$$\frac{X = Y}{E[z := X] = E[z := Y]}$$

Proving (15.21) $(-\,a)\cdot b = a\cdot(-\,b)$**:**

$(-\,a)\cdot b$

$=$ ⟨ (15.20) — via Leibniz (1.5) with E chosen as $z\cdot b$ ⟩

$((-\,1)\cdot a)\cdot b$

$=$ ⟨ Associativity (15.1) and Symmetry (15.2) of $\cdot$ ⟩

$a\cdot((-\,1)\cdot b)$

$=$ ⟨ (15.20) with $a := b$ — via Leibniz (1.5) with E chosen as $a\cdot z$ ⟩

$a\cdot(-\,b)$

Using Leibniz together with Substitution in (15.21)

Theorem (15.21): $(-\,a)\cdot b = a\cdot(-\,b)$

Proof:

$(-\,a)\cdot b$

$=$ ⟨ Substitution ⟩

$(z\cdot b)[z := -\,a]$

$=$ ⟨ (15.20) — via "Leibniz" with `z · b` as `E` ⟩

$(z\cdot b)[z := (-\,1)\cdot a]$

$=$ ⟨ Substitution ⟩

$(-\,1)\cdot a\cdot b$

$=$ ⟨ "Symmetry of $\cdot$" ⟩

$a\cdot(-\,1)\cdot b$

$=$ ⟨ Substitution ⟩

$(a\cdot z)[z := (-\,1)\cdot b]$

$=$ ⟨ (15.20) with \`$a := b$\` — via "Leibniz" with `a · z` as `E` ⟩

$(a\cdot z)[z := -\,b]$

$=$ ⟨ Substitution ⟩

$a\cdot(-\,b)$

"Leibniz": $$\frac{X = Y}{E[z := X] = E[z := Y]}$$

(15.20) $-\,a = (-\,1)\cdot a$

Combining Leibniz' Rule with Substitution

(1.5) **Leibniz:** $$\frac{X = Y}{E[z := X] = E[z := Y]}$$

(15.20) $-\,a = (-\,1)\cdot a$

(1.1) **Substitution:** $$\frac{F}{F[v := R]}$$

Using Leibniz:	**Using them together:**	Example:
$E[z := X]$	$E[z := X[v := R]]$	$a\cdot((-\,1)\cdot b)$
$=$ ⟨ $X = Y$ ⟩	$=$ ⟨ $X = Y$ ⟩	$=$ ⟨ (15.20) with $a := b$ — E is $a\cdot z$ ⟩
$E[z := Y]$	$E[z := Y[v := R]]$	$a\cdot(-\,b)$

Justification:

$$\dfrac{\dfrac{X = Y}{X[v := R] = Y[v := R]}\ \text{Substitution (1.1)}}{E[z := X[v := R]] = E[z := Y[v := R]]}\ \text{Leibniz (1.5)}$$

Automatic Application of Associativity and Symmetry Laws

Axiom (15.1) (15.1a) "Associativity of +": $(a + b) + c = a + (b + c)$
Axiom (15.1) (15.1b) "Associativity of ·": $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
Axiom (15.2) (15.2a) "Symmetry of +": $a + b = b + a$
Axiom (15.2) (15.2b) "Symmetry of ·": $a \cdot b = b \cdot a$

- You have been trained to reason "up to symmetry and associativity"
- Making symmetry and associativity steps explicit is
  - **always allowed**
  - sometimes **very useful for readability**
- CalcCheck allows selective activation of symmetry and associativity laws

  $\Longrightarrow$ "Exercise . . . / Assignment . . . : [. . .] **without automatic associativity and symmetry**"

  $\Longrightarrow$ Having to make symmetry and associativity steps explicit can be tedious. . .

(15.17) with Explicit Associativity and Symmetry Steps

(15.3) **Identity of +**	$0 + a = a$	(15.13) **Unary minus**	$a + (-a) = 0$

Proving (15.17) $-(-a) = a$**:**

$$
\begin{aligned}
& -(-a) \\
= \; & \langle \text{Identity of + (15.3)} \rangle \\
& 0 + -(-a) \\
= \; & \langle \text{Unary minus (15.13)} \rangle \\
& (a + (-a)) + -(-a) \\
= \; & \langle \text{Associativity of + (15.1)} \rangle \\
& a + ((-a) + -(-a)) \\
= \; & \langle \text{Unary minus (15.13)} \rangle \\
& a + 0 \\
= \; & \langle \text{Symmetry of + (15.2)} \rangle \\
& 0 + a \\
= \; & \langle \text{Identity of + (15.3)} \rangle \\
& a
\end{aligned}
$$

Some Property Names

Let $\odot$ and $\oplus$ be binary operators and $\square$ be a constant.

*($\odot$ and $\oplus$ and $\square$ are **metavariables** for operators respectively constants.)*

- "**$\odot$ is symmetric**": $x \odot y = y \odot x$
- "**$\odot$ is associative**": $(x \odot y) \odot z = x \odot (y \odot z)$
- "**$\odot$ is mutually associative with $\oplus$** (from the left)":

  $(x \odot y) \oplus z = x \odot (y \oplus z)$

  For example:
  - $+$ **is** mutually associative with $-$:

    $(x + y) - z = x + (y - z)$
  - $-$ **is not** mutually associative with $+$:

    $(5 - 2) + 3 \neq 5 - (2 + 3)$

Some Property Names (ctd.)

Let $\odot$ and $\oplus$ be binary operators and $\Box$ be a constant.

*($\odot$ and $\oplus$ and $\Box$ are **metavariables** for operators respectively constants.)*

- "**$\odot$ is idempotent**": $x \odot x \ = \ x$
- "**$\Box$ is a left-identity (or left-unit) of $\odot$**": $\Box \odot x \ = \ x$
- "**$\Box$ is a right-identity (or right-unit) of $\odot$**": $x \odot \Box \ = \ x$
- "**$\Box$ is a identity (or unit) of $\odot$**": $\Box \odot x \ = \ x \ = \ x \odot \Box$
- "**$\Box$ is a left-zero of $\odot$**": $\Box \odot x \ = \ \Box$
- "**$\Box$ is a right-zero of $\odot$**": $x \odot \Box \ = \ \Box$
- "**$\Box$ is a zero of $\odot$**": $\Box \odot x \ = \ \Box \ = \ x \odot \Box$
- "**$\odot$ distributes over $\oplus$ from the left**": $x \odot (y \oplus z) \ = \ (x \odot y) \oplus (x \odot z)$
- "**$\odot$ distributes over $\oplus$ from the right**": $(y \oplus z) \odot x \ = \ (y \odot x) \oplus (z \odot x)$
- "**$\odot$ distributes over $\oplus$**": $\odot$ distributes over $\oplus$ from the left **and** $\odot$ distributes over $\oplus$ from the right

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-06

Part 2: Boolean Expression

Truth Values

Boolean constants/values: *false*, *true*

The type of Boolean values: $\mathbb{B}$

— This is the type of propositions, for example: $(x = 1) : \mathbb{B}$

— For any type t, equality $\_{=}\_$ can be used on expressions of that type: $\_{=}\_ : t \to t \to \mathbb{B}$

Boolean operators:

- $\neg\_ : \mathbb{B} \to \mathbb{B}$ — negation, complement, "logical not", `\lnot`
- $\_\land\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — conjunction, "logical and", `\land`
- $\_\lor\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — disjunction, "logical or", "inclusive or", `\lor`
- $\_\Rightarrow\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — implication, "implies", "if … then …", `\=>`, `\implies`
- $\_\equiv\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — equivalence, "if and only if", "iff", `\==`, `\equiv`
- $\_\not\equiv\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — inequivalence, "exclusive or", `\nequiv`

Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- . (function application)
- unary prefix operators $+$, $-$, $\neg$, $\#$, $\sim$, $\mathcal{P}$
- $**$
- $\cdot$ $/$ $\div$ mod gcd
- $+$ $-$ $\cup$ $\cap$ $\times$ $\circ$ $\bullet$
- $\downarrow$ $\uparrow$
- $\#$
- $\triangleleft$ $\triangleright$ $\hat{}$
- $=$ $\neq$ $<$ $>$ $\in$ $\subset$ $\subseteq$ $\supset$ $\supseteq$ $|$ (conjunctional)
- $\vee$ $\wedge$
- $\Rightarrow$ $\not\Rightarrow$ $\Leftarrow$ $\not\Leftarrow$
- $\equiv$ $\not\equiv$ **(lowest precedence)**

All non-associative binary infix operators associate to the left, except $**$, $\triangleleft$, $\Rightarrow$, $\rightarrow$, which associate to the right.

Binary Boolean Operators: Conjunction

Args.		$\wedge$	
F	F	F	The moon is green, and $2 + 2 = 7$.
F	T	F	The moon is green, and $1 + 1 = 2$.
T	F	F	$1 + 1 = 2$, and the moon is green.
T	T	T	$1 + 1 = 2$, and the sun is a star.

Binary Boolean Operators: Disjunction

Args.		$\vee$	
F	F	F	The moon is green, or $2 + 2 = 7$.
F	T	T	The moon is green, or $1 + 1 = 2$.
T	F	T	$1 + 1 = 2$, or the moon is green.
T	T	T	$1 + 1 = 2$, or the sun is a star.

This is known as "inclusive or" — see textbook p.34.

Binary Boolean Operators: Implication

Args.		$\Rightarrow$	
F	F	T	If the moon is green, then $2 + 2 = 7$.
F	T	T	If the moon is green, then $1 + 1 = 2$.
T	F	F	If $1 + 1 = 2$, then the moon is green.
T	T	T	If $1 + 1 = 2$, then the sun is a star.

$$p \Rightarrow q \quad \equiv \quad \neg p \vee q$$
$$\neg p \Rightarrow q \quad \equiv \quad \neg\neg p \vee q$$
$$\neg p \Rightarrow q \quad \equiv \quad p \vee q$$

If you don't eat your spinach, I'll spank you.	$\equiv$	You eat your spinach, or I'll spank you.

Binary Boolean Operators: Consequence

Args.		$\Leftarrow$	
F	F	T	The moon is green **if** $2 + 2 = 7$.
F	T	F	The moon is green **if** $1 + 1 = 2$.
T	F	T	$1 + 1 = 2$ **if** the moon is green.
T	T	T	$1 + 1 = 2$ **if** the sun is a star.

$$p \Leftarrow q \quad \equiv \quad p \vee \neg q$$

Binary Boolean Operators: Equivalence

Equality of Boolean values is also called **equivalence** and written $\equiv$
(In some other places: $\Leftrightarrow$)

$p \equiv q$ can be read as: p is equivalent to q
or: p exactly when q
or: p if-and-only-if q
or: p iff q

p	q	$p \equiv q$	
false	*false*	*true*	The moon is green **iff** $2 + 2 = 7$.
false	*true*	*false*	The moon is green **iff** $1 + 1 = 2$.
true	*false*	*false*	$1 + 1 = 2$ **iff** the moon is green.
true	*true*	*true*	$1 + 1 = 2$ **iff** the sun is a star.

Binary Boolean Operators: Inequivalence ("exclusive or")

Args.		$\not\equiv$	
F	F	F	Either the moon is green, or $2 + 2 = 7$.
F	T	T	Either the moon is green, or $1 + 1 = 2$.
T	F	T	Either $1 + 1 = 2$, or the moon is green.
T	T	F	Either $1 + 1 = 2$, or the sun is a star.

Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- $.$ (function application)
- unary prefix operators $+$, $-$, $\neg$, $\#$, $\sim$, $\mathcal{P}$
- $**$
- $\cdot$ $/$ $\div$ mod gcd
- $+$ $-$ $\cup$ $\cap$ $\times$ $\circ$ $\bullet$
- $\downarrow$ $\uparrow$
- $\#$
- $\triangleleft$ $\triangleright$ $\hat{}$
- $=$ $\neq$ $<$ $>$ $\in$ $\subset$ $\subseteq$ $\supset$ $\supseteq$ $|$ (conjunctional)
- $\vee$ $\wedge$
- $\Rightarrow$ $\not\Rightarrow$ $\Leftarrow$ $\not\Leftarrow$
- $\equiv$ $\not\equiv$ **(lowest precedence)**

All non-associative binary infix operators associate to the left, except $**$, $\triangleleft$, $\Rightarrow$, $\rightarrow$, which associate to the right.

Expression Evaluation (LADM 1.1 end)

- $2 \cdot 3 + 4$
- $2 \cdot (3 + 4)$
- $2 \cdot y + 4$

A **state** is a "list of variables with associated values". E.g.:

$$s_1 = [\ (x, 5),\ (y, 6)\ ]$$ — (using Haskell notation for informal lists)

Evaluating an expression in a state:
"Replace variables with their values; then evaluate":

- $x - y + 2$ in state s_1
  $\longrightarrow\ 5 - 6 + 2\ \longrightarrow\ (5 - 6) + 2\ \longrightarrow\ (-1) + 2\ \longrightarrow\ 1$
- $x \cdot 2 + y$
- $x \cdot (2 + y)$
- $x \cdot (z + y)$

Evaluation of Boolean Expressions

Example: Using the state $\langle (p, \mathit{false}), (q, \mathit{true}), (r, \mathit{false}) \rangle$:

$$
\begin{aligned}
& p \vee (q \wedge \neg r) \\
= \; & \langle \text{ replace variables with state values } \rangle \\
& \mathit{false} \vee (\mathit{true} \wedge \neg \mathit{false}) \\
= \; & \langle \quad \neg \mathit{false} = \mathit{true} \quad \rangle \\
& \mathit{false} \vee (\mathit{true} \wedge \mathit{true}) \\
= \; & \langle \quad \mathit{true} \wedge \mathit{true} = \mathit{true} \quad \rangle \\
& \mathit{false} \vee \mathit{true} \\
= \; & \langle \quad \mathit{false} \vee \mathit{true} = \mathit{true} \quad \rangle \\
& \mathit{true}
\end{aligned}
$$

			$\wedge$					$\neq$	$\vee$	nor	$=$		$\Leftarrow$		$\Rightarrow$	nand	
F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
T	F	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T

Evaluation of Boolean Expressions Using Truth Tables

p	q	$\neg p$	$q \wedge \neg p$	$p \vee (q \wedge \neg p)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

- Identify variables
- Identify subexpressions
- Enumerate possible states (of the variables)
- Evaluate (sub-)expressions in all states

Validity and Satisfiability

p	q	$\neg p$	$q \wedge \neg p$	$p \vee (q \wedge \neg p)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

- A boolean expression is **satisfied** in state s iff it evaluates to *true* in state s.
- A boolean expression is **satisfiable** iff there is a state in which it is satisfied.
- A boolean expression is **valid** iff it is satisfied in every state.
- A valid boolean expression is called a **tautology**.
- A boolean expression is called a **contradiction** iff it evaluates to *false* in every state.
- Two boolean expressions are called **logically equivalent** iff they evaluate to the same truth value in every state.

These definitions rely on states / truth tables: **Semantic concepts**

Modeling English Propositions 1

- Henry VIII had one son and Cleopatra had two.

  Henry VIII had one son and Cleopatra had two sons.

  Declarations:

  h :≡ Henry VIII had one son

  c :≡ Cleopatra had two sons

  Formalisation:

  $h \wedge c$

Modeling English Propositions — Recipe

- Transform into shape with clear subpropositions
- Introduce Boolean variables to denote subpropositions
- Replace these subpropositions by their corresponding Boolean variables
- Translate the result into a Boolean expression, using (no perfect translation rules are possible!) **for example**:

and, but	becomes	$\wedge$
or	becomes	$\vee$
not	becomes	$\neg$
it is not the case that	becomes	$\neg$
if p then q	becomes	$p \Rightarrow q$

Ladies or Tigers

Raymond Smullyan provides, in **The Lady or the Tiger?**, the following context for a number of puzzles to follow:

> [...] the king explained to the prisoner that each of the two rooms contained either a lady or a tiger, but it *could* be that there were tigers in both rooms, or ladies in both rooms, or then again, maybe one room contained a lady and the other room a tiger.

In the first case, the following signs are on the doors of the rooms:

1	2
In this room there is a lady, and in the other room there is a tiger.	In one of these rooms there is a lady, and in one of these rooms there is a tiger.

We are told that one of the signs is true, and the other one is false.

"Which door would you open (assuming, of course, that you preferred the lady to the tiger)?"

Ladies or Tigers — The First Case — Starting Formalisation

Raymond Smullyan provides, in **The Lady or the Tiger?**, the following context for a number of puzzles to follow:

> [...] the king explained to the prisoner that each of the two rooms contained either a lady or a tiger, but it *could* be that there were tigers in both rooms, or ladies in both rooms, or then again, maybe one room contained a lady and the other room a tiger.

$R1L$:= There is a lady in room 1

$R1T$:= There is a tiger in room 1

$R2L$:= There is a lady in room 2

$R2T$:= There is a tiger in room 2

> [...] We are told that one of the signs is true, and the other one is false.

S_1 := Sign 1 is true

S_2 := Sign 2 is true

Equality "=" versus Equivalence "≡"

The operators = (**as Boolean operator**) and ≡

- have the **same meaning** (represent the same function),
- but **are used with different notational conventions:**
  - different precedences (≡ has lowest)
  - different **chaining behaviour**:
    - ≡ is associative:

$$(p \equiv q \equiv r) \quad = \quad ((p \equiv q) \equiv r) \quad = \quad (p \equiv (q \equiv r))$$

    - = is **conjunctional**:

$$(x = y = z) \quad = \quad ((x = y) \quad \wedge \quad (y = z))$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-10

Command Correctness, Propositional Calculus

Plan for Today

- **Reasoning about Assignment Commands in Imperative Programs** (≈ LADM 1.6):
  - Correctness of programs with respect to pre-/post-condition specifications
  - Reasoning using "Hoare logic"

$\Longrightarrow$ Homework 3 – due Friday, 8:30

- **Propositional Calculus (LADM Chapter 3)**
  - Equivalence
  - Negation, Inequivalence
  - Disjunction
  - Conjunction

$\Longrightarrow$ Exercises 2.4–2.7

$\Longrightarrow$ **Work through at least Exercise 2.4 before your tutorial!**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-10

Part 1: Correctness of Assignment Commands

States as Program States

LADM 1.1: A **state** is a "list of variables with associated values". E.g.:

$s_1 = [\ (x, 5),\ (y, 6)\ ]$ — (using Haskell notation for informal lists)

Evaluating an expression in a state:
"Replace variables with their values; then evaluate"

- In logic, "states" are usually called "variable assignments"
- States can serve as a mathematical model of **program states**
- Execution of imperative programs induces **state transformation**:

$$
\begin{array}{ll}
 & [\ (x,5),\ (y,6)\ ] \\
\leadsto & \langle \quad x := x + y \quad \rangle \\
 & [\ (x,11),\ (y,6)\ ] \\
\leadsto & \langle \quad y := x - y \quad \rangle \\
 & [\ (x,11),\ (y,5)\ ]
\end{array}
$$

State Predicates

- Execution of imperative programs induces **state transformation**:

$[\ (x,5),\ (y,6)\ ]$ ▪▪▪▪▪▪ \`$x < y$\` holds

$\leadsto \langle \quad x := x + y \quad \rangle$

$[\ (x,11),\ (y,6)\ ]$ ▪▪▪▪▪▪ \`$x < y$\` does not hold

$\leadsto \langle \quad y := x - y \quad \rangle$

$[\ (x,11),\ (y,5)\ ]$ ▪▪▪▪▪▪ \`$x < y$\` does not hold

- Boolean expressions containing variables can be used as **state predicates**:

P "holds in state s" iff P evaluates to *true* in state s

Precondition-Postcondition Specifications

- Program correctness statement in LADM (and much current use):

$$\{\ P\ \}\ C\ \{\ Q\ \}$$

This is called a "Hoare triple".

- **Meaning:** If command C is started in a state in which the **precondition** P holds, then it will terminate only in a state in which the **postcondition** Q holds.

- Hoare's original notation:

$$P\ \{\ C\ \}\ Q$$

- **Dynamic logic** notation (will be used in CALCCHECK):

$$P \Rightarrow\!\![\ C\ ]\ Q$$

Correctness of Assignment Commands

- *Recall:* Hoare triple: $\{\ P\ \}\ C\ \{\ Q\ \}$
- **Dynamic logic** notation (will be used in CALCCHECK): $P \Rightarrow\!\![\ C\ ]\ Q$
- **Meaning:** If command C is started in a state in which the **precondition** P holds, then it will terminate only in a state in which the **postcondition** Q holds.
- **Assignment Axiom:** $\{\ Q[x := E]\ \}\ x := E\ \{\ Q\ \}$ $\boxed{Q[x := E] \quad \Rightarrow\!\![\ x := E\ ] \quad Q}$
- **Example:**
  - $(x = 5)[x := x + 1] \quad \Rightarrow\!\![\ x := x + 1\ ] \quad x = 5$
  - $(x + 1 = 5) \quad \Rightarrow\!\![\ x := x + 1\ ] \quad x = 5$
  -
    ```
          x + 1 = 5
    ≡                 ⟨ Substitution ⟩
          (x = 5)[x := x + 1]
    ⇒[ x := x + 1 ]   ⟨ Assignment ⟩
          x = 5
    ```
- **Substitution ":=": One Unicode character; type "\:="** | **Assignment ":=": Two characters; type ":="**

Correctness of Assignment Commands — Longer Example

- *Recall:* Hoare triple: $\{ P \}\ C\ \{ Q \}$
- **Dynamic logic** notation (will be used in CALCCHECK):

$$P \Rrightarrow[\ C\ ]\ Q$$

- **Meaning:** If command C is started in a state in which the **precondition** P holds, then it will terminate only in a state in which the **postcondition** Q holds.
- **Assignment Axiom:** $\{ Q[x := E] \}\ x \mathrel{\textbf{:=}} E\ \{ Q \}$ $\qquad Q[x := E] \quad \Rrightarrow[\ x \mathrel{\textbf{:=}} E\ ] \quad Q$
- **Longer example** (**these proofs are developed from the bottom to the top!**):

$$
\begin{array}{ll}
 & true \\
\equiv & \langle \text{Zero of } \vee \rangle \\
 & 1 = 0 \vee true \\
\equiv & \langle \text{Reflexivity of } = \rangle \\
 & 1 = 0 \vee 1 = 1 \\
\equiv & \langle \text{Substitution} \rangle \\
 & (x = 0 \vee x = 1)[x := 1] \\
\Rrightarrow[\ x \mathrel{\textbf{:=}} 1\ ] & \langle \text{Assignment} \rangle \\
 & x = 0 \vee x = 1
\end{array}
$$

Example Proof for a Sequence of Assignments

Lemma (4):

$$
\begin{array}{l}
\quad x = 5 \\
\Rrightarrow[\ y := x + 1\ ;\\
\quad\ x := y + y \\
\ ] \\
\quad x = 12
\end{array}
$$

Proof:

$$
\begin{array}{l}
\quad x = 5 \\
\equiv \langle \text{“Cancellation of +”} \rangle \\
\quad x + 1 = 5 + 1 \\
\equiv \langle \text{Fact `5 + 1 = 6`} \rangle \\
\quad x + 1 = 6 \\
\equiv \langle \text{Substitution} \rangle \\
\quad (y = 6)[y := x + 1] \\
\Rrightarrow[\ y := x + 1\ ]\ \langle \text{“Assignment”} \rangle \\
\quad y = 6 \\
\equiv \langle \text{“Cancellation of} \cdot \text{” with Fact `2} \neq 0\text{`} \rangle \\
\quad 2 \cdot y = 2 \cdot 6 \\
\equiv \langle \text{Evaluation} \rangle \\
\quad (1 + 1) \cdot y = 12 \\
\equiv \langle \text{“Distributivity of} \cdot \text{over +”} \rangle \\
\quad 1 \cdot y + 1 \cdot y = 12 \\
\equiv \langle \text{“Identity of} \cdot \text{”} \rangle \\
\quad y + y = 12 \\
\equiv \langle \text{Substitution} \rangle \\
\quad (x = 12)[x := y + y] \\
\Rrightarrow[\ x := y + y\ ]\ \langle \text{“Assignment”} \rangle \\
\quad x = 12
\end{array}
$$

Read and write such “_⇛[_]_” proofs from the bottom to the top!

Sequential Composition of Commands

Primitive inference rule “SEQ”:

$$\frac{`\{ P \}\ C_1\ \{ Q \}`\ ,\ `\{ Q \}\ C_2\ \{ R \}`}{`\{ P \}\ C_1\ ;\ C_2\ \{ R \}`}$$

Primitive inference rule “Sequence”:

$$\frac{`P \Rrightarrow[\ C_1\ ]\ Q`\ ,\ `Q \Rrightarrow[\ C_2\ ]\ R`}{`P \Rrightarrow[\ C_1\ ;\ C_2\ ]\ R`}$$

- Activated as transitivity rule
- Therefore used implicitly in calculations, e.g., proving $P \Rrightarrow[\ C_1\ ;\ C_2\ ]\ R$ by:

$$
\begin{array}{ll}
 & P \\
\Rrightarrow[\ C_1\ ] & \langle \ \dots\ \rangle \\
 & Q \\
\Rrightarrow[\ C_2\ ] & \langle \ \dots\ \rangle \\
 & R
\end{array}
$$

- No need to refer to this rule explicitly.

Specification Pattern: "Auxiliary Variables"

Lemma: $x = x_0 \quad \Rightarrow[\ x := x + 1\ ] \quad x = x_0 + 1$

Proof:

$$\begin{array}{ll} & x = x_0 \\ \equiv & \langle \text{Cancellation of } + \rangle \\ & x + 1 = x_0 + 1 \\ \equiv & \langle \text{Substitution} \rangle \\ & (x = x_0 + 1)[x := x + 1] \\ \Rightarrow[\ x := x + 1\ ] & \langle \text{Assignment} \rangle \\ & x = x_0 + 1 \end{array}$$

Variable x_0

- is not assigned in the program
- "remembers" the value of x in the start state for referencing it in the postcondition

Such variables are called "auxiliary variables" in the context of pre-/post-condition specification.

What Does this C Program Fragment Do?

Let x and y be variables of type **int**.

```
x = x + y;

y = x − y;

x = x − y;
```

(There is a similar-looking program in H3...)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-10

Part 2: LADM Propositional Calculus: $\equiv$, $\neg$, $\not\equiv$, $\vee$, $\wedge$

Propositional Calculus

Calculus: method of reasoning by calculation with symbols

Propositional Calculus: calculating

- with Boolean expressions
- containing propositional variables

The Textbook's Propositional Calculus: **Equational Logic E**

- a set of **axioms** defining operator properties
- **four inference rules**:
  - (1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$ — We can apply equalities inside expressions.
  - (1.4) **Transitivity:** $\dfrac{X = Y \qquad Y = Z}{X = Z}$ — We can chain equalities.
  - (1.1) **Substitution:** $\dfrac{E}{E[x := R]}$ — We can can use substitution instances of theorems.
  - **Equanimity:** $\dfrac{X = Y \qquad X}{Y}$ — — This is …

Theorems — Remember!

A **theorem** is

- **either** an **axiom**
- **or** the **conclusion of an inference rule** where the premises are theorems
- **or** a Boolean expression **proved** (using the inference rules) **equal** to an axiom or a previously proved **theorem**. ("— This is …")

  Such proofs will be presented in the **calculational style**.

Note:

- **The theorem definition does not use evaluation/validity**
- But:
  - All theorems in **E** are valid
  - All valid Boolean expressions are theorems in **E**
- **Important:**
  - **We will prove theorems without using validity!**
  - This trains an **essential mathematical skill!**

Equivalence Axioms

(3.1) **Axiom, Associativity of** $\equiv$: $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of** $\equiv$: $p \equiv q \equiv q \equiv p$

Can be used as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

Example theorem — shown differently in the textbook:

Proving $p \equiv p \equiv q \equiv q$**:**

$$p \equiv p \equiv q \equiv q$$

$=$ ⟨ (3.2) Symmetry of $\equiv$, with $p, q := p, q \equiv q$ ⟩

$$p \equiv q \equiv q \equiv p$$ — This is (3.2) Symmetry of $\equiv$

Equivalence Axioms — Example Proof with Parentheses

(3.1) **Axiom, Associativity of ≡ :** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡ :** $p \equiv q \equiv q \equiv p$

Can be used as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

Example theorem — shown differently in the textbook:

Proving $p \equiv p \equiv q \equiv q$**:**

$p \equiv (p \equiv (q \equiv q))$

$\equiv$ ⟨ (3.2) Symmetry of ≡, with $p, q := p, (q \equiv q)$ — via Leibniz with $p \equiv z$ as E ⟩

$p \equiv ((q \equiv q) \equiv p)$ — This is (3.2) Symmetry of ≡

Equivalence Axioms — Introducing *true*

(3.1) **Axiom, Associativity of ≡ :** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡ :** $p \equiv q \equiv q \equiv p$

Can be used as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

(3.3) **Axiom, Identity of ≡ :** $true \equiv q \equiv q$

Can be used as:

- $(true \equiv q) = q$
- $true = (q \equiv q)$

Equivalence Axioms, and Theorem (3.4)

(3.1) **Axiom, Associativity of ≡ :** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡ :** $p \equiv q \equiv q \equiv p$

(3.3) **Axiom, Identity of ≡ :** $true \equiv q \equiv q$

Can be used as: $true = (q \equiv q)$

The least interesting theorem:

Proving (3.4) *true***:**

$true$

$=$ ⟨ Identity of ≡ (3.3), with $q := true$ ⟩

$true \equiv true$

$=$ ⟨ Identity of ≡ (3.3), with $q := q$ — via Leibniz with $true \equiv z$ as E ⟩

$true \equiv q \equiv q$ — This is Identity of ≡ (3.3)

Equivalence Axioms and Theorems

(3.1) **Axiom, Associativity of $\equiv$:** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of $\equiv$:** $p \equiv q \equiv q \equiv p$ —

(3.3) **Axiom, Identity of $\equiv$:** $true \equiv q \equiv q$

Can be used as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

Theorems and Metatheorems:

(3.4) *true*

(3.5) **Reflexivity of $\equiv$:** $p \equiv p$

(3.6) **Proof Method**: To prove that $P \equiv Q$ is a theorem, transform P to Q or Q to P using Leibniz.

(3.7) **Metatheorem**: Any two theorems are equivalent.

Proof Method Equanimity: To prove P, prove $P \equiv Q$ where Q is a theorem. (Document via "– This is . . .".)

Special case: To prove P, prove $P \equiv true$.

Negation Axioms

(3.8) **Axiom, Definition of *false*:** $false \equiv \neg true$

(3.9) **Axiom, Commutativity of $\neg$ with $\equiv$:** $\neg(p \equiv q) \equiv \neg p \equiv q$

(LADM: "Distributivity of $\neg$ over $\equiv$")

Can be used as:

- $\neg(p \equiv q) \quad = \quad (\neg p \equiv q)$
- $(\neg(p \equiv q) \equiv \neg p) \quad = \quad q$
- $(\neg(p \equiv q) \equiv q) \quad = \quad \neg p$

(3.10) **Axiom, Definition of $\not\equiv$:** $(p \not\equiv q) \equiv \neg(p \equiv q)$

Negation Axioms and Theorems

(3.8) **Axiom, Definition of *false*:** $false \equiv \neg true$

(3.9) **Axiom, Commutativity of $\neg$ with $\equiv$:** $\neg(p \equiv q) \equiv \neg p \equiv q$

(3.10) **Axiom, Definition of $\not\equiv$:** $(p \not\equiv q) \equiv \neg(p \equiv q)$

Theorems:

(3.11) $\neg p \equiv q \equiv p \equiv \neg q$

— can be used as **"$\neg$ connection"**: $(\neg p \equiv q) \equiv (p \equiv \neg q)$

— can be used as **"Cancellation of $\neg$"**: $(\neg p \equiv \neg q) \equiv (p \equiv q)$

(3.12) **Double negation:** $\neg\neg p \equiv p$

(3.13) **Negation of *false*:** $\neg false \equiv true$

(3.14) $(p \not\equiv q) \equiv \neg p \equiv q$

(3.15) **Definition of $\neg$ via $\equiv$:** $\neg p \equiv p \equiv false$

Inequivalence Theorems

(3.16) **Symmetry of $\not\equiv$**: $(p \not\equiv q) \quad \equiv \quad (q \not\equiv p)$

(3.17) **Associativity of $\not\equiv$**: $((p \not\equiv q) \not\equiv r) \quad \equiv \quad (p \not\equiv (q \not\equiv r))$

(3.18) **Mutual associativity**: $((p \not\equiv q) \equiv r) \quad \equiv \quad (p \not\equiv (q \equiv r))$

(3.19) **Mutual interchangeability**: $p \not\equiv q \equiv r \quad \equiv \quad p \equiv q \not\equiv r$

Note: Mutual associativity is not (yet...) automated!

(But omission of parentheses is implemented, similar to

- $k - m + n$
- $k + m - n$
- $k - m - n$

— None of these has $m - n$ as subexpression!
— But the second one is equal to $k + (m - n)$ ...)

(3.23) Heuristic of Definition Elimination

To prove a theorem concerning an operator $\circ$ that is defined in terms of another, say $\bullet$, expand the definition of $\circ$ to arrive at a formula that contains $\bullet$; exploit properties of $\bullet$ to manipulate the formula, and then (possibly) reintroduce $\circ$ using its definition.

Textbook, p. 48

"Unfold-Fold strategy"

Inequivalence Theorems: Symmetry

(3.16) **Symmetry of $\not\equiv$**: $(p \not\equiv q) \equiv (q \not\equiv p)$

Proving (3.16) **Symmetry of $\not\equiv$:**

$p \not\equiv q$
= ⟨ (3.10) Definition of $\not\equiv$ ⟩ ······ **Unfold**
$\neg(p \equiv q)$
= ⟨ (3.2) Symmetry of $\equiv$ ⟩
$\neg(q \equiv p)$
= ⟨ (3.10) Definition of $\not\equiv$ ⟩ ······ **Fold**
$q \not\equiv p$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-12

Propositional Calculus: $\neg$, $\not\equiv$, $\vee$, $\wedge$

Equivalence Axioms

LADM p. 42 Footnote 2: "Remember that $=$ and $\equiv$ are interchangeable in formulas, without special mention (subject to the caveats mentioned in Sec. 2.2)."

Note: In CALCCHECK, "without special mention" is replaced with:
"Definition of $\equiv$": $(p \equiv q) = (p = q)$ (only for Boolean p and q)

(3.1) **Axiom, Associativity of $\equiv$:** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of $\equiv$:** $p \equiv q \equiv q \equiv p$

By associativity, can be read as:

- $(p \equiv q) \equiv (q \equiv p)$
- $p \equiv (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) \equiv p$

Therefore can be used for Leibniz as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

(3.3) **Axiom, Identity of $\equiv$:** $true \equiv q \equiv q$

Can be used as:

- $(true \equiv q) = q$
- $true = (q \equiv q)$

Equivalence Axioms, and Theorem (3.4)

(3.1) **Axiom, Associativity of $\equiv$:** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of $\equiv$:** $p \equiv q \equiv q \equiv p$

(3.3) **Axiom, Identity of $\equiv$:** $true \equiv q \equiv q$

Can be used as: $true = (q \equiv q)$

The least interesting theorem:

Proving (3.4) *true*:

$true$
$=$ ⟨ Identity of $\equiv$ (3.3), with $q := true$ ⟩
$true \equiv true$
$=$ ⟨ Identity of $\equiv$ (3.3), with $q := q$ — via Leibniz with $true \equiv z$ as E ⟩
$true \equiv q \equiv q$ — This is Identity of $\equiv$ (3.3)

Equivalence Axioms and Theorems

(3.1) **Axiom, Associativity of ≡:** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡:** $p \equiv q \equiv q \equiv p$ —

Can be used as:
- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

(3.3) **Axiom, Identity of ≡:** $true \equiv q \equiv q$

Theorems and Metatheorems:

(3.4) *true*

(3.5) **Reflexivity of ≡:** $p \equiv p$

(3.6) **Proof Method**: To prove that $P \equiv Q$ is a theorem, transform P to Q or Q to P using Leibniz.

(3.7) **Metatheorem**: Any two theorems are equivalent.

Proof Method Equanimity: To prove P, prove $P \equiv Q$ where Q is a theorem. (Document via "– This is . . . ".)

Special case: To prove P, prove $P \equiv true$.

Negation Axioms

(3.8) **Axiom, Definition of** ***false*****:** $false \equiv \neg true$

(3.9) **Axiom, Commutativity of ¬ with ≡:** $\neg(p \equiv q) \equiv \neg p \equiv q$

(LADM: "Distributivity of ¬ over ≡")

Can be used as:
- $\neg(p \equiv q) \quad = \quad (\neg p \equiv q)$
- $(\neg(p \equiv q) \equiv \neg p) \quad = \quad q$
- $(\neg(p \equiv q) \equiv q) \quad = \quad \neg p$

(3.10) **Axiom, Definition of ≢:** $(p \not\equiv q) \equiv \neg(p \equiv q)$

Negation Axioms and Theorems

(3.8) **Axiom, Definition of** ***false*****:** $false \equiv \neg true$

(3.9) **Axiom, Commutativity of ¬ with ≡:** $\neg(p \equiv q) \equiv \neg p \equiv q$

(3.10) **Axiom, Definition of ≢:** $(p \not\equiv q) \equiv \neg(p \equiv q)$

Theorems:

(3.11) $\neg p \equiv q \equiv p \equiv \neg q$

— can be used as "**¬ connection**": $(\neg p \equiv q) \equiv (p \equiv \neg q)$

— can be used as "**Cancellation of ¬**": $(\neg p \equiv \neg q) \equiv (p \equiv q)$

(3.12) **Double negation**: $\neg\neg p \equiv p$

(3.13) **Negation of** ***false*****:** $\neg false \equiv true$

(3.14) $(p \not\equiv q) \;\equiv\; \neg p \equiv q$

(3.15) **Definition of ¬ via ≡:** $\neg p \equiv p \equiv false$

Inequivalence Theorems

(3.16) **Symmetry of** $\not\equiv$:	$(p \not\equiv q)$	$\equiv$	$(q \not\equiv p)$
(3.17) **Associativity of** $\not\equiv$:	$((p \not\equiv q) \not\equiv r)$	$\equiv$	$(p \not\equiv (q \not\equiv r))$
(3.18) **Mutual associativity**:	$((p \not\equiv q) \equiv r)$	$\equiv$	$(p \not\equiv (q \equiv r))$
(3.19) **Mutual interchangeability**:	$p \not\equiv q \equiv r$	$\equiv$	$p \equiv q \not\equiv r$

Note: Mutual associativity is not (yet...) automated!

(But omission of parentheses is implemented, similar to

- $k - m + n$
- $k + m - n$
- $k - m - n$

— None of these has $m - n$ as subexpression!
— But the second one is equal to $k + (m - n)$ ...)

(3.23) Heuristic of Definition Elimination

To prove a theorem concerning an operator $\circ$ that is defined in terms of another, say $\bullet$, expand the definition of $\circ$ to arrive at a formula that contains $\bullet$; exploit properties of $\bullet$ to manipulate the formula, and then (possibly) reintroduce $\circ$ using its definition.

Textbook, p. 48

"Unfold-Fold strategy"

Inequivalence Theorems: Symmetry

(3.16) **Symmetry of** $\not\equiv$: $(p \not\equiv q) \equiv (q \not\equiv p)$

Proving (3.16) **Symmetry of** $\not\equiv$**:**

$p \not\equiv q$
$=$ ⟨ (3.10) Definition of $\not\equiv$ ⟩ ······ **Unfold**
$\neg(p \equiv q)$
$=$ ⟨ (3.2) Symmetry of $\equiv$ ⟩
$\neg(q \equiv p)$
$=$ ⟨ (3.10) Definition of $\not\equiv$ ⟩ ······ **Fold**
$q \not\equiv p$

Disjunction Axioms

(3.24) **Axiom, Symmetry of** $\vee$: $p \vee q \equiv q \vee p$

(3.25) **Axiom, Associativity of** $\vee$: $(p \vee q) \vee r \equiv p \vee (q \vee r)$

(3.26) **Axiom, Idempotency of** $\vee$: $p \vee p \equiv p$

(3.27) **Axiom, Distributivity of** $\vee$ **over** $\equiv$:

$$p \vee (q \equiv r) \equiv p \vee q \equiv p \vee r$$

(3.28) **Axiom, Excluded middle**: $p \vee \neg p$

The Law of the Excluded Middle (LEM)

Aristotle:

> ...there cannot be an **intermediate** between contradictories, but of one subject we must either affirm or deny any one predicate...

Bertrand Russell in "The Problems of Philosophy":

> Three "Laws of Thought":
> 1. Law of identity: "Whatever is, is."
> 2. Law of noncontradiction: "Nothing can both be and not be."
> 3. Law of excluded middle: "Everything must either be or not be."
>
> These three laws are samples of self-evident logical principles...

(3.28) **Axiom, Excluded Middle**: $p \vee \neg p$

— this will often be used as: $p \vee \neg p \;\equiv\; true$

Disjunction Axioms and Theorems

(3.24) **Axiom, Symmetry of** $\vee$: $p \vee q \equiv q \vee p$

(3.25) **Axiom, Associativity of** $\vee$: $(p \vee q) \vee r \equiv p \vee (q \vee r)$

(3.26) **Axiom, Idempotency of** $\vee$: $p \vee p \equiv p$

(3.27) **Axiom, Distr. of** $\vee$ **over** $\equiv$: $p \vee (q \equiv r) \equiv p \vee q \equiv p \vee r$

(3.28) **Axiom, Excluded Middle**: $p \vee \neg p$

Theorems:

(3.29) **Zero of** $\vee$: $p \vee true \equiv true$

(3.30) **Identity of** $\vee$: $p \vee false \equiv p$

(3.31) **Distrib. of** $\vee$ **over** $\vee$: $p \vee (q \vee r) \equiv (p \vee q) \vee (p \vee r)$

(3.32) **(3.32)** $p \vee q \;\equiv\; p \vee \neg q \;\equiv\; p$

Heuristics of Directing Calculations

(3.33) **Heuristic:** To prove $P \equiv Q$, transform the expression with the most structure (either P or Q) into the other.

Proving (3.29) $p \vee true \equiv true$:

$$
\begin{array}{ll}
 & p \vee true \\
\equiv & \langle \text{ Identity of } \equiv (3.3) \rangle \\
 & p \vee (q \equiv q) \\
\equiv & \langle \text{ Distr. of } \vee \text{ over } \equiv (3.27) \rangle \\
 & p \vee q \equiv p \vee q \\
\equiv & \langle \text{ Identity of } \equiv (3.3) \rangle \\
 & true
\end{array}
$$

Proving (3.29) $p \vee true \equiv true$:

$$
\begin{array}{ll}
 & true \\
\equiv & \langle \text{ Identity of } \equiv (3.3) \rangle \\
 & p \vee p \equiv p \vee p \\
\equiv & \langle \text{ Distr. of } \vee \text{ over } \equiv (3.27) \rangle \\
 & p \vee (p \equiv p) \\
\equiv & \langle \text{ Identity of } \equiv (3.3) \rangle \\
 & p \vee true
\end{array}
$$

?

(3.34) **Principle:** Structure proofs to minimize the number of rabbits pulled out of a hat — make each step seem obvious, based on the structure of the expression and the goal of the manipulation.

The Conjunction Axiom: The "Golden Rule"

(3.35) **Axiom, Golden rule**: $p \wedge q \quad \equiv \quad p \equiv q \quad \equiv \quad p \vee q$

Can be used as:

- $p \wedge q \quad = \quad (p \equiv q \quad \equiv \quad p \vee q)$ — **Definition of** $\wedge$
- $(p \equiv q) \quad = \quad (p \wedge q \quad \equiv \quad p \vee q)$
- ...

Theorems:

(3.36) **Symmetry of** $\wedge$: $p \wedge q \quad \equiv \quad q \wedge p$

(3.37) **Associativity of** $\wedge$: $(p \wedge q) \wedge r \quad \equiv \quad p \wedge (q \wedge r)$

(3.38) **Idempotency of** $\wedge$: $p \wedge p \quad \equiv \quad p$

(3.39) **Identity of** $\wedge$: $p \wedge true \quad \equiv \quad p$

(3.40) **Zero of** $\wedge$: $p \wedge false \quad \equiv \quad false$

(3.41) **Distributivity of** $\wedge$ **over** $\wedge$: $p \wedge (q \wedge r) \equiv (p \wedge q) \wedge (p \wedge r)$

(3.42) **Contradiction**: $p \wedge \neg p \quad \equiv \quad false$

Conjunction Theorems: Symmetry

(3.36) **Symmetry of** $\wedge$: $(p \wedge q) \equiv (q \wedge p)$

Proving (3.36) **Symmetry of** $\wedge$**:**

$$
\begin{array}{ll}
 & p \wedge q \\
\equiv & \langle \text{ (3.35) Definition of } \wedge \text{ (Golden rule) } \rangle \quad \text{— Unfold} \\
 & p \equiv q \quad \equiv \quad p \vee q \\
\equiv & \langle \text{ (3.2) Symmetry of } \equiv \text{, (3.24) Symmetry of } \vee \rangle \\
 & q \equiv p \quad \equiv \quad q \vee p \\
\equiv & \langle \text{ (3.35) Definition of } \wedge \text{ (Golden rule) } \rangle \quad \text{— Fold} \\
 & q \wedge p
\end{array}
$$

Theorems Relating $\wedge$ and $\vee$

(3.43) **Absorption**:
$$p \wedge (p \vee q) \quad \equiv \quad p$$
$$p \vee (p \wedge q) \quad \equiv \quad p$$

(3.44) **Absorption**:
$$p \wedge (\neg p \vee q) \quad \equiv \quad p \wedge q$$
$$p \vee (\neg p \wedge q) \quad \equiv \quad p \vee q$$

(3.45) **Distributivity of $\vee$ over $\wedge$**: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

(3.46) **Distributivity of $\wedge$ over $\vee$**: $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

(3.47) **De Morgan**:
$$\neg(p \wedge q) \quad \equiv \quad \neg p \vee \neg q$$
$$\neg(p \vee q) \quad \equiv \quad \neg p \wedge \neg q$$

Boolean Lattice Duality

A **Boolean-lattice expression** is

- either a variable,
- or *true* or *false*
- or an application of $\neg\_$ to a Boolean-lattice expression
- or an application of $\_\wedge\_$ or $\_\vee\_$ to two Boolean-lattice expressions.

The **dual** of a Boolean-lattice expressions is obtained by

- replacing *true* with *false* and vice versa,
- replacing $\_\wedge\_$ with $\_\vee\_$ and vice versa.

The **dual** of a Boolean-lattice equation (equivalence) is the equation between the duals of the LHS and the RHS.

Metatheorem "Boolean lattice duality":
Every Boolean-lattice equation is valid iff its dual is valid.

Metatheorem "Boolean lattice duality":
Every Boolean-lattice equation is a theorem iff its dual is a theorem.

Theorems Relating $\wedge$ and $\equiv$

(3.48) **(3.48)** $\quad p \wedge q \quad \equiv \quad p \wedge \neg q \quad \equiv \quad \neg p$

(3.49) Semi-distributivity of $\wedge$ over $\equiv$ $\quad p \wedge (q \equiv r) \quad \equiv \quad p \wedge q \quad \equiv \quad p \wedge r \quad \equiv \quad p$

(3.50) Strong modus ponens for $\equiv$ $\quad p \wedge (q \equiv p) \quad \equiv \quad p \wedge q$

(3.51) **Replacement**: $\quad (p \equiv q) \wedge (r \equiv p) \quad \equiv \quad (p \equiv q) \wedge (r \equiv q)$

Alternative Definitions of ≡ and ≢

(3.52) **Alternative definition of ≡:** $p \equiv q \quad \equiv \quad (p \wedge q) \vee (\neg p \wedge \neg q)$

(3.53) **Alternative definition of ≢:** $p \not\equiv q \quad \equiv \quad (\neg p \wedge q) \vee (p \wedge \neg q)$

(3.21) Heuristic

> Identify applicable theorems by matching the structure of expressions or subexpressions. The operators that appear in a boolean expression and the shape of its subexpressions can focus the choice of theorems to be used in manipulating it.

Obviously, the more theorems you know by heart and the more practice you have in pattern matching, the easier it will be to develop proofs.

Textbook, p. 47

What is a natural number?

How is the set ℕ of all natural numbers defined?

(Without referring to the integers)

(From first principles. . .)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-13

- **Natural Numbers, Natural Induction**
- **Propositional Calculus: Implication $\Rightarrow$**

Natural Numbers — $\mathbb{N}$

- The set of all **natural numbers** is written $\mathbb{N}$.
- In Computing, zero "0" is a natural number.
- If n is a natural number, then its successor "suc n" is a natural number, too.
- We write
  - "1" for "suc 0"
  - "2" for "suc 1"
  - "3" for "suc 2"
  - "4" for "suc 3"
  - ...
- **In Haskell** (data constructors start with upper-case letters):

```
data Nat = Zero | Suc Nat
```

Natural Numbers — Rigorous Definition

- The set of all **natural numbers** is written $\mathbb{N}$.
- Zero "0" is a natural number.
- If n is a natural number, then its successor "suc n" is a natural number, too.
- Nothing else is a natural number.
- Two natural numbers are equal **if and only if** they are constructed in the same way.
  Example: suc suc suc 0 $\neq$ suc suc suc suc 0

This is an inductive definition.

(Like the definition of expressions...)

Every inductive definition gives rise to an induction principle
— a way to prove statements about the inductively defined elements

Factorial — Inductive Definition

- The set of all **natural numbers** is written ℕ.
- zero "0" is a natural number.
- If n is a natural number, then its successor "suc n" is a natural number, too.
- Nothing else is a natural number.
- Two natural numbers are only equal if constructed in the same way.

ℕ is an inductively-defined set.

The factorial operator "_!" on ℕ can be defined as follows:

- The factorial of a natural number is a natural number again:
  $\_! : \mathbb{N} \to \mathbb{N}$
- $0\,! = 1$
- For every $n : \mathbb{N}$, we have:

$$(\mathsf{suc}\ n)\,! = (\mathsf{suc}\ n) \cdot (n\,!)$$

_! is an inductively-defined function.

Natural Number Addition — Inductive Definition

- The set of all **natural numbers** is written ℕ.
- zero "0" is a natural number.
- If n is a natural number, then its successor "suc n" is a natural number, too.
- Nothing else is a natural number.
- Two natural numbers are only equal if constructed in the same way.

ℕ is an inductively-defined set.

Addition on ℕ can be defined as follows:

- The (infix) **addition operator** "+", when applied to two natural numbers, produces again a natural number
  $\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$
- For every $q : \mathbb{N}$, we have:
  - $0 + q = q$
  - For every $n : \mathbb{N}$ we have: $(\mathsf{suc}\ n) + q = \mathsf{suc}\ (n + q)$

_+_ is an inductively-defined function.

Natural Numbers — Induction Principle

- The set of all **natural numbers** is written ℕ.
- Zero "0" is a natural number.
- If n is a natural number, then its successor "suc n" is a natural number, too.

Proving properties of inductively-defined functions on ℕ frequently requires use of the induction principle for ℕ.

Induction principle for the natural numbers:

- if $P(0)$ — If P holds for 0
- and if $P(m)$ implies $P(\mathsf{suc}\ m)$,
  and whenever P holds for m, it also holds for suc m,
- then for all $m : \mathbb{N}$ we have $P(m)$.
  then P holds for all natural numbers.

Natural Numbers — Induction Proofs

Induction principle for the natural numbers:

- if $P[m := 0]$ — If P holds for 0
- and if we can obtain $P[m := \mathsf{suc}\ m]$ from P, — and whenever P holds for m, it also holds for $\mathsf{suc}\ m$,
- then P holds. — then P holds for all natural numbers.

An **induction proof** using this looks as follows:

Theorem: P
Proof:
  By induction on $m : \mathbb{N}$**:**
    Base case:
      Proof for $P[m := 0]$
    Induction step:
      Proof for $P[m := \mathsf{suc}\ m]$
        using **Induction hypothesis** P

$$\frac{P[m := 0] \qquad \begin{array}{c}\ulcorner P \urcorner\\ \vdots \\ P[m := \mathsf{suc}\ m]\end{array}}{P}$$

Proving "Right-Identity of +"

Theorem "Right-identity of +": $m + 0 = m$
Proof:
  By induction on \`$m : \mathbb{N}$\`:
    Base case:
      $0 + 0$
      $=$ ⟨ "Definition of + for 0" ⟩
      0
    Induction step:
      $\mathsf{suc}\ m + 0$
      $=$ ⟨ "Definition of + for \`suc\`" ⟩
      $\mathsf{suc}\ (m + 0)$
      $=$ ⟨ Induction hypothesis ⟩
      $\mathsf{suc}\ m$

An **induction proof** looks as follows:

Theorem: P
Proof:
  By induction on $m : \mathbb{N}$**:**
    Base case:
      Proof for $P[m := 0]$
    Induction step:
      Proof for $P[m := \mathsf{suc}\ m]$
        using **Induction hypothesis** P

Proving "Right-Identity of +" — With Details

Theorem "Right-identity of +": $m + 0 = m$
Proof:
  By induction on \`$m : \mathbb{N}$\`:
    Base case \`$0 + 0 = 0$\`:
      $0 + 0$
      $=$ ⟨ "Definition of + for 0" ⟩
      0
    Induction step \`$\mathsf{suc}\ m + 0 = \mathsf{suc}\ m$\`:
      $\mathsf{suc}\ m + 0$
      $=$ ⟨ "Definition of + for \`suc\`" ⟩
      $\mathsf{suc}\ (m + 0)$
      $=$ ⟨ Induction hypothesis \`$m + 0 = m$\` ⟩
      $\mathsf{suc}\ m$

An **induction proof** looks as follows:

Theorem: P
Proof:
  By induction on $m : \mathbb{N}$**:**
    Base case:
      Proof for $P[m := 0]$
    Induction step:
      Proof for $P[m := \mathsf{suc}\ m]$
        using **Induction hypothesis** P

Proving "Right-Identity of +" — Indentation!

```
Theorem "Right-identity of +": m + 0 = m
Proof:
␣␣By induction on `m : ℕ`:
␣␣␣␣Base case:
␣␣␣␣␣␣␣␣0 + 0
␣␣␣␣␣␣=⟨ "Definition of + for 0" ⟩
␣␣␣␣␣␣␣␣0
␣␣␣␣Induction step:
␣␣␣␣␣␣␣␣suc m + 0
␣␣␣␣␣␣=⟨ "Definition of + for `suc`" ⟩
␣␣␣␣␣␣␣␣suc (m + 0)
␣␣␣␣␣␣=⟨ Induction hypothesis ⟩
␣␣␣␣␣␣␣␣suc m
```

Press "Ctrl-Shift-v" to toggle "visible spaces".

Read Parse Error Messages!

≡ ⟨ Substitution ⟩
— CalcCheck: Due to parse error in the expression below, this calculation step cannot be checked.
⟪ Parse error: "Cell 12" (line 19, column 16):
unexpected "="
expecting white space, "------", ",", or ≔ «expressions»
⟫
⇒[y := z - y] ⟨ "Assignment" ⟩
— CalcCheck: Found "Assignment"
— CalcCheck: Due to parse error in the expression above, this calculation step cannot be checked.

```
18:    ≡⟨ Substitution ⟩
19:      (y = 42)[y = z - y]
20:    ⇒[ y := z - y ]   ⟨ "Assignment" ⟩
```

Submitting parse errors is unprofessional!

Carefully Check Indentation: Each Level ≥ 2 Spaces!

≡ ⟨ Substitution ⟩
— CalcCheck: Due to parse error in the expression below, this calculation step cannot be checked
⟪ Parse error: "Cell 12" (line 18, column 25):
unexpected "“"
expecting white space, "------", or «expression»
⟫

```
16:    ≡⟨ Substitution ⟩
17:      (y = z - y)[y ≔ z - y]
18:    ■⇒[ y := z - y ]   ⟨ "Assignment" ⟩
19:      y = 42
```

Hint item where the parser expects an expression —

calculation operators need to be aligned
two spaces to the left of calculation expressions!

You need to solve the Homeworks yourself!

- Assuming that you can pass this course without actually acquiring the expected reasoning skills is most likely unrealistic.
- You acquire the skills by doing Homeworks and Assignments yourself!
- **If you provide your solutions to others,**

  that constitutes academic dishonesty as well!
- **If you provide your solutions to others,**

  that actually reduces their chances of acquiring the skills and passing the course!
- Large/many clusters of extremely similar submissions strongly suggest that large numbers of students are not getting the expected learning: $\Longrightarrow$ **I need to act!**
- If homeworks were to be done with pen and paper, you would submit imperfect solutions without hesitation...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-13

Part 2: Propositional Calculus: Implication $\Rightarrow$

Implication

(3.57) **Axiom, Definition of implication,**
Definition of $\Rightarrow$ from $\vee$: $\boxed{p \Rightarrow q \ \equiv\ p \vee q \ \equiv\ q}$

(3.58) **Axiom, Consequence:** $\boxed{p \Leftarrow q \quad \equiv \quad q \Rightarrow p}$

Rewriting Implication:

(3.59) Material implication,
(Alternative) **Definition of implication**: $p \Rightarrow q \ \equiv\ \neg p \vee q$

(3.60) (Dual) **Definition of implication,**
Definition of $\Rightarrow$ from $\wedge$: $p \Rightarrow q \ \equiv\ p \wedge q \ \equiv\ p$

(3.61) **Contrapositive:** $p \Rightarrow q \quad \equiv \quad \neg q \Rightarrow \neg p$

All Propositional Axioms of the Equational Logic E

1. (3.1) **Axiom, Associativity of** $\equiv$
2. (3.2) **Axiom, Symmetry of** $\equiv$
3. (3.3) **Axiom, Identity of** $\equiv$
4. (3.8) **Axiom, Definition of** *false*
5. (3.9) **Axiom, Commutativity of** $\neg$ **with** $\equiv$
6. (3.10) **Axiom, Definition of** $\not\equiv$
7. (3.24) **Axiom, Symmetry of** $\vee$
8. (3.25) **Axiom, Associativity of** $\vee$
9. (3.26) **Axiom, Idempotency of** $\vee$
10. (3.27) **Axiom, Distributivity of** $\vee$ **over** $\equiv$
11. (3.28) **Axiom, Excluded middle**
12. (3.35) **Axiom, Golden rule**
13. (3.57) **Axiom, Definition of implication**
14. (3.58) **Axiom, Definition of consequence**

The "Golden Rule" and Implication

(3.35) **Axiom, Golden rule:** $\boxed{p \wedge q \quad \equiv \quad p \equiv q \quad \equiv \quad p \vee q}$

Can be used as:

- $p \wedge q \quad = \quad (p \equiv q \quad \equiv \quad p \vee q)$
- $(p \equiv q) \quad = \quad (p \wedge q \quad \equiv \quad p \vee q)$
- ...
- $(p \wedge q \quad \equiv \quad p) \quad \equiv \quad (q \quad \equiv \quad p \vee q)$

(3.57) **Axiom, Definition of implication:** $p \Rightarrow q \;\equiv\; p \vee q \;\equiv\; q$

(3.60) (Dual) **Definition of implication:** $p \Rightarrow q \;\equiv\; p \wedge q \;\equiv\; p$

Some Implication Theorems

(3.62) $p \Rightarrow (q \equiv r) \quad \equiv \quad p \wedge q \quad \equiv \quad p \wedge r$

(3.63) **Distributivity of** $\Rightarrow$ **over** $\equiv$**:** $p \Rightarrow (q \equiv r) \quad \equiv \quad p \Rightarrow q \quad \equiv \quad p \Rightarrow r$

(3.64) **Self-distributivity of** $\Rightarrow$**:** $p \Rightarrow (q \Rightarrow r) \quad \equiv \quad (p \Rightarrow q) \Rightarrow (p \Rightarrow r)$

(3.65) **Shunting:** $p \wedge q \Rightarrow r \quad \equiv \quad p \Rightarrow (q \Rightarrow r)$

How do start to prove the following? (For example, ...)

(3.66) $p \wedge (p \Rightarrow q) \quad \equiv \quad p \wedge q$ $\qquad \langle \ldots \quad p \wedge q \equiv p \rangle$

(3.67) $p \wedge (q \Rightarrow p) \quad \equiv \quad p$ $\qquad \langle \ldots \quad p \wedge q \equiv p \rangle$

(3.68) $p \vee (p \Rightarrow q) \quad \equiv \quad true$ $\qquad \langle \ldots \quad \neg p \vee q \rangle$

(3.69) $p \vee (q \Rightarrow p) \quad \equiv \quad q \Rightarrow p$ $\qquad \langle \ldots \quad p \vee q \equiv q \rangle$

(3.70) $p \vee q \Rightarrow p \wedge q \quad \equiv \quad p \equiv q$ $\qquad \langle \ldots$ Golden Rule $\ldots \rangle$

Additional Important Implication Theorems

(3.71) **Reflexivity of ⇒:** $p \Rightarrow p \quad \equiv \quad true$

(3.72) **Right-zero of ⇒:** $p \Rightarrow true \quad \equiv \quad true$

(3.73) **Left-identity of ⇒:** $true \Rightarrow p \quad \equiv \quad p$

(3.74) **Definition of ¬ from ⇒:** $p \Rightarrow false \quad \equiv \quad \neg p$

(3.15) **Definition of ¬ from ≡:** $\neg p \quad \equiv \quad p \equiv false$

(3.75) ***ex falso quodlibet:*** $false \Rightarrow p \quad \equiv \quad true$

(3.65) **Shunting:** $p \wedge q \Rightarrow r \quad \equiv \quad p \Rightarrow (q \Rightarrow r)$

(3.77) **Modus ponens:** $p \wedge (p \Rightarrow q) \quad \Rightarrow \quad q$

(3.78) **Case analysis:** $(p \Rightarrow r) \wedge (q \Rightarrow r) \equiv (p \vee q \Rightarrow r)$

(3.79) **Case analysis:** $(p \Rightarrow r) \wedge (\neg p \Rightarrow r) \equiv r$

Weakening/Strengthening Theorems

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

(3.76a) **Weakening/Strengthening:** $p \quad \Rightarrow p \vee q$

(3.76b) **Weakening/Strengthening:** $p \wedge q \quad \Rightarrow p$

(3.76c) **Weakening/Strengthening:** $p \wedge q \quad \Rightarrow p \vee q$

(3.76d) **Weakening/Strengthening:** $p \vee (q \wedge r) \quad \Rightarrow p \vee q$

(3.76e) **Weakening/Strengthening:** $p \wedge q \quad \Rightarrow p \wedge (q \vee r)$

Implication as Order on Propositions

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

— similar to "$x \leq y$" as "x is less-or-equal y"

— similar to "$x \geq y$" as "x is greater-or-equal y"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

— similar to "$x \leq y$" as "x is at most y"

— similar to "$x \geq y$" as "x is at least y"

(3.57) **Axiom, Definition of ⇒** from disjunction: $p \Rightarrow q \equiv p \vee q \equiv q$

— defines the order from maximum: $p \Rightarrow q \equiv ((p \vee q) = q)$

— analogous to: $x \leq y \equiv ((x \uparrow y) = y)$

— analogous to: $k \mid n \equiv ((\mathsf{lcm}(k, n) = n)$

(3.60) (Dual) **Definition of ⇒** from conjunction: $p \Rightarrow q \equiv p \wedge q \equiv p$

— defines the order from minimum: $p \Rightarrow q \equiv ((p \wedge q) = p)$

— analogous to: $x \leq y \equiv ((x \downarrow y) = x)$

— analogous to: $k \mid n \equiv ((\mathsf{gcd}(k, n) = k)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-17

Implication as Order, Replacement, Monotonicity

Plan for Today

- **Continuing Propositional Calculus (LADM chapter 3)**
  - Implication as order, order relations
  - Leibniz as axiom, and "Replacement" theorems
- Transitivity Calculations, Monotonicity (LADM section 4.1)
- (Coming up: LADM chapter 4, and then chapters 8 and 9.)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-17

Part 1: Nested (Induction) Proofs

Recall: Simple Natural Induction Proofs

Addition

$\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$

is defined by **induction over the first argument:**

Axiom "Definition of + for 0"
"Left-identity of +":
$0 + n = n$

Axiom "Definition of + for `suc`":
$(\mathsf{suc}\ m) + n = \mathsf{suc}\ (m + n)$

Many properties of _+_ can be proven by induction over one of the first arguments to _+_:

Theorem "Right-identity of +": $m + 0 = m$

Proof:
By induction on `$m : \mathbb{N}$`:
Base case:
$0 + 0$
$= \langle$ "Definition of + for 0" $\rangle$
0
Induction step:
$\mathsf{suc}\ m + 0$
$= \langle$ "Definition of + for `suc`" $\rangle$
$\mathsf{suc}\ (m + 0)$
$= \langle$ Induction hypothesis $\rangle$
$\mathsf{suc}\ m$

Defining (Monus) Subtraction Inductively

Axiom "Subtraction from zero": $0 - n = 0$
Axiom "Subtraction of zero from successor": $(\mathsf{suc}\ m) - 0 = \mathsf{suc}\ m$
Axiom "Subtraction of successor from successor": $(\mathsf{suc}\ m) - (\mathsf{suc}\ n) = m - n$

Note: In the natural numbers $\mathbb{N}$, we have: $2 - 5 = 0$

Why does this define _−_ for all possible arguments?

Because:

- _−_ takes **two** arguments of type $\mathbb{N}$
- **Each of these arguments** is **always** either 0, or $\mathsf{suc}\ k$ for some **smaller** $k : \mathbb{N}$
- Of the four possible combinations, two are covered by "Subtraction from zero"
- The remaining two clauses cover one of the remaining cases each.
- The third clause "builds up" the domain of definition of _−_ **from smaller to larger** m and n.

Using Subtraction Defined Inductively Using Three Clauses

Axiom "Subtraction from zero": $0 - n = 0$
Axiom "Subtraction of zero from successor": $(\mathsf{suc}\ m) - 0 = \mathsf{suc}\ m$
Axiom "Subtraction of successor from successor": $(\mathsf{suc}\ m) - (\mathsf{suc}\ n) = m - n$

$\Longrightarrow$ **Some properties of subtraction need nested induction proofs!**

... Syntactically, **where one kind of proof can go, any kind of proof can be used** ...

$\Longrightarrow$ **Inside nested induction steps, used induction hypotheses must be made explicit!**

... see Exercise 3.3.

Nested Induction Proofs For Subtraction Defined Inductively Using Three Clauses

Axiom "Subtraction from zero": $0 - n = 0$
Axiom "Subtraction of zero from successor": $(\mathsf{suc}\ m) - 0 = \mathsf{suc}\ m$
Axiom "Subtraction of successor from successor": $(\mathsf{suc}\ m) - (\mathsf{suc}\ n) = m - n$

... see Ex3.3, e.g.:

Theorem "Subtraction after addition": $(m + n) - n = m$
Proof:
  By induction on `$m : \mathbb{N}$`:
    Base case:
      $(0 + n) - n$
    $= \langle\ ?\ \rangle$
      0
    Induction step `$(\mathsf{suc}\ m + n) - n = \mathsf{suc}\ m$`:
      By induction on `$n : \mathbb{N}$`:
        Base case:
          $(\mathsf{suc}\ m + 0) - 0$
        $= \langle\ ?\ \rangle$
          $\mathsf{suc}\ m$
        Induction step:
          $(\mathsf{suc}\ m + \mathsf{suc}\ n) - \mathsf{suc}\ n$
        $= \langle\ ?\ \rangle$
          $(\mathsf{suc}\ m + n) - n$
        $= \langle$ Induction hypothesis `$(\mathsf{suc}\ m + n) - n = \mathsf{suc}\ m$` $\rangle$
          $\mathsf{suc}\ m$

... Syntactically,
where one kind of proof can go, any kind of proof can be used ...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-17

Part 2: Implication as Order, Order Relations

Recall: Weakening/Strengthening Theorems

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

(3.76a) $p \Rightarrow p \vee q$

(3.76b) $p \wedge q \Rightarrow p$

(3.76c) $p \wedge q \Rightarrow p \vee q$

(3.76d) $p \vee (q \wedge r) \Rightarrow p \vee q$

(3.76e) $p \wedge q \Rightarrow p \wedge (q \vee r)$

Implication as Order on Propositions

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

— similar to "$x \leq y$" as "x is less-or-equal y"

— similar to "$x \geq y$" as "x is greater-or-equal y"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

— similar to "$x \leq y$" as "x is at most y"

— similar to "$x \geq y$" as "x is at least y"

(3.57) **Axiom, Definition of** $\Rightarrow$ from disjunction: $p \Rightarrow q \equiv p \vee q \equiv q$

— defines the order from maximum: $p \Rightarrow q \equiv ((p \vee q) = q)$

— analogous to: $x \leq y \equiv ((x \uparrow y) = y)$

— analogous to: $k \mid n \equiv ((\mathsf{lcm}(k, n) = n)$

(3.60) (Dual) **Definition of** $\Rightarrow$ from conjunction: $p \Rightarrow q \equiv p \wedge q \equiv p$

— defines the order from minimum: $p \Rightarrow q \equiv ((p \wedge q) = p)$

— analogous to: $x \leq y \equiv ((x \downarrow y) = x)$

— analogous to: $k \mid n \equiv ((\mathsf{gcd}(k, n) = k)$

One View of Relations

- Let T_1 and T_2 be two types.
- A function of type $T_1 \rightarrow T_2 \rightarrow \mathbb{B}$ can be considered as *one view of* a **relation from T_1 to T_2**
  - We will see a different view of relations later ...
  - ... and **the** way to switch between these views.
  - With such a way of switching, the two views "are the same" in colloquial mathematics
  - Therefore we will occasionally just use the term "relation" also for functions of types $T_1 \rightarrow T_2 \rightarrow \mathbb{B}$
- A function of type $T \rightarrow T \rightarrow \mathbb{B}$ may then be called **a relation on T**.
- Some relations you are familar with:

$$
\begin{array}{ll}
\_{=}\_ & : T \rightarrow T \rightarrow \mathbb{B} \\
\_{=}\_ & : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B} \\
\_{\neq}\_ & : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B} \\
\_{<}\_ & : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B} \\
\_{\not\equiv}\_ & : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B} \\
\_{\in}\_ & : T \rightarrow \mathbf{set}\ T \rightarrow \mathbb{B}
\end{array}
$$

Order Relations

- Let T be a type.
- A relation $\_{\leq}\_$ on T is called:
  - **reflexive** iff $x \leq x$ is valid
  - **transitive** iff $x \leq y \wedge y \leq z \Rightarrow x \leq z$ is valid
  - **antisymmetric** iff $x \leq y \wedge y \leq x \Rightarrow x = y$ is valid
  - an **order** (or **ordering**) iff it is reflexive, transitive, and antisymmetric
- Orders you are familiar with:

$$
\begin{array}{ll}
\_{=}\_ & : T \rightarrow T \rightarrow \mathbb{B} \\
\_{\leq}\_ & : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B} \\
\_{\geq}\_ & : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B} \\
\_{\leq}\_ & : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B} \\
\_{\geq}\_ & : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B} \\
\_{\mid}\_ & : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B} \\
\_{\equiv}\_ & : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B} \\
\_{\Rightarrow}\_ & : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B} \\
\_{\subseteq}\_ & : \mathbf{set}\ T \rightarrow \mathbf{set}\ T \rightarrow \mathbb{B}
\end{array}
$$

Order Properties of Implication in LADM Chapter 3

(3.71) **Reflexivity of ⇒:** $p \Rightarrow p$

(3.80.1) **Reflexivity of ⇒ wrt. ≡:** $(p \equiv q) \Rightarrow (p \Rightarrow q)$

(3.80) **Mutual implication:** $(p \Rightarrow q) \wedge (q \Rightarrow p) \ \equiv\ p \equiv q$

(3.81) **Antisymmetry:** $(p \Rightarrow q) \wedge (q \Rightarrow p) \ \Rightarrow\ (p \equiv q)$

(3.82a) **Transitivity:** $(p \Rightarrow q) \wedge (q \Rightarrow r) \ \Rightarrow\ (p \Rightarrow r)$

(3.82b) **Transitivity:** $(p \equiv q) \wedge (q \Rightarrow r) \ \Rightarrow\ (p \Rightarrow r)$

(3.82c) **Transitivity:** $(p \Rightarrow q) \wedge (q \equiv r) \ \Rightarrow\ (p \Rightarrow r)$

Monotonicity, Isotonicity, Antitonicity

- Let _≤_ be an order on T
- Let $f : T \to T$ be a function on T
- Then f is called
  - **monotonic** iff $x \le y \ \Rightarrow\ f\,x \le f\,y$ is a theorem
  - **isotonic** iff $x \le y \ \equiv\ f\,x \le f\,y$ is a theorem
  - **antitonic** iff $x \le y \ \Rightarrow\ f\,y \le f\,x$ is a theorem
- Examples:
  - suc _ : $\mathbb{N} \to \mathbb{N}$ is isotonic
  - pred : $\mathbb{N} \to \mathbb{N}$ is monotonic, but not isotonic
  - _+_ : $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is isotonic in the first argument:
    $x \le y \ \equiv\ x + z \le y + z$ is a theorem
  - _+_ : $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is isotonic in the second argument:
    $x \le y \ \equiv\ z + x \le z + y$ is a theorem
  - _−_ : $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **monotonic in the first argument**:
    $x \le y \ \Rightarrow\ x - z \le y - z$ is a theorem
  - _−_ : $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **antitonic in the second argument**:
    $x \le y \ \Rightarrow\ z - y \le z - x$ is a theorem

Monotonicity and Antitonicity Theorems for ⇒

(4.2) **Left-monotonicity of ∨:** $(p \Rightarrow q) \ \Rightarrow\ (p \vee r \Rightarrow q \vee r)$

(4.3) **Left-monotonicity of ∧:** $(p \Rightarrow q) \ \Rightarrow\ p \wedge r \ \Rightarrow\ q \wedge r$

— You can prove these already in the context of chapter 3!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-17

Part 3: Leibniz as Axiom, Replacement Theorems

(LADM pp. 60–61, end of chapter 3)

Leibniz's Rule as an Axiom

Recall the **inference rule** (scheme):

(1.5) **Leibniz:** $$\frac{X = Y}{E[z := X] = E[z := Y]}$$

Axiom scheme (E can be any expression, and z any variable):

(3.83) **Axiom, Leibniz:** $(e = f) \Rightarrow (E[z := e] = E[z := f])$

What is the difference?

- Given a theorem $X = Y$ and an expression E, the inference rule (1.5) **produces** a new theorem $E[z := X] = E[z := Y]$
- (3.83) **is** a theorem
- $((e = f) \Rightarrow (E[z := e] = E[z := f])) \quad = \quad true$

  Can be used **deep inside nested expressions**
  — making use of **local assumptions (that are typically not theorems)**

Leibniz's Rule as an Axiom — Examples

Recall the **inference rule** (scheme):

(1.5) **Leibniz:** $$\frac{X = Y}{E[z := X] = E[z := Y]}$$

Axiom scheme (E can be any expression, and z any variable):

(3.83) **Axiom, Leibniz:** $(e = f) \Rightarrow (E[z := e] = E[z := f])$

Examples

- $n = k + 1 \Rightarrow n \cdot (k - 1) = (k + 1) \cdot (k - 1)$
- $n = k + 1 \Rightarrow (z \cdot (k - 1))[z := n] = (z \cdot (k - 1))[z := k + 1]$
- $$\begin{array}{ll} & (n = k + 1 \Rightarrow n \cdot (k - 1) = k^2 - 1) = true \\ \Rightarrow & (n > 5 \Rightarrow (n = k + 1 \Rightarrow n \cdot (k - 1) = k^2 - 1)) \\ & = (n > 5 \Rightarrow true) \end{array}$$

Leibniz's Rule Axiom, and Further Replacement Rules

Axiom scheme (E can be any expression; $z, e, f : t$ can be of **any type** t):

(3.83) **Axiom, Leibniz:** $(e = f) \Rightarrow (E[z := e] = E[z := f])$

— Axiom (3.83) is rarely useful directly!

— Almost all applications are via derived **"Replacement"** theorems

Replacement rules: (P can be any expression **of type** $\mathbb{B}$)

(3.84a) **"Replacement":** $(e = f) \wedge P[z := e] \quad \equiv \quad (e = f) \wedge P[z := f]$

(3.84b) **"Replacement":** $(e = f) \Rightarrow P[z := e] \quad \equiv \quad (e = f) \Rightarrow P[z := f]$

(3.84c) **"Replacement":** $q \wedge (e = f) \Rightarrow P[z := e] \equiv q \wedge (e = f) \Rightarrow P[z := f]$

Using a Replacement (LADM: "Substitution") Rule

Replacement rule: (P can be any expression **of type** $\mathbb{B}$)

(3.84a) **"Replacement":** $(e = f) \wedge P[z := e] \quad \equiv \quad (e = f) \wedge P[z := f]$

Textbook-style application:

$$k = n + 1 \quad \wedge \quad k \cdot (n - 1) = n^2 - 1$$
$$= \langle \text{(3.84a) "Replacement"} \rangle$$
$$k = n + 1 \quad \wedge \quad (n + 1) \cdot (n - 1) = n^2 - 1$$

Not so fast! — CalcCheck cannot do second-order matching (yet)

$$k = n + 1 \quad \wedge \quad k \cdot (n - 1) = n \cdot n - 1$$
$$= \langle \text{Substitution} \rangle$$
$$k = n + 1 \quad \wedge \quad (z \cdot (n - 1) = n \cdot n - 1)[z := k]$$
$$= \langle \text{(3.84a) "Replacement"} \rangle$$
$$k = n + 1 \quad \wedge \quad (z \cdot (n - 1) = n \cdot n - 1)[z := n + 1]$$
$$= \langle \text{Substitution} \rangle$$
$$k = n + 1 \quad \wedge \quad (n + 1) \cdot (n - 1) = n \cdot n - 1$$

Some Replacements

$$((x > f\ 5) \equiv (y < g\ 7)) \quad \wedge \quad ((f\ x \leq g\ y) \equiv (x > f\ 5))$$
$$\equiv \langle \quad ? \quad \rangle$$
$$((x > f\ 5) \equiv (y < g\ 7)) \quad \wedge \quad ((f\ x \leq g\ y) \equiv (y < g\ 7))$$

$$((f\ 5) = (g\ y)) \quad \wedge \quad ((f\ x \leq g\ y) \equiv x > (f\ 5))$$
$$\equiv \langle \quad ? \quad \rangle$$
$$((f\ 5) = (g\ y)) \quad \wedge \quad ((f\ x \leq g\ y) \equiv x > g\ y))$$

$$((x > f\ 5) \equiv (y < g\ 7)) \quad \wedge \quad ((f\ x \leq g\ y) \Rightarrow p(x - 1) \vee (x > f\ 5))$$
$$\equiv \langle \quad ? \quad \rangle$$
$$((x > f\ 5) \equiv (y < g\ 7)) \quad \wedge \quad ((f\ x \leq g\ y) \Rightarrow p(x - 1) \vee (y < g\ 7))$$

Replacements 1 & 2

$$((x > f\,5) \;\equiv\; (y < g\,7)) \quad\wedge\quad ((f\,x \le g\,y) \;\equiv\; (x > f\,5))$$

$\equiv$ ⟨ (3.51) **"Replacement"** $(p \equiv q) \wedge (r \equiv p) \;\equiv\; (p \equiv q) \wedge (r \equiv q)$ ⟩

$$((x > f\,5) \;\equiv\; (y < g\,7)) \quad\wedge\quad ((f\,x \le g\,y) \;\equiv\; (y < g\,7))$$

$$((f\,5) \;=\; (g\,y)) \quad\wedge\quad ((f\,x \le g\,y) \;\equiv\; x > (f\,5))$$

$\equiv$ ⟨ Substitution ⟩

$$((f\,5) \;=\; (g\,y)) \quad\wedge\quad \underline{((f\,x \le g\,y) \;\equiv\; x > z)}[z := (f\,5)]$$

$\equiv$ ⟨ (3.84a) **"Replacement"**
$(e = f) \wedge \underline{P}[z := e] \;\equiv\; (e = f) \wedge \underline{P}[z := f]$,
Substitution ⟩

$$((f\,5) \;=\; (g\,y)) \quad\wedge\quad ((f\,x \le g\,y) \;\equiv\; x > g\,y))$$

Replacement 3

$$((x > f\,5) \;\equiv\; (y < g\,7)) \quad\wedge\quad ((f\,x \le g\,y) \;\Rightarrow\; p(x-1) \vee (x > f\,5))$$

$\equiv$ ⟨ Substitution ⟩

$$((x > f\,5) \;\equiv\; (y < g\,7)) \wedge \underline{((f\,x \le g\,y) \;\Rightarrow\; p(x-1) \vee z)}[z := (x > f\,5)]$$

$\equiv$ ⟨ (3.84a) **"Replacement"**
$(e = f) \wedge \underline{P}[z := e] \;\equiv\; (e = f) \wedge \underline{P}[z := f]$,
"Definition of ≡" $(p \equiv q) = (p = q)$, Substitution ⟩

$$((x > f\,5) \;\equiv\; (y < g\,7)) \quad\wedge\quad ((f\,x \le g\,y) \;\Rightarrow\; p(x-1) \vee (y < g\,7))$$

In CalcCheck, ≡ does not match =!

Explicit conversions using "Definition of ≡" are necessary.

Replacing Variables by Boolean Constants

In each of the following, P can be any expression **of type** $\mathbb{B}$:

(3.85a) **Replace by** *true*: $p \Rightarrow P[z := p] \;\equiv\; p \Rightarrow P[z := true]$

(3.85b) $q \wedge p \Rightarrow P[z := p] \;\equiv\; q \wedge p \Rightarrow P[z := true]$

(3.86a) **Replace by** *false*: $P[z := p] \Rightarrow p \;\equiv\; P[z := false] \Rightarrow p$

(3.86b) $P[z := p] \Rightarrow p \vee q \;\equiv\; P[z := false] \Rightarrow p \vee q$

(3.87) **Replace by** *true*: $p \wedge P[z := p] \;\equiv\; p \wedge P[z := true]$

(3.88) **Replace by** *false*: $p \vee P[z := p] \;\equiv\; p \vee P[z := false]$

(3.89) **Shannon:** $P[z := p] \;\equiv\; (p \wedge P[z := true]) \vee (\neg p \wedge P[z := false])$

Note: Using Shannon on all propositional variables in sequence
is equivalent to producing a truth table.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-17

Part 4: Transitivity Calculations, Monotonicity

?

$$
\begin{array}{ll}
 & 7 \cdot 8 \\
= & \langle \text{ Evaluation } \rangle \\
 & (10 - 3) \cdot (12 - 4) \\
\leq & \langle \text{ Fact: } 3 \leq 4 \text{ } \rangle \\
 & (10 - 4) \cdot (12 - 4) \\
\leq & \langle \text{ Fact: } 4 \leq 5 \text{ } \rangle \\
 & (10 - 4) \cdot (12 - 5) \\
= & \langle \text{ Evaluation } \rangle \\
 & 6 \cdot 7 \\
= & \langle \text{ Evaluation } \rangle \\
 & 42
\end{array}
$$

This proves: $7 \cdot 8 \leq 42$

Recall: Calculational Proof Format

$$
\begin{array}{ll}
 & E_0 \\
= & \langle \text{ Explanation of why } E_0 = E_1 \text{ } \rangle \\
 & E_1 \\
= & \langle \text{ Explanation of why } E_1 = E_2 \text{ — with comment } \rangle \\
 & E_2 \\
= & \langle \text{ Explanation of why } E_2 = E_3 \text{ } \rangle \\
 & E_3
\end{array}
$$

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 = E_1 \qquad \wedge \qquad E_1 = E_2 \qquad \wedge \qquad E_2 = E_3$$

Because $=$ is **transitive**, this justifies:

$$E_0 = E_3$$

Extended Calculational Proof Format (1)

$$
\begin{array}{ll}
 & E_0 \\
\leq & \langle \text{Explanation of why } E_0 \leq E_1 \rangle \\
 & E_1 \\
\leq & \langle \text{Explanation of why } E_1 \leq E_2 \text{ — with comment} \rangle \\
 & E_2 \\
\leq & \langle \text{Explanation of why } E_2 \leq E_3 \rangle \\
 & E_3
\end{array}
$$

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \leq E_1 \qquad \wedge \qquad E_1 \leq E_2 \qquad \wedge \qquad E_2 \leq E_3$$

Because $\leq$ is **transitive**, this justifies:

$$E_0 \leq E_3$$

Extended Calculational Proof Format (2)

$$
\begin{array}{ll}
 & E_0 \\
\leq & \langle \text{Explanation of why } E_0 \leq E_1 \rangle \\
 & E_1 \\
= & \langle \text{Explanation of why } E_1 = E_2 \text{ — with comment} \rangle \\
 & E_2 \\
\leq & \langle \text{Explanation of why } E_2 \leq E_3 \rangle \\
 & E_3
\end{array}
$$

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \leq E_1 \qquad \wedge \qquad E_1 = E_2 \qquad \wedge \qquad E_2 \leq E_3$$

Because $\leq$ is **reflexive and transitive**, this justifies:

$$E_0 \leq E_3$$

Extended Calculational Proof Format (3)

$$
\begin{array}{ll}
 & E_0 \\
\Rightarrow & \langle \text{Explanation of why } E_0 \Rightarrow E_1 \rangle \\
 & E_1 \\
\equiv & \langle \text{Explanation of why } E_1 \equiv E_2 \text{ — with comment} \rangle \\
 & E_2 \\
\Rightarrow & \langle \text{Explanation of why } E_2 \Rightarrow E_3 \rangle \\
 & E_3
\end{array}
$$

Because the **calculational presentation** is **conjunctional**, this reads as:

$$(E_0 \Rightarrow E_1) \qquad \wedge \qquad (E_1 \equiv E_2) \qquad \wedge \qquad (E_2 \Rightarrow E_3)$$

Because $\Rightarrow$ is **reflexive and transitive**, this justifies:

$$E_0 \Rightarrow E_3$$

Extended Calculational Proof Format (4)

$$\begin{array}{ll} & E_0 \\ \leq & \langle \text{ Explanation of why } E_0 \leq E_1 \rangle \\ & E_1 \\ = & \langle \text{ Explanation of why } E_1 = E_2 \text{ — with comment } \rangle \\ & E_2 \\ < & \langle \text{ Explanation of why } E_2 < E_3 \rangle \\ & E_3 \end{array}$$

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \leq E_1 \quad \wedge \quad E_1 = E_2 \quad \wedge \quad E_2 < E_3$$

Because $<$ is **transitive**, and because $\leq$ is the reflexive closure of $<$, this justifies:

$$E_0 < E_3$$

Calculational Non-Proofs

$$\begin{array}{ll} & E_0 \\ \leq & \langle \text{ Explanation of why } E_0 \leq E_1 \rangle \\ & E_1 \\ = & \langle \text{ Explanation of why } E_1 = E_2 \text{ — with comment } \rangle \\ & E_2 \\ \geq & \langle \text{ Explanation of why } E_2 \geq E_3 \rangle \\ & E_3 \end{array}$$

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \leq E_1 \quad \wedge \quad E_1 = E_2 \quad \wedge \quad E_2 \geq E_3$$

This justifies nothing about the relation between E_0 and E_3 !

Leibniz is Special to Equality

How about the following?

$$\begin{array}{ll} & x - 3 \\ \leq & \langle \text{ Fact: } 3 \leq 4 \rangle \\ & x - 4 \end{array}$$

Remember:

(1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$

Leibniz is available <u>only for equality</u>

Example Application of "Monotonicity of –"

- $\_-\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **monotonic in the first argument**:
  $x \leq y \quad \Rightarrow \quad x - z \leq y - z$ is a theorem

---

Theorem "Monotonicity of – ": $a \leq b \Rightarrow a - c \leq b - c$

```
Calculation:
    12 - n
  ≤⟨ “Monotonicity of -” with Fact `12 ≤ 20` ⟩
    20 - n
```

This step can be justified without "with" as follows:

```
Calculation:
    12 - n  ≤  20 - n
  ≡⟨ “Left-identity of ⇒” ⟩
    true  ⇒  (12 - n  ≤  20 - n)
  ≡⟨ Fact `12 ≤ 20` ⟩
    (12 ≤ 20)  ⇒  (12 - n  ≤  20 - n)
  — This is “Monotonicity of -”
```

Modus Pones via with$_2$

Modus ponens theorem: (3.77) **Modus ponens:** $p \wedge (p \Rightarrow q) \Rightarrow q$

Modus ponens inference rule:
("Implication elimination" rule)

$$\frac{P \Rightarrow Q \qquad P}{Q} \Rightarrow\text{-Elim} \qquad \frac{f : A \to B \qquad x : A}{(f\ x) : B} \text{ Fct. app.}$$

Applying implication theorems:

A proof for $P \Rightarrow Q$ can be used as a recipe for turning a proof for P into a proof for Q.

Q_1
$\sqsubseteq$ ⟨ "Theorem 1" \`$P \Rightarrow (Q_1 \sqsubseteq Q_2)$\` with "Theorem 2" \`$P$\` ⟩
Q_2

Theorem "Monotonicity of – ": $a \leq b \Rightarrow a - c \leq b - c$

```
Calculation:
    12 - n
  ≤⟨ “Monotonicity of -” with Fact `12 ≤ 20` ⟩
    20 - n
```

Example Application of "Antitonicity of –"

- $\_-\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **antitonic in the second argument**:
  $x \leq y \quad \Rightarrow \quad z - y \leq z - x$ is a theorem

---

Theorem "Antitonicity of – ": $b \leq c \quad \Rightarrow \quad a - c \leq a - b$

```
Calculation:
    m - 3
  ≤⟨ “Antitonicity of -” with Fact `2 ≤ 3` ⟩
    m - 2
```

Multiplication on ℕ is Monotonic...

Calculation:

$$
\begin{array}{ll}
& 42 \\
= & \langle \text{ Evaluation } \rangle \\
& 6 \cdot 7 \\
= & \langle \text{ Evaluation } \rangle \\
& (10 - 4) \cdot (12 - 5) \\
\leq & \langle \text{ “Monotonicity of } \cdot \text{” with} \\
& \quad \text{“Antitonicity of } - \text{” with Fact `} 3 \leq 4 \text{`} \\
& \rangle \\
& (10 - 3) \cdot (12 - 5) \\
\leq & \langle \text{ “Monotonicity of } \cdot \text{” with} \\
& \quad \text{“Antitonicity of } - \text{” with Fact `} 4 \leq 5 \text{`} \\
& \rangle \\
& (10 - 3) \cdot (12 - 4) \\
= & \langle \text{ Evaluation } \rangle \\
& 7 \cdot 8
\end{array}
$$

with_2 Works Also With ≡ — Example Using “Isotonicity of +”

- _+_ : ℕ → ℕ → ℕ is isotonic in the first argument:
  $x \leq y \quad \equiv \quad x + z \leq y + z$ is a theorem

---

```
Calculation:
    2 + n
  ≤⟨ “Isotonicity of +” with Fact `2 ≤ 3` ⟩
    3 + n
```

This step can be justified without “with” as follows:

```
Calculation:
    2 + n ≤ 3 + n
  ≡⟨ “Identity of ≡” ⟩
    true  ≡  2 + n ≤ 3 + n
  ≡⟨ Fact `2 ≤ 3` ⟩
    2 ≤ 3  ≡  2 + n ≤ 3 + n
      — This is “Isotonicity of +”
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-19

LADM Chapter 4: “Relaxing the Proof Style” — New Proof Structures

Plan for Today

- **LADM Chapter 4: "Relaxing the Proof Style"** — **New Proof Structures**
  - Transitivity calculations with implication ⇒ or consequence ⇐
  - Proving implications: **Assuming** the antecedent
  - Proving **By cases**
  - **Using** theorems as proof methods
    - Proof by Contrapositive
    - Proof by Mutual Implication
- Coming up: LADM chapters 8 and 9.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-19

Part 1: Subproofs, Abbreviated Proofs for Implications

CalcCheck: Subproof Hint Items

You have used the following kinds of hint items:

- Theorem name references **"Identity of ≡"**
- Theorem number references **(3.32)**
- Certain key words and key phrases: Substitution, Evaluation, Induction hypothesis
- Fact \`*Expression*\`
- Composed hint items: "Identity of +" with \`*Substitution*\`
  "Monotonicity of +" with *HintItem*

A new kind of hint item: Subproof for \`*Expression*\`:
  Proof

For example, Fact \`3 = 2 + 1\` is really syntactic sugar for a subproof:

```
   3 · x
=  ⟨ Subproof for `3 = 2 + 1`:
        By evaluation ⟩
   (2 + 1) · x
```

Abbreviated Proofs for Implications

This:

$$\begin{array}{ll} & p \\ \equiv & \langle \text{Why} \quad p \equiv q \rangle \\ & q \\ \Rightarrow & \langle \text{Why} \quad q \Rightarrow r \rangle \\ & r \end{array}$$

proves: $p \Rightarrow r$

Because:

$$\begin{array}{ll} & (p \equiv q) \wedge (q \Rightarrow r) \\ \Rightarrow & \langle \text{(3.82b) Transitivity of } \Rightarrow \rangle \\ & p \Rightarrow r \end{array}$$

This proof style will not be allowed in questions "belonging" to LADM Chapter 3!

(4.1) — Creating the Proof "Bottom-up"

Proving (4.1) $p \Rightarrow (q \Rightarrow p)$**:**

$$\begin{array}{ll} & p \\ \Rightarrow & \langle \text{(3.76a) Weakening } p \Rightarrow p \vee q \ \rangle \\ & \neg q \vee p \\ \equiv & \langle \text{(3.59) Material implication} \rangle \\ & q \Rightarrow p \end{array}$$

...... **Rabbit!**

We have: **Axiom (3.58) Consequence**: $p \Leftarrow q \quad \equiv \quad q \Rightarrow p$

This means that the $\Leftarrow$ relation is the **converse** of the $\Rightarrow$ relation.

Theorem: The converse of a transitive relation is transitive again, and the converse of an order is an order again.

CalcCheck supports **activation** of converse properties, enabling **reversed presentations following mathematical habits** of transitivity calculations such as the above.

— "... propositional logic following LADM chapters 3 and 4 ..."

(4.1) Using "Consequence" Implicitly

Theorem (4.1): $p \Rightarrow (q \Rightarrow p)$
Proof:

$$\begin{array}{ll} & q \Rightarrow p \\ \equiv & \langle \text{"Material implication"} \rangle \\ & \neg q \vee p \\ \Leftarrow & \langle \text{"Strengthening" (3.76a) — used as } \texttt{p} \vee \texttt{q} \Leftarrow \texttt{p} \rangle \\ & p \end{array}$$

In CalcCheck, this requires that

Axiom (3.58) "Consequence" "Definition of $\Leftarrow$": $p \Leftarrow q \quad \equiv \quad q \Rightarrow p$

is **activated** as **converse property**.

(4.1) Using "Consequence" Explicitly — "Proof for this:"

In CALCCHECK, if "Consequence" is not **activated** as **converse property**, then $\Leftarrow$ is a separate operator requiring explicit conversion:

Theorem (4.1): $p \Rightarrow (q \Rightarrow p)$
Proof:

$p \Rightarrow (q \Rightarrow p)$
$\equiv$ ⟨ "Consequence" ⟩
$(q \Rightarrow p) \Leftarrow p$
Proof for this:
$q \Rightarrow p$
$\equiv$ ⟨ "Material implication" ⟩
$\neg q \vee p$
$\Leftarrow$⟨ "Strengthening" (3.76a), "Consequence" ⟩
p

("Proof for this:" is shorthand for the subproof to the right. It implements the frequent proof presentation pattern of transforming the goal, and then using a different kind of proof for the transformed goal.)

Theorem (4.1): $p \Rightarrow (q \Rightarrow p)$
Proof:

$p \Rightarrow (q \Rightarrow p)$
$\equiv$ ⟨ "Consequence" ⟩
$(q \Rightarrow p) \Leftarrow p$
$\equiv$ ⟨ **Subproof for** \`$(q \Rightarrow p) \Leftarrow p$\`:
$q \Rightarrow p$
$\equiv$ ⟨ "Material implication" ⟩
$\neg q \vee p$
$\Leftarrow$⟨ "Strengthening" (3.76a), "Consequence" ⟩
p
⟩
true

(4.2) Left-Monotonicity of $\vee$: $(p \Rightarrow q) \Rightarrow (p \vee r \Rightarrow q \vee r)$

Start from the right because there is more structure — therefore aim for "$\Leftarrow$" at the end:

$p \vee r \Rightarrow q \vee r$
$\equiv$ ⟨ (3.57) Definition of $\Rightarrow$ $p \Rightarrow q \equiv p \vee q \equiv q$ ⟩
$p \vee r \vee q \vee r \equiv q \vee r$
$\equiv$ ⟨ (3.26) Idempotency of $\vee$ ⟩
$p \vee q \vee r \equiv q \vee r$
$\equiv$ ⟨ (3.27) Distributivity of $\vee$ over $\equiv$ ⟩
$(p \vee q \equiv q) \vee r$
$\equiv$ ⟨ (3.57) Definition of $\Rightarrow$ $p \Rightarrow q \equiv p \vee q \equiv q$ ⟩
$(p \Rightarrow q) \vee r$
$\Leftarrow$ ⟨ (3.76a) Strengthening $p \Rightarrow p \vee q$ ⟩
$p \Rightarrow q$

(4.3) Left-Monotonicity of $\wedge$: $(p \Rightarrow q) \Rightarrow p \wedge r \Rightarrow q \wedge r$

($\Rightarrow$ associates to the right...)

$p \wedge r \Rightarrow q \wedge r$
$\equiv$ ⟨ (3.60) Definition of $\Rightarrow$ ⟩
$p \wedge r \wedge q \wedge r \equiv p \wedge r$
$\equiv$ ⟨ (3.38) Idempotency of $\wedge$ ⟩
$(p \wedge q) \wedge r \equiv p \wedge r$
$\equiv$ ⟨ (3.49) Semi-distributivity of $\wedge$ ⟩
$((p \wedge q) \equiv p) \wedge r \equiv r$
$\equiv$ ⟨ (3.60) Definition of $\Rightarrow$ ⟩
$(p \Rightarrow q) \wedge r \equiv r$
$\equiv$ ⟨ (3.60) Definition of $\Rightarrow$ ⟩
$r \Rightarrow (p \Rightarrow q)$
$\Leftarrow$ ⟨ (4.1) $p \Rightarrow (q \Rightarrow p)$ ⟩
$p \Rightarrow q$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-19

Part 2: Assuming the Antecedent

Proving Implications...

How to prove the following?

"=-Congruence of +": $b = c \quad \Rightarrow \quad a + b = a + c$

"Leibniz as Axiom" can help:

Lemma "=-congruence of +": $b = c \Rightarrow a + b = a + c$
Proof:
$b = c \Rightarrow a + b = a + c$
$\equiv$ ⟨ Substitution ⟩
$b = c \Rightarrow (a + z)[z := b] = (a + z)[z := c]$
— This is "Leibniz"

It may be nicer to turn this into a situation where the inference rule Leibniz (1.5) can be used again. . .

Lemma "=-congruence of +": $b = c \Rightarrow a + b = a + c$
Proof:
Assuming \`$b = c$\`: ▪▪▪▪▪▪ Assuming the Antecedent
$a + b$
$=$ ⟨ Assumption \`$b = c$\` ⟩
$a + c$

Assuming the Antecedent

To prove an implication $p \Rightarrow q$
we can prove its conclusion q using p as **assumption**:

Assuming \`$p$\`:
Proof of q
possibly using: Assumption \`$p$\`

Justification:

(4.4) **(Extended) Deduction Theorem:** Suppose adding $P_1, \ldots, P_n$ as axioms to propositional logic **E**, **with the free variables of the P_i considered to be constants**, allows Q to be proved.

Then $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ is a theorem.

That is:

Assumptions **cannot** be used with substitutions (with '$a, b := e, f$')
— just like induction hypotheses.

"Assuming the Antecedent" is not allowed in questions "belonging to" LADM chapt. 3!

Inference Rule for Proving Implications: ⇒-Introduction

One way to prove $P \Rightarrow Q$:

> **Assuming** \`$P$\`:
>
> > *Proof of* Q
> > *possibly using:* Assumption \`$P$\`

(And **Assuming** \`$P$\`: … can only prove theorems of shape $P \Rightarrow \cdots$.)

This directly corresponds to an application of the inference rule "⇒-Introduction" (which is missing in the Rosen book used in COMPSCI 1DM3):

$$\frac{\begin{array}{c}\ulcorner P \urcorner\\ \vdots \\ Q\end{array}}{P \Rightarrow Q}\ \Rightarrow\text{-Intro} \qquad\qquad \frac{\begin{array}{c}\ulcorner x : A \urcorner\\ \vdots \\ e : B\end{array}}{(\lambda x : A \bullet e) : A \to B}\ \lambda\text{-Abstraction}$$

Proving and Using Implication Theorems: `Assuming` and `with`[2]

Using "Cancellation of ·": $z \neq 0 \Rightarrow (z \cdot x = z \cdot y \quad \equiv \quad x = y)$

Theorem "Non-zero multiplication": $a \neq 0 \Rightarrow b \neq 0 \Rightarrow a \cdot b \neq 0$
Proof:
  Assuming \`$a \neq 0$\`, \`$b \neq 0$\`:
    $a \cdot b \neq 0$
  $\equiv$ ⟨ "Definition of ≠" ⟩
    $\neg (a \cdot b = 0)$
  $\equiv$ ⟨ "Zero of ·" ⟩
    $\neg (a \cdot b = a \cdot 0)$
  $\equiv$ ⟨ "Cancellation of ·" with assumption \`$a \neq 0$\` ⟩
    $\neg (b = 0)$
  $\equiv$ ⟨ "Definition of ≠", Assumption \`$b \neq 0$\` ⟩
    true

- *HintItem1* `with` *HintItem2* `and` *HintItem3*, *HintItem4*   parses as
  (*HintItem1* `with` (*HintItem2* `and` *HintItem3*)), *HintItem4*

(4.3) Left-Monotonicity of ∧ (shorter proof, LADM-style)

(4.3) $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

PROOF:

  Assume $p \Rightarrow q$ (which is equivalent to $p \wedge q \equiv p$)

    $p \wedge r$
  $\equiv$ ⟨ Assumption $p \wedge q \equiv p$ ⟩
    $p \wedge q \wedge r$
  $\Rightarrow$ ⟨ (3.76b) Weakening ⟩
    $q \wedge r$

How to do "which is equivalent to" in CALCCHECK?

- Transform before assuming
- or transform the assumption when using it
- or "Assuming … and using with …"

Transform Before Assuming — Proof for this:

Theorem (4.3) "Left-monotonicity of ∧" "Monotonicity of ∧":

$(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

Proof:

$(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

≡ ⟨ "Definition of ⇒ from ∧" ⟩

$(p \wedge q \equiv p) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

Proof for this:

Assuming \`$p \wedge q \equiv p$\`:

$p \wedge r$

≡ ⟨ Assumption \`$p \wedge q \equiv p$\` ⟩

$p \wedge q \wedge r$

⇒⟨ "Weakening" ⟩

$q \wedge r$

Transform Assumption When Used — `with`$_3$

(4.3) $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$ — LADM

PROOF:

Assume $p \Rightarrow q$ (which is equivalent to $p \wedge q \equiv p$)

$p \wedge r$

≡ ⟨ Assumption $p \wedge q \equiv p$ ⟩

$p \wedge q \wedge r$

⇒ ⟨ (3.76b) Weakening ⟩

$q \wedge r$

Theorem (4.3) "Left-monotonicity of ∧" "Monotonicity of ∧": —CALCCHECK

$(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

Proof:

Assuming \`$p \Rightarrow q$\`:

$p \wedge r$

≡ ⟨ Assumption \`$p \Rightarrow q$\` with "Implication via ∧" ⟩

$p \wedge q \wedge r$

⇒⟨ "Weakening" ⟩

$q \wedge r$

Assuming ... and using with ...

(4.3) $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$ — LADM

PROOF:

Assume $p \Rightarrow q$ (which is equivalent to $p \wedge q \equiv p$)

$p \wedge r$

≡ ⟨ Assumption $p \wedge q \equiv p$ ⟩

$p \wedge q \wedge r$

⇒ ⟨ (3.76b) Weakening ⟩

$q \wedge r$

Theorem (4.3) "Left-monotonicity of ∧" "Monotonicity of ∧": — CALCCHECK

$(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

Proof:

Assuming \`$p \Rightarrow q$\` **and using with** "Implication via ∧":

$p \wedge r$

≡ ⟨ Assumption \`$p \Rightarrow q$\` ⟩

$p \wedge q \wedge r$

⇒⟨ "Weakening" ⟩

$q \wedge r$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-20

- **Proof Structures (LADM ch. 4)**
- **Introduction to Quantification (LADM ch. 8)**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-20

Part 1: Case Analysis and Other Structured Proofs

LADM General Case Analysis

(4.6) $(p \vee q \vee r) \wedge (p \Rightarrow s) \wedge (q \Rightarrow s) \wedge (r \Rightarrow s) \ \Rightarrow \ s$

Proof pattern for general case analysis:

Prove: S
By cases: P, Q, R
(proof of $P \vee Q \vee R$ — omitted if obvious)
Case P **:** (proof of $P \Rightarrow S$)
Case Q **:** (proof of $Q \Rightarrow S$)
Case R **:** (proof of $R \Rightarrow S$)

Case Analysis Example (4.2) $(p \Rightarrow q) \Rightarrow p \vee r \Rightarrow q \vee r$ — LADM vs. CalcCheck

Assume $p \Rightarrow q$
 Assume $p \vee r$
  Prove: $q \vee r$
  By Cases: p, r
   — $p \vee r$ holds by assumption
  Case p **:**

$$p$$
$\Rightarrow$ ⟨ Assumption $p \Rightarrow q$ ⟩
$$q$$
$\Rightarrow$ ⟨ Weakening (3.76a) ⟩
$$q \vee r$$

  Case r **:**

$$r$$
$\Rightarrow$ ⟨ Weakening (3.76a) ⟩
$$q \vee r$$

Theorem "Monotonicity of ∨":
 $(p \Rightarrow q) \Rightarrow (p \vee r) \Rightarrow (q \vee r)$
Proof:
 Assuming \`$p \Rightarrow q$\`, \`$p \vee r$\`:
  By cases: \`$p$\`, \`$r$\`
   Completeness: By assumption \`$p \vee r$\`
   Case \`$p$\`:
     p — **This is** assumption \`$p$\`
    ⇒⟨ Assumption \`$p \Rightarrow q$\` ⟩
     q
    ⇒⟨ "Weakening" ⟩
     $q \vee r$
   Case \`$r$\`:
     r — **This is** assumption \`$r$\`
    ⇒⟨ "Weakening" ⟩
     $q \vee r$

"By cases:" with Calculation for "Completeness:" . . .

- In **By** cases: \`$P_1$\`, \`$P_2$\`, . . . , \`$P_n$\`,
  after "**Completeness:**", a proof for $P_1 \vee P_2 \vee \cdots \vee P_n$ is needed
- This can be any kind of proof.
- Inside the **Case** \`$p$\`**:** block, you may use **Assumption** \`$p$\`.

Theorem (15.34) "Positivity of squares": $b \neq 0 \Rightarrow$ pos $(b \cdot b)$
Proof:
 Assuming \`$b \neq 0$\`:
  By cases: \`pos $b$\`, \`¬ pos $b$\`
   Completeness:
     pos b ∨ ¬ pos b
    ≡ ⟨ "Excluded middle" ⟩
     true
   Case \`pos $b$\`:
     pos $(b \cdot b)$

The Predecessor Function pred on ℕ

The "predecessor function" pred is total; since zero has no predecessor, it maps 0 to 0.

Declaration: pred : ℕ → ℕ
Axiom "Predecessor of zero": pred 0 = 0
Axiom "Predecessor of successor": pred (suc n) = n

We then have:

Theorem "Zero or successor of predecessor": $n = 0 \vee n =$ suc (pred n)

This is useful for case analysis proofs of properties that so far you have shown "By induction" without using the induction hypothesis:

Theorem "Right-identity of subtraction": $m - 0 = m$
Proof:
 By cases: \`$m = 0$\`, \`$m =$ suc (pred $m$)\`
  Completeness: By "Zero or successor of predecessor"
  Case \`$m = 0$\`:
    ?
  Case \`$m =$ suc (pred $m$)\`:

Proof by Contrapositive

(3.61) **Contrapositive:** $p \Rightarrow q \quad \equiv \quad \neg q \Rightarrow \neg p$

(4.12) **Proof method:** Prove $P \Rightarrow Q$ by proving its contrapositive $\neg Q \Rightarrow \neg P$

Proving $x + y \geq 2 \quad \Rightarrow \quad x \geq 1 \lor y \geq 1$:

$$
\begin{array}{ll}
& \neg(x \geq 1 \lor y \geq 1) \\
\equiv & \langle \text{ De Morgan (3.47) } \rangle \\
& \neg(x \geq 1) \land \neg(y \geq 1) \\
\equiv & \langle \text{ Def. } \geq \text{ (15.39) with Trichotomy (15.44) } \rangle \\
& x < 1 \land y < 1 \\
\Rightarrow & \langle \text{ Monotonicity of + (15.42) } \rangle \\
& x + y < 1 + 1 \\
\equiv & \langle \text{ Def. 2 } \rangle \\
& x + y < 2 \\
\equiv & \langle \text{ Def. } \geq \text{ (15.39) with Trichotomy (15.44) } \rangle \\
& \neg(x + y \geq 2)
\end{array}
$$

Proof by Contrapositive in CalcCheck — `Using`

- "**Using** *HintItem1*: *subproof1 subproof2*"
  is processed as "**By** *HintItem1* **with** *subproof1* **and** *subproof2*"
- If you get the subproof goals wrong, the `with` heuristic has no chance to succeed...

Theorem "Example for use of `*Contrapositive*`":
$x + y \geq 2 \Rightarrow x \geq 1 \lor y \geq 1$
Proof:
  Using "Contrapositive":
    Subproof for `$\neg (x \geq 1 \lor y \geq 1) \Rightarrow \neg (x + y \geq 2)$`:
      $\neg (x \geq 1 \lor y \geq 1)$
    $\equiv$ ⟨ "De Morgan" ⟩
      $\neg (x \geq 1) \land \neg (y \geq 1)$
    $\equiv$ ⟨ "Complement of <" with (3.14) ⟩
      $x < 1 \land y < 1$
    $\Rightarrow$ ⟨ "<-Monotonicity of +" ⟩
      $x + y < 1 + 1$
    $\equiv$ ⟨ Evaluation ⟩
      $x + y < 2$
    $\equiv$ ⟨ "Complement of <" with (3.14) ⟩
      $\neg (x + y \geq 2)$

Proof by Contradiction

(3.74) $p \Rightarrow false \quad \equiv \quad \neg p$

(4.9) **Proof by contradiction:** $\neg p \Rightarrow false \quad \equiv \quad p$

"This proof method is overused"

Theorem "Reflexivity of ≡": $b \equiv b$
Proof:
  Using "Proof by contradiction":
    Subproof for `$\neg (b \equiv b) \Rightarrow$ false`:
      $\neg (b \equiv b)$
    $\equiv$ ⟨ "Commutativity of ¬ with ≡" ⟩
      $\neg b \equiv b$
    $\equiv$ ⟨ "Definition of ¬ from ≡" ⟩
      false

Use short proof via (3.3)!

If you intuitively try to do a proof by contradiction:

- Formalise your proof
- This may already contain a direct proof!
- So check whether contradiction is still necessary
- ..., or whether your proof can be transformed into one that does not use contradiction.

Proof by Mutual Implication — `Using`

(3.80) **Mutual implication:** $(p \Rightarrow q) \wedge (q \Rightarrow p) \ \equiv\ p \equiv q$

Theorem (15.47) "Indirect Equality" "Indirect Equality from below":
  $a = b \ \equiv\ (\forall z \bullet z \leq a \ \equiv\ z \leq b)$
Proof:
  Using "Mutual implication":   ▪▪▪▪▪▪"Antisymmetry of ⇒" would work as well
    Subproof for \`$a = b \ \Rightarrow\ (\forall z \bullet z \leq a \ \equiv\ z \leq b)$\`:
      Assuming \`$a = b$\`:
        For any \`$z$\`:
          By Assumption \`$a = b$\`
    Subproof for \`$(\forall z \bullet z \leq a \ \equiv\ z \leq b) \ \Rightarrow\ a = b$\`:
      Assuming "A" \`$(\forall z \bullet z \leq a \ \equiv\ z \leq b)$\`:
          $a = b$
        $\equiv$ ⟨ "Antisymmetry of ≤" ⟩
          $a \leq b \ \wedge\ b \leq a$
        $\equiv$ ⟨ Assumption "A" ⟩
          $a \leq a \ \wedge\ b \leq b$
        $\equiv$ ⟨ "Reflexivity of ≤", "Idempotency of ∧" ⟩
          true

Proof by Mutual Implication — `Using`

(3.80) **Mutual implication:** $(p \Rightarrow q) \wedge (q \Rightarrow p) \ \equiv\ p \equiv q$

Theorem "Cancellation of unary minus": $-\,a = -\,b \ \equiv\ a = b$
Proof:
  Using "Mutual implication":
    Subproof: ▪▪▪▪▪▪ Subproof goals determined by the enclosed proof can be omitted.
      Assuming \`$a = b$\`:
          $-\,a$
        $=$ ⟨ Assumption \`$a = b$\` ⟩
          $-\,b$
    Subproof:
      Assuming \`$-\,a = -\,b$\`:
          a
        $=$ ⟨ "Self-inverse of unary minus" ⟩
          $-\,-\,a$
        $=$ ⟨ Assumption \`$-\,a = -\,b$\` ⟩
          $-\,-\,b$
        $=$ ⟨ "Self-inverse of unary minus" ⟩
          b

Opportunities for Structured Proofs: LADM Theory of Integers — Positivity and Ordering

(15.30) **Axiom, Addition in pos:** $\text{pos}\ a \wedge \text{pos}\ b \ \Rightarrow\ \text{pos}\ (a + b)$
(15.31) **Axiom, Multiplication in pos:** $\text{pos}\ a \wedge \text{pos}\ b \ \Rightarrow\ \text{pos}\ (a \cdot b)$
(15.32) **Axiom:** $\neg\,\text{pos}\ 0$
(15.33) **Axiom:** $b \neq 0 \ \Rightarrow\ (\text{pos}\ b \ \equiv\ \neg\text{pos}\ (-b))$
(15.34) **Positivity of Squares:** $b \neq 0 \ \Rightarrow\ \text{pos}\ (b \cdot b)$
(15.35) $\text{pos}\ a \ \Rightarrow\ (\text{pos}\ b \ \equiv\ \text{pos}\ (a \cdot b)$
(15.36) **Axiom, Less:** $a < b \ \equiv\ \text{pos}\ (b - a)$
(15.37) **Axiom, Greater:** $a > b \ \equiv\ \text{pos}\ (a - b)$
(15.38) **Axiom, At most:** $a \leq b \ \equiv\ a < b \vee a = b$
(15.39) **Axiom, At least:** $a \geq b \ \equiv\ a > b \vee a = b$
(15.40) **Positive elements:** $\text{pos}\ b \ \equiv\ 0 < b$
(15.41) **Transitivity:** $(a)\ \ a < b \ \wedge\ b < c \ \Rightarrow\ a < c$ $(b)\ \ a \leq b \ \wedge\ b < c \ \Rightarrow\ a < c$
  $(c)\ \ a < b \ \wedge\ b \leq c \ \Rightarrow\ a < c$ $(d)\ \ a \leq b \ \wedge\ b \leq c \ \Rightarrow\ a \leq c$
(15.42) **Monotonicity of +:** $a < b \ \equiv\ a + d \ <\ b + d$
(15.43) **Monotonicity of ·:** $0 < d \ \Rightarrow\ (a < b \ \equiv\ a \cdot d \ <\ b \cdot d)$
(15.44) **Trichotomy:** $(a < b \ \equiv\ a = b \ \equiv\ a > b) \ \wedge\ \neg(a < b \ \wedge\ a = b \ \wedge\ a > b)$
(15.45) **Antisymmetry of ≤:** $a \leq b \ \wedge\ a \geq b \ \equiv\ a = b$
(15.46) **Reflexivity of ≤:** $a \leq a$

Proof Structures Can Be Freely Combined…

Theorem (15.35) "Positivity under positive ·": pos a ⇒ (pos b ≡ pos (a · b))
Proof:
  Assuming `pos $a$`:
    Using "Mutual implication":
      Subproof for `pos $b$ ⇒ pos ($a$ · $b$)`:
          pos b ⇒ pos (a · b)
        ⇐⟨ "Positivity under ·" ⟩
          pos a — **This is** Assumption `pos $a$`
      Subproof for `pos ($a$ · $b$) ⇒ pos $b$`:
        Using "Contrapositive":
          Subproof for `¬ pos $b$ ⇒ ¬ pos ($a$ · $b$)`:
            By cases: `$b$ = 0`, `$b$ ≠ 0`
              Completeness: By "Definition of ≠", "LEM"
              Case `$b$ = 0`:
                  ¬ pos b ⇒ ¬ pos (a · b)
                ≡ ⟨ Assumption `$b$ = 0`, "Zero of ·" ⟩
                  ¬ pos 0 ⇒ ¬ pos 0 — **This is** "Reflexivity of ⇒"
              Case `$b$ ≠ 0`:
                  ¬ pos b
                ≡ ⟨ (15.33b) with Assumption `$b$ ≠ 0` ⟩

The CalcCheck Language — Calculational Proofs on Steroids

Besides calculations, CalcCheck has the following proof structures:

- By *hint* — for discharging simple proof obligations,
- Assuming '*expression*': — for assuming the antecedent,
- By cases: '$expression_1$',…,'$expression_n$' — for proofs by case analysis
- By induction on '*var* : *type*': — for proofs by induction
- Using *hint*: — for turning theorems into inference rules
- For any '*var* : *type*': — corresponding to ∀-introduction

This does not sound that different from LADM —

  — but in CalcCheck, these are actually used!

Structured Proof Example from LADM — And Fully Formal in CalcCheck

Theorems for *pos*
(15.34) $b \neq 0 \Rightarrow pos(b{\cdot}b)$

We prove (15.34). For arbitrary nonzero b in D, we prove $pos(b{\cdot}b)$ by case analysis: either $pos.b$ or $\neg pos.b$ holds (see (15.33)).

Case $pos.b$. By axiom (15.31) with $a,b := b,b$, $pos(b{\cdot}b)$ holds.

Case $\neg pos.b \wedge b \neq 0$. We have the following.

$$\begin{array}{ll} & pos(b{\cdot}b) \\ = & \langle (15.23), \text{ with } a,b := b,b \rangle \\ & pos((-b){\cdot}(-b)) \\ \Leftarrow & \langle \text{Multiplication } (15.31) \rangle \\ & pos(-b) \wedge pos(-b) \\ = & \langle \text{Idempotency of } \wedge \ (3.38) \rangle \\ & pos(-b) \\ = & \langle \text{Double negation } (3.12) \text{ —note that } b \neq 0;\ (15.33) \rangle \\ & \neg pos.b \quad \text{—the case under consideration} \end{array}$$

Theorem (15.34) "Positivity of squares": b ≠ 0 ⇒ pos (b · b)
Proof:
  Assuming `$b$ ≠ 0`:
    By cases: `pos $b$`, `¬ pos $b$`
      Completeness: By "Excluded middle"
      Case `pos $b$`:
        By "Positivity under ·" (15.31) with assumption `pos $b$`
      Case `¬ pos $b$`:
          pos (b · b)
        ≡ ⟨ (15.23) `− $a$ · − $b$ = $a$ · $b$` ⟩
          pos ((− b) · (− b))
        ⇐⟨ "Positivity under ·" (15.31) ⟩
          pos (− b) ∧ pos (− b)
        ≡ ⟨ "Idempotency of ∧", "Double negation" ⟩
          ¬ ¬ pos (− b)
        ≡ ⟨ "Positivity under unary minus" (15.33) with assumption `$b$ ≠ 0` ⟩
          ¬ pos b — **This is** assumption `¬ pos $b$`

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-20

Part 2: Introduction to Quantification (start LADM chapt. 8),

Quantification expansion

Counting Integral Points

How many integral points are in the triangle

$$\begin{array}{ccc} (0,n) & & \\ | & \diagdown & \\ (0,0) & — & (n,0) \end{array}$$

?

$(\sum x,y : \mathbb{N} \mid x + y \le n \bullet 1)$

How many integral points are in the circle of radius n around $(0,0)$?

$(\sum x,y : \mathbb{Z} \mid x \cdot x + y \cdot y \le n \cdot n \bullet 1)$

Sum Quantification Examples

$(\sum k : \mathbb{N} \mid k < 5 \bullet k)$

- "The sum of all natural numbers less than five"

$(\sum k : \mathbb{N} \mid k < 5 \bullet k \cdot k)$

- "For all natural numbers k that are less than 5, adding up the value of $k \cdot k$"
- "The sum of all squares of natural numbers less than five"

$(\sum x,y : \mathbb{N} \mid x \cdot y = 120 \bullet 2 \cdot (x + y))$

- "For all natural numbers x and y with product 120, adding up the value of $2 \cdot (x + y)$"
- "The sum of the perimeters of all integral rectangles with area 120"

Product Quantification Examples

- "The factorial of n is the product of all positive integers up to n"

  $\mathit{factorial} : \mathbb{N} \to \mathbb{N}$

  $\mathit{factorial}\ n = (\prod k : \mathbb{N} \mid 0 < k \leq n \bullet k)$

- "The product of all odd natural numbers below 50."

  $(\prod n : \mathbb{N} \mid \neg(2 \mid n) \wedge n < 50 \bullet n)$

  $(\prod k : \mathbb{N} \mid 2 \cdot k + 1 < 50 \bullet 2 \cdot k + 1)$

  $(\prod k : \mathbb{N} \mid k < 25 \bullet 2 \cdot k + 1)$

Sum and Product Quantification

$(\sum x \mid R \bullet E)$

- "For all x satisfying R, summing up the value of E"
- "The sum of all E for x with R"

$(\sum x : T \bullet E)$

- "For all x of type T, summing up the value of E"
- "The sum of all E for x of type T"

$(\prod x \mid R \bullet E)$

- "The product of all E for x with R"

$(\prod x : T \bullet E)$

- "The product of all E for x of type T"

General Shape of Sum and Product Quantifications

$$(\sum x : t_1;\ y, z : t_2 \mid R \bullet E)$$

$$(\prod x : t_1;\ y, z : t_2 \mid R \bullet E)$$

- Any number of **variables** x, y, z can be quantified over
- The quantified variables may have **type annotations** (which act as **type declarations**)
- Expression $R : \mathbb{B}$ is the **range** of the quantification
- Expression E is the **body** of the quantification
- E will have a number type ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$)
- Both R and E may refer to the **quantified variables** x, y, z
- The type of the whole quantification expression is the type of E.

LADM/CalcCheck Quantification Notation

Conventional sum quantification notation: $\sum_{i=1}^{n} e \quad = \quad e[i := 1] + \ldots + e[i := n]$

The textbook uses a different, but systematic **linear** notation:

$$(\sum i \mid 1 \leq i \leq n : e) \qquad \text{or} \qquad (+ i \mid 1 \leq i \leq n : e)$$

We use a variant with a "spot" "•" instead of the colon ":" and only use "big" operators:

$(\sum i \mid 1 \leq i \leq n \bullet e)$ — `(\sum i \with 1 \leq i \<= n \spot e)`

Reasons for using this kind of **linear** quantification notation:

- Clearly delimited introduction of **quantified variables** (**dummies**)
- **Arbitrary** Boolean expressions can define the **range**
  $(\sum i \mid 1 \leq i \leq 7 \wedge even\ i \bullet i) = 2 + 4 + 6$
- The notation extends easily to multiple quantified variables:
  $(\sum i, j : \mathbb{Z} \mid 1 \leq i < j \leq 4 \bullet i/j) = 1/2 + 1/3 + 1/4 + 2/3 + 2/4 + 3/4$

Meaning of Sum Quantification

Let i be a variable list, R a Boolean expression, and E an expression of a number type.

The **meaning** of $(\sum i \mid R \bullet E)$ in state s is:

- the sum of the meanings of E
- in all those states that satisfy R
- and are different from s at most in variables in i.

Examples:

- $(\sum i, j \mid i = j = i + 1 \bullet i \cdot j) \quad = \quad 0$
- $(\sum i, j \mid 0 < i < j < 4 \bullet i \cdot j) \quad = \quad 1 \cdot 2 + 1 \cdot 3 + 2 \cdot 3$
- $(\sum i, j \mid 1 \leq i \leq 2 \wedge 3 \leq j \leq 4 \bullet i \cdot j) \quad = \quad 1 \cdot 3 + 1 \cdot 4 + 2 \cdot 3 + 2 \cdot 4$
- In state $[(i, 7), (j, 11), (k, 3)]$, we have:
  $(\sum i, j \mid 0 < i < j < k \bullet i \cdot j) \quad = \quad 1 \cdot 2$

Expanding Sum and Product Quantification

Sum quantification ($\sum$) is **"addition (+) of arbitrarily many terms"**:

$(\sum i \mid 5 \leq i < 9 \bullet i \cdot (i + 1))$

= ⟨ Quantification expansion ⟩

$(i \cdot (i+1))[i := 5] + (i \cdot (i+1))[i := 6] + (i \cdot (i+1))[i := 7] + (i \cdot (i+1))[i := 8]$

= ⟨ Substitution ⟩

$5 \cdot (5 + 1) + 6 \cdot (6 + 1) + 7 \cdot (7 + 1) + 8 \cdot (8 + 1)$

Product quantification ($\prod$) is **"multiplication ($\cdot$) of arbitrarily many factors"**:

$(\prod i \mid 0 \leq i < 3 \bullet 5 \cdot i + 1)$

= ⟨ Quantification expansion ⟩

$(5 \cdot i + 1)[i := 0] \cdot (5 \cdot i + 1)[i := 1] \cdot (5 \cdot i + 1)[i := 2]$

= ⟨ Substitution ⟩

$(5 \cdot 0 + 1) \cdot (5 \cdot 1 + 1) \cdot (5 \cdot 2 + 1)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-24

General Quantification — LADM Chapter 8

Quantification Examples

$(\sum i \mid 0 \le i < 4 \bullet i \cdot 8)$
$= \langle$ Quantification expansion, substitution $\rangle$
$0 \cdot 8 + 1 \cdot 8 + 2 \cdot 8 + 3 \cdot 8$

$(\prod i \mid 0 \le i < 3 \bullet i + (i + 1))$
$= \langle$ Quantification expansion, substitution $\rangle$
$(0 + 1) \cdot (1 + 2) \cdot (2 + 3)$

$(\forall i \mid 1 \le i < 3 \bullet i \cdot d \ne 6)$
$= \langle$ Quantification expansion, substitution $\rangle$
$1 \cdot d \ne 6 \wedge 2 \cdot d \ne 6$

$(\exists i \mid 0 \le i < 6 \bullet b\, i = 0)$
$= \langle$ Quantification expansion, substitution $\rangle$
$b\,0 = 0 \vee b\,1 = 0 \vee b\,2 = 0 \vee b\,3 = 0 \vee b\,4 = 0 \vee b\,5 = 0$

General Quantification

It works not only for $+$, $\wedge$, $\vee$ . . .

Let a type T and an operator $\star : T \times T \to T$ be given.
If for an appropriate $u : T$ we have:

- **Symmetry:** $b \star c = c \star b$
- **Associativity:** $(b \star c) \star d = b \star (c \star d)$
- **Identity u:** $u \star b = b = b \star u$

we may use $\star$ as quantification operator:

$$(\star\, x : T_1, y : T_2 \mid R \bullet E)$$

- $R : \mathbb{B}$ is the **range** of the quantification
- $E : T$ is the **body** of the quantification
- E and R may refer to the **quantified variables** x and y
- The type of the whole quantification expression is T.

General Quantification: Instances

Let a type T and an operator $\star : T \times T \to T$ be given.
If for an appropriate $u : T$ we have:

- **Symmetry:** $b \star c = c \star b$
- **Associativity:** $(b \star c) \star d = b \star (c \star d)$
- **Identity** u**:** $u \star b = b = b \star u$

we may use $\star$ as quantification operator: $(\star\, x : T_1, y : T_2 \mid R \bullet E)$

- $\_\lor\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.24), associative (3.25), and has *false* as identity (3.30) — the "big operator" for $\lor$ is $\exists$":
  $$(\exists\, k : \mathbb{N} \mid k > 0 \bullet k \cdot k < k + 1)$$
- $\_\land\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.36), associative (3.27), and has *true* as identity (3.39) — the "big operator" for $\land$ is $\forall$":
  $$(\forall\, k : \mathbb{N} \mid k > 2 \bullet \mathit{prime}\ k \Rightarrow \neg\, \mathit{prime}\ (k + 1))$$
- $\_+\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is symmetric (15.2), associative (15.1), and has 0 as identity (15.3) — the "big operator" for $+$ is $\sum$":
  $$(\textstyle\sum n : \mathbb{Z} \mid 0 < n < 100 \land \mathit{prime}\ n \bullet n \cdot n)$$

Meaning of General Quantification

Let a type T, and a symmetric and associative operator $\star : T \times T \to T$ with identity $u : T$ be given.
Further let x be a **variable list**, R a Boolean expression, and E an expression of type T.

LADM: "Expression $(\star\, x : X \mid R \bullet E)$ denotes the application of operator $\star$ to the values of E for all x in X for which range R is true."

The **meaning** of $(\star\, x \mid R \bullet E)$ in state s is:

- the nested application of $\star$ to the meanings of E
- in all those states that satisfy R
- and are different from s at most in variables in x,

or u, if there are no such states.

Examples:

- $(\exists\, i,j \mid i = j = i + 1 \bullet i < j) \quad = \quad \mathit{false}$
- $(\forall\, i,j \mid i = j = i + 1 \bullet i < j) \quad = \quad \mathit{true}$
- $(\prod\, i,j \mid 5 = j = i + 1 \bullet i \cdot j) \quad = \quad 4 \cdot 5$
- $(\exists\, i,j \mid 0 < i \leq j < 3 \bullet i \geq j) \quad = \quad 1 \geq 1 \lor 1 \geq 2 \lor 2 \geq 2$

Bound / Free Variable Occurrences

$(\sum i : \mathbb{N} \mid i < x \bullet i + 1) = 10$ example expression

Is this true or false? In which states?

We have: $(\sum i : \mathbb{N} \mid i < x \bullet i + 1) = 10 \quad \equiv \quad x = 4$

The value of this example expression in a state depends only on x, not on i!

Renaming quantified variables <u>does not change the meaning</u>:

$$(\textstyle\sum i : \mathbb{N} \mid i < x \bullet i + 1) \quad = \quad (\textstyle\sum j : \mathbb{N} \mid j < x \bullet j + 1)$$

- **Occurrences** of quantified variables inside the quantified expression are **bound**
- Non-bound **variable occurences** are called **free**
- Variables of the same name may occur both free and bound in the same expression, e.g.: $3 \cdot i + (\sum i : \mathbb{N} \mid i < x \bullet 2 \cdot i)$
- The variable declarations after the quantification operator may be called **binding occurrences**.

Variable Binding is Everywhere! Including in Substitution!

Another example expression: $(x + 3 = 5 \cdot i)[i := 9]$

Is this true or false? In which states?

$$\begin{aligned} & (x + 3 = 5 \cdot i)[i := 9] \\ \equiv \; & \langle \text{Substitution}, \ldots \rangle \\ & x = 42 \end{aligned}$$

The value of $(x + 3 = 5 \cdot i)[i := 9]$ in a state depends only on x, not on i!

Renaming substituted variables does not change the meaning:

$$(x + 3 = 5 \cdot i)[i := 9] \qquad \equiv \qquad (x + 3 = 5 \cdot j)[j := 9]$$

- **Occurrences** of substituted variables inside the target expression are **bound**
- The variable occurrences to the left of := in substitutions may be called **binding occurrences**.
- Non-bound **variable occurences** are called **free**.

$$i > 0 \wedge (x + 3 = 5 \cdot i)[i := 7 + i]$$

- **Substitution does not bind to the right of := !**

Trivial Range Axioms

(8.13) **Axiom, Empty Range** (where u is the identity of $\star$):

$$(\star\, x \mid false \bullet P) \quad = \quad u$$

$$(\forall\, x \mid false \bullet P) \quad = \quad true$$

$$(\exists\, x \mid false \bullet P) \quad = \quad false$$

$$(\sum\, x \mid false \bullet P) \quad = \quad 0$$

$$(\prod\, x \mid false \bullet P) \quad = \quad 1$$

(8.14) **Axiom, One-point Rule:** **Provided $\neg occurs(`x', `E')$,**

$$(\star x \mid x = E \bullet P) \quad = \quad P[x := E]$$

The *occurs* Meta-Predicate

Definition: $occurs(`v', `e')$ means that at least one variable in the list v of variables occurs **free** in at least one expression in expression list e.

$occurs(`i, n', `(\sum\, i, n \mid 1 \le i \cdot n \le k \bullet n^i), (\sum\, n \mid 0 \le n < k \bullet n^i)')$ ✓

$occurs(`i', `(i \cdot (5 + i))[i := k + 2]')$ × **Substitution is a variable binder, too!**

$occurs(`i', `(i \cdot (5 + i))[i := i + 2]')$ ✓

The $\neg occurs$ Proviso for the One-point Rule

(8.14) **Axiom, One-point Rule for $\sum$:** Provided $\neg occurs(`x', `E')$,

$$(\sum x \mid x = E \bullet P) \quad = \quad P[x := E]$$

(8.14) **Axiom, One-point Rule for $\prod$:** Provided $\neg occurs(`x', `E')$,

$$(\prod x \mid x = E \bullet P) \quad = \quad P[x := E]$$

Examples:

- $(\sum x \mid x = 1 \bullet x \cdot y) \quad = \quad 1 \cdot y$
- $(\prod x \mid x = y + 1 \bullet x \cdot x) \quad = \quad (y+1) \cdot (y+1)$
- $(\sum x \mid x = (\sum x \mid 1 \le x < 4 \bullet x) \bullet x \cdot y) \quad = \quad (\sum x \mid 1 \le x < 4 \bullet x) \cdot y \quad = \quad 6 \cdot y$

Counterexamples:

- $(\sum x \mid x = x + 1 \bullet x) \quad ? \quad x + 1$ — "=" not valid!
- $(\prod x \mid x = 2 \cdot x \bullet y + x) \quad ? \quad y + 2 \cdot x$ — "=" not valid!

The $\neg occurs$ Proviso for the One-point Rule

(8.14) **Axiom, One-point Rule:** Provided $\neg occurs(`x', `E')$,

$$(\star x \mid x = E \bullet P) \quad = \quad P[x := E]$$

$$(\forall x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

$$(\exists x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

Examples:

- $(\forall x \mid x = 1 \bullet x \cdot y = y) \quad \equiv \quad 1 \cdot y = y$
- $(\exists x \mid x = y + 1 \bullet x \cdot x > 42) \quad \equiv \quad (y+1) \cdot (y+1) > 42$

Counterexamples:

- $(\forall x \mid x = x + 1 \bullet x = 42) \quad ? \quad x + 1 = 42$ — "≡" not valid!
- $(\exists x \mid x = 2 \cdot x \bullet y + x = 42) \quad ? \quad y + 2 \cdot x = 42$ — "≡" not valid!

One-point Rule with Example Calculation

(8.14) **Axiom, One-point Rule:** Provided $\neg occurs(`x', `E')$,

$$(\star\, x \mid x = E \bullet P) \quad = \quad P[x := E]$$

Example:

$$\begin{array}{ll}
 & (\sum\, i : \mathbb{N} \bullet 5 + 2 \cdot i < 7 \mid 5 + 7 \cdot i) \\
= & \langle \ldots \rangle \\
 & (\sum\, i : \mathbb{N} \bullet i = 0 \mid 5 + 7 \cdot i) \\
= & \langle \text{One-point rule} \rangle \\
 & (5 + 7 \cdot i)[i := 0] \\
= & \langle \text{Substitution} \rangle \\
 & 5 + 7 \cdot 0
\end{array}$$

Automatic extraction of $\neg occurs$ Provisos

(8.14) **Axiom, One-point Rule:** Provided $\neg occurs('x', 'E')$,

$$(\forall x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$
$$(\exists x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

Investigate the binders in scope at the metavariables P and E:

- P on the LHS occurs in scope of the binder $\forall x$
- P on the RHS occurs in scope of the binder $\_[x := \ldots]$

Therefore: Whether x occurs in P or not does not raise any problems.

- E on the LHS occurs in scope of the binder $\forall x$
- E on the RHS occurs in scope no binders

Therefore: An x that is free in E would be **bound** on the LHS,
but **escape** into freedom on the RHS!

CalcCheck derives and checks $\neg occurs$ provisos automatically.

Textual Substitution Revisited

Let E and R be expressions and let x be a variable. **Original definition:**

> We write: $E[x := R]$ or E^x_R
> to denote an expression that is the same as E but with all occurrences of x replaced by (R).

This was for expressions E built from **constants, variables, operator applications** only!

In presence of **variable binders**, such as $\sum$, $\prod$, $\forall$, $\exists$ and substitution,

- only **free** occurrences of x can be replaced
- **and** we need to avoid **"capture of free variables"**:

(8.11) Provided $\neg occurs('y', 'x, F')$,

$$(\star y \mid R \bullet P)[x := F] \quad = \quad (\star y \mid R[x := F] \bullet P[x := F])$$

LADM Chapter 8:
"$\star$ is a **metavariable** for operators $\_+\_$, $\_\cdot\_$, $\_\wedge\_$, $\_\vee\_$" (resp. $\sum$, $\prod$, $\forall$, $\exists$)

(8.11) is part of the Substitution keyword in CalcCheck.

Read LADM Chapter 8!

Substitution Examples

(8.11) Provided $\neg occurs('y', 'x, F')$,

$$(\star y \mid R \bullet P)[x := F] \quad = \quad (\star y \mid R[x := F] \bullet P[x := F])$$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[y := y + z]$
  $= \langle$ substitution $\rangle$
  $(\sum x \mid 1 \le x \le 2 \bullet y + z)$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[y := y + x]$
  $= \langle$ (8.21) Variable renaming $\rangle$
  $(\sum z \mid 1 \le z \le 2 \bullet y)[y := y + x]$
  $= \langle$ substitution $\rangle$
  $(\sum z \mid 1 \le z \le 2 \bullet y + x)$

Substitution Examples (ctd.)

(8.11) Provided $\neg occurs('y', 'x, F')$,

$$(\star y \mid R \bullet P)[x := F] \quad = \quad (\star y \mid R[x := F] \bullet P[x := F])$$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[x := y + x]$
  $= \langle$ (8.21) Variable renaming $\rangle$
  $(\sum z \mid 1 \le z \le 2 \bullet y)[x := y + x]$
  $= \langle$ (8.11) $\rangle$
  $(\sum z \mid (1 \le z \le 2)[x := y + x] \bullet (y)[x := y + x])$
  $= \langle$ Substitution $\rangle$
  $(\sum z \mid 1 \le z \le 2 \bullet y)$
  $= \langle$ (8.21) Variable renaming $\rangle$
  $(\sum x \mid 1 \le x \le 2 \bullet y)$

(8.11f) Provided $\neg occurs('x', 'E')$,

$$E[x := F] \quad = \quad E$$

Renaming of Bound Variables

(8.21) **Axiom, Dummy renaming** (α-conversion):

$(\star x \mid R \bullet P) \quad = \quad (\star y \mid R[x := y] \bullet P[x := y])$ provided $\neg occurs('y', 'R, P')$.

$(\sum i \mid 0 \le i < k \bullet n^i)$
$= \langle$ Dummy renaming (8.21), $\neg occurs('j', '0 \le i < k,\ n^i')$ $\rangle$
$(\sum j \mid 0 \le j < k \bullet n^j)$

$(\sum i \mid 0 \le i < k \bullet n^i)$
? $\langle$ Dummy renaming (8.21)$\rangle$ ×
$(\sum k \mid 0 \le k < k \bullet n^k)$ ...... k **captured!**

Generally, use <u>fresh</u> variables for renaming to avoid <u>variable capture</u>!

In CalcCheck, renaming of bound variables is part of "Reflexivity of =",
but can also be mentioned explicitly.

Leibniz Rules for Quantification

Try to use $x + x = 2 \cdot x$ and Leibniz (1.5) $\dfrac{X \quad = \quad Y}{E[z := X] \quad = \quad E[z := Y]}$ to obtain:

$$(\sum x \mid 0 \le x < 9 \bullet x + x) = (\sum x \mid 0 \le x < 9 \bullet 2 \cdot x)$$

- Choose E as: $(\sum x \mid 0 \le x < 9 \bullet z)$
- Perform substitution: $(\sum x \mid 0 \le x < 9 \bullet z)[z := x + x]$
  $(\sum y \mid 0 \le y < 9 \bullet x + x)$
- Not possible with (1.5)!
  — $E[z := X] = E[z := Y]$ **renames** x**!**

Special Leibniz rule for quantification:

$$\frac{P \quad = \quad Q}{(\star x \mid R \bullet E[z := P]) \quad = \quad (\star x \mid R \bullet E[z := Q])}$$

LADM Leibniz Rules for Quantification

Rewrite equalities in the **range** context of quantifications:

$$\textbf{(8.12) Leibniz} \quad \frac{P \quad = \quad Q}{(\star\, x \mid E[z := P] \bullet S) \quad = \quad (\star\, x \mid E[z := Q] \bullet S)}$$

Rewrite equalities in the **body** context of quantifications:

$$\textbf{(8.12) Leibniz} \quad \frac{R \quad \Rightarrow \quad (P \quad = \quad Q)}{(\star\, x \mid R \bullet E[z := P]) \quad = \quad (\star\, x \mid R \bullet E[z := Q])}$$

(These inference rules will also be used **implicitly**.)

Important: $P = Q$, repectively $R \Rightarrow (P = Q)$, needs to be a **theorem**!
These rules are **not** available for local **Assumption**s!
(Because x may occur in R, P, Q.)

The CalcCheck versions use **universally-quantified antecedents**.

Axiom "Leibniz for $\sum$ range": $(\forall\, x \bullet R_1 \equiv R_2) \Rightarrow (\sum\, x \mid R_1 \bullet E) = (\sum\, x \mid R_2 \bullet E)$
Axiom "Leibniz for $\sum$ body": $(\forall\, x \bullet R \Rightarrow E_1 = E_2) \Rightarrow (\sum\, x \mid R \bullet E_1) = (\sum\, x \mid R \bullet E_2)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-24

General Quantification (ctd.) — LADM Chapter 8

Predicate Logic — LADM Chapter 9

Bound / Free Variable Occurrences — The *occurs* Meta-Predicate

Renaming quantified variables does not change the meaning:

$$(\forall\, i \bullet x \cdot i = 0) \qquad \equiv \qquad (\forall\, j \bullet x \cdot j = 0)$$

- **Occurrences** of quantified variables inside the quantified expression are **bound**
- **Variable occurences** in an expression where they are not bound are **free**

$$i > 0 \lor (\forall\, i \mid 0 \le i \bullet x \cdot i = 0)$$

- The variable declarations after the quantification operator may be called **binding occurrences**.

Definition: *occurs*('v', 'e') means that at least one variable in the list v of variables occurs **free** in at least one expression in expression list e.

CalcCheck derives and checks $\neg$*occurs* **provisos automatically.**

Leibniz Rules for Quantification: LADM and CalcCheck

Rewrite equalities in the **range** context of quantifications:

(8.12) **Leibniz** $$\frac{P = Q}{(\star x \mid E[z := P] \bullet S) = (\star x \mid E[z := Q] \bullet S)}$$

Rewrite equalities in the **body** context of quantifications:

(8.12) **Leibniz** $$\frac{R \Rightarrow (P = Q)}{(\star x \mid R \bullet E[z := P]) = (\star x \mid R \bullet E[z := Q])}$$

(These inference rules will also be used **implicitly**.)

Important: $P = Q$, repectively $R \Rightarrow (P = Q)$, needs to be a **theorem**!
These rules are **not** available for local **Assumption**s!
(Because x may occur in R, P, Q.)

The CalcCheck versions use **universally-quantified antecedents**.

Axiom "Leibniz for $\sum$ range": $(\forall x \bullet R_1 \equiv R_2) \Rightarrow (\sum x \mid R_1 \bullet E) = (\sum x \mid R_2 \bullet E)$
Axiom "Leibniz for $\sum$ body": $(\forall x \bullet R \Rightarrow E_1 = E_2) \Rightarrow (\sum x \mid R \bullet E_1) = (\sum x \mid R \bullet E_2)$

Distributivity

(8.15) **Axiom,** (Quantification) **Distributivity:**

$$(\star x \mid R \bullet P) \star (\star x \mid R \bullet Q) = (\star x \mid R \bullet P \star Q),$$

provided each quantification is defined.

CalcCheck currently has no way to express or check this proviso —
— it remains in **your responsibility!**

$(\sum i : \mathbb{N} \mid i < n \bullet f\ i) + (\sum i : \mathbb{N} \mid i < n \bullet g\ i)$
$=$ ⟨ Quantification Distributivity (8.15) ⟩
$(\sum i : \mathbb{N} \mid i < n \bullet f\ i + g\ i)$

Note: Some quantifications are not defined, e.g.: $(\sum n : \mathbb{N} \bullet n)$

Note that quantifications over $\wedge$ or $\vee$ are always defined:

$$(\forall x \mid R \bullet P \wedge Q) = (\forall x \mid R \bullet P) \wedge (\forall x \mid R \bullet Q)$$
$$(\exists x \mid R \bullet P \vee Q) = (\exists x \mid R \bullet P) \vee (\exists x \mid R \bullet Q)$$

Distributivity

(8.15) **Axiom,** (Quantification) **Distributivity:**

$$(\star x \mid R \bullet P) \star (\star x \mid R \bullet Q) = (\star x \mid R \bullet P \star Q),$$

provided each quantification is defined.

Calculation:

$(1 + 1 \cdot 1) + (2 + 2 \cdot 2) + (3 + 3 \cdot 3)$
$=$ ⟨ Quantification expansion, substitution ⟩
$\sum i : \mathbb{N} \mid 1 \leq i < 4 \bullet (i + i \cdot i)$
$=$ ⟨ "**Distributivity** of $\sum$ over $+$" ⟩
$(\sum i : \mathbb{N} \mid 1 \leq i < 4 \bullet i) + (\sum i : \mathbb{N} \mid 1 \leq i < 4 \bullet i \cdot i)$
$=$ ⟨ Quantification expansion, substitution ⟩
$(1 + 2 + 3) + (1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3)$

Disjoint Range Split — LADM

(8.16) **Axiom, Range split:**

$$(\star x \mid R \vee S \bullet P) \quad = \quad (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each quantification is defined.

$$(\Sigma x \mid R \vee S \bullet P) \quad = \quad (\Sigma x \mid R \bullet P) + (\Sigma x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each sum is defined.

$$(\forall x \mid R \vee S \bullet P) \quad = \quad (\forall x \mid R \bullet P) \wedge (\forall x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$.

$$(\exists x \mid R \vee S \bullet P) \quad = \quad (\exists x \mid R \bullet P) \vee (\exists x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$.

Disjoint Range Split for ∑ (LADM and CalcCheck)

(8.16) **Axiom, Range Split:** $(\Sigma x \mid R \vee S \bullet P) = (\Sigma x \mid R \bullet P) + (\Sigma x \mid S \bullet P)$
provided $R \wedge S = \mathit{false}$ and each sum is defined.

CalcCheck currently cannot deal with "provided each sum is defined".
But once ∀ is available, $Q \wedge R = \mathit{false}$ does not need to be a proviso:

Theorem "Disjoint range split for ∑ ":
$(\forall x \bullet R \wedge S \equiv \mathsf{false}) \Rightarrow$
$((\sum x \mid R \vee S \bullet E) = (\sum x \mid R \bullet E) + (\sum x \mid S \bullet E))$

That is: Summing up over a large range can be done by adding the results of summing up two disjoint and complementary subranges.

$\Longrightarrow$ "**Divide and conquer**" algorithm design pattern

DIVIDE ET IMPERA
— Gaius Julius Caesar

Range Split "Axioms"

(8.16) **Axiom, Range split:**

$$(\star x \mid R \vee S \bullet P) \quad = \quad (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each quantification is defined.

(8.17) **Axiom, Range Split:**

$$(\star x \mid R \vee S \bullet P) \star (\star x \mid R \wedge S \bullet P) \quad = \quad (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided each quantification is defined.

(8.18) Axiom, **Range Split for idempotent** $\star$**:**

$$(\star x \mid R \vee S \bullet P) \quad = \quad (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided each quantification is defined.

$$(\forall x \mid R \vee S \bullet P) \quad = \quad (\forall x \mid R \bullet P) \wedge (\forall x \mid S \bullet P)$$

$$(\exists x \mid R \vee S \bullet P) \quad = \quad (\exists x \mid R \bullet P) \vee (\exists x \mid S \bullet P)$$

Variable Binding Rearrangements

(8.19) **Axiom, Interchange of dummies:**

$$(\star x \mid R \bullet (\star y \mid S \bullet P)) \quad = \quad (\star y \mid S \bullet (\star x \mid R \bullet P))$$

provided $\neg occurs(`y', `R')$ and $\neg occurs(`x', `S')$, and each quantification is defined.

Apparently not provable for general quantification from the quantification axioms in LADM:

(8.19.1) **Dummy list permutation:** $(\star x, y \mid R \bullet P) \quad = \quad (\star y, x \mid R \bullet P)$

(without side conditions restricting variable occurrences!)

(8.20) **Axiom, Nesting:** $(\star x, y \mid R \wedge S \bullet P) \quad = \quad (\star x \mid R \bullet (\star y \mid S \bullet P))$
provided $\neg occurs(`y', `R')$.

(8.21) **Axiom, Dummy renaming** (α-conversion):
$(\star x \mid R \bullet P) \quad = \quad (\star y \mid R[x := y] \bullet P[x := y])$ provided $\neg occurs(`y', `R, P')$.

***Substitution* (8.11) *prevents capture of* y *by binders in* R *or* P**

Formalise, and prove:

- The sum of the first n odd natural numbers is equal to n^2.

Formalise it in a way that makes it easy to prove!

One option:

```
Theorem “Odd-number sum”:
    (∑ i : ℕ ❙ i < n • suc i + i) = n · n
```

How do you prove this?

The sum of the first n odd natural numbers is equal to n^2

```
Theorem “Odd-number sum”:
    (∑ i : ℕ ❙ i < n • suc i + i) = n · n
Proof:
  By induction on `n : ℕ`:
    Base case:
        (∑ i : ℕ ❙ i < 0 • suc i + i)
      =⟨ ? ⟩



      =⟨ ? ⟩
        0 · 0
    Induction step:
        (∑ i : ℕ ❙ i < suc n • suc i + i)
      =⟨ ? ⟩






      =⟨ ? ⟩
        suc n · suc n
```

The sum of the first n odd natural numbers is equal to n^2

```
Theorem “Odd-number sum”:
    (∑ i : ℕ ❙ i < n • suc i + i) = n · n
Proof:
  By induction on `n : ℕ`:
    Base case:
        (∑ i : ℕ ❙ i < 0 • suc i + i)
      =⟨ “Nothing is less than zero” ⟩
        (∑ i : ℕ ❙ false • suc i + i)
      =⟨ “Empty range for ∑”⟩
        0
      =⟨ “Definition of · for 0” ⟩
        0 · 0
    Induction step:
        (∑ i : ℕ ❙ i < suc n • suc i + i)
      =⟨ “Split off term at top”, Substitution ⟩
        (∑ i : ℕ ❙ i < n • suc i + i) + (suc n + n)
      =⟨ Induction hypothesis ⟩
        suc n + n + n · n
      =⟨ “Definition of · for `suc`” ⟩
        suc n + n · suc n
      =⟨ “Definition of · for `suc`” ⟩
        suc n · suc n
```

Manipulating Ranges (General Quantfication Version)

(8.23) **Theorem Split off term**: For $n : \mathbb{N}$ and dummies $i : \mathbb{N}$,

$$(\star i \mid 0 \le i < n+1 \bullet P) \;=\; (\star i \mid 0 \le i < n \bullet P) \star P[i := n]$$

$$(\star i \mid 0 \le i < n+1 \bullet P) \;=\; P[i := 0] \star (\star i \mid 0 < i < n+1 \bullet P)$$

- Typical uses: Induction proofs, verification of loops
- Generalisation: $\mathbb{N} \longrightarrow \mathbb{Z}$, $\quad 0 \longrightarrow m : \mathbb{Z}$ (with $m \le n$)

The following work both with $m, n, i : \mathbb{N}$ and with $m, n, i : \mathbb{Z}$:

Theorem: Split off term from top:

$$m \le n \quad \Rightarrow \quad (\star i \mid m \le i < n+1 \bullet P) = (\star i \mid m \le i < n \bullet P) \star P[i := n]$$

Theorem: Split off term from bottom:

$$m \le n \quad \Rightarrow \quad (\star i \mid m \le i < n+1 \bullet P) = P[i := m] \star (\star i \mid m+1 \le i < n+1 \bullet P)$$

Manipulating Ranges (Sum Version)

(8.23) **Theorem Split off term**: For $n : \mathbb{N}$ and dummies $i : \mathbb{N}$,

$$(\textstyle\sum i \mid 0 \le i < n+1 \bullet P) \;=\; (\sum i \mid 0 \le i < n \bullet P) + P[i := n]$$

$$(\textstyle\sum i \mid 0 \le i < n+1 \bullet P) \;=\; P[i := 0] + (\sum i \mid 0 < i < n+1 \bullet P)$$

- Typical uses: Induction proofs, verification of loops
- Generalisation: $\mathbb{N} \longrightarrow \mathbb{Z}$, $\quad 0 \longrightarrow m : \mathbb{Z}$ (with $m \le n$)

The following work both with $m, n, i : \mathbb{N}$ and with $m, n, i : \mathbb{Z}$:

Theorem: Split off term from top:

$$m \le n \quad \Rightarrow \quad (\textstyle\sum i \mid m \le i < n+1 \bullet P) = (\sum i \mid m \le i < n \bullet P) + P[i := n]$$

Theorem: Split off term from bottom:

$$m \le n \quad \Rightarrow \quad (\textstyle\sum i \mid m \le i < n+1 \bullet P) = P[i := m] + (\sum i \mid m+1 \le i < n+1 \bullet P)$$

Proving Split-off Term

We have:

(8.16) **Axiom, Range Split:**

$$(\Sigma x \mid R \vee S \bullet P) = (\Sigma x \mid R \bullet P) + (\Sigma x \mid S \bullet P)$$

provided $R \wedge S = false$ and each sum is defined.

How can you prove theorems like the following?

Theorem "Split off $\sum$-term from top of _<-suc range":

$$(\sum i : \mathbb{N} \mid i < \mathsf{suc}\, n \bullet E) = (\sum i : \mathbb{N} \mid i < n \bullet E) + E[i := n]$$

- Use range split first —
  $\Longrightarrow$ need to transform the LHS range expression $i < \mathsf{suc}\, n$ into an appropriate disjunction
  $\Longrightarrow$ the first disjunct should be the range expression $i < n$ from the RHS
- The second range will have one element
  $\Longrightarrow$ The second sum from the (8.16) RHS has range $i = n$
  $\Longrightarrow$ That second sum disappears via the **one-point rule**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-26

Part 2: Predicate Logic 1

Generalising De Morgan to Quantification

$\neg(\exists i \mid 0 \leq i < 4 \bullet P)$

= ⟨ Expand quantification ⟩

$\neg(P[i := 0] \vee P[i := 1] \vee P[i := 2] \vee P[i := 3])$

= ⟨ (3.47) De Morgan ⟩

$\neg P[i := 0] \wedge \neg P[i := 1] \wedge \neg P[i := 2] \wedge \neg P[i := 3]$

= ⟨ Contract quantification ⟩

$(\forall i \mid 0 \leq i < 4 \bullet \neg P)$

(9.18b,c,a) **Generalised De Morgan**:

$$\neg(\exists x \mid R \bullet P) \equiv (\forall x \mid R \bullet \neg P)$$
$$(\exists x \mid R \bullet \neg P) \equiv \neg(\forall x \mid R \bullet P)$$
$$\neg(\exists x \mid R \bullet \neg P) \equiv (\forall x \mid R \bullet P)$$

(9.17) **Axiom, Generalised De Morgan**:

$$(\exists x \mid R \bullet P) \equiv \neg(\forall x \mid R \bullet \neg P)$$

"Trading" Range Predicates with Body Predicates in ∀ and ∃

(9.2) Axiom, Trading: $(\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet R \Rightarrow P)$

Trading Theorems for ∀:

(9.3a) $(\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet \neg R \vee P)$

(9.3b) $(\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet R \wedge P \equiv R)$

(9.3c) $(\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet R \vee P \equiv P)$

(9.4a) $(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet R \Rightarrow P)$

(9.4b) $(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet \neg R \vee P)$

(9.4c) $(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet R \wedge P \equiv R)$

(9.4d) $(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet R \vee P \equiv P)$

(9.17) **Axiom, Generalised De Morgan**: $(\exists x \mid R \bullet P) \quad \equiv \quad \neg(\forall x \mid R \bullet \neg P)$

(9.19) Trading for ∃: $(\exists x \mid R \bullet P) \quad \equiv \quad (\exists x \bullet R \wedge P)$

(9.20) **Trading for** ∃: $(\exists x \mid Q \wedge R \bullet P) \equiv (\exists x \mid Q \bullet R \wedge P)$

Instantiation for ∀

$P[x := E]$

$\equiv$ ⟨ (8.14) One-point rule ⟩

$(\forall x \mid x = E \bullet P)$

$\Leftarrow$ ⟨ (9.10) Range weakening for ∀ ⟩

$(\forall x \mid true \vee x = E \bullet P)$

$\equiv$ ⟨ (3.29) Zero of ∨ ⟩

$(\forall x \mid true \bullet P)$

$\equiv$ ⟨ *true* range in quantification ⟩

$(\forall x \bullet P)$

$$\frac{\forall x \bullet P}{P[x := E]} \ \forall\text{-Elim}$$

This proves: (9.13) **Instantiation:** $(\forall x \bullet P) \quad \Rightarrow \quad P[x := E]$

The one-point rule is **"sharper"** than Instantiation.

Using sharper rules often means fewer dead ends...

A sharp version obtained via (3.60):

$$(\forall x \bullet P) \quad \equiv \quad (\forall x \bullet P) \wedge P[x := E]$$

Using Instantiation for ∀

(9.13) **Instantiation:** $(\forall x \bullet P) \quad \Rightarrow \quad P[x := E]$

A sharp version of Instantiation obtained via (3.60): $(\forall x \bullet P) \equiv (\forall x \bullet P) \wedge P[x := E]$

Proving $(\forall x \bullet x + 1 > x) \quad \Rightarrow \quad y + 2 > y$**:**

$(\forall x \bullet x + 1 > x)$

$=$ ⟨ **Instantiation (9.13) with (3.60)** ⟩

$(\forall x \bullet x + 1 > x) \quad \wedge \quad y + 1 > y$

$\Rightarrow$ ⟨ **Left-monotonicity of ∧ (4.3) with Instantiation (9.13)** ⟩

$(y + 1) + 1 > y + 1 \quad \wedge \quad y + 1 > y$

$\Rightarrow$ ⟨ Transitivity of > (15.41) ⟩

$y + 1 + 1 > y$

$=$ ⟨ $1 + 1 = 2$ ⟩

$y + 2 > y$

Recall: with$_2$

$$\neg (a \cdot b = a \cdot 0)$$
$$\equiv \langle \text{"Cancellation of } \cdot \text{" with assumption `} a \neq 0 \text{`} \rangle$$
$$\neg (b = 0)$$

In a hint of shape "*HintItem1* `with` *HintItem2* `and` *HintItem3*":

- If *HintItem1* refers to a theorem of shape $p \Rightarrow q$,
- then *HintItem2* and *HintItem3* are used to prove p
- and q is used in the surrounding proof.

Here:

- *HintItem1* is "Cancellation of ·": $z \neq 0 \Rightarrow (z \cdot x = z \cdot y \quad \equiv \quad x = y)$
- *HintItem2* is "Assumption $a \neq 0$"
- The surrounding proof uses: $a \cdot b = a \cdot 0 \quad \equiv \quad b = 0$

Monotonicity with ...

$$(\forall x \bullet x + 1 > x) \quad \wedge \quad y + 1 > y$$
$$\Rightarrow \langle \text{ Left-monotonicity of } \wedge \text{ (4.3) with Instantiation (9.13) } \rangle$$
$$(y + 1) + 1 > y + 1 \quad \wedge \quad y + 1 > y$$

In a hint of shape "*HintItem1* `with` *HintItem2* `and` *HintItem3*":

- If *HintItem1* refers to a theorem of shape $p \Rightarrow q$,
- then *HintItem2* and *HintItem3* are used to prove p
- and q is used in the surrounding proof.

Here:

- *HintItem1* is "Left-monotonicity of ∧": $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
- *HintItem2* is "Instantiation":

$$(\forall x \bullet x + 1 > x)$$
$$\Rightarrow \quad (y + 1) + 1 > y + 1$$

- The surrounding proof uses:

$$(\forall x \bullet x + 1 > x) \quad \wedge \quad y + 1 > y$$
$$\Rightarrow \quad (y + 1) + 1 > y + 1 \quad \wedge \quad y + 1 > y$$

with$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$:
  Rewrite *ThmA* with $L \mapsto R$
- E.g.: Assumption `$p \Rightarrow q$` with (3.60) `$p \Rightarrow q \equiv p \wedge q \equiv q$`

  The local theorem $p \Rightarrow q$ (resulting from the Assumption)

  rewrites via: $p \Rightarrow q \mapsto p \equiv p \wedge q$ (from (3.60))

  to: $p \equiv p \wedge q$

  which can be used for the rewrite: $p \mapsto p \wedge q$

Theorem (4.3) "Left-monotonicity of ∧": $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

Proof:

Assuming `$p \Rightarrow q$`:

$$p \wedge r$$
$$\equiv \langle \text{ Assumption `} p \Rightarrow q \text{` with "Definition of } \Rightarrow \text{ from } \wedge \text{"} \rangle$$
$$p \wedge q \wedge r$$
$$\Rightarrow \langle \text{ "Weakening" } \rangle$$
$$q \wedge r$$

Using Instantiation for ∀

(9.13) **Instantiation:** $(\forall\, x \bullet P) \quad \Rightarrow \quad P[x := E]$

A sharp version of Instantiation obtained via (3.60): $(\forall\, x \bullet P) \equiv (\forall\, x \bullet P) \wedge P[x := E]$

Theorem: $(\forall\, x : \mathbb{Z} \bullet x < x + 1) \Rightarrow y < y + 2$

Proof:

$(\forall\, x : \mathbb{Z} \bullet x < x + 1)$

$\equiv$ ⟨ "Instantiation" (9.13) with "Definition of ⇒ via ∧" (3.60) — explicit substitution needed! ⟩

$(\forall\, x : \mathbb{Z} \bullet x < x + 1) \wedge (x < x + 1)[x := y + 1]$

$\equiv$ ⟨ Substitution, Fact `1 + 1 = 2` ⟩

$(\forall\, x : \mathbb{Z} \bullet x < x + 1) \wedge y + 1 < y + 2$

$\Rightarrow$ ⟨ "Monotonicity of ∧" with "Instantiation" ⟩

$(x < x + 1)[x := y] \wedge y + 1 < y + 2$

$\equiv$ ⟨ Substitution ⟩

$y < y + 1 \wedge y + 1 < y + 2$

$\Rightarrow$ ⟨ "Transitivity of <" ⟩

$y < y + 2$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-09-27

Predicate Logic — LADM Chapter 9 (ctd.)

Warm-Up

- What does "assuming the antecedent" mean?
- Give the rule for quantification nesting.
- State the one-point rule and the empty range axiom.
- State the quantification distributivity axiom.
- Give the rule for disjoint range split.
- Give the rule for substitution into quantification.
- State the basic trading laws for ∀ and ∃.
- State the theorem of instantiation for ∀.

Using Instantiation for $\forall$

(9.13) **Instantiation:** $(\forall x \bullet P) \Rightarrow P[x := E]$

A sharp version of Instantiation obtained via (3.60): $(\forall x \bullet P) \equiv (\forall x \bullet P) \wedge P[x := E]$

Proving $(\forall x \bullet x+1 > x) \Rightarrow y+2 > y$**:**

$$
\begin{array}{ll}
& (\forall x \bullet x+1 > x) \\
\equiv & \langle \text{“Instantiation” (9.13) with “Implication via } \wedge\text{” (3.60=)} \rangle \\
& (\forall x \bullet x+1 > x) \quad \wedge \quad y+1 > y \\
\Rightarrow & \langle \text{“Left-monotonicity of } \wedge\text{” (4.3) with “Instantiation” (9.13)} \rangle \\
& (y+1)+1 > y+1 \quad \wedge \quad y+1 > y \\
\Rightarrow & \langle \text{“Transitivity of } >\text{” (15.41)} \rangle \\
& y+1+1 > y \\
\equiv & \langle 1+1 = 2 \rangle \\
& y+2 > y
\end{array}
$$

Recall: `with`$_2$

$$
\begin{array}{ll}
& \neg (a \cdot b = a \cdot 0) \\
\equiv & \langle \text{“Cancellation of } \cdot\text{” with assumption `} a \neq 0\text{`} \rangle \\
& \neg (b = 0)
\end{array}
$$

In a hint of shape “*HintItem1* `with` *HintItem2* `and` *HintItem3*”:

- If *HintItem1* refers to a theorem of shape $p \Rightarrow q$,
- then *HintItem2* and *HintItem3* are used to prove p
- and q is used in the surrounding proof.

Here:

- *HintItem1* is “Cancellation of $\cdot$”: $z \neq 0 \Rightarrow (z \cdot x = z \cdot y \equiv x = y)$
- *HintItem2* is “Assumption $a \neq 0$”
- The surrounding proof uses: $a \cdot b = a \cdot 0 \equiv b = 0$

Recall: `with`$_2$ in: Monotonicity `with` ...

$$
\begin{array}{ll}
& (\forall x \bullet x+1 > x) \quad \wedge \quad y+1 > y \\
\Rightarrow & \langle \text{“Left-monotonicity of } \wedge\text{” (4.3) with “Instantiation” (9.13)} \rangle \\
& (y+1)+1 > y+1 \quad \wedge \quad y+1 > y
\end{array}
$$

In a hint of shape “*HintItem1* `with` *HintItem2* `and` *HintItem3*”:

- If *HintItem1* refers to a theorem of shape $p \Rightarrow q$,
- then *HintItem2* and *HintItem3* are used to prove p
- and q is used in the surrounding proof.

Here:

- *HintItem1* is “Left-monotonicity of $\wedge$”: $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
- *HintItem2* is “Instantiation”: $(\forall x \bullet x+1 > x) \Rightarrow (y+1)+1 > y+1$
- The surrounding proof uses: $(\forall x \bullet x+1 > x) \wedge y+1 > y \Rightarrow (y+1)+1 > y+1 \wedge y+1 > y$

Modus Pones via with$_2$

Modus ponens theorem: (3.77) **Modus ponens:** $p \wedge (p \Rightarrow q) \Rightarrow q$

Modus ponens inference rule: ("Implication elimination" rule)

$$\frac{P \Rightarrow Q \qquad P}{Q} \Rightarrow\text{-Elim} \qquad \frac{f : A \to B \qquad x : A}{(f\ x) : B} \text{ Fct. app.}$$

Applying implication theorems:

A proof for $P \Rightarrow Q$ can be used as a recipe for turning a proof for P into a proof for Q.

Q_1
$\sqsubseteq$ ⟨ "Theorem 1" `$P \Rightarrow (Q_1 \sqsubseteq Q_2)$` with "Theorem 2" `$P$` ⟩
Q_2

Theorem "Left-monotonicity of ∧": $(p \Rightarrow q) \quad \Rightarrow \quad ((p \wedge r) \Rightarrow (q \wedge r))$

$(\forall x \bullet x + 1 > x) \quad \wedge \quad y + 1 > y$
$\Rightarrow$ ⟨ **"Left-monotonicity of ∧" (4.3) with "Instantiation" (9.13)** ⟩
$(y + 1) + 1 > y + 1 \quad \wedge \quad y + 1 > y$

with$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$: Rewrite *ThmA* with $L \mapsto R$
- E.g.: "Instantiation" (9.13) with "Implication via ∧" `$(p \Rightarrow q) \;=\; (p \wedge q \equiv q)$`

The theorem $(\forall x \bullet P) \;\Rightarrow\; P[x := E]$ "Instantiation" (9.13)
rewrites via the rule $p \Rightarrow q \;\mapsto\; p \equiv p \wedge q$ (from "Implication via ∧" (3.60=))
to $(\forall x \bullet P) \;\equiv\; (\forall x \bullet P) \wedge P[x := E]$,
which instantiated with $x + 1 > x$ **for** P **and** y **for** E **to:**
$$(\forall x \bullet x + 1 > x) \;\equiv\; (\forall x \bullet x + 1 > x) \wedge (x + 1 > x)[x := y]$$

In LADM, this substitution can be implicitly applied:

$(\forall x \bullet x + 1 > x)$
$\equiv$ ⟨ **"Instantiation" (9.13) with "Implication via ∧" (3.60=)** ⟩
$(\forall x \bullet x + 1 > x) \quad \wedge \quad y + 1 > y$

(CALCCHECK need it explicit — see the next slide.)

with$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$: Rewrite *ThmA* with $L \mapsto R$
- E.g.: "Instantiation" (9.13) with "Implication via ∧" `$(p \Rightarrow q) \;=\; (p \wedge q \equiv q)$`

The theorem $(\forall x \bullet P) \;\Rightarrow\; P[x := E]$ "Instantiation" (9.13)
rewrites via the rule $p \Rightarrow q \;\mapsto\; p \equiv p \wedge q$ (from "Implication via ∧" (3.60=))
to $(\forall x \bullet P) \;\equiv\; (\forall x \bullet P) \wedge P[x := E]$,
which can be used right-to-left† **as rewrite rule** $(\forall x \bullet P) \wedge P[x := E] \;\mapsto\; (\forall x \bullet P)$
and instantiated with $x + 1 > x$ **for** P **and** y **for** E **to:**
$$(\forall x \bullet x + 1 > x) \wedge (x + 1 > x)[x := y] \;\mapsto\; (\forall x \bullet x + 1 > x)$$

$(\forall x : \mathbb{Z} \bullet x < x + 1)$
$\equiv$ ⟨ "Instantiation" (9.13) with "Implication via ∧" (3.60 =) — explicit substitution needed! ⟩
$(\forall x : \mathbb{Z} \bullet x < x + 1) \;\wedge\; (x < x + 1)[x := y + 1]$

† Trying this left-to-right would not gain an instantiation for E from the matching of $(\forall x \bullet P)$ against $(\forall x \bullet x + 1 > x)$.

Using Instantiation for ∀

(9.13) **Instantiation:** $(\forall x \bullet P) \Rightarrow P[x := E]$

A sharp version of Instantiation obtained via (3.60): $(\forall x \bullet P) \equiv (\forall x \bullet P) \land P[x := E]$

Theorem: $(\forall x : \mathbb{Z} \bullet x < x + 1) \Rightarrow y < y + 2$
Proof:

$(\forall x : \mathbb{Z} \bullet x < x + 1)$
≡ ⟨ "Instantiation" (9.13) with "Definition of ⇒ via ∧" (3.60) — explicit substitution needed! ⟩
$(\forall x : \mathbb{Z} \bullet x < x + 1) \land (x < x + 1)[x := y + 1]$
≡ ⟨ Substitution, Fact `1 + 1 = 2` ⟩
$(\forall x : \mathbb{Z} \bullet x < x + 1) \land y + 1 < y + 2$
⇒⟨ "Monotonicity of ∧" with "Instantiation" ⟩
$(x < x + 1)[x := y] \land y + 1 < y + 2$
≡ ⟨ Substitution ⟩
$y < y + 1 \land y + 1 < y + 2$
⇒⟨ "Transitivity of <" ⟩
$y < y + 2$

Theorems and Universal Quantification

(9.16) **Metatheorem**: P is a theorem iff $(\forall x \bullet P)$ is a theorem.

This is another justification for **implicit use of "Instantiation" (9.13)**
$(\forall x \bullet P) \Rightarrow P[x := E]$:

Theorem: $(\forall x : \mathbb{Z} \bullet x < x + 1) \Rightarrow y < y + 2$
Proof:

Assuming (1) `∀ x : ℤ • x < x + 1`:

y
< ⟨ Assumption (1) — implicit instantiation with `y` for `E` ⟩
$y + 1$
< ⟨ Assumption (1) — implicit instantiation with `y + 1` for `E` ⟩
$y + 1 + 1$
= ⟨ Fact `1 + 1 = 2` ⟩
$y + 2$

Implicit Universal Quantification in Theorems 1

(9.16) **Metatheorem**: P is a theorem iff $(\forall x \bullet P)$ is a theorem.

(If proving "$x + 1 > x$" is considered to *really mean* proving "$\forall x \bullet x + 1 > x$", then the x in "$x + 1 > x$" is called *implicitly universally quantified*.)

Proof method: To prove $(\forall x \bullet P)$,
we prove P for arbitrary x.

That is really a prose version of the following **inference rule:**

$$\frac{P}{\forall x \bullet P} \quad \forall\text{-Intro} \quad (\text{prov. } x \text{ not free in assumptions})$$

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \bullet P)$:

For any '$v : \mathbb{N}$': *Proof for P*

(Non-local assumptions with free v are not usable.)

Using “For any” for “Proof by Generalisation”

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \bullet P)$:

For any ‘$v : \mathbb{N}$’:   *Proof for P*

Proving $\forall x : \mathbb{N} \bullet x < x + 1$**:**

For any \`$x : \mathbb{N}$\`:
$$
\begin{array}{ll}
 & x < x + 1 \\
\equiv & \langle \text{Identity of } + \rangle \\
 & x + 0 < x + 1 \\
\equiv & \langle \text{Cancellation of } + \rangle \\
 & 0 < 1 \\
\equiv & \langle \text{Fact `} 1 = \mathsf{suc}\, 0 \text{`} \rangle \\
 & 0 < \mathsf{suc}\, 0 \\
\equiv & \langle \text{Zero is less than successor} \rangle \\
 & \mathit{true}
\end{array}
$$

Implicit Universal Quantification in Theorems 2

(9.16) **Metatheorem**: P is a theorem iff $(\forall x \bullet P)$ is a theorem.

LADM Proof method: To prove $(\forall x \mid R \bullet P)$,
we prove P for arbitrary x in range R.

That is:

- Assume R to prove P (and assume nothing else that mentions x)
- This proves $R \Rightarrow P$
- Then, by (9.16), $(\forall x \bullet R \Rightarrow P)$ is a theorem.
- With (9.2) Trading for $\forall$, this is transformed into $(\forall x \mid R \bullet P)$.

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \bullet P)$:

For any ‘$v : \mathbb{N}$’:   *Proof for P*

- Proving $(\forall v : \mathbb{N} \mid R \bullet P)$:

For any ‘$v : \mathbb{N}$’ **satisfying** ‘R’:   *Proof for P using* **Assumption** ‘R’

Using “For any … satisfying” for “Proof by Generalisation”

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \mid R \bullet P)$:

For any ‘$v : \mathbb{N}$’ **satisfying** ‘R’:   *Proof for P using* **Assumption** ‘R’

Proving $\forall x : \mathbb{N} \mid x < 2 \bullet x < 3$**:**

For any \`$x : \mathbb{N}$\` satisfying \`$x < 2$\`:
$$
\begin{array}{ll}
 & x \\
< & \langle \text{Assumption `} x < 2 \text{`} \rangle \\
 & 2 \\
< & \langle \text{Fact `} 2 < 3 \text{`} \rangle \\
 & 3
\end{array}
$$

Combined Quantification Examples

- "There is a least integer."
- "There exists an integer b such that every integer n is at least b".
- "There exists an integer b such that for every integer n, we have $b \leq n$".

$(\exists\, b : \mathbb{Z} \bullet (\forall\, n : \mathbb{Z} \bullet b \leq n))$

- "π can be enclosed within rational bounds that are less than any ε apart"
- "For every positive real number ε, there are rational numbers r and s with $r < s < r + \varepsilon$, such that $r < \pi < s$"

$$(\forall\, \varepsilon : \mathbb{R} \mid 0 < \varepsilon \\ \bullet (\exists\, r, s : \mathbb{Q} \mid r < s < r + \varepsilon \bullet r < \pi < s))$$

- "$f : \mathbb{R} \to \mathbb{R}$ is continuous" — **Exercise!**

∃-Introduction

Recall: (9.13) **Instantiation:** $(\forall\, x \bullet P) \Rightarrow P[x := E]$

Dual: (9.28) **∃-Introduction:** $P[x := E] \Rightarrow (\exists\, x \bullet P)$

An expression E with $P[x := E]$ is called a "**witness**" of $(\exists\, x \bullet P)$.

Proving an existential quantification via ∃-Introduction requires "**exhibiting a witness**".

Inference rule:

$$\frac{P[x := E]}{\exists\, x \bullet P}\ \exists\text{-Intro} \qquad\qquad \frac{\forall\, x \bullet P}{P[x := E]}\ \forall\text{-Elim}$$

Using ∃-Introduction for "Proof by Example"

(9.28) **∃-Introduction:** $P[x := E] \Rightarrow (\exists\, x \bullet P)$

An expression E with $P[x := E]$ is called a "**witness**" of $(\exists\, x \bullet P)$.

Proving an existential quantification via ∃-Introduction requires "**exhibiting a witness**".

$(\exists\, x : \mathbb{N} \bullet x \cdot x < x + x)$

$\Leftarrow$ ⟨ ∃-Introduction ⟩

$(x \cdot x < x + x)[x := 1]$

$\equiv$ ⟨ Substitution ⟩

$1 \cdot 1 < 1 + 1$

$\equiv$ ⟨ Evaluation ⟩

true

Using ∃-Introduction for "Proof by Counter-Example"

(9.28) **∃-Introduction:** $P[x := E] \quad \Rightarrow \quad (\exists\, x \bullet P)$

$$
\begin{array}{ll}
 & \neg(\forall\, x : \mathbb{N} \bullet x + x < x \cdot x) \\
\equiv & \langle \text{ Generalised De Morgan } \rangle \\
 & (\exists\, x : \mathbb{N} \bullet \neg(x + x < x \cdot x)) \\
\Leftarrow & \langle\ \exists\text{-Introduction } \rangle \\
 & (\neg(x + x < x \cdot x))[x := 2] \\
\equiv & \langle \text{ Substitution } \rangle \\
 & \neg(2 + 2 < 2 \cdot 2) \\
\equiv & \langle \text{ Fact `} 2 + 2 < 2 \cdot 2 \equiv \mathit{false}\text{` } \rangle \\
 & \neg\mathit{false} \\
\equiv & \langle \text{ Negation of } \mathit{false} \ \rangle \\
 & \mathit{true}
\end{array}
$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-01

Part 1: Assuming witness ..., Monotonicity of ∀ and ∃

Witnesses

(9.30v) **Metatheorem Witness**: If $\neg occurs(\text{‘}x\text{’}, \text{‘}Q\text{’})$, then:

$(\exists\, x \mid R \bullet P) \Rightarrow Q$ is a theorem iff $(R \wedge P) \Rightarrow Q$ is a theorem

---

Theorem "Witness": $(\exists\, x \mid R \bullet P) \Rightarrow Q \quad \equiv \quad (\forall\, x \bullet R \wedge P \Rightarrow Q)$ prov. $\neg occurs(\text{‘}x\text{’}, \text{‘}Q\text{’})$

Proof:

$$
\begin{array}{ll}
 & (\exists\, x \mid R \bullet P) \Rightarrow Q \\
= & \langle \text{ (9.19) Trading for } \exists \ \rangle \\
 & (\exists\, x \bullet R \wedge P) \Rightarrow Q \\
= & \langle \text{ (3.59) Material implication } p \Rightarrow q \equiv \neg p \vee q, \text{ (9.18b) Gen. De Morgan } \rangle \\
 & (\forall\, x \bullet \neg(R \wedge P)) \vee Q \\
= & \langle \text{ (9.5) Distributivity of } \vee \text{ over } \forall \text{ — } \neg occurs(\text{‘}x\text{’}, \text{‘}Q\text{’}) \ \rangle \\
 & (\forall\, x \bullet \neg(R \wedge P) \vee Q) \\
= & \langle \text{ (3.59) Material implication } p \Rightarrow q \equiv \neg p \vee q \ \rangle \\
 & (\forall\, x \bullet R \wedge P \Rightarrow Q)
\end{array}
$$

The last line is, by Metatheorem (9.16), a theorem iff $(R \wedge P) \Rightarrow Q$ is.

LADM Theory of Integers — Axioms and Some Theorems

(15.1) **Axiom, Associativity:** $(a + b) + c = a + (b + c)$
$(a \cdot b) \cdot c = a \cdot (b \cdot c)$

(15.2) **Axiom, Symmetry:** $a + b = b + a$
$a \cdot b = b \cdot a$

(15.3) **Axiom, Additive identity:** $0 + a = a$

(15.4) **Axiom, Multiplicative identity:** $1 \cdot a = a$

(15.5) **Axiom, Distributivity:** $a \cdot (b + c) = a \cdot b + a \cdot c$

(15.6) Axiom, Additive Inverse: $(\exists x \bullet x + a = 0)$

(15.7) **Axiom, Cancellation of $\cdot$:** $c \neq 0 \Rightarrow (c \cdot a = c \cdot b \equiv a = b)$

(15.8) **Cancellation of $+$:** $a + b = a + c \equiv b = c$

(15.10b) **Unique mult. identity:** $a \neq 0 \Rightarrow (a \cdot z = a \equiv z = 1)$

(15.12) **Unique additive inverse:** $x + a = 0 \wedge y + a = 0 \Rightarrow x = y$

"Witness":

$(\exists x \mid R \bullet P) \Rightarrow Q$
$\equiv (\forall x \bullet R \wedge P \Rightarrow Q)$
prov. $\neg occurs(`x', `Q$

(15.6) **Additive Inverse:**
$(\exists x \bullet x + a = 0)$

(15.8) **Cancellation of $+$:**
$a + b = a + c \equiv b = c$

```
Theorem (15.8) “Cancellation of +”: a + b = a + c  ≡  b = c
Proof:
  Using “Mutual implication”:
    Subproof for `b = c  ⇒  a + b = a + c`:
      Assuming `b = c`:
          a + b
        =⟨ Assumption `b = c` ⟩
          a + c
    Subproof for `a + b = a + c  ⇒  b = c`:
          a + b = a + c  ⇒  b = c
        ≡⟨ “Left-identity of ⇒”, “Additive inverse” with `a ≔ a` ⟩
          (∃ x : ℤ • x + a = 0)  ⇒  a + b = a + c  ⇒  b = c
        ≡⟨ “Witness”, “Trading for ∀” ⟩
          ∀ x : ℤ ❙ x + a = 0  •  a + b = a + c  ⇒  b = c
      Proof for this:
        For any `x : ℤ` satisfying `x + a = 0`:
          Assuming `a + b = a + c`:
              b
            =⟨ “Identity of +” ⟩
              0 + b
            =⟨ Assumption `x + a = 0` ⟩
              x + a + b
            =⟨ Assumption `a + b = a + c` ⟩
              x + a + c
            =⟨ Assumption `x + a = 0` ⟩
              0 + c
            =⟨ “Identity of +” ⟩
              c
```

(15.6) Additive Inverse
$(\exists x \bullet x + a = 0)$

$$\frac{(\exists x \bullet P) \qquad \begin{array}{c} \ulcorner P \urcorner \\ \vdots \\ R \end{array}}{R} \ \exists\text{-Elim}$$

(prov. x not free in R, assumptions)

```
Theorem (15.8) “Cancellation of +”: a + b = a + c  ≡  b = c
Proof:
  Using “Mutual implication”:
    Subproof for `b = c  ⇒  a + b = a + c`:
      Assuming `b = c`:
          a + b
        =⟨ Assumption `b = c` ⟩
          a + c
    Subproof for `a + b = a + c  ⇒  b = c`:
          a + b = a + c  ⇒  b = c
        ≡⟨ “Left-identity of ⇒”, “Additive inverse” ⟩
          (∃ x : ℤ • x + a = 0)  ⇒  a + b = a + c  ⇒  b = c
      Proof for this:
        Assuming witness `x : ℤ` satisfying `x + a = 0`:
          Assuming `a + b = a + c`:
              b
            =⟨ “Identity of +” ⟩
              0 + b
            =⟨ Assumption `x + a = 0` ⟩
              x + a + b
            =⟨ Assumption `a + b = a + c` ⟩
              x + a + c
            =⟨ Assumption `x + a = 0` ⟩
              0 + c
            =⟨ “Identity of +” ⟩
              c
```

New Proof Strutures: Assuming witness

Assuming witness \`$x\{ :\ type\}^?$\` **satisfying** \`$P$\` :

- introduces the bound variable 'x'
- makes P available as assumption to the contained proof.
- This proves $(\exists\, x : type \bullet P) \Rightarrow R$
  if the contained proof proves R,

Assuming witness \`$x\{:\ type\}^?$\` **satisfying** \`$P$\` **by** *hint* :

- introduces the bound variable 'x'
- makes P available as assumption to the contained proof.
- *hint* needs to prove $(\exists\, x : type \bullet P)$
- This then proves R
  if the contained proof proves R
  (with the additional assumnption P)
- This can be understood as providing $\exists$-elimination:
  It uses *hint* to discharge the antecedent $(\exists\, x : type \bullet P)$
  and then has inferred proof goal R.

$$\frac{(\exists x \bullet P) \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix}}{R}\ \exists\text{-Elim}$$

(prov. x not free in R, assumptions)

(15.6) Additive Inverse
$(\exists\, x \bullet x + a = 0)$

$$\frac{(\exists x \bullet P) \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix}}{R}\ \exists\text{-Elim}$$

(prov. x not free in R, assumptions)

```
Theorem (15.8) “Cancellation of +”: a + b = a + c  ≡  b = c
Proof:
  Using “Mutual implication”:
    Subproof for `b = c  ⇒  a + b = a + c`:
      Assuming `b = c`:
          a + b
        =⟨ Assumption `b = c` ⟩
          a + c
    Subproof for `a + b = a + c  ⇒  b = c`:
      Assuming witness `x : ℤ` satisfying `x + a = 0`
        by “Additive inverse”:
        Assuming `a + b = a + c`:
            b
          =⟨ “Identity of +” ⟩
            0 + b
          =⟨ Assumption `x + a = 0` ⟩
            x + a + b
          =⟨ Assumption `a + b = a + c` ⟩
            x + a + c
          =⟨ Assumption `x + a = 0` ⟩
            0 + c
          =⟨ “Identity of +” ⟩
            c
```

Recall: Monotonicity With Respect To $\Rightarrow$

Let $\_\leq\_$ be an order on T, and let $f : T \to T$ be a function on T. Then f is called

- **monotonic** iff $x \leq y \;\Rightarrow\; f\, x \leq f\, y$,
- **antitonic** iff $x \leq y \;\Rightarrow\; f\, y \leq f\, x$.

(4.2) Left-Monotonicity of $\vee$: $(p \Rightarrow q) \Rightarrow (p \vee r \Rightarrow q \vee r)$

(4.3) Left-Monotonicity of $\wedge$: $(p \Rightarrow q) \Rightarrow p \wedge r \Rightarrow q \wedge r$

Antitonicity of $\neg$: $(p \Rightarrow q) \Rightarrow \neg q \Rightarrow \neg p$

Left-Antitonicity of $\Rightarrow$: $(p \Rightarrow q) \Rightarrow (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$

Right-Monotonicity of $\Rightarrow$: $(p \Rightarrow q) \Rightarrow (r \Rightarrow p) \Rightarrow (r \Rightarrow q)$

Guarded Right-Monotonicity of $\Rightarrow$: $(r \Rightarrow (p \Rightarrow q)) \Rightarrow (r \Rightarrow p) \Rightarrow (r \Rightarrow q)$

Transitivity Laws are Monotonicity Laws

Notice: The following two “are” transitivity of $\Rightarrow$:

- **Left-Antitonicity of $\Rightarrow$:** $(p \Rightarrow q) \Rightarrow (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$
- **Right-Monotonicity of $\Rightarrow$:** $(p \Rightarrow q) \Rightarrow (r \Rightarrow p) \Rightarrow (r \Rightarrow q)$

This works also for other orders — with general monotonicity: Let

- $\_\leq_1\_$ be an order on T_1, and $\_\leq_2\_$ be an order on T_2,
- $f : T_1 \rightarrow T_2$ be a function from T_1 to T_2.

Then f is called

- **monotonic** iff $x \leq_1 y \;\Rightarrow\; f\ x \leq_2 f\ y$,
- **antitonic** iff $x \leq_1 y \;\Rightarrow\; f\ y \leq_2 f\ x$.

Transitivity of $\leq$ is antitonitcity of $(\_ \leq r) : \mathbb{Z} \rightarrow \mathbb{B}$:

- **Left-Antitonicity of $\leq$:** $(p \leq q) \Rightarrow (q \leq r) \Rightarrow (p \leq r)$
- **Right-Monotonicity of $\leq$:** $(p \leq q) \Rightarrow (r \leq p) \Rightarrow (r \leq q)$

Weakening/Strengthening for ∀ and ∃ — “Cheap Antitonicity/Monotonicity”

(9.10) **Range weakening/strengthening for ∀:** $(\forall x \mid Q \vee R \bullet P) \Rightarrow (\forall x \mid Q \bullet P)$

(9.11) **Body weakening/strengthening for ∀:** $(\forall x \mid R \bullet P \wedge Q) \Rightarrow (\forall x \mid R \bullet P)$

(9.25) **Range weakening/strengthening for ∃:** $(\exists x \mid R \bullet P) \Rightarrow (\exists x \mid Q \vee R \bullet P)$

(9.26) **Body weakening/strengthening for ∃:** $(\exists x \mid R \bullet P) \Rightarrow (\exists x \mid R \bullet P \vee Q)$

Recall:

(9.2) **Trading for ∀:** $(\forall x \mid R \bullet P) \equiv (\forall x \bullet R \Rightarrow P)$

(9.19) **Trading for ∃:** $(\exists x \mid R \bullet P) \equiv (\exists x \bullet R \wedge P)$

Monotonicity for ∀

(9.12) **Monotonicity of ∀:**

$$(\forall x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\forall x \mid R \bullet P_1) \Rightarrow (\forall x \mid R \bullet P_2))$$

Range-Antitonicity of ∀:

$$(\forall x \bullet R_2 \Rightarrow R_1) \Rightarrow ((\forall x \mid R_1 \bullet P) \Rightarrow (\forall x \mid R_2 \bullet P))$$

$(\forall x \bullet R_2 \Rightarrow R_1)$

$\Rightarrow$ ⟨ (9.12) with shunted (3.82a) Transitivity of $\Rightarrow$ ⟩

$(\forall x \bullet (R_1 \Rightarrow P) \Rightarrow (R_2 \Rightarrow P))$

$\Rightarrow$ ⟨ (9.12) Monotonicity of ∀ ⟩

$(\forall x \bullet R_1 \Rightarrow P) \Rightarrow (\forall x \bullet R_2 \Rightarrow P)$

$=$ ⟨ (9.2) Trading for ∀ ⟩

$(\forall x \mid R_1 \bullet P) \Rightarrow (\forall x \mid R_2 \bullet P)$

Monotonicity for ∃

(9.27) (Body) **Monotonicity of ∃:**

$$(\forall x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\exists x \mid R \bullet P_1) \Rightarrow (\exists x \mid R \bullet P_2))$$

Range-Monotonicity of ∃:

$$(\forall x \bullet R_1 \Rightarrow R_2) \Rightarrow ((\exists x \mid R_1 \bullet P) \Rightarrow (\exists x \mid R_2 \bullet P))$$

Predicate Logic Laws You Really Need To Know Already Now

(8.13) **Empty Range**: $(\forall x \mid \mathit{false} \bullet P) = \mathit{true}$

$(\exists x \mid \mathit{false} \bullet P) = \mathit{false}$

(8.14) **One-point Rule:** Provided $\neg occurs(`x', `E')$, $(\forall x \mid x = E \bullet P) \equiv P[x := E]$

$(\exists x \mid x = E \bullet P) \equiv P[x := E]$

(9.17) **Generalised De Morgan**: $(\exists x \mid R \bullet P) \equiv \neg(\forall x \mid R \bullet \neg P)$

(9.2) **Trading for ∀:** $(\forall x \mid R \bullet P) \equiv (\forall x \bullet R \Rightarrow P)$

(9.4a) **Trading for ∀**: $(\forall x \mid Q \wedge R \bullet P) \equiv (\forall x \mid Q \bullet R \Rightarrow P)$

(9.19) **Trading for ∃:** $(\exists x \mid R \bullet P) \equiv (\exists x \bullet R \wedge P)$

(9.20) **Trading for ∃**: $(\exists x \mid Q \wedge R \bullet P) \equiv (\exists x \mid Q \bullet R \wedge P)$

(9.13) **Instantiation:** $(\forall x \bullet P) \Rightarrow P[x := E]$

(9.28) **∃-Introduction**: $P[x := E] \Rightarrow (\exists x \bullet P)$

...and correctly handle substitution, Leibniz, renaming of bound variables, monotonicity/antitonicity, For any ...

Sentences: Predicate Logic Formulae without Free Variables

Definition: A sentence is a Boolean expression without free variables.

- Expressions without free variables are also called "closed":
  A sentence is a closed Boolean expression.
- Recall: The value of an expression (in a state) only depends on its free variables.
- Therefore: **The value of a closed expression does not depend on the state.**
- That is, a closed Boolean expression, or sentence,
  - either always evaluates to *true*
  - or always evaluates to *false*
- In other words: A closed Boolean expression, or sentence,
  - is either valid
  - or a contradiction
- Also: For a closed Boolean expression, or sentence, φ
  - either φ is valid
  - or $\neg\varphi$ is valid
- This means: For a closed Boolean expression, or sentence, φ,
  only one of φ and $\neg\varphi$ can have a proof!

Closed Boolean Expressions . . . — 2018 Midterm 2

Prove one of the following two theorem statements — **only one is valid.** (Should be easy in less than ten steps.)

```
Theorem "M2-3A-1-yes": (∃ x : ℤ • ∀ y : ℤ • (x - 2) · y + 1 = x - 1)

Theorem "M2-3A-1-no": ¬ (∃ x : ℤ • ∀ y : ℤ • (x - 2) · y + 1 = x - 1)
```

- For a closed Boolean expression, or sentence, φ,
  only one of φ and $\neg\varphi$ can have a proof!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-01

Part 2: More Command Correctness

Recall: Partial Correctness for Pre-Postcond. Specs in Dynamic Logic Notation

- Program correctness statement in LADM (and much current use):

$$\{\ P\ \}\ C\ \{\ Q\ \}$$

  This is called a "Hoare triple".

- **Partial Correctness Meaning:**

  If **command** C is started in a state in which the **precondition** P holds
  then it will terminate **only in states** in which the **postcondition** Q holds.

- **Dynamic logic** notation (used in CalcCheck):

$$P \Rightarrow[\ C\ ]\ Q$$

- **Assignment Axiom:**

  — Hoare triple: $\{\ Q[x := E]\ \}\ x := E\ \{\ Q\ \}$

  — **Dynamic logic** notation (used in CalcCheck): $Q[x := E] \Rightarrow[\ x := E\ ]\ Q$

Transitivity Rules for Calculational Command Correctness Reasoning

Primitive inference rule "Sequence":

$$\frac{\text{`}P \Rrightarrow[\, C_1 \,]\ Q\text{`}, \quad \text{`}Q \Rrightarrow[\, C_2 \,]\ R\text{`}}{\text{`}P \Rrightarrow[\, C_1 ; C_2 \,]\ R\text{`}}$$

Strengthening the precondition:

```
  `P₁ ⇒ P₂`,    `P₂ ⇒[ C ] Q`
⊢────────────────────────────
        `P₁ ⇒[ C ] Q`
```

Weakening the postcondition:

```
  `P ⇒[ C ] Q₁`,    `Q₁ ⇒ Q₂`
⊢────────────────────────────
        `P ⇒[ C ] Q₂`
```

- Activated as transitivity rules
- Therefore used implicitly in calculations, e.g., proving $P \Rrightarrow[\, C_1 ; C_2 \,]\ R$ below
- No need to refer to these rules explicitly

$$\begin{array}{ll} & P \\ \Rrightarrow[\, C_1 \,] & \langle \ldots \rangle \\ & Q \\ \Rightarrow & \langle \ldots \rangle \\ & Q' \\ \Rrightarrow[\, C_2 \,] & \langle \ldots \rangle \\ & R \end{array}$$

Conditional Commands

- Pascal:

```
if condition then
  statement₁
else
  statement₂
```

- Ada:

```
if condition then
  statement₁
else
  statement₂
end if;
```

- C/Java:

```
if (condition)
  statement₁
else
  statement₂
```

- Python:

```
if condition:
  statement₁
else:
  statement₂
```

- sh:

```
if condition
then
  statement₁
else
  statement₂
fi
```

Conditional Rule

Primitive inference rule "Conditional":

$$\frac{\text{`}B \wedge P \Rrightarrow[\, C_1 \,]\ Q\text{`}, \quad \text{`}\neg B \wedge P \Rrightarrow[\, C_2 \,]\ Q\text{`}}{\text{`}P \Rrightarrow[\, \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \,]\ Q\text{`}}$$

```
Fact "Simple COND":
   true ⇒[ if x = 1 then y := 42 else x := 1 fi ] x = 1
Proof:
    true
  ⇒[ if x = 1 then y := 42 else x := 1 fi ] ⟨ Subproof:
      Using "Conditional":
        Subproof for `(true ∧ x = 1) ⇒[ y := 42 ] x = 1`:
            ?
        Subproof for `(true ∧ ¬ (x = 1)) ⇒[ x := 1 ] x = 1`:
            ?
    ⟩
    x = 1
```

```
Fact “Simple COND”:
   true ⇒[ if x = 1 then y := 42 else x := 1 fi ] x = 1
Proof:
    true
  ⇒[ if x = 1 then y := 42 else x := 1 fi ] ⟨ Subproof:
      Using “Conditional”:
        Subproof for `(true ∧ x = 1) ⇒[ y := 42 ] x = 1`:
            true ∧ x = 1
          ≡⟨ “Identity of ∧” ⟩
            x = 1
          ≡⟨ Substitution ⟩
            (x = 1)[y ≔ 42]
          ⇒[ y := 42 ] ⟨ “Assignment” ⟩
            x = 1
        Subproof for `(true ∧ ¬ (x = 1)) ⇒[ x := 1 ] x = 1`:
            true ∧ ¬ (x = 1)
          ⇒⟨ “Right-zero of ⇒” ⟩
            true
          ≡⟨ “Reflexivity of =” ⟩
            1 = 1
          ≡⟨ Substitution ⟩
            (x = 1)[x ≔ 1]
          ⇒[ x := 1 ]  ⟨ “Assignment” ⟩
            x = 1
    ⟩
    x = 1
```

The “While” Rule

The constituents of a while loop “while B do C od” are:

- The **loop condition** $B : \mathbb{B}$
- The **(loop) body** C : Cmd

The conventional **while rule** allows to infer only correctness statements for while loops that are in the shape of the conclusion of this inference rule, involving an **invariant** condition $Q : \mathbb{B}$:

```
         `B ∧ Q   ⇒[ C ]   Q`
⊢─────────────────────────────────────────
   `Q  ⇒[ while B do C od ]  ¬ B ∧ Q`
```

This rule reads:

- If you can prove that execution of the loop body C starting in states satisfying the loop condition B **preserves** the invariant Q,
- then you have proof that the whole loop also preserves the invariant Q, and in addition establishes the negation of the loop condition.

The “While” Rule — Induction for Partial Correctness

```
         `B ∧ Q   ⇒[ C ]   Q`
⊢─────────────────────────────────────────
   `Q  ⇒[ while B do C od ]  ¬ B ∧ Q`
```

The invariant will need to hold

- immediately before the loop starts,
- after each execution of the loop body,
- and therefore also after the loop ends.

The invariant will typically mention all variables that are changed by the loop, and explain how they are related.

Frequent pattern: Generalised postcondition using the negated loop condition

In general, you have to identify an appropriate invariant yourself!

Well-written programs contain documentation of invariants for all loops.

Using the “While” Rule

Theorem “While-example”:

```
    Pre
  ⇒[ INIT ;
      while B
        do
          C
        od ;
      FINAL
    ]
    Post
```

Proof:

```
    Pre    ······Precondition
  ⇒[ INIT ] ⟨ ? ⟩
    Q     ······Invariant
  ⇒[ while B do
          C
       od ] ⟨ “While” with subproof:
            B ∧ Q    ······Loop condition and invariant
          ⇒[ C ] ⟨ ? ⟩
            Q       ······Invariant
    ⟩
    ¬ B ∧ Q  ······Negated loop condition, and invariant
  ⇒[ FINAL ] ⟨ ? ⟩
    Post    ······Postcondition
```

“Quantification is Somewhat Like Loops”

```
Theorem “Summing up”:
      true
    ⇒[  s := 0 ;
        i := 0 ;
        while i ≠ n
          do
            s := s + f i ;
            i := i + 1
          od
      ]
      s = ∑ j : ℕ ❙ j < n • f j
```

Invariant: $s = \sum j : \mathbb{N} \mid j < i \bullet f\, j$

— Generalised postcondition using the negated loop condition

(This is a frequent pattern.)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-03

While, Sequences

Plan for Today

- Correctness proofs for while-loops (ctd.)
- Sequences — a brief start (LADM chapter 13)

Coming up:

- Some remarks about Types (see also LADM section 8.1)
- “A Theory of Sets” (LADM chapter 11)
- Relations (see also LADM chapter 14)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-03

Part 1: Correctness of while-Loops

The “While” Rule — Induction for Partial Correctness

```
          `B ∧ Q  ⇒⁅ C ⁆  Q`
⊢ ─────────────────────────────────
   `Q  ⇒⁅ while B do C od ⁆  ¬ B ∧ Q`
```

The invariant will need to hold

- immediately before the loop starts,
- after each execution of the loop body,
- and therefore also after the loop ends.

The invariant will typically mention all variables that are changed by the loop, and explain how they are related.

Frequent pattern: Generalised postcondition using the negated loop condition

In general, you have to identify an appropriate invariant yourself!

Well-written programs contain documentation of invariants for all loops.

Using the “While” Rule

Theorem “While-example”:

```
        Pre
     ⇒[ INIT ;
          while B
            do
              C
            od ;
          FINAL
        ]
        Post
```

Proof:

```
    Pre  ▪▪▪▪▪▪Precondition
  ⇒[ INIT ] ⟨ ? ⟩
    Q    ▪▪▪▪▪▪Invariant
  ⇒[ while B do
          C
       od ]  ⟨ “While” with subproof:
           B ∧ Q    ▪▪▪▪▪▪Loop condition and invariant
         ⇒[ C ] ⟨ ? ⟩
           Q      ▪▪▪▪▪▪Invariant
    ⟩
    ¬ B ∧ Q  ▪▪▪▪▪▪Negated loop condition, and invariant
  ⇒[ FINAL ] ⟨ ? ⟩
    Post   ▪▪▪▪▪▪Postcondition
```

“Quantification is Somewhat Like Loops”

```
Theorem “Summing up”:
        true
     ⇒[   s := 0 ;
          i := 0 ;
          while i ≠ n
            do
              s := s + f i ;
              i := i + 1
            od
        ]
        s = ∑ j : ℕ ❙ j < n • f j
```

Invariant: $s = \sum j : \mathbb{N} \mid j < i \bullet f\, j$

— Generalised postcondition using the negated loop condition

(This is a frequent pattern.)

Using the “While” Rule — An Example

Theorem “Adding$_1$”:

```
        n = n₀
     ⇒[ i := 0 ;
          while i ≠ m
            do
              i := i + 1 ;
              n := n + 1
            od
        ]
        n = m + n₀
```

Proof:

```
      n = n₀             ▪▪▪▪▪▪Precondition
  ≡ ⟨ “Identity of +” ⟩
      n = 0 + n₀
  ⇒[ i := 0 ] ⟨ “Assignment” with substitution ⟩
      n = i + n₀          ▪▪▪▪▪▪Invariant
  ⇒[ while i ≠ m do
          i := i + 1 ;
          n := n + 1
       od ] ⟨ “While” with subproof:
            i ≠ m ∧ n = i + n₀   ▪▪▪▪▪▪Loop condition and invariant
         ⇒⟨ “Weakening” ⟩
            n = i + n₀
         ≡ ⟨ “Cancellation of +” ⟩
            n + 1 = i + 1 + n₀
         ⇒[ i := i + 1 ] ⟨ “Assignment” with substitution ⟩
            n + 1 = i + n₀
         ⇒[ n := n + 1 ] ⟨ “Assignment” with substitution ⟩
            n = i + n₀        ▪▪▪▪▪▪Invariant
      ⟩
      ¬ (i ≠ m) ∧ n = i + n₀  ▪▪▪▪▪▪Negated loop condition, and inv.
```

Using the "While" Rule — Another Example...

Theorem "Answering...":

```
      true
   ⇒[ i := 0 ;
        while i = 0
          do
            n := n + 1
          od
      ]
      n = 42
```

This program will terminate only in states satisfying $n = 42$.

Using the "While" Rule — Another Example...

Theorem "Answering...":

```
      true
   ⇒[ i := 0 ;
        while i = 0
          do
            n := n + 1
          od
      ]
      n = 42
```

Proof:

```
    true                  ▪▪▪▪▪▪Precondition
  ≡ ⟨ "Reflexivity of =" ⟩
    0 = 0
  ⇒[ i := 0 ] ⟨ "Assignment" with substitution ⟩
    i = 0                 ▪▪▪▪▪▪Invariant
  ⇒[ while i = 0 do
         n := n + 1
       od ] ⟨ "While" with subproof:
           i = 0 ∧ i = 0      ▪▪▪▪▪▪Loop condition and invariant
         ≡ ⟨ "Idempotency of ∧" ⟩
           i = 0
         ⇒[ n := n + 1 ] ⟨ "Assignment" with substitution ⟩
           i = 0          ▪▪▪▪▪▪Invariant
    ⟩
    ¬ (i = 0) ∧ i = 0     ▪▪▪▪▪▪Negated loop condition, and inv.
  ⇒⟨ ? ⟩
    n = 42                ▪▪▪▪▪▪Postcondition
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-03

Part 2: Sequences

Sequences

- We may write $[33, 22, 11]$ (Haskell notation) for the sequence that has
  - "33" as its first element,
  - "22" as its second element,
  - "11" as its third element, and
  - no further elements.

  (Notation "$[\ldots]$" for sequences is not supported by CalcCheck. LADM writes "$\langle \ldots \rangle$".)
- Sequence matters: $[33, 22, 11]$ and $[11, 22, 33]$ are different!
- Multiplicity matters: $[33, 22, 11]$ and $[33, 22, 22, 11]$ are different!
- We consider the type $\mathsf{Seq}\ A$ of sequences with elements of type A as generated inductively by the following two constructors:

ϵ	:	$\mathsf{Seq}\ A$	`\eps`	empty sequence
$\_\triangleleft\_$	:	$A \to \mathsf{Seq}\ A \to \mathsf{Seq}\ A$	`\cons`	"cons"

  $\triangleleft$ associates to the right.
- Therefore: $[33, 22, 11] = 33 \triangleleft [22, 11]$
  $= 33 \triangleleft 22 \triangleleft [11]$
  $= 33 \triangleleft 22 \triangleleft 11 \triangleleft \epsilon$

Sequences — "cons" and "snoc"

- We consider the type $\mathsf{Seq}\ A$ of sequences with elements of type A as generated inductively by the following two constructors:

ϵ	:	$\mathsf{Seq}\ A$	`\eps`	empty sequence
$\_\triangleleft\_$	:	$A \to \mathsf{Seq}\ A \to \mathsf{Seq}\ A$	`\cons`	"cons"

  $\triangleleft$ associates to the right.
- Therefore: $[33, 22, 11] = 33 \triangleleft [22, 11]$
  $= 33 \triangleleft 22 \triangleleft [11]$
  $= 33 \triangleleft 22 \triangleleft 11 \triangleleft \epsilon$
- Appending single elements "at the end":

  $\_\triangleright\_$: $\mathsf{Seq}\ A \to A \to \mathsf{Seq}\ A$ `\snoc` "snoc"

  $\triangleright$ associates to the left.
- (Con-)catenation:

  $\_\frown\_$: $\mathsf{Seq}\ A \to \mathsf{Seq}\ A \to \mathsf{Seq}\ A$ `\catenate`

  $\frown$ associates to the right.

Sequences — Induction Principle

- The set of all **sequences over type A** is written $\mathsf{Seq}\ A$.
- The empty sequence "ϵ" is a sequence over type A.
- If x is an element of A and xs is a sequence over type A, then "$x \triangleleft xs$" (pronounced: "x cons xs") is a sequence over type A, too.
- Two sequences are equal **iff** they are constructed the same way from ϵ and $\triangleleft$.

Induction principle for sequences:

- if $P(\epsilon)$ — If P holds for ϵ
- and if $P(xs)$ implies $P(x \triangleleft xs)$ for all $x : A$,

  and whenever P holds for xs, it also holds for any $x \triangleleft xs$,
- then for all $xs : \mathsf{Seq}\ A$ we have $P(xs)$.

  then P holds for all sequences over A.

Sequences — Induction Proofs

Induction principle for sequences:

- if $P(\epsilon)$ — If P holds for ϵ
- and if $P(xs)$ implies $P(x \triangleleft xs)$ **for all** $x : A$, — and whenever P holds for xs, it also holds for any $x \triangleleft xs$,
- then for all $xs : \mathsf{Seq}\ A$ we have $P(xs)$. — then P holds for all sequences over A.

An **induction proof** using this looks as follows:

Theorem: P

Proof:

By induction on $xs : \mathsf{Seq}\ A$**:**

Base case:

Proof for $P[xs := \epsilon]$

Induction step:

Proof for $(\forall x : A \bullet P[xs := x \triangleleft xs])$

using **Induction hypothesis** P

Concatenation

```
Axiom (13.17) “Left-identity of ⌢”
              “Definition of ⌢ for ϵ”:            ϵ ⌢ ys = ys
Axiom (13.18) “Mutual associativity of ◃ with ⌢”
              “Definition of ⌢ for ◃”:            (x ◃ xs) ⌢ ys = x ◃ (xs ⌢ ys)
```

$\Longrightarrow$ H9._, Ex6._

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-04

Types, Sets

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-04

Part 1: Types

Types

A **type denotes a set of values** that

- can be associated with a variable
- an expression might evaluate to

Some basic types: $\mathbb{B}, \mathbb{Z}, \mathbb{N}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$

Some constructed types: $\mathsf{Seq}\ \mathbb{N}$, $\mathbb{N} \to \mathbb{B}$, $\mathsf{Seq}\ (\mathsf{Seq}\ \mathbb{N}) \to \mathsf{Seq}\ \mathbb{B}$, $\mathbf{set}\ \mathbb{Z}$

"E : t" means: "Expression E is declared to have type t".

Examples:

- constants: $\mathit{true} : \mathbb{B}$, $\pi : \mathbb{R}$, $2 : \mathbb{Z}$, $2 : \mathbb{N}$
- variable declarations: $p : \mathbb{B}$, $k : \mathbb{N}$, $d : \mathbb{R}$
- type annotations in expressions:
  - $(x + y) \cdot x \quad \longrightarrow \quad (x : \mathbb{N} + y) \cdot x$
  - $(x + y) \cdot x \quad \longrightarrow \quad ((((x : \mathbb{N}) + (y : \mathbb{N})) : \mathbb{N}) \cdot (x : \mathbb{N})) : \mathbb{N}$

Function Types — LADM Version

- If the parameters of function f have types $t_1, \ldots, t_n$
- and the result has type r,
- then f has type $t_1 \times \cdots \times t_n \to r$

We write: $\boxed{f : t_1 \times \cdots \times t_n \to r}$

Examples: $\neg\_ : \mathbb{B} \to \mathbb{B}$ $\quad \_+\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ $\quad \_<\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$

Forming expressions using $\_<\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$:

- if expression a_1 has type $\mathbb{Z}$, and a_2 has type $\mathbb{Z}$
- then $a_1 < a_2$ is a (well-typed) expression
- and has type $\mathbb{B}$.

Mechanised Mathematics Version

- If the parameters of function f have types $t_1, \ldots, t_n$
- and the result has type r,
- then f has type $t_1 \to \cdots \to t_n \to r$

We write: $\boxed{f : t_1 \to \cdots \to t_n \to r}$

(The function type constructor "$\to$" **associates to the right!**)

Example: $\_<\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$

Forming expressions using $\_<\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$:

$$\frac{a_1 : \mathbb{Z} \qquad a_2 : \mathbb{Z}}{(a_1 < a_2) : \mathbb{B}}$$

In general: $$\frac{f : A \to B \qquad x : A}{f\ x : B}$$

Non-well-typed expressions make no sense!

Function Application — LADM Version

Consider function g defined by: (1.6) $g(z) = 3 \cdot z + 6$

- Special **function application** syntax for argument that is identifier or constant:

$$g.z = 3 \cdot z + 6$$

LADM Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- $.$ **(function application)**
- unary prefix operators $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot \quad / \quad \div \quad$ **mod** **gcd**
- $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \bullet$
- $\downarrow \quad \uparrow$
- $\#$
- $\triangleleft \quad \triangleright \quad \hat{}$
- $= \quad \neq \quad < \quad > \quad \in \quad \subset \quad \subseteq \quad \supset \quad \supseteq \quad |$ (conjunctional)
- $\vee \quad \wedge$
- $\Rightarrow \quad \not\Rightarrow \quad \Leftarrow \quad \not\Leftarrow$
- $\equiv \quad \not\equiv$ **(lowest precedence)**

All non-associative binary infix operators **associate to the left**, **except** $**$, $\triangleleft$, $\Rightarrow$, $\rightarrow$, which **associate to the right**.

Function Application — Mechanised Mathematics Version

Consider function g defined by: (1.6) $g\ z = 3 \cdot z + 6$

- **Function application** is denoted by **juxtaposition** ("putting side by side")
- **Lexical separation** for argument that is identifier or constant: **space required:**

  $$h\ z = g\ (g\ z)$$

  Superfluous parentheses (e.g., "$h(z) = g(g(z))$") are allowed, **ugly**, and bad style.
- Function application still has **higher precedence than other binary operators**.
- As non-associative binary infix operator, function application **associates to the left:** If $f : \mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z})$, then $f\ 2\ 3 = (f\ 2)\ 3$, and $f\ 2 : \mathbb{Z} \rightarrow \mathbb{Z}$
- Typing rule for function application:

$$\frac{f : A \rightarrow B \qquad x : A}{f\ x : B}$$

COMPSCI 2LC3 Fall 2024 CalcCheck Default Table of Precedences

- (∞): _[_ :=_] (textual substitution) **(highest precedence)**
- 140: unary postfix operators: _! _˘ _* _⁺ _(|_|)
- 130: unary prefix operators: +_ −_ ¬_ #_ ~_ ℙ_ suc_
- 120: __ (**function application**), @
- 115: **
- 110: · / ÷ mod gcd
- 105: ⨾ ╱ ╲
- 100: + − ∪ ∩ × ∘ ⊕ ⇨ ◁ ⩤ ▷ ⩥
- 97: ↔ (relation type)
- 95: → (function type)
- 90: ↓ ↑
- 70: #
- 60: ◃ ▹ ⌢
- 50: = ≠ < > ∈ ⊂ ⊆ ⊃ ⊇ | _(_)_ **(conjunctional)**
- 40: ∨ ∧
- 20: ⇒ ⇏ ⇐ ⇍
- 10: ≡ ≢
- 9: := (assignment command, two characters)
- 5: ; (command sequencing)
- (−∞): ⊛_|_•_ (quantification notation, for ⊛ ∈ {∀, ∃, ⋃, ⋂, ∑, ∏, . . .}) **(lowest precedence)**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-04

Part 2: Sets

LADM Chapter 11: A Theory of Sets

"A *set* is simply a collection of distinct (different) elements."

- 11.1 Set comprehension and membership
- 11.2 Operations on sets
- 11.3 Theorems concerning set operations (many! — mostly easy. . .)
- 11.4 Union and intersection of families of sets (quantification over ∪ and ∩)
- . . .

The Axioms of Set Theory — Overview

(11.2) Provided $\neg occurs(`x', `e_0, \ldots, e_{n-1}')$,

$$\{e_0, \ldots, e_{n-1}\} = \{x \mid x = e_0 \lor \cdots \lor x = e_{n-1} \bullet x\}$$

(11.3) **Axiom, Set membership:** Provided $\neg occurs(`x', `F')$,

$$F \in \{x \mid R \bullet E\} \quad \equiv \quad (\exists x \mid R \bullet E = F)$$

(11.2f) **Empty Set:** $v \in \{\} \equiv \mathit{false}$

(11.4) **Axiom, Extensionality:** Provided $\neg occurs(`x', `S, T')$,

$$S = T \quad \equiv \quad (\forall x \bullet x \in S \equiv x \in T)$$

(11.13T) **Axiom, Subset:** Provided $\neg occurs(`x', `S, T')$,

$$S \subseteq T \quad \equiv \quad (\forall x \bullet x \in S \Rightarrow x \in T)$$

(11.14) **Axiom, Proper subset:** $S \subset T \equiv S \subseteq T \land S \neq T$
(11.20) **Axiom, Union:** $v \in S \cup T \equiv v \in S \lor v \in T$
(11.21) **Axiom, Intersection:** $v \in S \cap T \equiv v \in S \land v \in T$
(11.22) **Axiom, Set difference:** $v \in S - T \equiv v \in S \land v \notin T$
(11.23) **Axiom, Power set:** $v \in \mathbb{P}\, S \equiv v \subseteq S$

Set Membership versus Type Annotation

Like in Haskell:
- **Sets** are datastructures (Data.Set.Set)
- **Types** aid program correctness

Therefore: **Types are not sets!**

Let T be a **type**; let S be a **set**, that is, an expression of type **set** T,
and let e be an expression ot type T, then

- $e \in S$ is an expression
- of type $\mathbb{B}$
- and denotes "e is **in** S"
  or "e is an **element of** S"

Because: $\_\in\_ : T \to \mathbf{set}\ T \to \mathbb{B}$

Note:
- $e : T$ is nothing but the expression e, with type annotation T.
- If e has type T, then $e : T$ has the same value as e.

Cardinality of Finite Sets

(11.12) **Axiom, Size:** Provided $\neg occurs(`x', `S')$,

$$\# S = (\Sigma x \mid x \in S \bullet 1)$$

This uses: $\#\_ : \mathbf{set}\ t \to \mathbb{N}$

Note:
- $(\Sigma x \mid x \in S \bullet 1)$ is defined if and only if S is finite.
- $\# \{n : \mathbb{N} \mid \mathit{true} \bullet n\}$ **is undefined!**
- "$\# \mathbb{N}$" **is a type error!** — because $\mathbb{N}$: Type
- Types are not sets — like in Haskell:

```
Integer :: *
Data.Set.Set Integer :: *
```

Set Comprehension

Set comprehension examples:

$$\{i : \mathbb{N} \mid i < 4 \bullet 2 \cdot i + 1\} = \{1, 3, 5, 7\}$$
$$\{x : \mathbb{Z} \mid 1 \le x < 5 \bullet x \cdot x\} = \{1, 4, 9, 16\}$$
$$\{i : \mathbb{Z} \mid 5 \le i < 8 \bullet i \triangleleft i \triangleleft \epsilon\} = \{(5 \triangleleft 5 \triangleleft \epsilon), (6 \triangleleft 6 \triangleleft \epsilon), (7 \triangleleft 7 \triangleleft \epsilon)\}$$

(11.1) **Set comprehension general shape:** $\{x : t \mid R \bullet E\}$

— This set comprehension **binds** variable x in R and E!

Evaluated in state s, this denotes the set containing the values of E evaluated in those states resulting from s by changing the binding of x to those values from type t that satisfy R.

Note: The braces "$\{\ldots\}$" are **only** used for set notation!

Abbreviation for special case: $\{x \mid R\} = \{x \mid R \bullet x\}$

(11.2) Provided $\neg occurs('x', 'e_0, \ldots, e_{n-1}')$,

$$\{e_0, \ldots, e_{n-1}\} = \{x \mid x = e_0 \lor \cdots \lor x = e_{n-1} \bullet x\}$$

Note: This is covered by "Reflexivity of =" in CalcCheck.

Set Membership

(11.3) **Axiom, Set membership:** Provided $\neg occurs('x', 'F')$,

$$F \in \{x \mid R \bullet E\} \equiv (\exists x \mid R \bullet E = F)$$

$F \in \{x \mid R\}$

$=$ ⟨ Expanding abbreviation ⟩

$F \in \{x \mid R \bullet x\}$

$=$ ⟨ (11.3) Axiom, Set membership — **provided** $\neg occurs('x', 'F')$ ⟩

$(\exists x \mid R \bullet x = F)$

$=$ ⟨ (9.19) Trading for $\exists$ ⟩

$(\exists x \mid x = F \bullet R)$

$=$ ⟨ (8.14) One-point rule — **provided** $\neg occurs('x', 'F')$ ⟩

$R[x := F]$

This proves: **Simple set compr. membership:** Prov. $\neg occurs('x', 'F')$,

$$F \in \{x \mid R\} \equiv R[x := F]$$

Set Equality and Inclusion

(11.4) **Axiom, Extensionality:** Provided $\neg occurs('x', 'S, T')$,

$$S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$$

(11.13T) **Axiom, Subset:** Provided $\neg occurs('x', 'S, T')$,

$$S \subseteq T \equiv (\forall x \bullet x \in S \Rightarrow x \in T)$$

(11.11b) **Metatheorem Extensionality:**
Let S and T be set expressions and v be a variable.
Then $S = T$ is a theorem iff $v \in S \equiv v \in T$ is a theorem. — Using "Set extensionality"

(11.13m) **Metatheorem Subset:** — Using "Set inclusion"
Let S and T be set expressions and v be a variable.
Then $S \subseteq T$ is a theorem iff $v \in S \Rightarrow v \in T$ is a theorem.

Extensionality (11.11b) and Subset (11.13m) will, **by LADM**, mostly be used as the following inference rules:

$$\frac{v \in S \equiv v \in T}{S = T} \qquad \frac{v \in S \Rightarrow v \in T}{S \subseteq T}$$

LADM Set Equality via Equivalence

(11.4) **Axiom, Extensionality:** Provided $\neg occurs('x', 'S, T')$,

$$S = T \quad \equiv \quad (\forall x \bullet x \in S \equiv x \in T)$$

(11.9) **"Simple set comprehension equality":** $\{x \mid Q\} = \{x \mid R\} \equiv (\forall x \bullet Q \equiv R)$

(11.10) **Metatheorem set comprehension equality:**

$\{x \mid Q\} = \{x \mid R\}$ is valid iff $Q \equiv R$ is valid.

(11.11) **Methods for proving set equality** $S = T$**:**

(a) Use Leibniz directly
(b) Use axiom Extensionality (11.4) and prove $v \in S \equiv v \in T$
(c) Prove $Q \equiv R$ and conclude $\{x \mid Q\} = \{x \mid R\}$ via (11.9)/(11.10)

Note:

- In the informal setting, confusion about variable binding is easy!
- Using "Set extensionality" or Using (11.9)
  followed by For any ... make variable binding clear.

Using Set Extensionality — LADM-Style

Extensionality (11.11b) inference rule: $\dfrac{v \in S \equiv v \in T}{S = T}$

Ex. 8.2(a) Prove: $\{E, E\} = \{E\}$ for each expression E.

By extensionality (11.11b):

Proving $v \in \{E, E\} \equiv v \in \{E\}$**:**

$v \in \{E, E\}$
$\equiv$ ⟨ Set enumerations (11.2) ⟩
$v \in \{x \mid x = E \vee x = E\}$
$\equiv$ ⟨ Idempotency of $\vee$ (3.26) ⟩
$v \in \{x \mid x = E\}$
$\equiv$ ⟨ Set enumerations (11.2) ⟩
$v \in \{E\}$

Using Set Extensionality — CalcCheck Example

Axiom (11.4) "Set extensionality": $S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$
— provided $\neg occurs('x', 'S, T')$

Theorem (11.26) "Symmetry of ∪": $S \cup T = T \cup S$
Proof:
Using "Set extensionality":
Subproof for `∀ e • e ∈ S ∪ T ≡ e ∈ T ∪ S`:
For any `e`:

```
    e ∈ S ∪ T
  ≡⟨ "Union" ⟩
    e ∈ S ∨ e ∈ T
  ≡⟨ "Symmetry of ∨" ⟩
    e ∈ T ∨ e ∈ S
  ≡⟨ "Union" ⟩
    e ∈ T ∪ S
```

Anything Wrong?

Let the set Q be defined by the following:

(R) $\quad Q = \{S \mid S \notin S\}$

Then:

$$\begin{array}{ll} & Q \in Q \\ \equiv & \langle \text{ (R) } \rangle \\ & Q \in \{S \mid S \notin S\} \\ \equiv & \langle \text{ (11.3) Membership in set comprehension } \rangle \\ & (\exists S \mid S \notin S \bullet Q = S) \\ \equiv & \langle \text{ (9.19) Trading for } \exists\text{, (8.14) One-point rule } \rangle \\ & Q \notin Q \\ \equiv & \langle \text{ (11.0) Def. } \notin \rangle \\ & \neg(Q \in Q) \end{array}$$

With (3.15) $p \equiv \neg p \equiv \mathit{false}$, this proves:

(R′) $\quad \mathit{false}$ — **"Russell's paradox"**

$\_\in\_ , \_\notin\_ : A \to \mathbf{set}\ A \to \mathbb{B}$

"The mother of all type errors"

$\Longrightarrow$ **birth of type theory…**

"The Universe" in LADM

THE UNIVERSE

A theory of sets concerns sets constructed from some collection of elements. There is a theory of sets of integers, a theory of sets of characters, a theory of sets of sets of integers, and so forth. This collection of elements is called the *domain of discourse* or the *universe of values*; it is denoted by $\mathbf{U}$. The universe can be thought of as the type of every set variable in the theory. For example, if the universe is $set(\mathbb{Z})$, then $v{:}set(\mathbb{Z})$.

When several set theories are being used at the same time, there is a different universe for each. The name $\mathbf{U}$ is then overloaded, and we have to distinguish which universe is intended in each case. This overloading is similar to using the constant 1 as a denotation of an integer, a real, the identity matrix, and even (in some texts, alas) the boolean $true$.

Overloading via type polymorphism: $\quad \{\}, \mathbf{U} : \mathbf{set}\ t$

$$\begin{array}{lcllcl} (\{\} : \mathbf{set}\ \mathbb{B}) & = & \{\} & \quad (\mathbf{U} : \mathbf{set}\ \mathbb{B}) & = & \{\mathit{false}, \mathit{true}\} \\ (\{\} : \mathbf{set}\ \mathbb{N}) & = & \{\} & \quad (\mathbf{U} : \mathbf{set}\ \mathbb{N}) & = & \{k : \mathbb{N} \mid \mathit{true}\} \end{array}$$

"The Universe" and Complement in LADM

the *domain of discourse* or the *universe of values*; it is denoted by $\mathbf{U}$. The universe can be thought of as the type of every set variable in the theory. For example, if the universe is $set(\mathbb{Z})$, then $v{:}set(\mathbb{Z})$.

COMPLEMENT

The *complement* of S, written $\sim S$,[4] is the set of elements that are not in S (but are in the universe). In the Venn diagram in this paragraph, we have shown set S and universe $\mathbf{U}$. The non-filled area represents $\sim S$.

(11.17) **Axiom, Complement:** $v \in \sim S \quad \equiv \quad v \in \mathbf{U} \land v \notin S$

For example, for $\mathbf{U} = \{0,1,2,3,4,5\}$, we have

$$\sim \{3,5\} \;=\; \{0,1,2,4\} \quad ,$$
$$\sim \mathbf{U} \;=\; \emptyset \quad , \qquad \sim \emptyset \;=\; \mathbf{U} \quad .$$

We can easily prove

(11.18) $v \in \ \sim S \quad \equiv \quad v \notin S \quad$ (for v in $\mathbf{U}$).

“The” Universe

Frequently, a “domain of discourse” is assumed, that is, a set of “all objects under consideration”.

This is often called a “**universe**”. Special notation: **U** — `\universe`

Declaration: $\mathbf{U} : \mathbf{set}\ t$

Axiom “Universal set”: $x \in \mathbf{U}$ — remember: $\_\in\_ : t \to \mathbf{set}\ t \to \mathbb{B}$

Theorem: $(\mathbf{U} : \mathbf{set}\ t) = \{x : t \bullet x\}$

Types are not sets! — $(\mathbf{U} : \mathbf{set}\ t)$ is the set containing all values of type t.

We define a nicer notation: $\llcorner\ t\ \lrcorner = (\mathbf{U} : \mathbf{set}\ t)$

“Definition of $\llcorner\_\lrcorner$”: $\forall x : t \bullet x \in \llcorner\ t\ \lrcorner$

Example: $\llcorner\ \mathbb{B}\ \lrcorner = \{false, true\}$

Set Complement

(11.17) **Axiom, Complement:** $v \in \sim S \quad \equiv \quad v \in \mathbf{U} \wedge v \notin S$

Complement can be expressed via difference: $\sim S \quad = \quad \mathbf{U} - S$

Complement $\sim$ **always implicitly depends on the universe U!**

Example: $\sim \{true\} \quad = \quad \llcorner\ \mathbb{B}\ \lrcorner - \{true\} \quad = \quad \{false, true\} - \{true\} \quad = \quad \{false\}$

LADM: “We can easily prove
(11.18) $v \in \sim S \quad \equiv \quad v \notin S$ (for v in **U**).”

Consider $\mathbb{Z}_+ : \mathbf{set}\ \mathbb{Z}$ defined as $\mathbb{Z}_+ = \{x : \mathbb{Z} \mid \mathrm{pos}\ x\}$:

- Let S be a subset of $\mathbb{Z}_+$. For example: $S = \{2, 3, 7\}$
- Consider the complement $\sim S$
- Is $-5 \in \sim S$ true or false?

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-08

Sets (ctd.)

Recall: The Axioms of Set Theory — Overview

(11.2) Provided $\neg occurs(`x', `e_0, \ldots, e_{n-1}')$, $\{e_0, \ldots, e_{n-1}\} = \{x \mid x = e_0 \vee \cdots \vee x = e_{n-1} \bullet x\}$

(11.3) **Axiom, Set membership:** Provided $\neg occurs(`x', `F')$,
$$F \in \{x \mid R \bullet E\} \equiv (\exists x \mid R \bullet E = F)$$

Empty Set: $v \in \{\} \equiv \mathit{false}$

(11.12) **Axiom, Size:** Provided $\neg occurs(`x', `S')$, $\# S = (\Sigma x \mid x \in S \bullet 1)$

(11.4) **Axiom, Extensionality:** Provided $\neg occurs(`x', `S, T')$,
$$S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$$

Axiom, Subset: Provided $\neg occurs(`x', `S, T')$,
$$S \subseteq T \equiv (\forall x \bullet x \in S \Rightarrow x \in T)$$

(11.14) **Axiom, Proper subset:** $S \subset T \equiv S \subseteq T \wedge S \neq T$

Axiom, Complement: $v \in {\sim} S \equiv v \notin S$

(11.20) **Axiom, Union:** $v \in S \cup T \equiv v \in S \vee v \in T$

(11.21) **Axiom, Intersection:** $v \in S \cap T \equiv v \in S \wedge v \in T$

(11.22) **Axiom, Set difference:** $v \in S - T \equiv v \in S \wedge v \notin T$

(11.23) **Axiom, Power set:** $v \in \mathbb{P}\, S \equiv v \subseteq S$

(14.3) **Axiom, Cross product:** $S \times T = \{b, c \mid b \in S \wedge c \in T \bullet \langle b, c \rangle\}$

Set Comprehension and Quantification Semantics

- Evaluated in state s, the expression $\{ x : t \mid R \bullet E \}$ denotes the set containing the values of E evaluated in those states resulting from s by changing the binding of x to those values from type t that satisfy R.
- Evaluated in state s, the expression $(\sum x : t \mid R \bullet E)$ denotes the sum of the values of E evaluated in those states resulting from s by changing the binding of x to those values from type t that satisfy R.
- Evaluated in state s, the expression $(\forall x : t \mid R \bullet P)$ evaluates to true iff P evaluates to true in **all** those states resulting from s by changing the binding of x to those values from type t that satisfy R.
- Evaluated in state s, the expression $(\exists x : t \mid R \bullet P)$ evaluates to true iff P evaluates to true in **at least one** state resulting from s by changing the binding of x to a value from type t that satisfies R.

Cardinality Example

(11.12) **Axiom, Size:** Provided $\neg occurs(`x', `S')$, $\# S = (\Sigma x \mid x \in S \bullet 1)$

$\# \{1, 1, 2\}$

$=$ ⟨ (11.12) Axiom, Size ⟩

$(\Sigma x \mid x \in \{1, 1, 2\} \bullet 1)$

$=$ ⟨ (11.2) Set enumeration ⟩

$(\Sigma x \mid x \in \{y \mid y = 1 \vee y = 1 \vee y = 2 \bullet y\} \bullet 1)$

$=$ ⟨ (11.3) Set membership, (9.19) Trading for $\exists$ ⟩

$(\Sigma x \mid (\exists y \mid y = x \bullet y = 1 \vee y = 1 \vee y = 2) \bullet 1)$

$=$ ⟨ (8.14) One-point rule: $(\star x \mid x = E \bullet P) = P[x := E]$ prov. $\neg occurs(`x', `E')$ ⟩

$(\Sigma x \mid x = 1 \vee x = 1 \vee x = 2 \bullet 1)$

$=$ ⟨ (3.26) Idempotency of $\vee$ ⟩

$(\Sigma x \mid x = 1 \vee x = 2 \bullet 1)$

$=$ ⟨ (8.16) Disjoint range split: $(x = 1 \wedge x = 2) \equiv \mathit{false}$ ⟩

$(\Sigma x \mid x = 1 \bullet 1) + (\Sigma x \mid x = 2 \bullet 1)$

$=$ ⟨ (8.14) One-point rule ⟩

$1 + 1$

$=$ ⟨ Arithmetic ⟩

2

"The" Universe

Frequently, a "domain of discourse" is assumed, that is, a set of "all objects under consideration".

This is often called a "**universe**". Special notation: **U** — `\universe`

Declaration: $\mathbf{U} : \mathbf{set}\ t$

Axiom "Universal set": $x \in \mathbf{U}$ — remember: $\_\in\_ : t \to \mathbf{set}\ t \to \mathbb{B}$

Theorem: $(\mathbf{U} : \mathbf{set}\ t) = \{x : t \bullet x\}$

Types are not sets! — $(\mathbf{U} : \mathbf{set}\ t)$ is the set containing all values of type t.

We define a nicer notation: $\llcorner t \lrcorner = (\mathbf{U} : \mathbf{set}\ t)$ — `\llcorner ... \lrcorner`

- $\llcorner t \lrcorner$ is the **set of all values of type** t
- "Definition of $\llcorner \_ \lrcorner$": $\forall x : t \bullet x \in \llcorner t \lrcorner$
- Example: $\llcorner \mathbb{B} \lrcorner = \{\mathit{false}, \mathit{true}\}$

Set Complement

(11.17) **Axiom, Complement:** $v \in {\sim} S \quad \equiv \quad v \in \mathbf{U} \land v \notin S$

Complement can be expressed via difference: ${\sim} S \quad = \quad \mathbf{U} - S$

Complement $\sim$ **always implicitly depends on the universe U!**

Example: ${\sim}\{\mathit{true}\} \quad = \quad \llcorner \mathbb{B} \lrcorner - \{\mathit{true}\} \quad = \quad \{\mathit{false}, \mathit{true}\} - \{\mathit{true}\} \quad = \quad \{\mathit{false}\}$

LADM: "We can easily prove
(11.18) $v \in {\sim} S \quad \equiv \quad v \notin S$ (for v in **U**)."

Consider $\mathbb{Z}_+ : \mathbf{set}\ \mathbb{Z}$ defined as $\mathbb{Z}_+ = \{x : \mathbb{Z} \mid \mathsf{pos}\ x\}$:

- Let S be a subset of $\mathbb{Z}_+$. For example: $S = \{2, 3, 7\}$
- Consider the complement ${\sim} S$
- Is $-5 \in {\sim} S$ true or false?

We will just rely on the type system to provide the relevant universe, and use:

Axiom "Set complement": $v \in {\sim} S \quad \equiv \quad v \notin S$

Set Complement via Set Difference

Theorem (11.55.1) "Set complement via difference": ${\sim} S = \mathbf{U} - S$

Proof:

Using "Set extensionality":

For any \`$x$\`:

$x \in \mathbf{U} - S$

$\equiv \langle$ "Set difference" $\rangle$

$x \in \mathbf{U} \land \neg(x \in S)$

$\equiv \langle$ "Universal set", "Identity of $\land$" $\rangle$

$\neg(x \in S)$

$\equiv \langle$ "Set complement" $\rangle$

$x \in {\sim} S$

Let c be defined by: $x \le c \quad \equiv \quad x \le 5$

What do you know about c? Why? (Prove it!)

Note: x is implicitly univerally quantified!

Proving $5 \le c$**:**

$$\begin{array}{ll} & 5 \le c \\ \equiv & \langle \text{ The given equivalence, with } x := 5 \ \rangle \\ & 5 \le 5 \quad \text{— This is Reflexivity of } \le \end{array}$$

Proving $c \le 5$**:**

$$\begin{array}{ll} & c \le 5 \\ \equiv & \langle \text{ Given equivalence, with } x := c \ \rangle \\ & c \le c \quad \text{— This is Reflexivity of } \le \end{array}$$

With antisymmetry of $\le$ (that is, $a \le b \ \wedge \ b \le a \ \Rightarrow \ a = b$), we obtain $c = 5$ — An instance of:

(15.47) **Indirect equality**: $a = b \quad \equiv \quad (\forall z \ \bullet \ z \le a \quad \equiv \quad z \le b)$

Relative Pseudocomplement

Let A, B : **set** t be two sets of the same type.

The **relative pseudocomplement** $A \Rightarrow B$ of A with respect to B is defined by:

$$X \subseteq (A \Rightarrow B) \quad \equiv \quad X \cap A \subseteq B$$

Calculate the **relative pseudocomplement** $A \Rightarrow B$ as a set expression
not using $\Rightarrow$! That is:

$$\text{Calculate} \quad A \Rightarrow B \quad = \quad ?$$

Using set extensionality, that is:

$$\text{Calculate} \quad x \in A \Rightarrow B \quad \equiv \quad x \in ?$$

Characterisation of relative pseudocomplement of sets: $X \subseteq (A \Rightarrow B) \quad \equiv \quad X \cap A \subseteq B$

$$\begin{array}{ll} & x \in A \Rightarrow B \\ \equiv & \langle \ e \in S \equiv \{e\} \subseteq S \quad \text{—} \quad \text{Exercise!} \ \rangle \\ & \{x\} \subseteq A \Rightarrow B \\ \equiv & \langle \text{ Def. } \Rightarrow \text{, with } X := \{x\} \ \rangle \\ & \{x\} \cap A \subseteq B \\ \equiv & \langle \text{ (11.13) Subset } \rangle \\ & (\forall y \mid y \in \{x\} \cap A \bullet y \in B) \\ \equiv & \langle \text{ (11.21) Intersection } \rangle \\ & (\forall y \mid y \in \{x\} \wedge y \in A \bullet y \in B) \\ \equiv & \langle \ y \in \{x\} \equiv y = x \quad \text{—} \quad \text{Exercise!} \ \rangle \\ & (\forall y \mid y = x \wedge y \in A \bullet y \in B) \\ \equiv & \langle \text{ (9.4b) Trading for } \forall \text{, Def. } \notin \ \rangle \\ & (\forall y \mid y = x \bullet y \notin A \vee y \in B) \\ \equiv & \langle \text{ (8.14) One-point rule } \rangle \\ & x \notin A \vee x \in B \\ \equiv & \langle \text{ (11.17) Set complement, (11.20) Union } \rangle \\ & x \in {\sim}A \cup B \end{array}$$

Theorem: $A \Rightarrow B \quad = \quad {\sim}A \cup B$

Characterisation of relative pseudocomplement of sets: $X \subseteq A \Rightarrow B \quad \equiv \quad X \cap A \subseteq B$

Theorem "Pseudocomplement via ∪": $A \Rightarrow B \quad = \quad {\sim} A \cup B$

Calculation:

$$
\begin{array}{ll}
& x \in A \Rightarrow B \\
\equiv & \langle \text{ Pseudocomplement via } \cup \rangle \\
& x \in {\sim} A \cup B \\
\equiv & \langle \text{ (11.20) Union, (11.17) Set complement } \rangle \\
& \neg(x \in A) \lor x \in B \\
\equiv & \langle \text{ (3.59) Material implication } \rangle \\
& x \in A \Rightarrow x \in B
\end{array}
$$

Corollary "Membership in pseudocomplement":
$x \in A \Rightarrow B \quad \equiv \quad x \in A \Rightarrow x \in B$

Easy to see: On sets, relative pseudocomplement wrt. {} is complement:
$A \Rightarrow \{\} \quad = \quad {\sim} A$

Power Set

(11.23) **Axiom, Power set:** $v \in \mathbb{P}\, S \quad \equiv \quad v \subseteq S$

Declaration: $\mathbb{P}\_ \ :\ \textbf{set}\ t \to \textbf{set}\ (\textbf{set}\ t)$

— remember: **set** : Type → Type

$\mathbb{P}\ \{0,1\} = \{\{\}, \{0\}, \{1\}, \{0,1\}\}$

- For a set S, the set of its **subsets** is $\mathbb{P}\, S$
- For a type t, the **type of sets of elements of type t** is **set** t
- Therefore we have: $\llcorner\ \textbf{set}\ t\ \lrcorner = \mathbb{P} \llcorner\ t\ \lrcorner$
- According to the textbook, **type annotations $v : t$**, in particular in variable declarations in quantifications and in set comprehensions, **may only use types t**.
- (The **specification notation Z** allows the use of sets in variable declarations — **this makes ∀ and ∃ rules more complicated.**)

Calculate!

The size of a finite set S is written $\# S$.
(11.23) **Axiom, Power set:** $v \in \mathbb{P}\, S \quad \equiv \quad v \subseteq S$

- $\#(\mathbb{P}\, \mathbb{B})$
- $\#(\mathbb{P}\ \{1,2,3\})$
- $\#(\mathbb{P}\ \{1,2,3,4,5\})$
- $\#(\mathbb{P}\ \{2,3,4,5\})$
- $\#(\mathbb{P}\ \{1,2,3\} \quad \cap \quad \mathbb{P}\ \{2,3,4,5\})$
- $\#(\mathbb{P}\ \{1,2,3\} \quad \cup \quad \mathbb{P}\ \{2,3,4,5\})$
- $\#(\mathbb{P}\ \{2,3,4,5\} \quad - \quad \mathbb{P}\ \{1,2,3\})$
- $(\Sigma\, S : \mathbb{P}\ \{1,2,3,4,5\} \bullet (\Sigma n \mid n \in S \bullet n))$
- $(\Sigma\, S : \mathbb{P}\ \{1,2,3,4,5\} \mid \# S > 1 \bullet (\Sigma\, n \mid n \in S \bullet n))$
- $(\Sigma\, S : \mathbb{P}\ \{1,2,3,4,5\} \mid \# S > 2 \bullet (\Sigma\, n \mid n \in S \bullet n))$

Calculate!

The **size** of a finite set S, that is, the number of its elements, is written $\# S$.

Calculate:

- $\# \llcorner \mathbb{B} \lrcorner$
- $\# \{S : \textbf{set}\ \mathbb{B} \mid \mathit{true} \in S \bullet S\}$
- $\# \{T : \textbf{set set}\ \mathbb{B} \mid \{\} \notin T \bullet T\}$
- $\# \{S : \textbf{set}\ \mathbb{N} \mid (\forall x : \mathbb{N} \mid x \in S \bullet x < n) \land \# S = k \bullet S\}$

- $\llcorner \mathbb{B} \lrcorner = \{\mathit{false}, \mathit{true}\}$
- $S \in \llcorner \textbf{set}\ \mathbb{B} \lrcorner \quad \equiv \quad S \subseteq \llcorner \mathbb{B} \lrcorner$
- $\llcorner \textbf{set}\ \mathbb{B} \lrcorner = \{\{\}, \{\mathit{false}\}, \{\mathit{true}\}, \{\mathit{false}, \mathit{true}\}\}$
- $T \in \llcorner \textbf{set set}\ \mathbb{B} \lrcorner \quad \equiv \quad T \subseteq \mathbb{P} \llcorner \mathbb{B} \lrcorner$

Metatheorem (11.25): Sets $\Longleftrightarrow$ Propositions

Let

- $P, Q, R, \ldots$ be set variables
- $p, q, r, \ldots$ be propositional variables
- E, F be expressions built from these set variables and $\cup$, $\cap$, $\sim$, $\mathbf{U}$, $\{\}$.

Define the Boolean expressions E_p and F_p by replacing

$P, Q, R, \ldots$	with	$p, q, r, \ldots$	$\sim$	with	$\neg$
$\cup$	with	$\lor$	$\mathbf{U}$	with	true
$\cap$	with	$\land$	$\{\}$	with	false

Then:

- $E = F$ is valid iff $E_p \equiv F_p$ is valid.
- $E \subseteq F$ is valid iff $E_p \Rightarrow F_p$ is valid.
- $E = \mathbf{U}$ is valid iff E_p is valid.

Metatheorem (11.25): Sets $\Longleftrightarrow$ Propositions — Examples

Let E, F be expressions built from set variables $P, Q, R, \ldots$ and $\cup$, $\cap$, $\sim$, $\mathbf{U}$, $\{\}$.

Define the Boolean expressions E_p and F_p by replacing

$P, Q, R, \ldots$	with	$p, q, r, \ldots$	$\sim$	with	$\neg$
$\cup$	with	$\lor$	$\mathbf{U}$	with	true
$\cap$	with	$\land$	$\{\}$	with	false

Then:

- $E = F$ is valid iff $E_p \equiv F_p$ is valid.
- $E \subseteq F$ is valid iff $E_p \Rightarrow F_p$ is valid.
- $E = \mathbf{U}$ is valid iff E_p is valid.

"Free" theorems! — (typically proven via set extensionality/inclusion and unfold-fold)

$$P \cap (P \cup Q) = P$$
$$P \cap (Q \cup R) = (P \cap Q) \cup (P \cap R)$$
$$P \cup (Q \cap R) \subseteq P \cup Q$$
$$\vdots$$

Tuples and Tuple Types in CALCCHECK

Tuples can have arbitrary "arity" at least 2.

Example: A triple with type: $\langle 2, true, "Hello" \rangle : \langle\!\langle\, \mathbb{Z}, \mathbb{B}, String \,\rangle\!\rangle$

Example: A seven-tuple: $\langle 3, true, 5 \triangleleft \epsilon, \langle 5, false \rangle, "Hello", \{2,8\}, \{42 \triangleleft \epsilon\} \rangle$

The type of this: $\langle\!\langle\, \mathbb{Z}, \mathbb{B}, Seq\ \mathbb{Z}, \langle\!\langle\, \mathbb{Z}, \mathbb{B} \,\rangle\!\rangle, String, \mathbf{set}\ \mathbb{Z}, \mathbf{set}\ (Seq\ \mathbb{Z}) \,\rangle\!\rangle$

- Tuples are enclosed in $\langle \ldots \rangle$ as in LADM. (type "\<" and "\>")
- Tuple types are enclosed in $\langle\!\langle \ldots \rangle\!\rangle$. (type "\<!" and "\>!")
- Otherwise, tuples and tuple types "work" as in Haskell.
- In particular, there is no implicit nesting:

  $\langle\!\langle \langle\!\langle A, B \rangle\!\rangle, C \rangle\!\rangle$ and $\langle\!\langle A, B, C \rangle\!\rangle$ and $\langle\!\langle A, \langle\!\langle B, C \rangle\!\rangle \rangle\!\rangle$ are three different types!

Pairs and Pair Projections

Cartesian product of types: Two-tuple types, pair types:

$b : t_1$ and $c : t_2$ are well-typed iff $\langle b, c \rangle : \langle\!\langle\, t_1, t_2 \,\rangle\!\rangle$ is well-typed.

Pair projections: $\mathsf{fst} : \langle\!\langle\, t_1, t_2 \,\rangle\!\rangle \to t_1$ $\qquad \mathsf{fst}\ \langle b, c \rangle = b$

$\mathsf{snd} : \langle\!\langle\, t_1, t_2 \,\rangle\!\rangle \to t_2$ $\qquad \mathsf{snd}\ \langle b, c \rangle = c$

Pair equality: For $p, q : \langle\!\langle\, t_1, t_2 \,\rangle\!\rangle$, $\qquad p = q \quad \equiv \quad \mathsf{fst}\ p = \mathsf{fst}\ q \wedge \mathsf{snd}\ p = \mathsf{snd}\ q$

Theorem "Pair extensionality": For $p : \langle\!\langle\, t_1, t_2 \,\rangle\!\rangle$, $\qquad p \quad = \quad \langle\ \mathsf{fst}\ p,\ \mathsf{snd}\ p\ \rangle$

Proof:

$p = \langle \mathsf{fst}\ p, \mathsf{snd}\ p \rangle$

$=$ ⟨ Pair equality ⟩

$\mathsf{fst}\ p = \mathsf{fst}\ \langle \mathsf{fst}\ p, \mathsf{snd}\ p \rangle \quad \wedge \quad \mathsf{snd}\ p = \mathsf{snd}\ \langle \mathsf{fst}\ p, \mathsf{snd}\ p \rangle$

$=$ ⟨ Pair projections ⟩

$\mathsf{fst}\ p = \mathsf{fst}\ p \quad \wedge \quad \mathsf{snd}\ p = \mathsf{snd}\ p$

$=$ ⟨ Reflexivity of equality, Idempotency of $\wedge$ ⟩

$true$

LADM: Pairs and Cross Products

If b and c are expressions, then $\langle b, c \rangle$ is their **2-tuple** or **ordered pair**

— "ordered" means that there is a **first** constituent (b) and a **second** constituent (c).

(14.2) **Axiom, Pair equality:** $\langle b, c \rangle = \langle b', c' \rangle \quad \equiv \quad b = b' \wedge c = c'$

(14.3) **Axiom, Cross product:** $S \times T \ = \ \{b, c \mid b \in S \wedge c \in T \bullet \langle b, c \rangle\}$

— This uses: $\_\times\_ : \mathbf{set}\ t_1 \to \mathbf{set}\ t_2 \to \mathbf{set}\ \langle\!\langle t_1, t_2 \rangle\!\rangle$

(14.4) **Membership:** $\langle b, c \rangle \in S \times T \quad \equiv \quad b \in S \wedge c \in T$

(14.5) $\langle x, y \rangle \in S \times T \quad \equiv \quad \langle y, x \rangle \in T \times S$

(14.6) $S = \{\} \quad \Rightarrow \quad S \times T = T \times S = \{\}$

(14.7) $S \times T = T \times S \quad \equiv \quad S = \{\} \vee T = \{\} \vee S = T$

(14.8) **Distributivity of $\times$ over $\cup$:** $S \times (T \cup U) \ = \ (S \times T) \cup (S \times U)$

$(S \cup T) \times U \ = \ (S \times U) \cup (T \times U)$

(14.9) **Distributivity of $\times$ over $\cap$:** $S \times (T \cap U) \ = \ (S \times T) \cap (S \times U)$

$(S \cap T) \times U \ = \ (S \times U) \cap (T \times U)$

(14.10) **Distributivity of $\times$ over $-$:** $S \times (T - U) \ = \ (S \times T) - (S \times U)$

$(S - T) \times U \ = \ (S \times U) - (T \times U)$

(14.12) **Monotonicity:** $S \subseteq S' \wedge T \subseteq T' \quad \Rightarrow \quad S \times T \subseteq S' \times T'$

Some Spice...

Converting between "different ways to take two arguments":

$$\begin{array}{lcl} \mathsf{curry} & : & (\langle A, B \rangle \to C) \to (A \to B \to C) \\ \mathsf{curry}\ f\ x\ y & = & f\ \langle x, y \rangle \end{array}$$

$$\begin{array}{lcl} \mathsf{uncurry} & : & (A \to B \to C) \to (\langle A, B \rangle \to C) \\ \mathsf{uncurry}\ g\ \langle x, y \rangle & = & g\ x\ y \end{array}$$

These functions correspond to the "Shunting" law:

(3.65) **Shunting:** $p \wedge q \Rightarrow r \quad \equiv \quad p \Rightarrow (q \Rightarrow r)$

The "currying" concept is named for Haskell Brooks Curry (1900–1982),
but goes back to Moses Ilyich Schönfinkel (1889–1942)
and Gottlob Frege (1848–1925).

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-10

General Induction

Descending Chains in Numbers

Consider numbers with the usual strict-order <, and consider descending chains, like $17 > 12 > 9 > 8 > 3 > \ldots$

Are there infinite descending chains in

- $\mathbb{Z}$? — $0 > -1 > -2 > -3 > \ldots$
- $\mathbb{N}$? — **No**
- $\mathbb{R}$? — $0 > -1 > -2 > -3 > \ldots$
- $\mathbb{R}_+$? — $\pi^0 > \pi^{-1} > \pi^{-2} > \pi^{-3} > \ldots$
- $\mathbb{Q}_+$? — $1 > 1/2 > 1/3 > 1/4 > \ldots$
- $\mathbb{C}$? — no "default" order!

Relations $\precsim$ with **no** infinite (descending) $\succsim$-chains are **well-founded**.

while-loops **terminate** iff they are "going down" some well-founded relation.

Induction over inductive datatypes like $\mathbb{N}$ and Seq A is based on of their well-founded respective `_part – of_` relation.

Idea Behind Induction — How Does It Work? — Informally

Proving $(\forall\, x : t \bullet P)$ by induction, **for an appropriate type t**:

- You are familiar with proving a base case and an induction step
- The base cases establish $P[x := S]$, for each S that are "simplest t"
- The induction steps work for $x : t$ for which we already know $P[x := x]$ and from that establish $P[x := C\,x]$ for elements $C\,x : t$ that "are slightly more complicated than x".
- Since the construction principle(s) ("C") used in the induction step is/are sufficiently powerful to construct all $x : t$, this justifies $(\forall\, x : t \bullet P)$.

Idea Behind Induction — How Does It Work? — Informally

Proving $(\forall\, x : t \bullet P)$ by induction, **for an appropriate type t**:

- You are familiar with proving a base case and an induction step
- The base cases establish $P[x := S]$, for each S that are "simplest t"
- The induction steps work for $x : t$ for which we already know $P[x := x]$ and from that establish $P[x := C\,x]$ for elements $C\,x : t$ that "are slightly more complicated than x".
- Since the construction principle(s) ("C") used in the induction step is/are sufficiently powerful to construct all $x : t$, this justifies $(\forall\, x : t \bullet P)$.

Looking at this from the other side:

- Each element $x : t$ is either a "simplest element" ("S"), or constructed via a construction principle ("C") from "slightly simpler elements" y, that is, $x = C\,y$.
- In the first case, the base case gives you the proof for $P[x := S]$.
- In the second case, you obtain $P[x := Cy]$ via the induction step from a proof for $P[x := y]$, if you can find that.
- You can find that proof if repeated decomposition into S or C always terminates.

Idea Behind Induction — Reduction via Well-founded Relations

- Goal: prove $(\forall\, x : T \bullet P\,x)$ for some property $P : T \to \mathbb{B}$ (with $\neg occurs(`x', `P')$)
- Situation: Elements of T are related via $\_\succ\_ : T \to T \to \mathbb{B}$ with "simpler" elements (constituents, predecessors, parts, ...)
  "$y \prec x$" may read "y precedes x" or "y is an (immediate) constituent of x" or "y is simpler than x" or "y is below x"...
- If for every $x : T$ there is a proof that

  if $P\,y$ for all predecessors y of x, then $P\,x$,

  then for every $z : T$ with $\neg(P\,z)$:
  - there is a predecessor u of z with $\neg(P\,u)$
  - and so there is an infinite $\succ$-chain (of elements c with $\neg(P\,c)$) starting at z.

Theorem "Mathematical induction over $\langle T, \prec \rangle$":
If there are no infinite $\succ$-chains in T, (that is, **if $\prec$ is noetherian**), then:

$$(\forall\, x \bullet P\,x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \prec x \bullet P\,y) \Rightarrow P\,x)$$

“$\langle T, \precsim \rangle$ Admits Induction” (LADM Section 12.4)

Definition (12.19): $\langle T, \precsim \rangle$ **admits induction** iff the following principle of **mathematical induction over** $\langle T, \precsim \rangle$ holds for all properties $P : T \to \mathbb{B}$:

$$(\forall\, x \bullet P\, x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \precsim x \bullet P\, y) \Rightarrow P\, x)$$

Definition (12.21): $\langle T, \precsim \rangle$ is **well-founded** iff every non-empty subset of T has a minimal element wrt. $\precsim$, that is:

$$\forall\, S : \mathbf{set}\, T \quad \bullet \quad S \neq \{\} \quad \equiv \quad \exists\, x : T \bullet x \in S \wedge \forall\, y : T \mid y \precsim x \bullet y \notin S$$

Theorem (12.22): $\langle T, \precsim \rangle$ is well-founded iff it admits induction.

Definition (12.25′): $\langle T, \precsim \rangle$ is **noetherian** iff there are no infinite $\succsim$-chains in T.

Definition (12.25″): $\langle T, \precsim \rangle$ is **noetherian** iff $\neg(\exists\, s : \mathbb{N} \to T \bullet \forall\, n : \mathbb{N} \bullet s\,(n+1) \precsim s\, n)$

Theorem (12.26): $\langle T, \precsim \rangle$ is well-founded iff it is noetherian.

Theorem “Mathematical induction over $\langle T, \precsim \rangle$”:
If there are no infinite $\succsim$-chains in T, that is, **if $\precsim$ is noetherian**, then:

$$(\forall\, x \bullet P\, x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \precsim x \bullet P\, y) \Rightarrow P\, x)$$

Mathematical Induction in $\mathbb{N}$

Consider $\_\precsim\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ with $(x \precsim y) \;=\; (y \succsim x) \;=\; (y = \mathsf{suc}\, x)$. $\quad \_\precsim\_ = \ulcorner\mathsf{suc}\_\urcorner$

Mathematical induction over $(\mathbb{N}, \precsim)$:

$$
\begin{aligned}
& (\forall\, x : \mathbb{N} \bullet P\, x) \\
= \;& \langle \text{(12.19) Math. induction; Def. } \precsim \rangle \\
& (\forall\, x : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid \mathsf{suc}\, y = x \bullet P\, y) \Rightarrow P\, x) \\
= \;& \langle \text{Disjoint range split, with } \mathit{true} \equiv x = 0 \vee x > 0 \rangle \\
& (\forall\, x : \mathbb{N} \mid x = 0 \bullet (\forall\, y : \mathbb{N} \mid \mathsf{suc}\, y = x \bullet P\, y) \Rightarrow P\, x) \wedge \\
& (\forall\, x : \mathbb{N} \mid x > 0 \bullet (\forall\, y : \mathbb{N} \mid \mathsf{suc}\, y = x \bullet P\, y) \Rightarrow P\, x) \\
= \;& \langle \text{One-point rule; (8.22) Change of dummy — } x \mapsto \mathsf{suc}\, z \rangle \\
& ((\forall\, y : \mathbb{N} \mid \mathsf{suc}\, y = 0 \bullet P\, y) \Rightarrow P\, 0) \wedge \\
& (\forall\, z : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid \mathsf{suc}\, y = \mathsf{suc}\, z \bullet P\, y) \Rightarrow P\,(\mathsf{suc}\, z)) \\
= \;& \left\langle \begin{array}{l} \text{(8.13) Empty range, with } \mathsf{suc}\, y = 0 \equiv \mathit{false}; \\ \text{Cancellation of } \mathsf{suc}\text{, (8.14) One-point rule for } \forall \end{array} \right\rangle \\
& P\, 0 \wedge (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\,(\mathsf{suc}\, z))
\end{aligned}
$$

Mathematical Induction in $\mathbb{N}$ (ctd.)

Mathematical induction over $(\mathbb{N}, \ulcorner\mathsf{suc}\urcorner)$:

$$(\forall\, x : \mathbb{N} \bullet P\, x) \quad \equiv \quad P\, 0 \wedge (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\,(\mathsf{suc}\, z))$$

$$(\forall\, x : \mathbb{N} \bullet P\, x) \quad \equiv \quad P\, 0 \wedge (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\,(z + 1))$$

Absence of infinite **descending** $\ulcorner\mathsf{suc}\urcorner$ chains is due to the **inductive definition of** $\mathbb{N}$ **with constructors 0 and** suc : “…and nothing else is a natural number.”

Mathematical induction over $(\mathbb{N}, <)$ **“Complete induction over** $\mathbb{N}$**”:**

$$(\forall\, x : \mathbb{N} \bullet P\, x) \equiv (\forall\, x : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid y < x \bullet P\, y) \Rightarrow P\, x)$$

Complete induction gives you a **stronger induction hypothesis**
for non-zero x — some proofs become easier.

Natural Numbers Generated from 0 and suc — Explicit Induction Principle

Mathematical induction over $(\mathbb{N}, \ulcorner \mathsf{suc} \urcorner)$**:**

$$(\forall\, n : \mathbb{N} \bullet P\, n) \quad \equiv \quad P\, 0 \wedge (\forall\, n : \mathbb{N} \bullet P\, n \Rightarrow P\, (\mathsf{suc}\, n))$$

As **inference rule** underlying "By induction on `$n : \mathbb{N}$`":

*With **variable** $P : \mathbb{N} \to \mathbb{B}$:*

$$\frac{P\,0 \qquad \begin{array}{c}\ulcorner P\, n \urcorner \\ \vdots \\ P(\mathsf{suc}\, n)\end{array}}{P\, n}$$

*With $P : \mathbb{B}$ as **metavariable** for an expression:*

$$\frac{P[n := 0] \qquad \begin{array}{c}\ulcorner P \urcorner \\ \vdots \\ P[n := \mathsf{suc}\, n]\end{array}}{P}$$

As **axiom / theorem** — LADM p. 219: "weak induction":

Axiom "Induction over ℕ":

$P[n := 0]$
$\Rightarrow (\forall\, n : \mathbb{N} \mid P \bullet P[n := \mathsf{suc}\, n])$
$\Rightarrow (\forall\, n : \mathbb{N} \bullet P)$

Proving "Right-identity of +" Using the Induction Principle (v0)

```
Axiom "Induction over ℕ":
   P[n ≔ 0]
   ⇒ (∀ n : ℕ ❙ P • P[n ≔ suc n])
   ⇒ (∀ n : ℕ • P)

Theorem "Right-identity of +": ∀ m : ℕ • m + 0 = m
Proof:
  Using "Induction over ℕ":
    Subproof for `(m + 0 = m)[m ≔ 0]`:
      By substitution and "Definition of +"
    Subproof for `∀ m : ℕ ❙ m + 0 = m • (m + 0 = m)[m ≔ suc m]`:
      For any `m : ℕ` satisfying `m + 0 = m`:
          (m + 0 = m)[m ≔ suc m]
        =⟨ Substitution, "Definition of +" ⟩
          suc (m + 0) = suc m
        =⟨ Assumption `m + 0 = m`, "Reflexivity of =" ⟩
          true
```

(I never use this pattern with substitutions in the subproof goals.)

Proving "Right-identity of +" Using the Induction Principle (v1)

```
Axiom "Induction over ℕ":
   P[n ≔ 0]
   ⇒ (∀ n : ℕ ❙ P • P[n ≔ suc n])
   ⇒ (∀ n : ℕ • P)

Theorem "Right-identity of +": ∀ m : ℕ • m + 0 = m
Proof:
  Using "Induction over ℕ":
    Subproof for `0 + 0 = 0`:
      By "Definition of +"
    Subproof for `∀ m : ℕ ❙ m + 0 = m • suc m + 0 = suc m`:
      For any `m : ℕ` satisfying `m + 0 = m`:
          suc m + 0
        =⟨ "Definition of +" ⟩
          suc (m + 0)
        =⟨ Assumption `m + 0 = m` ⟩
          suc m
```

Proving "Right-identity of +" `Using` the Induction Principle (v2)

```
Theorem "Right-identity of +": ∀ m : ℕ • m + 0 = m
Proof:
  Using "Induction over ℕ":
    Subproof:
        0 + 0
      =⟨ "Definition of +" ⟩
        0
    Subproof:
      For any `m : ℕ` satisfying "IndHyp" `m + 0 = m`:
          suc m + 0
        =⟨ "Definition of +" ⟩
          suc (m + 0)
        =⟨ Assumption "IndHyp" ⟩
          suc m
```

```
Axiom "Induction over ℕ":
   P[n ≔ 0]
   ⇒ (∀ n : ℕ ❙ P • P[n ≔ suc n])
   ⇒ (∀ n : ℕ • P)
```

- (Subproof goals can be omitted where they are clear from the contained proof.)
- You need to understand (v0) and (v1) to be able to do (v2)!

"By induction on ..." versus `Using` Induction Principles

- Using induction principles directly is not much more verbose than "By induction on ..."
- "By induction on ..." only supports **very few** built-in induction principles
- Induction principles can be derived as theorems, or provided as axioms, and then can be used directly!

Mathematical Induction on Sequences

Cons induction: Mathematical induction over (Seq A, $\precsim$) where

$$xs \precsim ys \quad\equiv\quad \exists\, x : A \bullet x \triangleleft xs = ys$$

$$(\forall\, xs : \mathsf{Seq}\ A \bullet P\ xs) \quad\equiv\quad P\ \epsilon \wedge (\forall\, xs : \mathsf{Seq}\ A \mid P\ xs \bullet (\forall\, x : A \bullet P(x \triangleleft xs)))$$

Snoc induction: Mathematical induction over (Seq A, $\precsim$) where

$$xs \precsim ys \quad\equiv\quad \exists\, x : A \bullet xs \triangleright x = ys$$

$$(\forall\, xs : \mathsf{Seq}\ A \bullet P\ xs) \quad\equiv\quad P\ \epsilon \wedge (\forall\, xs : \mathsf{Seq}\ A \mid P\ xs \bullet (\forall\, x : A \bullet P(xs \triangleright x)))$$

Strict prefix induction: Mathematical induction over (Seq A, $\precsim$) where

$$xs \precsim ys \quad\equiv\quad \exists\, z : A; zs : \mathsf{Seq}\ A \bullet xs \frown z \triangleleft zs = ys$$

$$(\forall\, xs : \mathsf{Seq}\ A \bullet P\ xs) \quad\equiv\quad (\forall\, xs : \mathsf{Seq}\ A \bullet (\forall\, ys : \mathsf{Seq}\ A \mid ys \precsim xs \bullet P\ ys) \Rightarrow P\ xs)$$

Different induction hypotheses make certain proofs easier.

Sequences — Induction Principle

Cons induction: Mathematical induction over $(\mathsf{Seq}\ A, \precneqq)$ where

$$xs \precneqq ys \quad \equiv \quad \exists\, x : A \bullet x \triangleleft xs = ys$$

$$(\forall\, xs : \mathsf{Seq}\ A \bullet P\ xs) \quad \equiv \quad P\ \epsilon \wedge (\forall\, xs : \mathsf{Seq}\ A \mid P\ xs \bullet (\forall\, x : A \bullet P(x \triangleleft xs)))$$

As **inference rule** underlying "By induction on `$xs : \mathsf{Seq}\ A$`":

*With **variable** $P : \mathsf{Seq}\ A \to \mathbb{B}$:*

$$\frac{P\ \epsilon \qquad \begin{matrix}\ulcorner P\ xs \urcorner \\ \vdots \\ \forall\, x \bullet P\,(x \triangleleft xs)\end{matrix}}{P\ xs}$$

*With $P : \mathbb{B}$ as **metavariable** for an expression:*

$$\frac{P[xs := \epsilon] \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ \forall\, x \bullet P[xs := x \triangleleft xs]\end{matrix}}{P}$$

Axiomn "Induction over sequences":

$$\begin{aligned} & P[xs := \epsilon] \\ \Rightarrow\ & (\forall\, xs : \mathsf{Seq}\ A \mid P \bullet (\forall\, x : A \bullet P[xs := x \triangleleft xs]) \\ \Rightarrow\ & (\forall\, xs : \mathsf{Seq}\ A \bullet P) \end{aligned}$$

Recall: Tail is different — LADM Proof

Theorem (13.7) "Tail is different": $\forall\, \mathsf{xs} : \mathsf{Seq}\ A \bullet \forall\, x : A \bullet x \triangleleft \mathsf{xs} \neq \mathsf{xs}$

Proof:
- **By induction on** `$\mathsf{xs} : \mathsf{Seq}\ A$`:
  - **Base case**:
    - **For any** `$x : A$`:
      - $x \triangleleft \epsilon \neq \epsilon$
      - $\equiv$ ⟨ "Cons is not empty" ⟩
      - true
  - **Induction step**:
    - **For any** `$z : A$`, `$x : A$`:
      - $x \triangleleft z \triangleleft \mathsf{xs} \neq z \triangleleft \mathsf{xs}$
      - $\equiv$ ⟨ "Definition of $\neq$", "Cancellation of $\triangleleft$" ⟩
      - $\neg (x = z \wedge z \triangleleft \mathsf{xs} = \mathsf{xs})$
      - $\Leftarrow$ ⟨ "Consequence", "De Morgan", "Weakening", "Definition of $\neq$" ⟩
      - $z \triangleleft \mathsf{xs} \neq \mathsf{xs}$
      - $\equiv$ ⟨ Induction hypothesis `$\forall\, x : A \bullet x \triangleleft \mathsf{xs} \neq \mathsf{xs}$` ⟩
      - true

(For explanations about using "By induction on `$xs : \mathsf{Seq}\ A$`:" for proving "$\forall\, xs : \mathsf{Seq}\ A \bullet P$", see Ex6.1 and Ex6.2.)

Proving "Tail is different" `Using` the Induction Principle

Theorem "Induction over sequences":

$$\begin{aligned} & P[xs := \epsilon] \Rightarrow (\forall\, \mathsf{xs} : \mathsf{Seq}\ A \mid P \bullet (\forall\, x : A \bullet P[xs := x \triangleleft xs])) \\ & \qquad \Rightarrow (\forall\, \mathsf{xs} : \mathsf{Seq}\ A \bullet P) \end{aligned}$$

Theorem (13.7) "Tail is different": $\forall\, \mathsf{xs} : \mathsf{Seq}\ A \bullet \forall\, x : A \bullet x \triangleleft \mathsf{xs} \neq \mathsf{xs}$

Proof:
- **Using** "Induction over sequences":
  - **Subproof for** `$\forall\, x : A \bullet x \triangleleft \epsilon \neq \epsilon$`:
    - **For any** `$x : A$`:
      - **By** "Cons is not empty"
  - **Subproof for** `$\forall\, \mathsf{xs} : \mathsf{Seq}\ A \mid (\forall\, x : A \bullet x \triangleleft \mathsf{xs} \neq \mathsf{xs}) \bullet (\forall\, z : A \bullet (\forall\, x : A \bullet x \triangleleft z \triangleleft \mathsf{xs} \neq z \triangleleft \mathsf{xs}))$`:
    - **For any** `$\mathsf{xs} : \mathsf{Seq}\ A$` **satisfying** "Ind. Hyp." `$(\forall\, x : A \bullet x \triangleleft \mathsf{xs} \neq \mathsf{xs})$`:
      - **For any** `$z : A$`, `$x : A$`:
        - $x \triangleleft z \triangleleft \mathsf{xs} \neq z \triangleleft \mathsf{xs}$
        - $\equiv$ ⟨ "Definition of $\neq$", "Injectivity of $\triangleleft$" ⟩
        - $\neg (x = z \wedge z \triangleleft \mathsf{xs} = \mathsf{xs})$
        - $\Leftarrow$ ⟨ "De Morgan", "Weakening", "Definition of $\neq$" ⟩
        - $z \triangleleft \mathsf{xs} \neq \mathsf{xs}$
        - $\equiv$ ⟨ Assumption "Ind. Hyp." ⟩
        - true

Proving "Tail is different" Using the Induction Principle — Less Verbose

Theorem "Induction over sequences":

$P[xs := \epsilon]$
$\Rightarrow (\forall \mathsf{xs} : \mathsf{Seq}\ A \mid P \bullet (\forall x : A \bullet P[xs := x \triangleleft xs]))$
$\Rightarrow (\forall \mathsf{xs} : \mathsf{Seq}\ A \bullet P)$

Theorem (13.7) "Tail is different": $\forall \mathsf{xs} : \mathsf{Seq}\ A \bullet \forall x : A \bullet x \triangleleft \mathsf{xs} \neq \mathsf{xs}$

Proof:

Using "Induction over sequences":

Subproof for `$\forall x : A \bullet x \triangleleft \epsilon \neq \epsilon$`:

For any `$x : A$`:

By "Cons is not empty"

Subproof:

For any `$\mathsf{xs} : \mathsf{Seq}\ A$` **satisfying** "Ind. Hyp." `$(\forall x : A \bullet x \triangleleft \mathsf{xs} \neq \mathsf{xs})$`:

For any `$z : A$`, `$x : A$`:

$x \triangleleft z \triangleleft \mathsf{xs} \neq z \triangleleft \mathsf{xs}$
$\equiv$ ⟨ "Definition of ≠", "Injectivity of ◃" ⟩
$\neg (x = z \wedge z \triangleleft \mathsf{xs} = \mathsf{xs})$
$\Leftarrow$⟨ "De Morgan", "Weakening", "Definition of ≠" ⟩
$z \triangleleft \mathsf{xs} \neq \mathsf{xs}$
$\equiv$ ⟨ Assumption "Ind. Hyp." ⟩
true

Structural Induction

Structural induction is mathematical induction over, *e.g.*,

- **finite sequences** with the strict suffix relation
- **expressions** with the direct constituent relation
- **propositional formulae** with the strict subformula relation
- **trees** with the appropriate strict subtree relation
- **proofs** with appropriate strict sub-proof relation
- **programs** with appropriate strict sub-program relation
- ...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-11

Trees, with

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

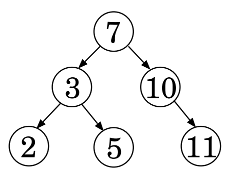
2024-10-11

Part 1: Inductive Datastructures: Trees

Inductively-defined Tree Data Structures — with Haskell Definitions

Binary (search) trees

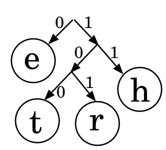
```
data BTree = EmptyB
  | Branch BTree Int BTree
```




```
bt1left = Branch
  (Branch EmptyB 2 EmptyB)
  3
  (Branch EmptyB 5 EmptyB)
bt1right = Branch
  EmptyB
  10
  (Branch EmptyB 11 EmptyB)
```

Huffman trees

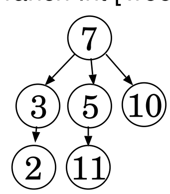
```
data HTree = Leaf Char
  | HBranch HTree HTree
```




```
hTree1 = HBranch (Leaf 'e')
  (HBranch
    (HBranch (Leaf 't') (Leaf 'r'))
    (Leaf 'h'))
decode hTree1 "100110" = "the"
```

Arbitrarily branching

```
data Tree
  = Branch Int [Tree]
```




```
t1left = Branch 7
  [Branch 3 [Branch 2 []]
  ,Branch 5 [Branch 11 []]
  ,Branch 10 []
  ]
```

Trees are Everywhere!

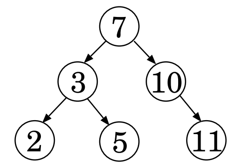
- Search trees, dictionary datastructures — BinTree, balanced trees
- Huffman trees — used for compression encoding e.g. in JPEG
- Abstract Syntax Trees (ASTs) — central datastructures in compilers
  — *Recall:* For expressions, we write strings, but we think trees...
- ...
- Every "data" in Haskell defines a (possibly degenerated) tree datastructure

Binary Trees

Declaration: △ : Tree A
Declaration: _◿_◺_ : Tree A → A → Tree A → Tree A

Declaration: t1 : Tree ℕ
Axiom "Definition of \`t1\`":

```
t1 = ((△ ◿ 2 ◺ △) ◿ 3 ◺ (△ ◿ 5 ◺ △))
        ◿ 7 ◺
        (△ ◿ 10 ◺ (△ ◿ 11 ◺ △))
```



Declaration: ⌜_⌟ : A → Tree A
Axiom "Singleton tree":

```
⌜ x ⌟ = △ ◿ x ◺ △
```

Fact "Alternative definition of \`t1\`":

```
t1 = (⌜ 2 ⌟ ◿ 3 ◺ ⌜ 5 ⌟)
        ◿ 7 ◺
        (△ ◿ 10 ◺ ⌜ 11 ⌟)
```

Axiom "Tree induction":

```
      P[t ≔ △]
  ∧  ( ∀ l, r : Tree A; x : A
         • P[t ≔ l] ∧ P[t ≔ r]  ⇒  P[t ≔ l ◿ x ◺ r]
      )
  ⇒  (∀ t : Tree A  •  P)
```

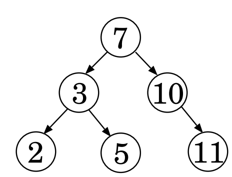
Using the Induction Principle for Binary Trees

```
Theorem "Self-inverse of tree mirror": ∀ t : Tree A • (t ˘) ˘ = t
Proof:
  Using "Tree induction":
    Subproof for `△ ˘ ˘ = △`: By "Mirror"
    Subproof for `∀ l, r : Tree A; x : A
         • (l ˘) ˘ = l ∧ (r ˘) ˘ = r
         ⇒ (l ◿ x ◺ r)˘ ˘ = (l ◿ x ◺ r)`:
       For any `l, r, x`:
         Assuming "IHL" `(l ˘) ˘ = l`,
                  "IHR" `(r ˘) ˘ = r`:
             (l ◿ x ◺ r) ˘ ˘
           =⟨ "Mirror" ⟩
             (l ˘ ˘) ◿ x ◺ (r ˘ ˘)
           =⟨ Assumptions "IHL" and "IHR" ⟩
             l ◿ x ◺ r
```

```
Axiom "Tree induction":
     P[t ≔ △]
  ∧  ( ∀ l, r : Tree A; x : A
       • P[t ≔ l] ∧ P[t ≔ r]  ⇒  P[t ≔ l ◿ x ◺ r]
     )
  ⇒  (∀ t : Tree A • P)
```

Induction Principle for Binary Trees

```
Declaration:          △   : Tree A
Declaration:        _◿_◺_ : Tree A → A → Tree A → Tree A

Fact "Alternative definition of `t1`":
  t1 = (⌜ 2 ⌟ ◿ 3 ◺ ⌜ 5 ⌟)
       ◿ 7 ◺
       (△ ◿ 10 ◺ ⌜ 11 ⌟)
```




```
Declaration: _≺_ : Tree A → Tree A → 𝔹
Axiom "HTree ≺":
     (t ≺ △              ≡  false)
  ∧  (t ≺ (l ◿ x ◺ r)    ≡  t = l   ∨   t = r)
```

Theorem (12.19) **Mathematical induction over $(T, \prec)$, if $\prec$ is well-founded**

$$(\forall x \bullet P\,x) \quad \equiv \quad (\forall x \bullet (\forall y \mid y \prec x \bullet P\,y) \Rightarrow P\,x)$$

Equivalently:

```
Axiom "Tree induction":
     P[t ≔ △]
  ∧  ( ∀ l, r : Tree A; x : A
       • P[t ≔ l] ∧ P[t ≔ r]  ⇒  P[t ≔ l ◿ x ◺ r]
     )
  ⇒  (∀ t : Tree A • P)
```

Structural Induction — Remember!

Theorem (12.19) **Mathematical induction over** $(T, \prec)$**, if** $\prec$ **is well-founded**

$$(\forall x \bullet P\, x) \quad \equiv \quad (\forall x \bullet (\forall y \mid y \prec x \bullet P\, y) \Rightarrow P\, x)$$

Structural induction is mathematical induction over, *e.g.*,

- **finite sequences** with the strict suffix relation
- **expressions** with the direct constituent relation
- **propositional formulae** with the strict subformula relation
- **trees** with the appropriate strict subtree relation
- **proofs** with appropriate strict sub-proof relation
- **programs** with appropriate strict sub-program relation
- ...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-11

Part 2: with$_2$ and with$_3$

`with` — Overview

CalcCheck currently knows three kinds of "`with`":

- "with$_1$": For explicit substitutions: **"Identity of +"** with '$x := 2$'
- *ThmA* `with` *ThmB* and *ThmB*$_2$ ...
  - "with$_2$": If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$:
    Perform **conditional rewriting**, rigidly applying $L\sigma \mapsto R\sigma$
    if using *ThmB* and *ThmB*$_2$ ... to prove $A_1\sigma, A_2\sigma, \ldots$ succeeds

Using hi_1: sp_1 sp_2	is essentially syntactic sugar for:	By hi_1 with sp_1 and sp_2

- "with$_3$": *ThmA* `with` *ThmB*
  - If *ThmB* gives rise to an equality/equivalence $L = R$:
    Rewrite *ThmA* with $L \mapsto R$ to *ThmA*$'$,
    and use *ThmA*$'$ for rewriting the goal.

with$_2$: Conditional Rewriting

ThmA with *ThmB* and *ThmB*$_2$ ...

- If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$, where $\mathsf{FVar}(L) = \mathsf{FVar}(A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R))$:
  - Find substitution σ such that $L\sigma$ matches goal
  - Resolve $A_1\sigma, A_2\sigma, \ldots$ using *ThmB* and *ThmB*$_2$ ...
  - Rewrite goal applying $L\sigma \mapsto R\sigma$ rigidly.
- E.g.: "Cancellation of $\cdot$" with Assumption '$m + n \neq 0$'

  when trying to prove $(m + n) \cdot (n + 2) = (m + n) \cdot 5 \cdot k$:
  - "Cancellation of $\cdot$" is: $c \neq 0 \Rightarrow (c \cdot a = c \cdot b \equiv a = b)$
  - We try to use: $c \cdot a = c \cdot b \mapsto a = b$, so L is $c \cdot a = c \cdot b$
  - Matching L against goal produces $\sigma = [a, b, c := (n + 2), (5 \cdot k), (m + n)]$
  - $(c \neq 0)\sigma$ is $(m + n) \neq 0$ and can be proven by "Assumption '$m + n \neq 0$'"
  - The goal is rewritten to $(a = b)\sigma$, that is, $(n + 2) = 5 \cdot k$.

Limitations of Conditional Rewriting Implementation of with$_2$

- If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$:
  - Find substitution σ such that $L\sigma$ matches goal
  - Resolve $A_1\sigma, A_2\sigma, \ldots$ using *ThmB* and *ThmB*$_2$ ...
  - Rewrite goal applying $L\sigma \mapsto R\sigma$ rigidly.

ThmA with *ThmB* and *ThmB*$_2$ ...

- E.g.: "Transitivity of $\subseteq$" with Assumptions \`$Q \cap S \subseteq Q$\` and \`$Q \subseteq R$\` when trying to prove \`$Q \cap S \subseteq R$\`
  - "Transitivity of $\subseteq$" is: $Q \subseteq R \Rightarrow R \subseteq S \Rightarrow Q \subseteq S$
  - For application, a **fresh renaming** is used: $q \subseteq r \Rightarrow r \subseteq s \Rightarrow q \subseteq s$
  - We try to use: $q \subseteq s \mapsto \mathit{true}$, so L is: $q \subseteq s$
  - Matching L against goal produces $\sigma = [q, s := Q \cap S, R]$
  - $(q \subseteq r)\sigma$ is $(Q \cap S \subseteq r)$, and $(r \subseteq s)\sigma$ is $r \subseteq R$

    — **which cannot be proven** by "Assumption '$Q \cap S \subseteq Q$'"
    resp. by "Assumption '$Q \subseteq R$'"
  - *Narrowing* or *unification* would be needed for such cases

    — **not yet implemented**
  - Adding an explicit substitution should help:

    "Transitivity of $\subseteq$" with \`$R := Q$\` and assumption \`$Q \cap S \subseteq Q$\` and assumption \`$Q \subseteq R$\`

with$_3$: Rewriting Theorems before Rewriting

ThmA with *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$:
  Rewrite *ThmA* with $L \mapsto R$
- E.g.: Assumption \`$p \Rightarrow q$\` with (3.60) \`$p \Rightarrow q \;\equiv\; p \wedge q \equiv q$\`

  The local theorem $p \Rightarrow q$ (resulting from the Assumption)

  rewrites via: $p \Rightarrow q \;\mapsto\; p \equiv p \wedge q$ (from (3.60))

  to: $p \;\equiv\; p \wedge q$

  which can be used for the rewrite: $p \;\mapsto\; p \wedge q$

Theorem (4.3) "Left-monotonicity of $\wedge$": $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
Proof:
Assuming \`$p \Rightarrow q$\`:

$p \wedge r$
$\equiv\langle$ Assumption \`$p \Rightarrow q$\` with "Definition of $\Rightarrow$ via $\wedge$" $\rangle$
$p \wedge q \wedge r$
$\Rightarrow\langle$ "Weakening" $\rangle$
$q \wedge r$

`with`$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$:
  Rewrite *ThmA* with $L \mapsto R$
- E.g.: "Instantiation" with (3.60)

"Instantiation" $`(\forall x \bullet P) \Rightarrow P[x := E]`$ rewrites via (3.60) $`q \Rightarrow r \;\mapsto\; q \equiv q \wedge r`$

to: $(\forall x \bullet P) \equiv (\forall x \bullet P) \wedge P[x := E]$

which can be used as: $(\forall x \bullet P) \mapsto (\forall x \bullet P) \wedge P[x := E]$

H11:

$(\forall x : \mathbb{Z} \bullet 5 < f\, x)$
$\equiv \langle$ "Instantiation" with "Definition of $\Rightarrow$ via $\wedge$" (3.60) $\rangle$ ▪▪▪▪▪▪with$_3$
$(\forall x : \mathbb{Z} \bullet 5 < f\, x) \wedge (5 < f\, x)[x := 9]$
$\Rightarrow \langle$ "Monotonicity of $\wedge$" with "Instantiation" $\rangle$ ▪▪▪▪▪▪with$_2$
$(5 < f\, x)[x := 8] \wedge (5 < f\, x)[x := 9]$

How can you simplify if you know $P_1 \Rightarrow P_2$?

$\vdots$
$\equiv \langle \ldots \rangle$
$\ldots \vee P_1 \vee P_2 \vee \ldots$
$\equiv \langle$? $\rangle$
?

$\vdots$
$\equiv \langle \ldots \rangle$
$\ldots \wedge P_1 \wedge P_2 \wedge \ldots$
$\equiv \langle$? $\rangle$
?

$\vdots$
$\equiv \langle \ldots \rangle$
$\ldots \vee P_1 \vee P_2 \vee \ldots$
$\equiv \langle$ "Reason for $P_1 \Rightarrow P_2$"
with "Def. of $\Rightarrow$ via $\vee$" $\rangle$
$\ldots \vee P_2 \vee \ldots$

$\vdots$
$\equiv \langle \ldots \rangle$
$\ldots \wedge P_1 \wedge P_2 \wedge \ldots$
$\equiv \langle$ "Reason for $P_1 \Rightarrow P_2$"
with "Def. of $\Rightarrow$ via $\wedge$" $\rangle$
$\ldots \wedge P_1 \wedge \ldots$

How can you simplify if you know $S_1 \subseteq S_2$?

$\vdots$
$= \langle \ldots \rangle$
$\ldots \cup S_1 \cup S_2 \cup \ldots$
$= \langle$? $\rangle$
?

$\vdots$
$= \langle \ldots \rangle$
$\ldots \cap S_1 \cap S_2 \cap \ldots$
$= \langle$? $\rangle$
?

$\longrightarrow$ Set Theory:

- "Set inclusion via $\cup$" $S \subseteq T \equiv S \cup T = T$
- "Set inclusion via $\cap$" $S \subseteq T \equiv S \cap T = S$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-22

Relations in Set Theory

Predicates and Tuple Types — Relations are Tuple Sets — Think Database Tables!

$\_\mathsf{called}\_ \; : \; P \to P \to \mathbb{B}$

$(\mathsf{uncurry}\ \_\mathsf{called}\_) \; : \; \langle P, P \rangle \to \mathbb{B}$ is the **characteristic function** of the set

$R_{\mathsf{called}} \; : \; \mathbf{set}\ \langle P, P \rangle$

$R_{\mathsf{called}} \; = \; \{p, q : P \mid p\ \mathsf{called}\ q \bullet \langle p, q \rangle\}$

R_{called} is a **(binary) relation**.

$D \; : \; P \to City \to City \to \mathbb{B}$

$D\ p\ a\ b \; \equiv \; \boxed{p \text{ drove from } a \text{ to } b}$

$R_D \; : \; \mathbf{set}\ \langle P, City, City \rangle$

$R_D \; = \; \{p : P; a, b : City \mid D\ p\ a\ b \bullet \langle p, a, b \rangle\}$

R_D is a **(ternary) relation**.

Relations

- LADM: A **(n-ary) relation** on $B_1 \times \dots \times B_n$ is a subset of $B_1 \times \dots \times B_n$
  — where $B_1, \dots, B_n$ are sets — avoiding to mention types...
- CalcCheck: Normally: A **relation** on $\langle t_1, \dots, t_n \rangle$ is a subset of $\llcorner \langle t_1, \dots, t_n \rangle \lrcorner$,
  that is, an item of type $\mathbf{set}\ \langle t_1, \dots, t_n \rangle$
  — where $t_1, \dots, t_n$ are types
- A relation on the tuple type $\langle t_1, \dots, t_n \rangle$ is an **n-ary relation**.

  "Tables" in relational databases are n-ary relations.
- A relation on the pair type $\langle t_1, t_2 \rangle$ is a **binary relation**.
- The **type** of binary relations on $\langle t_1, t_2 \rangle$ is also written $t_1 \leftrightarrow t_2$, with

  $t_1 \leftrightarrow t_2 \; = \; \mathbf{set}\ \langle t_1, t_2 \rangle$ — `\rel`
- The **set** of binary relations on the Cartesian product $B \times C$ will be written $B \leftrightarrow C$, with

  $B \leftrightarrow C \; = \; \mathbb{P}(B \times C)$ — `\Rel`

What is a Relation?

A **relation**
is a subset
of a Cartesian product.

What is a Binary Relation?

A **binary relation**
is a set of pairs.

Relations are Everywhere in Specification and Reasoning in CS

- Operations are easily defined and understood via set theory
- These operations satisfy many algebraic properties
- **Formalisation using relation-algebraic operations needs no quantifiers**
- **Similar to** how matrix operations do away with quantifications and indexed variables a_{ij} in **linear algebra**
- Like linear algebra, **relation algebra**
  - raises the level of abstraction
  - makes reasoning easier by reducing necessity for quantification
- Starting with lots of quantification over elements,
  while **proving properties via set theory**.
- Moving towards **abstract relation algebra**
  (avoiding any mention of and quantification over elements)

(Graphs), Simple Graphs

A **graph** consists of:

- a set of "nodes" or "vertices"
- a set of "edges" or "arrows"
- "incidence" information specifying how edges connect nodes

— *more details another day.*

A **simple graph** consists of:

- a set of "nodes", and
- a set of "edges", which **are** pairs of nodes.

(A simple graph has no "parallel edges".)

Formally: A **simple graph** $\langle N, E\rangle$ is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$,
  the element pairs of which are called "edges".

Simple Graphs

A **simple graph** consists of:

- a set of "nodes", and
- a set of "edges", which **are** pairs of nodes.

(A simple graph has no "parallel edges".)

Formally: A **simple graph** $\langle N, E\rangle$ is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$,
  the element pairs of which are called "edges".

Even more formally: A **simple graph** $\langle N, E\rangle$ is a pair consisting of

- a set N, and
- a relation E with $E \in N \leftrightarrow N$.

Given a simple graph $\langle N, E\rangle$, the elements of N are called "nodes" and the elements of E are called "edges".

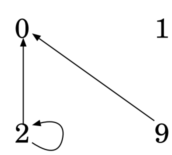
Simple Graphs: Example

Formally: A **simple graph** $\langle N, E\rangle$ is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$, the element pairs of which are called "edges".

Example: $G_1 = \langle\{2,0,1,9\}, \{\langle 2,0\rangle, \langle 9,0\rangle, \langle 2,2\rangle\}\rangle$

Graphs are normally visualised via **graph drawings**:



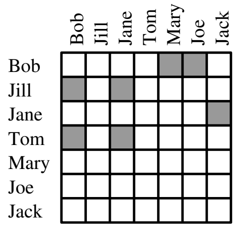
Simple graphs are essentially just relations!

Reasoning with relations is reasoning about graphs!

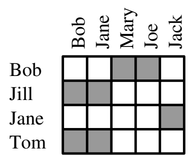
Visualising Binary Relations

$\llcorner$ *Person* $\lrcorner = \{Bob, Jill, Jane, Tom, Mary, Joe, Jack\}$

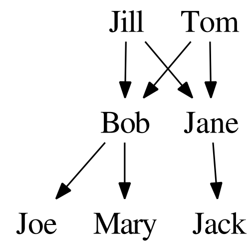
$parentOf = \{\langle Jill, Bob\rangle, \langle Jill, Jane\rangle, \langle Tom, Bob\rangle, \langle Tom, Jane\rangle,$
$\langle Bob, Mary\rangle, \langle Bob, Joe\rangle, \langle Jane, Jack\rangle\}$



$parentOf : Person \leftrightarrow Person$



$parentOf \in (parents \leftrightarrow\!\!\!\!\leftrightarrow children)$



$parents = \mathrm{Dom}\ parentOf = \{Bob, Jill, Jane, Tom\}$

$children = \mathrm{Ran}\ parentOf = \{Bob, Jane, Mary, Joe, Jack\}$

Expressing relationship: $\langle Jill, Bob\rangle \in parentOf \quad \equiv \quad Jill\ ⦗parentOf⦘\ Bob$

Notation for Relationship

Notations for "x is related via R with y":

- explicit membership notation: $\langle x, y\rangle \in R$
- ambiguous traditional infix notation: $x\,R\,y$
- CALCCHECK: $x\,⦗R⦘\,y$

Type "\((. . . \))" for these "tortoise shell bracket" Unicode codepoints

The operator $\_⦗\_⦘\_ : t_1 \to (t_1 \leftrightarrow t_2) \to t_2 \to \mathbb{B}$

- is conjunctional:
  $(1 = x\,⦗R⦘\,y < 5) \quad \equiv \quad (1 = x) \wedge (x\,⦗R⦘y) \wedge (y < 5)$
- and calculational:

$$\begin{array}{ll} & x \\ ⦗R⦘ & \langle\ \text{Reason why } x\,⦗R⦘y\ \rangle \\ & y \end{array}$$

Experimental Key Bindings

— US keyboard only! Firefox only?

- `Alt-=` for ≡ in addition to `\==`
- `Alt-<` for ⟨ in addition to `\<`
- `Alt->` for ⟩ in addition to `\>`
- `Alt-(` for ⦗ in addition to `\((`
- `Alt-)` for ⦘ in addition to `\))`

Set Operations Used as Operations on Binary Relations

Relation union:
$$\langle u,v\rangle \in (R \cup S) \quad\equiv\quad \langle u,v\rangle \in R \lor \langle u,v\rangle \in S$$
$$u \llparenthesis R \cup S \rrparenthesis v \quad\equiv\quad u \llparenthesis R \rrparenthesis v \lor u \llparenthesis S \rrparenthesis v$$

Relation intersection:
$$u \llparenthesis R \cap S \rrparenthesis v \quad\equiv\quad u \llparenthesis R \rrparenthesis v \land u \llparenthesis S \rrparenthesis v$$

Relation difference:
$$u \llparenthesis R - S \rrparenthesis v \quad\equiv\quad u \llparenthesis R \rrparenthesis v \land \neg(u \llparenthesis S \rrparenthesis v)$$

Relation complement:
$$u \llparenthesis \sim R \rrparenthesis v \quad\equiv\quad \neg\,(u \llparenthesis R \rrparenthesis v)$$

<u>Relation extensionality</u>:
$$R = S \quad\equiv\quad (\forall\, x \bullet \forall\, y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y)$$
$$R = S \quad\equiv\quad (\forall\, x,\, y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y)$$

<u>Relation inclusion</u>:
$$R \subseteq S \quad\equiv\quad (\forall\, x \bullet \forall\, y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y)$$
$$R \subseteq S \quad\equiv\quad (\forall\, x \bullet \forall\, y \mid x \llparenthesis R \rrparenthesis y \bullet x \llparenthesis S \rrparenthesis y)$$
$$R \subseteq S \quad\equiv\quad (\forall\, x,\, y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y)$$
$$R \subseteq S \quad\equiv\quad (\forall\, x,\, y \mid x \llparenthesis R \rrparenthesis y \bullet x \llparenthesis S \rrparenthesis y)$$

Empty and Universal Binary Relations

- The **empty relation** on $\langle t_1, t_2\rangle$ is $\{\} : t_1 \leftrightarrow t_2$
$$x \llparenthesis \{\} \rrparenthesis y \quad\equiv\quad \mathit{false}$$
$$\langle x,y\rangle \in \{\} \quad\equiv\quad \mathit{false}$$

- The **universal relation** on $\langle t_1, t_2\rangle$ is $\llcorner \langle t_1, t_2\rangle \lrcorner : t_1 \leftrightarrow t_2$ or $\mathbf{U} : t_1 \leftrightarrow t_2$
$$x \llparenthesis \llcorner \langle t_1, t_2\rangle \lrcorner \rrparenthesis y \quad\equiv\quad \mathit{true} \qquad\qquad x \llparenthesis \mathbf{U} \rrparenthesis y \quad\equiv\quad \mathit{true}$$
$$\langle x,y\rangle \in \llcorner \langle t_1, t_2\rangle \lrcorner \quad\equiv\quad \mathit{true} \qquad\qquad \langle x,y\rangle \in \mathbf{U} \quad\equiv\quad \mathit{true}$$

- The **universal relation on $B \times C$** is $B \times C$
$$x \llparenthesis B \times C \rrparenthesis y \quad\equiv\quad x \in B \land y \in C$$
(14.4)
$$\langle x,y\rangle \in B \times C \quad\equiv\quad x \in B \land y \in C$$

Relation-Algebraic Operations: Operations <u>on</u> Relations

- Set operations $\sim$, $\cup$, $\cap$, $-$, $\Rightarrow$ are all available.
- If $R : B \leftrightarrow C$, $\qquad B \xrightarrow{R} C$
  then its **converse** $R^{\smile} : C \leftrightarrow B$
  (in the textbook called "inverse" and written: R^{-1})
  stands for "going R backwards":
$$c \llparenthesis R^{\smile} \rrparenthesis b \quad\equiv\quad b \llparenthesis R \rrparenthesis c$$
- If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, $\qquad B \xrightarrow{R} C \xrightarrow{S} D$
  then their **composition** $R \mathbin{;} S$
  (in the textbook written: $R \circ S$)
  is a relation in $B \leftrightarrow D$, and stands for
  "going first a step via R, and then a step via S":
$$b \llparenthesis R \mathbin{;} S \rrparenthesis d \quad\equiv\quad (\exists c : C \bullet b \llparenthesis R \rrparenthesis c \llparenthesis S \rrparenthesis d)$$

The resulting **relation algebra**

- allows concise formalisations **without quantifications**,
- enables simple calculational proofs.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-24

Relations in Set Theory (ctd.)

What is a Binary Relation?

A **binary relation**
is a set of pairs.

Binary Relation Types Contain Subsets of Cartesian Products

- The **type** of binary relations between types t_1 and t_2:

  $t_1 \leftrightarrow t_2 \quad = \quad \mathbf{set}\ \langle t_1, t_2 \rangle$ — `\rel`

- The **set** of binary relations between sets B and C:

  $B \leftrightarrow\!\!\!\circ\; C \quad = \quad \mathbb{P}\,(B \times C)$ — `\Rel`

Note that for a type t, the universal set

$\mathbf{U} : \mathbf{set}\ t$

is the set of all members of t.

Or, ($\mathbf{U} : \mathbf{set}\ t$) is "type t as a set".

We **abbreviate:** $\llcorner\ t\ \lrcorner := (\mathbf{U} : \mathbf{set}\ t)$,
(`\llcorner` ... `\lrcorner`) and have:

$S \in \llcorner\ \mathbf{set}\ t\ \lrcorner \quad \equiv \quad S \subseteq \llcorner\ t\ \lrcorner$

"Universe of sets": $\llcorner\ \mathbf{set}\ t\ \lrcorner \quad = \quad \mathbb{P}\,\llcorner\ t\ \lrcorner$

Theorem "Universe of relations":
$\llcorner\ t_1 \leftrightarrow t_2\ \lrcorner \ = \ \llcorner\ t_1\ \lrcorner \leftrightarrow\!\!\!\circ\; \llcorner\ t_2\ \lrcorner$

Proof:
Using "Set extensionality":
For any `$R : t_1 \leftrightarrow t_2$`:

$R \in \llcorner\ t_1 \leftrightarrow t_2\ \lrcorner$
$\equiv \langle$ "Definition of $\leftrightarrow$" $\rangle$
$R \in \llcorner\ \mathsf{set}\ \langle t_1, t_2 \rangle\ \lrcorner$
$\equiv \langle$ "Universe of sets" $\rangle$
$R \in \mathbb{P}\,\llcorner\ \langle t_1, t_2 \rangle\ \lrcorner$
$\equiv \langle$ "Universe of pairs" $\rangle$
$R \in \mathbb{P}\,(\llcorner\ t_1\ \lrcorner \times \llcorner\ t_2\ \lrcorner)$
$\equiv \langle$ "Definition of $\leftrightarrow\!\!\!\circ$" $\rangle$
$R \in \llcorner\ t_1\ \lrcorner \leftrightarrow\!\!\!\circ\; \llcorner\ t_2\ \lrcorner$

Domain and Range of Binary Relations

For $R : t_1 \leftrightarrow t_2$, we define $Dom\ R\ :\ \mathbf{set}\ t_1$ and $Ran\ R\ :\ \mathbf{set}\ t_2$ as follows:

(14.16) $Dom\ R = \{x : t_1 \mid (\exists\, y : t_2 \bullet x\ (\!| R |\!)\ y)\} = \{p \mid p \in R \bullet \mathsf{fst}\ p\} = \mathrm{map}_{\mathbf{set}}\ fst\ R$

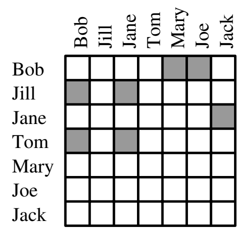
(14.17) $Ran\ R = \{y : t_2 \mid (\exists\, x : t_1 \bullet x\ (\!| R |\!)\ y)\} = \{p \mid p \in R \bullet \mathsf{snd}\ p\} = \mathrm{map}_{\mathbf{set}}\ snd\ R$

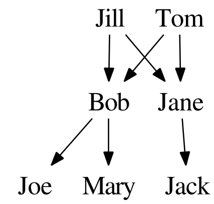
"Membership in `*Dom*`":

$x \in Dom\ R \ \equiv\ (\exists\, y : t_2 \bullet x\ (\!| R |\!)\ y)$

"Membership in `*Ran*`":

$y \in Ran\ R \ \equiv\ (\exists\, x : t_1 \bullet x\ (\!| R |\!)\ y)$





$$
\begin{array}{lclcl}
parents & = & Dom\ parentOf & = & \{Bob, Jill, Jane, Tom\} \\
children & = & Ran\ parentOf & = & \{Bob, Jane, Mary, Joe, Jack\}
\end{array}
$$

Formalise Without Quantifiers!

$$
\begin{array}{lcl}
P & = & \text{type of persons} \\
C & : & P \leftrightarrow P \\
p\ (\!| C |\!)\ q & \equiv & p \text{ called } q
\end{array}
$$

Remember: For $R : t_1 \leftrightarrow t_2$:

"Membership in `*Dom*`":

$x \in Dom\ R \ \equiv\ (\exists\, y : t_2 \bullet x (\!| R |\!) y)$

"Membership in `*Ran*`":

$y \in Ran\ R \ \equiv\ (\exists\, x : t_1 \bullet x (\!| R |\!) y)$

1. Helen called somebody.

   $Helen \in Dom\ C \qquad \equiv \qquad (\exists\, y : P \bullet Helen\ (\!| C |\!)\ y)$

2. For everybody, there is somebody they haven't called.

   $Dom\ (\sim C) = \llcorner\, P\, \lrcorner$

   $Dom\ (\sim C) = \mathbf{U}$

Relation-Algebraic Operations: Operations on Relations

- Set operations $\sim$, $\cup$, $\cap$, $-$, $\Rightarrow$ are all available.
- If $R : B \leftrightarrow C$, $\quad B \xrightarrow{R} C$
  then its **converse** $R^{\smile} : C \leftrightarrow B$
  (in the textbook called "inverse" and written: R^{-1})
  stands for "going R backwards": $\quad c\ (\!| R^{\smile} |\!)\ b \ \equiv\ b\ (\!| R |\!)\ c$
- If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, $\quad B \xrightarrow{R} C \xrightarrow{S} D$
  then their **composition** $R ; S$
  (in the textbook written: $R \circ S$)
  is a relation in $B \leftrightarrow D$, and stands for
  "going first a step via R, and then a step via S":

  $b (\!| R ; S |\!) d \ \equiv\ (\exists c : C \bullet b (\!| R |\!) c (\!| S |\!) d)$

The resulting **relation algebra**

- allows concise formalisations **without quantifications**,
- enables simple calculational proofs.

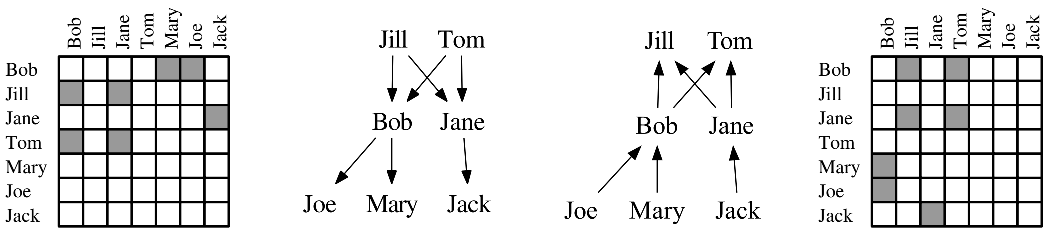
Operations on Relations: Converse

- If $R : B \leftrightarrow C$, then its **converse** $R^{\smile} : C \leftrightarrow B$ (in the textbook called "inverse" and written: R^{-1}) stands for "going R backwards":

$B \xrightarrow{R} C$

$$c \llparenthesis R^{\smile} \rrparenthesis b \quad \equiv \quad b \llparenthesis R \rrparenthesis c$$

— type "\converse" or "\u{}"



$parentOf : Person \leftrightarrow Person$ $\qquad$ $parentOf^{\smile} : Person \leftrightarrow Person$

Proving Self-inverse of Converse: $(R^{\smile})^{\smile} = R$

$(R^{\smile})^{\smile} = R$
$\equiv$ ⟨ Relation extensionality ⟩
$\forall x, y \bullet x \llparenthesis (R^{\smile})^{\smile} \rrparenthesis y \equiv x \llparenthesis R \rrparenthesis y$
$\equiv$ ⟨ … ⟩
$true$

Using "Relation extensionality":
Subproof for `$\forall x, y \bullet x \llparenthesis (R^{\smile})^{\smile} \rrparenthesis y \equiv x \llparenthesis R \rrparenthesis y$`:
For any x, y:
$x \llparenthesis (R^{\smile})^{\smile} \rrparenthesis y$
$\equiv$ ⟨ Converse ⟩
$y \llparenthesis R^{\smile} \rrparenthesis x$
$\equiv$ ⟨ Converse ⟩
$x \llparenthesis R \rrparenthesis y$

Proving Isotonicity of Converse

Proving $R \subseteq S \equiv R^{\smile} \subseteq S^{\smile}$**:**

$R^{\smile} \subseteq S^{\smile}$
$\equiv$ ⟨ Relation inclusion ⟩
$\forall y, x \mid y \llparenthesis R^{\smile} \rrparenthesis x \bullet y \llparenthesis S^{\smile} \rrparenthesis x$
$\equiv$ ⟨ Converse, dummy permutation ⟩
$\forall x, y \mid x \llparenthesis R \rrparenthesis y \bullet x \llparenthesis S \rrparenthesis y$
$\equiv$ ⟨ Relation inclusion ⟩
$R \subseteq S$

Properties of Converse $\quad B \xrightarrow{R} C$

If $R : B \leftrightarrow C$, then its **converse** $R^{\smile} : C \leftrightarrow B$ is defined by:

(14.18) $\quad \langle c, b\rangle \in R^{\smile} \quad \equiv \quad \langle b, c\rangle \in R \qquad$ (for $b : B$ and $c : C$)

(14.18) $\quad c \llparenthesis R^{\smile} \rrparenthesis b \quad \equiv \quad b \llparenthesis R \rrparenthesis c \qquad$ (for $b : B$ and $c : C$)

(14.19) **Properties of Converse:** Let $R, S : B \leftrightarrow C$ be relations.

(a) $Dom\ (R^{\smile}) = Ran\ R$

(b) $Ran\ (R^{\smile}) = Dom\ R$

(c_0) If $R \in S \leftrightarrow T$, then $Dom\ R \subseteq S$ and $Ran\ R \subseteq T$

(c) If $R \in S \leftrightarrow T$, then $R^{\smile} \in T \leftrightarrow S$

(d) $(R^{\smile})^{\smile} = R$

(d) $R \subseteq S \quad \equiv \quad R^{\smile} \subseteq S^{\smile}$

Operations on Relations: Composition $\quad B \xrightarrow{R} C \xrightarrow{S} D$

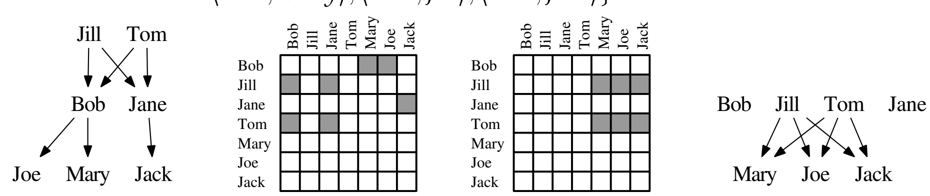
If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, then their **composition** $R \,⨾\, S : B \leftrightarrow D$ is defined by:

(14.20) $\quad b \llparenthesis R ⨾ S \rrparenthesis d \equiv (\exists c : C \bullet b \llparenthesis R \rrparenthesis c \llparenthesis S \rrparenthesis d) \qquad$ (for $b : B, d : D$)

(14.20) $\quad b \llparenthesis R ⨾ S \rrparenthesis d \equiv (\exists c : C \bullet b \llparenthesis R \rrparenthesis c \wedge c \llparenthesis S \rrparenthesis d) \qquad$ (for $b : B, d : D$)

$parentOf = \{\langle Jill, Bob\rangle, \langle Jill, Jane\rangle, \langle Tom, Bob\rangle, \langle Tom, Jane\rangle,$
$\langle Bob, Mary\rangle, \langle Bob, Joe\rangle, \langle Jane, Jack\rangle\}$

$grandparentOf = parentOf ⨾ parentOf$
$= \{\langle Jill, Mary\rangle, \langle Jill, Joe\rangle, \langle Jill, Jack\rangle$
$\langle Tom, Mary\rangle, \langle Tom, Joe\rangle, \langle Tom, Jack\rangle\}$



Sub-identity and Identity Relations

- The **(sub-)identity relation** on $B : \mathbf{set}\ t$ is $\mathrm{id}\ B : t \leftrightarrow t$

$\mathrm{id}\ B = \{x : t \mid x \in B \bullet \langle x, x\rangle\}$:

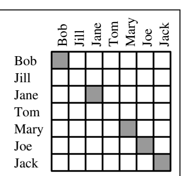
$x \llparenthesis \mathrm{id}\ B \rrparenthesis y \quad \equiv \quad x = y \in B$

$\langle x, y\rangle \in \mathrm{id}\ B \quad \equiv \quad x = y \wedge y \in B$

— LADM writes ι_B

— Writing "id B" follows the Z notation

id *children* =

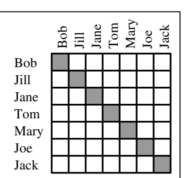


- The **identity relation** on t : Type is $\mathbb{I} : t \leftrightarrow t$ with $\mathbb{I} = \mathrm{id}\ \mathbf{U}$

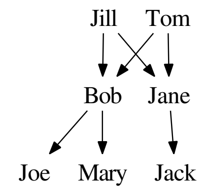
$x \llparenthesis \mathbb{I} \rrparenthesis y \quad \equiv \quad x = y$

$\langle x, y\rangle \in \mathbb{I} \quad \equiv \quad x = y$

$(\mathbb{I} : Person \leftrightarrow Person)$ =

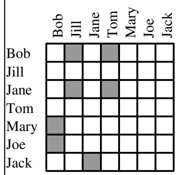


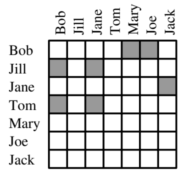
Combining Several Operations

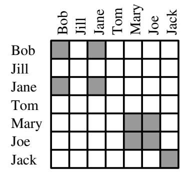


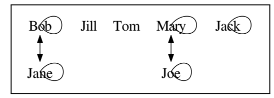
How to define siblings?

- First attempt: $\mathit{childOf} \fatsemi \mathit{parentOf}$, with $\mathit{childOf} = \mathit{parentOf}^{\smile}$

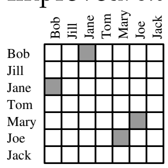


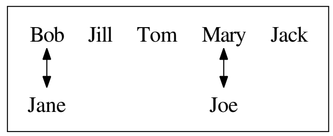


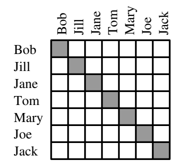




- Improved: $\mathit{sibling} = \mathit{childOf} \fatsemi \mathit{parentOf} - \mathrm{id} \llcorner \mathit{Person} \lrcorner$







P = type of persons

$C : P \leftrightarrow P$ — "called"

$B : P \leftrightarrow P$ — "brother of"

$\mathit{Aos} : P$

$\mathit{Jun} : P$

Convert into English (via predicate logic):

$\mathit{Aos} \llparenthesis C \rrparenthesis \mathit{Jun}$

$\mathit{Aos} \llparenthesis C \fatsemi B \rrparenthesis \mathit{Jun}$

$\mathit{Aos} \llparenthesis \sim (C \fatsemi \sim B) \rrparenthesis \mathit{Jun}$

$\mathit{Aos} \llparenthesis \sim (\sim C \fatsemi B) \rrparenthesis \mathit{Jun}$

$\mathit{Aos} \llparenthesis \sim ((C \cap \sim (B \fatsemi C^{\smile})) \fatsemi \sim B) \rrparenthesis \mathit{Jun}$

$(B \fatsemi (\{\mathit{Jun}\} \times \llcorner P \lrcorner)) \cap (C \fatsemi C^{\smile}) \subseteq \mathrm{id} \llcorner P \lrcorner$

Translating between Relation Algebra and Predicate Logic

$$
\begin{array}{rcl}
R = S & \equiv & (\forall x, y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y) \\
R \subseteq S & \equiv & (\forall x, y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y) \\
u \llparenthesis \{\} \rrparenthesis v & \equiv & \mathit{false} \\
u \llparenthesis \mathbf{U} \rrparenthesis v & \equiv & \mathit{true} \\
u \llparenthesis A \times B \rrparenthesis v & \equiv & u \in A \wedge v \in B \\
u \llparenthesis \sim S \rrparenthesis v & \equiv & \neg(u \llparenthesis S \rrparenthesis v) \\
u \llparenthesis S \cup T \rrparenthesis v & \equiv & u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v \\
u \llparenthesis S \cap T \rrparenthesis v & \equiv & u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v \\
u \llparenthesis S - T \rrparenthesis v & \equiv & u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v) \\
u \llparenthesis S \Rightarrow T \rrparenthesis v & \equiv & u \llparenthesis S \rrparenthesis v \Rightarrow (u \llparenthesis T \rrparenthesis v) \\
u \llparenthesis \mathbb{I} \rrparenthesis v & \equiv & u = v \\
u \llparenthesis \mathrm{id}\ A \rrparenthesis v & \equiv & u = v \in A \\
u \llparenthesis R^{\smile} \rrparenthesis v & \equiv & v \llparenthesis R \rrparenthesis u \\
u \llparenthesis R \fatsemi S \rrparenthesis v & \equiv & (\exists x \bullet u \llparenthesis R \rrparenthesis x \llparenthesis S \rrparenthesis v)
\end{array}
$$

$P \quad = \quad$ type of persons

$C \quad : \quad P \leftrightarrow P$ — "called"

$B \quad : \quad P \leftrightarrow P$ — "brother of"

$Aos \quad : \quad P$

$Jun \quad : \quad P$

Convert into English (via predicate logic):

$$\begin{aligned}
& Aos \,\llparenthesis\, C \,;\, B \,\rrparenthesis\, Jun \\
\equiv \; & \langle \text{(14.20) Relation composition} \rangle \\
& (\exists\, b \bullet Aos \,\llparenthesis\, C \,\rrparenthesis\, b \,\llparenthesis\, B \,\rrparenthesis\, Jun)
\end{aligned}$$

"Aos called some brother of Jun."

"Aos called a brother of Jun."

$$\begin{aligned}
& Aos \,\llparenthesis\, \sim (C \,;\, \sim B) \,\rrparenthesis\, Jun \\
\equiv \; & \langle \text{(11.17r) Relation complement} \rangle \\
& \neg(Aos \,\llparenthesis\, C \,;\, \sim B \,\rrparenthesis\, Jun) \\
\equiv \; & \langle \text{(14.20) Relation composition} \rangle \\
& \neg(\exists\, p \bullet Aos \,\llparenthesis\, C \,\rrparenthesis\, p \,\llparenthesis\, \sim B \,\rrparenthesis\, Jun) \\
\equiv \; & \langle \text{(11.17r) Relation complement} \rangle \\
& \neg(\exists\, p \bullet Aos \,\llparenthesis\, C \,\rrparenthesis\, p \wedge \neg(p \,\llparenthesis\, B \,\rrparenthesis\, Jun)) \\
\equiv \; & \langle \text{(9.18b) Generalised De Morgan} \rangle \\
& (\forall\, p \bullet \neg(Aos \,\llparenthesis\, C \,\rrparenthesis\, p \wedge \neg(p \,\llparenthesis\, B \,\rrparenthesis\, Jun))) \\
\equiv \; & \langle \text{(3.47) De Morgan, (3.12) Double negation} \rangle \\
& (\forall\, p \bullet \neg(Aos \,\llparenthesis\, C \,\rrparenthesis\, p) \vee p \,\llparenthesis\, B \,\rrparenthesis\, Jun) \\
\equiv \; & \langle \text{(9.3a) Trading for } \forall \rangle \\
& (\forall\, p \mid Aos \,\llparenthesis\, C \,\rrparenthesis\, p \bullet p \,\llparenthesis\, B \,\rrparenthesis\, Jun)
\end{aligned}$$

"Everybody Aos called is a brother of Jun."

"Aos called only brothers of Jun."

Formalise Without Quantifiers! (2)

$P \quad := \quad$ type of persons

$C \quad : \quad P \leftrightarrow P$

$p \,\llparenthesis\, C \,\rrparenthesis\, q \quad :\equiv \quad p$ called q

1. Helen called somebody who called her.

2. For arbitrary people x, z, if x called z, then there is sombody whom x called, and who was called by somebody who also called z.

3. For arbitrary people x, y, z, if x called y, and y was called by somebody who also called z, then x called z.

4. Obama called everybody directly, or indirectly via at most two intermediaries.

First Simple Properties of Composition

If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, then their **composition** $R \fatsemi S : B \leftrightarrow D$ is defined by:

(14.20) $b \,(R \fatsemi S)\, d \;\equiv\; (\exists c : C \bullet b \,(R)\, c \wedge c \,(S)\, d)$ (for $b : B, d : D$)

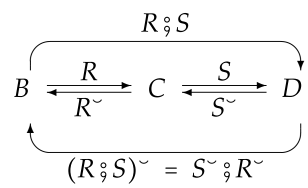
(14.22) **Associativity of ⨾:** $Q \fatsemi (R \fatsemi S) = (Q \fatsemi R) \fatsemi S$

Left- and Right-identities of ⨾: If $R \in X \leftrightarrow Y$, then: $\mathrm{id}\, X \fatsemi R = R = R \fatsemi \mathrm{id}\, Y$

We defined: $\mathbb{I} = \mathrm{id}\, \mathbf{U}$ with: **Relationship via $\mathbb{I}$:** $x \,(\mathbb{I})\, y \equiv x = y$

$\mathbb{I}$ is "the" identity of composition: **Identity of ⨾:** $\mathbb{I} \fatsemi R = R = R \fatsemi \mathbb{I}$

Contravariance: $(R \fatsemi S)^{\smile} = S^{\smile} \fatsemi R^{\smile}$



Some of the Predicate Logic Laws You Really Need To Know Now

(8.13) **Empty Range:** ...

(8.14) **One-point Rule:** Provided ..., ...

(8.15) **(Quantification) Distributivity:** ...

(8.16–18) **Range split:** ...

(9.17) **Generalised De Morgan**: ...

(9.2) **Trading for ∀:** ...

(9.19) **Trading for ∃:** ...

(9.13) **Instantiation:** ...

(9.28) **∃-Introduction**: ...

... and correctly handle substitution, Leibniz, bound variable rearrangements, monotonicity/antitonicity, For any ...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-25

Quantifier Reasoning, Some Properties of Relation Composition

Plan for Today

- Examples for the kind of quantifier reasoning required in the context of set-theoretical relations
- Some properties of relation composition, e.g., $⨾$ is monotonic is bijective”

Moving towards relation-algebraic formalisations and reasoning...

Translating between Relation Algebra and Predicate Logic

$$
\begin{array}{rcl}
R = S & \equiv & (\forall x, y \bullet x \,(\!|\, R \,|\!)\, y \equiv x \,(\!|\, S \,|\!)\, y) \\
R \subseteq S & \equiv & (\forall x, y \bullet x \,(\!|\, R \,|\!)\, y \Rightarrow x \,(\!|\, S \,|\!)\, y) \\
u \,(\!|\, \{\} \,|\!)\, v & \equiv & \mathit{false} \\
u \,(\!|\, \mathbf{U} \,|\!)\, v & \equiv & \mathit{true} \\
u \,(\!|\, A \times B \,|\!)\, v & \equiv & u \in A \wedge v \in B \\
u \,(\!|\, {\sim} S \,|\!)\, v & \equiv & \neg(u \,(\!|\, S \,|\!)\, v) \\
u \,(\!|\, S \cup T \,|\!)\, v & \equiv & u \,(\!|\, S \,|\!)\, v \vee u \,(\!|\, T \,|\!)\, v \\
u \,(\!|\, S \cap T \,|\!)\, v & \equiv & u \,(\!|\, S \,|\!)\, v \wedge u \,(\!|\, T \,|\!)\, v \\
u \,(\!|\, S - T \,|\!)\, v & \equiv & u \,(\!|\, S \,|\!)\, v \wedge \neg(u \,(\!|\, T \,|\!)\, v) \\
u \,(\!|\, S \Rightarrow T \,|\!)\, v & \equiv & u \,(\!|\, S \,|\!)\, v \Rightarrow (u \,(\!|\, T \,|\!)\, v) \\
u \,(\!|\, \mathbb{I} \,|\!)\, v & \equiv & u = v \\
u \,(\!|\, \mathrm{id}\ A \,|\!)\, v & \equiv & u = v \in A \\
u \,(\!|\, R^{\smile} \,|\!)\, v & \equiv & v \,(\!|\, R \,|\!)\, u \\
u \,(\!|\, R ⨾ S \,|\!)\, v & \equiv & (\exists x \bullet u \,(\!|\, R \,|\!)\, x \,(\!|\, S \,|\!)\, v)
\end{array}
$$

$$
\begin{array}{lcll}
P & = & \text{type of persons} & \\
C & : & P \leftrightarrow P & \text{— “called”} \\
B & : & P \leftrightarrow P & \text{— “brother of”} \\
Aos & : & P & \\
Jun & : & P &
\end{array}
$$

Convert into English (via predicate logic):

$Aos \,(\!|\, C \,|\!)\, Jun$

$Aos \,(\!|\, C ⨾ B \,|\!)\, Jun$

$Aos \,(\!|\, {\sim}(C ⨾ {\sim} B) \,|\!)\, Jun$

$Aos \,(\!|\, {\sim}({\sim} C ⨾ B) \,|\!)\, Jun$

$Aos \,(\!|\, {\sim}((C \cap {\sim}(B ⨾ C^{\smile})) ⨾ {\sim} B) \,|\!)\, Jun$

$(B ⨾ (\{Jun\} \times \mathbf{U})) \cap (C ⨾ C^{\smile}) \quad \subseteq \quad \mathbb{I}$

$Aos ⦇ {\sim} ((C \cap {\sim} (B ⨾ C^{\smile})) ⨾ {\sim} B) ⦈ Jun$

$\equiv$ ⟨ Relation complement ⟩

$\neg (Aos ⦇ (C \cap {\sim} (B ⨾ C^{\smile})) ⨾ {\sim} B ⦈ Jun)$

$\equiv$ ⟨ Relation composition ⟩

$\neg (\exists p \bullet Aos ⦇ C \cap {\sim} (B ⨾ C^{\smile}) ⦈ p ⦇ {\sim} B ⦈ Jun)$

$\equiv$ ⟨ Relation intersection ⟩

$\neg (\exists p \bullet Aos ⦇ C ⦈ p \wedge Aos ⦇ {\sim} (B ⨾ C^{\smile}) ⦈ p \wedge p ⦇ {\sim} B ⦈ Jun)$

$\equiv$ ⟨ Relation complement ⟩

$\neg (\exists p \bullet Aos ⦇ C ⦈ p \wedge \neg (Aos ⦇ B ⨾ C^{\smile} ⦈ p) \wedge \neg (p ⦇ B ⦈ Jun))$

$\equiv$ ⟨ Relation composition ⟩

$\neg (\exists p \bullet Aos ⦇ C ⦈ p \wedge \neg (\exists q \bullet Aos ⦇ B ⦈ q ⦇ C^{\smile} ⦈ p) \wedge \neg (p ⦇ B ⦈ Jun))$

$\equiv$ ⟨ (9.18b) Generalised De Morgan ⟩

...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-25

Part 1: Quantifier Reasoning Examples: H11

H11 — Domain of Union — Step 1

Theorem "Domain of union": $\mathsf{Dom}\ (R \cup S) = \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

Proof:

Using "Set extensionality":

For any \`$x$\`:

$x \in \mathsf{Dom}\ (R \cup S)$

$\equiv$ ⟨ ? ⟩

$x \in \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

H11 — Domain of Union — Step 2

Theorem "Domain of union": $\mathsf{Dom}\ (R \cup S) \ = \ \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

Proof:

Using "Set extensionality":

For any \`$x$\`:

$x \in \mathsf{Dom}\ (R \cup S)$

$\equiv$ ⟨ "Membership in \`Dom\`" ⟩

$\exists y \bullet x ⦗ R \cup S ⦘ y$

$\equiv$ ⟨ "Relation union" ⟩

$\exists y \bullet x ⦗ R ⦘ y \vee x ⦗ S ⦘ y$

$\equiv$ ⟨ ? ⟩

$(\exists y \bullet x ⦗ R ⦘ y) \vee (\exists y \bullet x ⦗ S ⦘ y)$

$\equiv$ ⟨ "Membership in \`Dom\`" ⟩

$x \in \mathsf{Dom}\ R \vee x \in \mathsf{Dom}\ S$

$\equiv$ ⟨ "Union" ⟩

$x \in \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

H11 — Domain of Union — Step 3

Theorem "Domain of union": $\mathsf{Dom}\ (R \cup S) \ = \ \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

Proof:

Using "Set extensionality":

For any \`$x$\`:

$x \in \mathsf{Dom}\ (R \cup S)$

$\equiv$ ⟨ "Membership in \`Dom\`" ⟩

$\exists y \bullet x ⦗ R \cup S ⦘ y$

$\equiv$ ⟨ "Relation union" ⟩

$\exists y \bullet x ⦗ R ⦘ y \vee x ⦗ S ⦘ y$

$\equiv$ ⟨ "Distributivity of $\exists$ over $\vee$" ⟩

$(\exists y \bullet x ⦗ R ⦘ y) \vee (\exists y \bullet x ⦗ S ⦘ y)$

$\equiv$ ⟨ "Membership in \`Dom\`" ⟩

$x \in \mathsf{Dom}\ R \vee x \in \mathsf{Dom}\ S$

$\equiv$ ⟨ "Union" ⟩

$x \in \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

H11 — Domain of $\cap$ — Step 1

Theorem "Domain of intersection": $\mathsf{Dom}\ (R \cap S) \ \subseteq \ \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

Proof:

Using "Set inclusion":

For any \`$x$\`:

$x \in \mathsf{Dom}\ (R \cap S)$

$\equiv$ ⟨ "Membership in \`Dom\`" ⟩

$\exists y \bullet x ⦗ R \cap S ⦘ y$

$\equiv$ ⟨ "Relation intersection" ⟩

$\exists y \bullet x ⦗ R ⦘ y \wedge x ⦗ S ⦘ y$

$\Rightarrow$ ⟨ ? ⟩

$(\exists y \bullet x ⦗ R ⦘ y) \wedge (\exists y \bullet x ⦗ S ⦘ y)$

$\equiv$ ⟨ "Membership in \`Dom\`" ⟩

$x \in \mathsf{Dom}\ R \wedge x \in \mathsf{Dom}\ S$

$\equiv$ ⟨ "Intersection" ⟩

$x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

H11 — Domain of ∩ — Step 2

Theorem "Domain of intersection": $\mathsf{Dom}\ (R \cap S) \subseteq \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\ (R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x ⦇ R \cap S ⦈ y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x ⦇ R ⦈ y \wedge x ⦇ S ⦈ y$
    $\equiv\langle$ "Idempotency of $\wedge$" $\rangle$
      $(\exists y \bullet x ⦇ R ⦈ y \wedge x ⦇ S ⦈ y) \wedge (\exists y \bullet x ⦇ R ⦈ y \wedge x ⦇ S ⦈ y)$

    $\Rightarrow\langle$? with "Weakening" $\rangle$

      $(\exists y \bullet x ⦇ R ⦈ y) \qquad \wedge (\exists y \bullet \qquad x ⦇ S ⦈ y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\ R \wedge x \in \mathsf{Dom}\ S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

H11 — Domain of ∩ — Step 3

Theorem "Domain of intersection": $\mathsf{Dom}\ (R \cap S) \subseteq \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\ (R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x ⦇ R \cap S ⦈ y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x ⦇ R ⦈ y \wedge x ⦇ S ⦈ y$
    $\equiv\langle$ "Idempotency of $\wedge$" $\rangle$
      $(\exists y \bullet x ⦇ R ⦈ y \wedge x ⦇ S ⦈ y) \wedge$
      $(\exists y \bullet x ⦇ R ⦈ y \wedge x ⦇ S ⦈ y)$
    $\Rightarrow\langle$ "Monotonicity of $\wedge$" with
      "Body monotonicity of $\exists$" with "Weakening" $\rangle$
      $(\exists y \bullet x ⦇ R ⦈ y) \wedge (\exists y \bullet x ⦇ S ⦈ y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\ R \wedge x \in \mathsf{Dom}\ S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

H11 — Domain of ∩ (B) — Step 1

Theorem "Domain of intersection": $\mathsf{Dom}\ (R \cap S) \subseteq \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\ (R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x ⦇ R \cap S ⦈ y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x ⦇ R ⦈ y \wedge x ⦇ S ⦈ y$

    $\Rightarrow\langle$? $\rangle$

      $(\exists y \bullet x ⦇ R ⦈ y) \wedge (\exists y \bullet x ⦇ S ⦈ y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\ R \wedge x \in \mathsf{Dom}\ S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

> **Theorem** (9.21) "Distributivity of $\wedge$ over $\exists$":
> $P \wedge (\exists x \mid R \bullet Q) \equiv (\exists x \mid R \bullet P \wedge Q)$
> provided $\neg occurs('x', 'P')$

H11 — Domain of ∩ (B) — Step 2

Theorem "Domain of intersection": $\mathsf{Dom}\ (R \cap S) \subseteq \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$
Proof:
Using "Set inclusion":
For any `$x$`:

$x \in \mathsf{Dom}\ (R \cap S)$
≡⟨ "Membership in `Dom`" ⟩
$\exists\, y \bullet x \llparenthesis R \cap S \rrparenthesis y$
≡⟨ "Relation intersection" ⟩
$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge x \llparenthesis S \rrparenthesis y$

⇒⟨ ? ⟩

$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge (\exists\, y \bullet x \llparenthesis S \rrparenthesis y)$
≡⟨ "Distributivity of ∧ over ∃" ⟩
$(\exists\, y \bullet x \llparenthesis R \rrparenthesis y) \wedge (\exists\, y \bullet x \llparenthesis S \rrparenthesis y)$
≡⟨ "Membership in `Dom`" ⟩
$x \in \mathsf{Dom}\ R \wedge x \in \mathsf{Dom}\ S$
≡⟨ "Intersection" ⟩
$x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

Theorem (9.21) "Distributivity of ∧ over ∃":
$P \wedge (\exists\, x \mid R \bullet Q) \equiv (\exists\, x \mid R \bullet P \wedge Q)$
provided $\neg occurs(`x', `P')$

H11 — Domain of ∩ (B) — Step 3

Theorem "Domain of intersection": $\mathsf{Dom}\ (R \cap S) \subseteq \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$
Proof:
Using "Set inclusion":
For any `$x$`:

$x \in \mathsf{Dom}\ (R \cap S)$
≡⟨ "Membership in `Dom`" ⟩
$\exists\, y \bullet x \llparenthesis R \cap S \rrparenthesis y$
≡⟨ "Relation intersection" ⟩
$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge x \llparenthesis S \rrparenthesis y$
≡⟨ Substitution ⟩
$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge (x \llparenthesis S \rrparenthesis y)[y := y]$
⇒⟨ ? with "∃-Introduction" ⟩
$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge (\exists\, y \bullet x \llparenthesis S \rrparenthesis y)$
≡⟨ "Distributivity of ∧ over ∃" ⟩
$(\exists\, y \bullet x \llparenthesis R \rrparenthesis y) \wedge (\exists\, y \bullet x \llparenthesis S \rrparenthesis y)$
≡⟨ "Membership in `Dom`" ⟩
$x \in \mathsf{Dom}\ R \wedge x \in \mathsf{Dom}\ S$
≡⟨ "Intersection" ⟩
$x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

H11 — Domain of ∩ (B) — Step 4

Theorem "Domain of intersection": $\mathsf{Dom}\ (R \cap S) \subseteq \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$
Proof:
Using "Set inclusion":
For any `$x$`:

$x \in \mathsf{Dom}\ (R \cap S)$
≡⟨ "Membership in `Dom`" ⟩
$\exists\, y \bullet x \llparenthesis R \cap S \rrparenthesis y$
≡⟨ "Relation intersection" ⟩
$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge x \llparenthesis S \rrparenthesis y$
≡⟨ Substitution ⟩
$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge (x \llparenthesis S \rrparenthesis y)[y := y]$
⇒⟨ "Body monotonicity of ∃" with "Monotonicity of ∧" with "∃-Introduction" ⟩
$\exists\, y \bullet x \llparenthesis R \rrparenthesis y \wedge (\exists\, y \bullet x \llparenthesis S \rrparenthesis y)$
≡⟨ "Distributivity of ∧ over ∃" ⟩
$(\exists\, y \bullet x \llparenthesis R \rrparenthesis y) \wedge (\exists\, y \bullet x \llparenthesis S \rrparenthesis y)$
≡⟨ "Membership in `Dom`" ⟩
$x \in \mathsf{Dom}\ R \wedge x \in \mathsf{Dom}\ S$
≡⟨ "Intersection" ⟩
$x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

Distributivity over ∀

(9.5) **Axiom, Distributivity of $\vee$ over $\forall$:** If $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$,

$$P \vee (\forall x \mid R \bullet Q) \equiv (\forall x \mid R \bullet P \vee Q)$$

(9.6) Provided $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$,

$$(\forall x \mid R \bullet P) \equiv P \vee (\forall x \bullet \neg R)$$

(9.7) **Distributivity of $\wedge$ over $\forall$:** If $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$,

$$\neg(\forall x \bullet \neg R) \Rightarrow (P \wedge (\forall x \mid R \bullet Q) \equiv (\forall x \mid R \bullet P \wedge Q))$$

(9.22.1) **Distributivity of $\wedge$ over $\forall$:** If $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$,

$$(\exists x \bullet R) \Rightarrow (P \wedge (\forall x \mid R \bullet Q) \equiv (\forall x \mid R \bullet P \wedge Q))$$

(9.8) $(\forall x \mid R \bullet true) \equiv true$

(9.9) $$(\forall x \mid R \bullet P \equiv Q) \Rightarrow ((\forall x \mid R \bullet P) \equiv (\forall x \mid R \bullet Q))$$

Distributivity over ∃

(9.21) **Distributivity of $\wedge$ over $\exists$:** If $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$,

$$P \wedge (\exists x \mid R \bullet Q) \equiv (\exists x \mid R \bullet P \wedge Q)$$

(9.22) Provided $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$,

$$(\exists x \mid R \bullet P) \equiv P \wedge (\exists x \bullet R)$$

(9.23) **Distributivity of $\vee$ over $\exists$:** If $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$,

$$(\exists x \bullet R) \Rightarrow ((\exists x \mid R \bullet P \vee Q) \equiv P \vee (\exists x \mid R \bullet Q))$$

(9.24) $(\exists x \mid R \bullet false) \equiv false$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-25

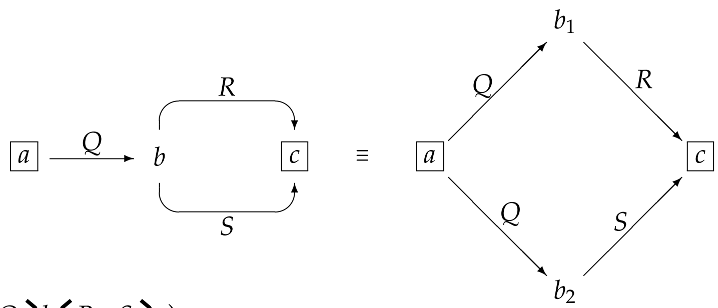
Part 2: Some Properties of Relation Composition

Distributivity of Relation Composition over Union

Composition distributes over **union** from both sides:

(14.23) $$Q ⨾ (R \cup S) \quad = \quad Q ⨾ R \cup Q ⨾ S$$
$$(P \cup Q) ⨾ R \quad = \quad P ⨾ R \cup Q ⨾ R$$

In **control flow** diagrams (NFA) — boxed variables are free; others existentially quantified; alternative paths correspond to **disjunction**:



$$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \cup S \rrparenthesis c) \quad \equiv$$
$$(\exists b_1 \bullet a \llparenthesis Q \rrparenthesis b_1 \llparenthesis R \rrparenthesis c) \quad \vee \quad (\exists b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \llparenthesis S \rrparenthesis c)$$

Proving Distributivity of Relation Composition over Union

Theorem "Distributivity of ⨾ over ∪": $Q ⨾ (R \cup S) \;=\; Q ⨾ R \;\cup\; Q ⨾ S$

Proof:

Using "Relation extensionality":

For any \`$a$\`, \`$c$\`:

$a \llparenthesis Q ⨾ (R \cup S) \rrparenthesis c$

$\equiv$ ⟨ "Relation composition" ⟩

$\exists b \bullet a \llparenthesis Q \rrparenthesis b \wedge b \llparenthesis R \cup S \rrparenthesis c$

$\equiv$ ⟨ "Relation union" ⟩

$\exists b \bullet a \llparenthesis Q \rrparenthesis b \wedge (b \llparenthesis R \rrparenthesis c \vee b \llparenthesis S \rrparenthesis c)$

$\equiv$ ⟨ ? ⟩

$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \wedge b \llparenthesis R \rrparenthesis c) \vee (\exists b \bullet a \llparenthesis Q \rrparenthesis b \wedge b \llparenthesis S \rrparenthesis c)$

$\equiv$ ⟨ "Relation composition" ⟩

$a \llparenthesis Q ⨾ R \rrparenthesis c \vee a \llparenthesis Q ⨾ S \rrparenthesis c$

$\equiv$ ⟨ "Relation union" ⟩

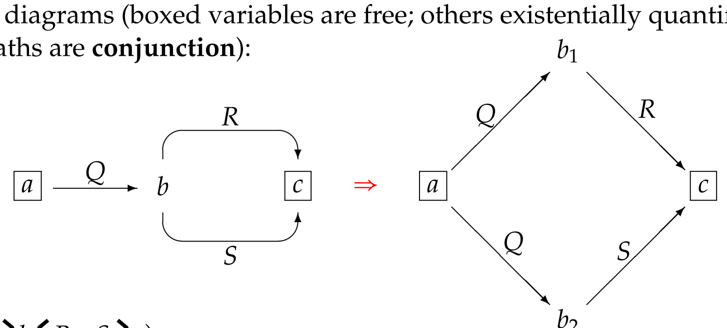
$a \llparenthesis Q ⨾ R \cup Q ⨾ S \rrparenthesis c$

Sub-Distributivity of Composition over Intersection

Composition **sub-**distributes over intersection from both sides:

(14.24) $$Q ⨾ (R \cap S) \quad \subseteq \quad Q ⨾ R \cap Q ⨾ S$$
$$(P \cap Q) ⨾ R \quad \subseteq \quad P ⨾ R \cap Q ⨾ R$$

In constraint diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \cap S \rrparenthesis c) \quad \Rightarrow$$
$$(\exists b_1 \bullet a \llparenthesis Q \rrparenthesis b_1 \llparenthesis R \rrparenthesis c) \quad \wedge \quad (\exists b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \llparenthesis S \rrparenthesis c)$$

Counterexample for $\Leftarrow$: Q := neighbour of R := brother of S := parent of

Proving Sub-Distributivity of Composition over Intersection

Theorem "Sub-distributivity of ⨾ over ∩": $Q ⨾ (R \cap S) \subseteq Q ⨾ R \cap Q ⨾ S$

Proof:

Using "Relation inclusion":

For any \`$a$\`, \`$c$\`:

$a \,⦗\, Q ⨾ (R \cap S) \,⦘\, c$

≡ ⟨ "Relation composition" ⟩

$\exists b \bullet a \,⦗\, Q \,⦘\, b \wedge b \,⦗\, R \cap S \,⦘\, c$

≡ ⟨ "Relation intersection" ⟩

$\exists b \bullet a \,⦗\, Q \,⦘\, b \wedge (b \,⦗\, R \,⦘\, c \wedge b \,⦗\, S \,⦘\, c)$

⇒⟨ ? ⟩

$(\exists b \bullet a \,⦗\, Q \,⦘\, b \wedge b \,⦗\, R \,⦘\, c) \wedge (\exists b \bullet a \,⦗\, Q \,⦘\, b \wedge b \,⦗\, S \,⦘\, c)$

≡ ⟨ "Relation composition" ⟩

$a \,⦗\, Q ⨾ R \,⦘\, c \wedge a \,⦗\, Q ⨾ S \,⦘\, c$

≡ ⟨ "Relation intersection" ⟩

$a \,⦗\, Q ⨾ R \cap Q ⨾ S \,⦘\, c$

Monotonicity of Relation Composition

Relation composition is monotonic in both arguments:

$Q \subseteq R \Rightarrow Q ⨾ S \subseteq R ⨾ S$

$Q \subseteq R \Rightarrow P ⨾ Q \subseteq P ⨾ R$

We could prove this via **"Relation inclusion"** *and* **"For any"**, *but we don't need to:*

Assume $Q \subseteq R$, which by "Definition of ⊆ via ∪" is equivalent to $Q \cup R = R$:

Proving $Q ⨾ S \subseteq R ⨾ S$**:**

$R ⨾ S$

= ⟨ Assumption $Q \cup R = R$ ⟩

$(Q \cup R) ⨾ S$

= ⟨ (14.23) Distributivity of ⨾ over ∪ ⟩

$Q ⨾ S \cup R ⨾ S$

⊇ ⟨ (11.31) Strengthening $S \subseteq S \cup T$ ⟩

$Q ⨾ S$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-29

Residuals of ⨾. Relation Properties

Plan for Today

- "Residuals": Left- and right-division with respect to $⨟$
- Some properties of homogeneous relations, e.g., "R is transitive", "E is an order"
- Some more properties of relations of arbitrary types, e.g., "R is univalent", "F is bijective"

Moving towards relation-algebraic formalisations and reasoning...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-29

Part 1: Residuals

Given: $x \leq z \quad \equiv \quad x \leq 5$

What do you know about z? Why? (Prove it!)

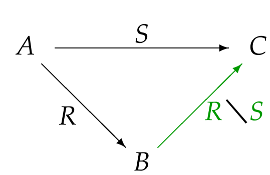
Given: $X \subseteq A \Rightarrow B \quad \equiv \quad X \cap A \subseteq B$

Calculate the **relative pseudocomplement** $A \Rightarrow B$!

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\quad X \subseteq R \backslash S \quad \equiv \quad R ⨟ X \subseteq S$

$R \backslash S$ is the largest solution $X : B \leftrightarrow C$ for $R ⨟ X \subseteq S$.

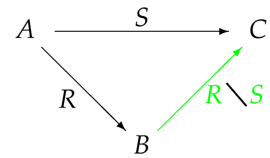
Calculate the **right residual** ("left division") $R \backslash S$!



Same idea as for "$\Rightarrow$":
Using extensionality, calculate $\quad b \,(\!|\, R \backslash S \,|\!)\, c \quad \equiv \quad b \,(\!|\, ? \,|\!)\, c$

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\qquad X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S$

Calculate the **right residual** ("left division") $R \backslash S$!



$$
\begin{array}{ll}
& b \llparenthesis R \backslash S \rrparenthesis c \\
= & \langle \text{Similar to the calculation for relative pseudocomplement} \rangle \\
& (\forall\, a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c) \\
= & \langle \text{Generalised De Morgan, Relation conversions — Ex. 6.3 (R1)} \rangle \\
& b \llparenthesis \sim (R^{\smile} \fatsemi \sim S) \rrparenthesis c
\end{array}
$$

Therefore: $\quad R \backslash S \; = \; \sim (R^{\smile} \fatsemi \sim S)$

— monotonic in second argument; antitonic in first argument

Proving $b \llparenthesis R \backslash S \rrparenthesis c \equiv (\forall\, a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c)$**:**

$$
\begin{array}{ll}
& b (R \backslash S) c \\
= & \langle e \in S \equiv \{e\} \subseteq S \text{ — Exercise!} \rangle \\
& \{\langle b, c \rangle\} \subseteq (R \backslash S) \\
= & \langle \text{Def. } \backslash\text{: } X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S \rangle \\
& R \fatsemi \{\langle b, c \rangle\} \subseteq S \\
= & \langle \text{(11.13r) Relation inclusion} \rangle \\
& (\forall\, a, c' \mid a \llparenthesis R \fatsemi \{\langle b, c \rangle\} \rrparenthesis c' \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(14.20) Relation composition} \rangle \\
& (\forall\, a, c' \mid (\exists\, b' \bullet a \llparenthesis R \rrparenthesis b' \wedge b' \llparenthesis \{\langle b, c \rangle\} \rrparenthesis c') \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle y \in \{x\} \equiv y = x \text{ — Exercise!} \rangle \\
& (\forall\, a, c' \mid (\exists\, b' \bullet a \llparenthesis R \rrparenthesis b' \wedge b' = b \wedge c = c') \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(9.19) Trading for } \exists \rangle \\
& (\forall\, a, c' \mid (\exists\, b' \mid b' = b \bullet a \llparenthesis R \rrparenthesis b' \wedge c = c') \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(8.14) One-point rule} \rangle \\
& (\forall\, a, c' \mid a \llparenthesis R \rrparenthesis b \wedge c = c' \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(8.20) Quantifier nesting} \rangle \\
& (\forall\, a \mid a \llparenthesis R \rrparenthesis b \bullet (\forall\, c' \mid c = c' \bullet a \llparenthesis S \rrparenthesis c')) \\
= & \langle \text{(1.3) Symmetry of } =\text{, (8.14) One-point rule} \rangle \\
& (\forall a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c)
\end{array}
$$

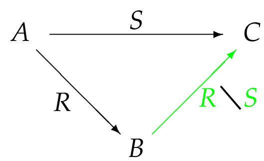
Right Residual: $\qquad X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S$

Proving $R \backslash S \; = \; \sim (R^{\smile} \fatsemi \sim S)$**:**

$$
\begin{array}{ll}
& b \llparenthesis R \backslash S \rrparenthesis c \\
= & \langle \text{previous slide} \rangle \\
& (\forall\, a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c) \\
= & \langle \text{(9.18a) Generalised De Morgan} \rangle \\
& \neg(\exists\, a \mid a \llparenthesis R \rrparenthesis b \bullet \neg(a \llparenthesis S \rrparenthesis c)) \\
= & \langle \text{(11.17r) Relation complement} \rangle \\
& \neg(\exists\, a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis \sim S \rrparenthesis c) \\
= & \langle \text{(9.19) Trading for } \exists\text{, (14.18) Converse} \rangle \\
& \neg(\exists\, a \bullet b \llparenthesis R^{\smile} \rrparenthesis a \wedge a \llparenthesis \sim S \rrparenthesis c) \\
= & \langle \text{(14.20) Relation composition} \rangle \\
& \neg(b \llparenthesis R^{\smile} \fatsemi \sim S \rrparenthesis c) \\
= & \langle \text{(11.17r) Relation complement} \rangle \\
& b \llparenthesis \sim (R^{\smile} \fatsemi \sim S) \rrparenthesis c
\end{array}
$$

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\qquad X \subseteq R \backslash S \quad \equiv \quad R \mathbin{;} X \subseteq S$

Calculate the **right residual** ("left division") $R \backslash S$! $\qquad$ ("R under S")



$$\begin{aligned}
& b \llparenthesis R \backslash S \rrparenthesis c \\
= \; & \langle \text{ Similar to the calculation for relative pseudocomplement } \rangle \\
& (\forall\, a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c) \\
= \; & \langle \text{ Generalised De Morgan, Relation conversions — Ex. 6.3 (R1) } \rangle \\
& b \llparenthesis \sim (R^{\smile} \mathbin{;} \sim S) \rrparenthesis c
\end{aligned}$$

Therefore: $\quad R \backslash S = \sim (R^{\smile} \mathbin{;} \sim S)$

— monotonic in second argument; antitonic in first argument

Formalisations Using Residuals

"Aos called only brothers of Jun."

"Everybody called by Aos is a brother of Jun."

$$\begin{aligned}
& (\forall\, p \mid Aos \llparenthesis C \rrparenthesis p \bullet p \llparenthesis B \rrparenthesis Jun) \\
\equiv \; & \langle \text{ (14.18) Relation converse } \rangle \\
& (\forall\, p \mid p \llparenthesis C^{\smile} \rrparenthesis Aos \bullet p \llparenthesis B \rrparenthesis Jun) \\
\equiv \; & \langle \text{ Right residual } \rangle \\
& Aos \llparenthesis C^{\smile} \backslash B \rrparenthesis Jun
\end{aligned}$$

Relationship via $\backslash$:

$$\begin{aligned}
& b \llparenthesis R \backslash S \rrparenthesis c \\
\equiv \; & (\forall\, a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c)
\end{aligned}$$

"Aos called every brother of Jun."

"Every brother of Jun has been called by Aos."

$$\begin{aligned}
& (\forall\, p \mid p \llparenthesis B \rrparenthesis Jun \bullet Aos \llparenthesis C \rrparenthesis p) \\
\equiv \; & \langle \text{ (14.18) Relation converse } \rangle \\
& (\forall\, p \mid p \llparenthesis B \rrparenthesis Jun \bullet p \llparenthesis C^{\smile} \rrparenthesis Aos) \\
\equiv \; & \langle \text{ Right residual } \rangle \\
& Jun \llparenthesis B \backslash C^{\smile} \rrparenthesis Aos
\end{aligned}$$

Some Properties of Right Residuals

Characterisation of right residual: $\forall\, R : A \leftrightarrow B;\; S : A \leftrightarrow C \bullet \quad X \subseteq R \backslash S \;\equiv\; R \mathbin{;} X \subseteq S$

Two sub-cancellation properties follow easily:

$$R \mathbin{;} (R \backslash S) \subseteq S$$
$$(Q \backslash R) \mathbin{;} (R \backslash S) \subseteq (Q \backslash S)$$

Theorem "$\mathbb{I} \backslash$": $\mathbb{I} \backslash R = R$

Proof:

Using "Mutual inclusion":

Subproof:

$$\begin{aligned}
& \mathbb{I} \backslash R \\
= \; & \langle \text{ "Identity of ;" } \rangle \\
& \mathbb{I} \mathbin{;} (\mathbb{I} \backslash R) \\
\subseteq \; & \langle \text{ "Cancellation of } \backslash \text{" } \rangle \\
& R
\end{aligned}$$

Subproof:

$$\begin{aligned}
& R \subseteq \mathbb{I} \backslash R \\
\equiv \; & \langle \text{ "Characterisation of } \backslash \text{" } \rangle \\
& \mathbb{I} \mathbin{;} R \subseteq R \\
\equiv \; & \langle \text{ "Identity of ;", "Reflexivity of } \subseteq \text{" } \rangle \\
& \mathsf{true}
\end{aligned}$$

Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall\, x,y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y)$
$R \subseteq S$	$\equiv$	$(\forall\, x,y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y)$
$u \llparenthesis \{\} \rrparenthesis v$	$\equiv$	false
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis \sim S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \Rightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\, A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^{\smile} \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \fatsemi S \rrparenthesis v$	$\equiv$	$(\exists\, x \bullet u \llparenthesis R \rrparenthesis x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \backslash S \rrparenthesis v$	$\equiv$	$(\forall\, x \mid x \llparenthesis R \rrparenthesis u \bullet x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall\, x \mid v \llparenthesis R \rrparenthesis x \bullet u \llparenthesis S \rrparenthesis x)$

Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall\, x,y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y)$
$R \subseteq S$	$\equiv$	$(\forall\, x,y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y)$
$u \llparenthesis \{\} \rrparenthesis v$	$\equiv$	false
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis \sim S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \Rightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\, A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^{\smile} \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \fatsemi S \rrparenthesis v$	$\equiv$	$(\exists\, x \mid u \llparenthesis R \rrparenthesis x \bullet x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \backslash S \rrparenthesis v$	$\equiv$	$(\forall\, x \mid x \llparenthesis R \rrparenthesis u \bullet x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall\, x \mid v \llparenthesis R \rrparenthesis x \bullet u \llparenthesis S \rrparenthesis x)$

Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall\, x,y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y)$
$R \subseteq S$	$\equiv$	$(\forall\, x,y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y)$
$u \llparenthesis \{\} \rrparenthesis v$	$\equiv$	false
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis \sim S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \Rightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\, A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^{\smile} \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \fatsemi S \rrparenthesis v$	$\equiv$	$(\exists\, x \bullet u \llparenthesis R \rrparenthesis x \wedge x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \backslash S \rrparenthesis v$	$\equiv$	$(\forall\, x \bullet x \llparenthesis R \rrparenthesis u \Rightarrow x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall\, x \bullet v \llparenthesis R \rrparenthesis x \Rightarrow u \llparenthesis S \rrparenthesis x)$

Symmetric Difference

Symmetric difference is usually defined on sets: $S \ominus T \;=\; (S - T) \cup (T - S)$

Theorem "Membership in $\ominus$": $x \in (S \ominus T) \;\equiv\; x \in S \not\equiv x \in T$

We can define it also on numbers, e.g., on $\mathbb{Z}$ or $\mathbb{N}$: $k \ominus m \;=\; (k - m) \uparrow (m - k)$

Then we have:

Theorem "Size of symmetric set difference": $(\# S) \ominus (\# T) \;\leq\; \# (S \ominus T)$

Proof: — *Exercise!*

Let the following sets be given:

S_1 : all students who normally attended lectures up to Midterm 1

S_s : all students who achieved a grade of at least 50% in Midterm 1

Observation: $(\# S_1) \ominus (\# S_2) \;\leq\; 20$

Conjecture: $\# (S_1 \ominus S_2) \;\leq\; 20$

Logical Reasoning for Computer Science

COMPSCI 2LC3

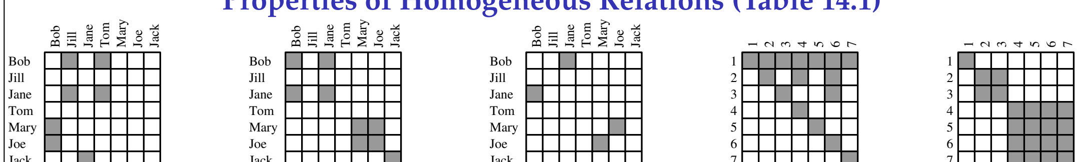
McMaster University, Fall 2024

Wolfram Kahl

2024-10-29

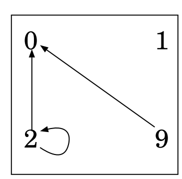
Part 2: Relation Properties

Properties of Homogeneous Relations (Table 14.1)



A relation $R : B \leftrightarrow C$ is called **homogeneous** iff $B = C$.

A (homogeneous) relation $R : B \leftrightarrow B$ is called:



reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \,(R)\, b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall\, b : B \bullet \neg(b \,(R)\, b))$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : B \bullet b \,(R)\, c \equiv c \,(R)\, b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall\, b, c \bullet b \,(R)\, c \wedge c \,(R)\, b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall\, b, c : B \bullet b \,(R)\, c \Rightarrow \neg(c \,(R)\, b))$
transitive	$R \,⨾\, R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \,(R)\, d \Rightarrow b \,(R)\, d)$
idempotent	$R \,⨾\, R = R$	

Properties of Homogeneous Relations (ctd.)

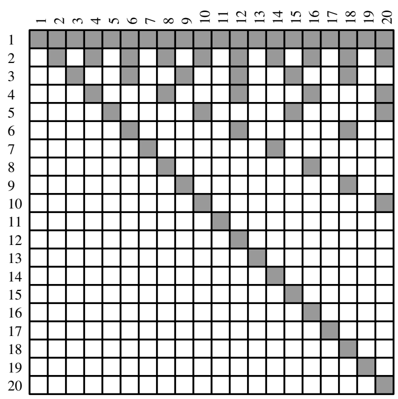
reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \,(\!|\, R \,|\!)\, b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall\, b : B \bullet \neg(b \,(\!|\, R \,|\!)\, b))$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : B \bullet b \,(\!|\, R \,|\!)\, c \equiv c \,(\!|\, R \,|\!)\, b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall\, b, c \bullet b \,(\!|\, R \,|\!)\, c \wedge c \,(\!|\, R \,|\!)\, b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall\, b, c : B \bullet b \,(\!|\, R \,|\!)\, c \Rightarrow \neg(c \,(\!|\, R \,|\!)\, b))$
transitive	$R ⨾ R \subseteq R$	$(\forall b, c, d \bullet b \,(\!|\, R \,|\!)\, c \wedge c \,(\!|\, R \,|\!)\, d \Rightarrow b \,(\!|\, R \,|\!)\, d)$

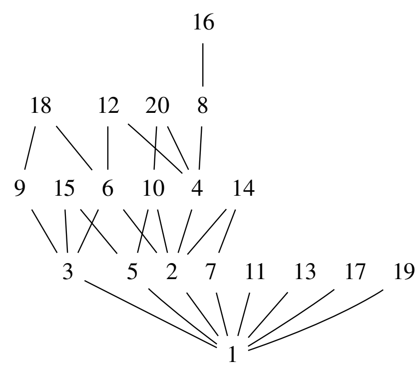
R is an **equivalence (relation) on** B iff it is reflexive, transitive, and symmetric.

R is a **(partial) order on** B iff it is reflexive, transitive, and antisymmetric.
(E.g., $\leq$, $\geq$, $\subseteq$, $\supseteq$, $|$)

R is a **strict-order on** B iff it is irreflexive, transitive, and asymmetric.
(E.g., $<$, $>$, $\subset$, $\supset$)

Divisibility Order with Hasse Diagram

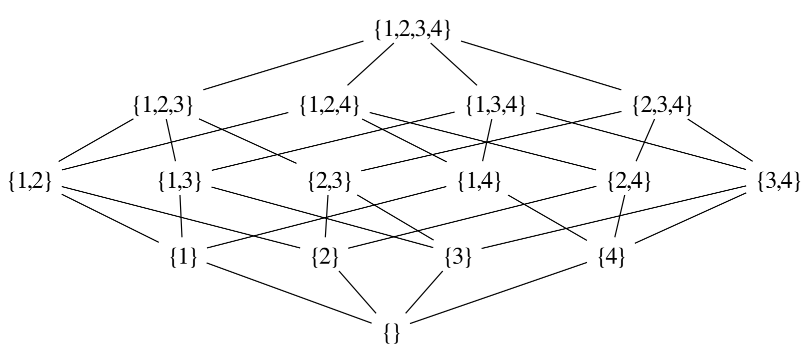




Hasse diagram for an **order**:

- Edge direction is **upwards** — **antisymmetric**
- Loops not drawn — **reflexive**
- Transitive edges not drawn — **transitive**

Inclusion Order on Power Set of $\{1, 2, 3, 4\}$



Hasse diagram for an **order**:

- Edge direction is **upwards** — **antisymmetric**
- Loops not drawn — **reflexive**
- Transitive edges not drawn — **transitive**

Properties of Heterogeneous Relations

A relation $R : B \leftrightarrow C$ is called:

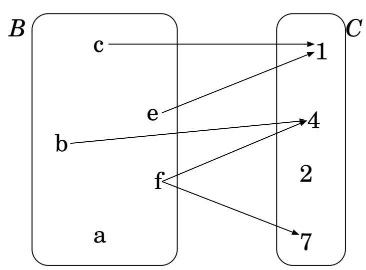
univalent determinate	$R^{\smile} ⨾ R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b \,(R)\, c_1 \wedge b \,(R)\, c_2 \Rightarrow c_1 = c_2$
total	$\mathit{Dom}\ R = \mathbf{U}$ $\mathit{Dom}\ R = \llcorner B \lrcorner$ $\mathbb{I} \subseteq R ⨾ R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b \,(R)\, c)$
injective	$R ⨾ R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 \,(R)\, c \wedge b_2 \,(R)\, c \Rightarrow b_1 = b_2$
surjective	$\mathit{Ran}\ R = \mathbf{U}$ $\mathit{Ran}\ R = \llcorner C \lrcorner$ $\mathbb{I} \subseteq R^{\smile} ⨾ R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b \,(R)\, c)$
a **mapping**	iff it is univalent and total	
bijective	iff it is injective and surjective	

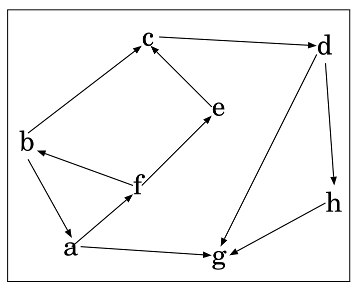
Univalent relations are also called **(partial) functions**.

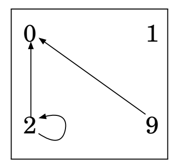
Mappings are also called **total functions**.

Properties of Heterogeneous Relations — Examples 1

univalent	$R^{\smile} ⨾ R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b \,(R)\, c_1 \wedge b \,(R)\, c_2 \Rightarrow c_1 = c_2$
total	$\mathit{Dom}\ R = \mathbf{U}$ $\mathbb{I} \subseteq R ⨾ R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b \,(R)\, c)$
a **mapping**	iff it is univalent and total	

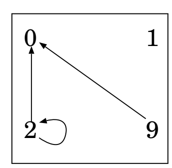


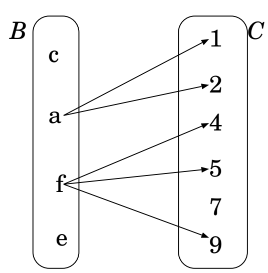


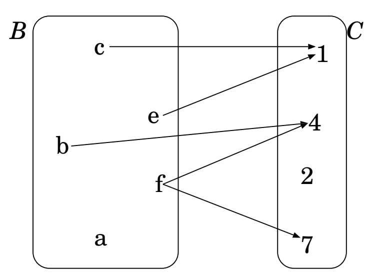


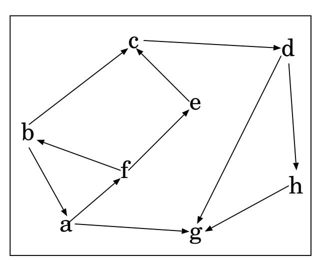
Properties of Heterogeneous Relations — Examples 2

injective	$R ⨾ R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 \,(R)\, c \wedge b_2 \,(R)\, c \Rightarrow b_1 = b_2$
surjective	$\mathit{Ran}\ R = \mathbf{U}$ $\mathbb{I} \subseteq R^{\smile} ⨾ R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b \,(R)\, c)$
bijective	iff it is injective and surjective	









Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-10-31

More Quantification Calculations, Relation Properties, . . .

Interchange

(9.29) **Interchange of quantifications::** Provided $\neg occurs(`y', `R') \wedge \neg occurs(`x', `Q')$,

$$(\exists x \mid R \bullet (\forall y \mid Q \bullet P)) \quad \Rightarrow \quad (\forall y \mid Q \bullet (\exists x \mid R \bullet P))$$

One direction only!

(9.29) **Interchange of quantifications::** Provided $\neg occurs(`y', `R') \wedge \neg occurs(`x', `Q')$,

$$(\exists x \mid R \bullet (\forall y \mid Q \bullet P)) \quad \Rightarrow \quad (\forall y \mid Q \bullet (\exists x \mid R \bullet P))$$

One direction only!

Understanding Interchange

Formalise: Every real number has an additive inverse.

true

= ⟨ Every real number does have an additive inverse ⟩

$(\forall y : \mathbb{R} \bullet (\exists x : \mathbb{R} \bullet y + x = 0))$

⇐ ⟨ (9.29) Interchange of quantifications ⟩

$(\exists x : \mathbb{R} \bullet (\forall y : \mathbb{R} \bullet y + x = 0))$

This says: "There is a real number x which is an additive inverse for all real numbers".

= ⟨ Different numbers have different additive inverses . . . ⟩

false

Interchange — Proof

(9.29) **Interchange of quantifications::** Provided $\neg occurs(`y', `R') \land \neg occurs(`x', `Q')$,

$$(\exists x \mid R \bullet (\forall y \mid Q \bullet P)) \quad \Rightarrow \quad (\forall y \mid Q \bullet (\exists x \mid R \bullet P))$$

Proof of simpler case ($R \equiv true$):

$(\exists x \bullet (\forall y \bullet P)) \quad \Rightarrow \quad (\forall y \bullet (\exists x \bullet P))$

= ⟨ (3.57) Definition of ⇒ ⟩

$(\exists x \bullet (\forall y \bullet P)) \lor (\forall y \bullet (\exists x \bullet P)) \equiv (\forall y \bullet (\exists x \bullet P))$

= ⟨ (9.5) Distributivity of ∨ over ∀ ⟩

$(\forall y \bullet (\exists x \bullet (\forall y \bullet P)) \lor (\exists x \bullet P)) \equiv (\forall y \bullet (\exists x \bullet P))$

= ⟨ (8.15) Distributivity of ∃ over ∨ ⟩

$(\forall y \bullet (\exists x \bullet (\forall y \bullet P) \lor P)) \quad \equiv \quad (\forall y \bullet (\exists x \bullet P))$

= ⟨ (9.13.1) Instantiation $(\forall y \bullet P) \Rightarrow P$, with (3.57): $(\forall y \bullet P) \lor P \equiv P$ ⟩

$(\forall y \bullet (\exists x \bullet P)) \quad \equiv \quad (\forall y \bullet (\exists x \bullet P))$

— This is (3.5) Reflexivity of ≡

`with`$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$:
  Rewrite *ThmA* with $L \mapsto R$
- E.g.: Assumption \`$Q \subseteq R$\` with "Relation inclusion":

  $Q \subseteq R$ rewrites via $Q \subseteq R \mapsto \forall x \bullet \forall y \bullet x ⦇ Q ⦈ y \Rightarrow x ⦇ R ⦈ y$

  to: $\forall x \bullet \forall y \bullet x ⦇ Q ⦈ y \Rightarrow x ⦇ R ⦈ y$

  which can be instantiated to: to: $a ⦇ Q ⦈ b \Rightarrow a ⦇ R ⦈ b$

```
  ∃ b • a ⦇ Q ⦈ b ∧ b ⦇ S ⦈ c
⇒⟨ "Body monotonicity of ∃" with "Monotonicity of ∧"
    with assumption `Q ⊆ R` with "Relation inclusion" ⟩
  ∃ b • a ⦇ R ⦈ b ∧ b ⦇ S ⦈ c
```

`with`$_2$ and `with`$_3$: Example

```
  ∃ b • a ⦇ Q ⦈ b ∧ b ⦇ S ⦈ c
⇒⟨ "Body monotonicity of ∃" with "Monotonicity of ∧"
    with assumption `Q ⊆ R` with "Relation inclusion" ⟩
  ∃ b • a ⦇ R ⦈ b ∧ b ⦇ S ⦈ c
```

- assumption '$Q \subseteq R$' gives you $Q \subseteq R$
- assumption '$Q \subseteq R$' with "Relation inclusion"
  gives you via `with`$_3$: $\forall x \bullet \forall y \bullet x ⦇ Q ⦈ y \Rightarrow x ⦇ R ⦈ y$
  and then via implicit "Instantiation" triggered by the next `with`$_2$:
  $a ⦇ Q ⦈ b \quad \Rightarrow \quad a ⦇ R ⦈ b$
- "Monotonicity of ∧" with
  assumption '$Q \subseteq R$' with "Relation inclusion"
  gives you via `with`$_2$: $a ⦇ Q ⦈ b \land b ⦇ S ⦈ c \quad \Rightarrow \quad a ⦇ R ⦈ b \land b ⦇ S ⦈ c$
- "Body monotonicity of ∃" with "Monotonicity of ∧" with
  assumption '$Q \subseteq R$' with "Relation inclusion"
  gives you via `with`$_2$:
  $(\exists b \bullet a ⦇ Q ⦈ b \land b ⦇ S ⦈ c) \quad \Rightarrow \quad (\exists b \bullet a ⦇ R ⦈ b \land b ⦇ S ⦈ c)$

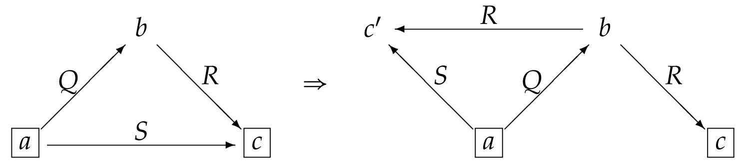
Modal Rules— Converse as Over-Approximation of Inverse

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q ; R \cap S \subseteq Q ; (R \cap Q^{\smile} ; S)$$
$$Q ; R \cap S \subseteq (Q \cap S ; R^{\smile}) ; R$$

Useful to "**make information available locally**" (Q is replaced with $Q \cap S ; R^{\smile}$) for use in further proof steps.

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a\,(Q)\,b\,(R)\,c \wedge a\,(S)\,c) \quad \Rightarrow$
$\quad (\exists b \bullet \exists c' \bullet a\,(Q)\,b\,(R)\,c \wedge b\,(R)\,c' \wedge a\,(S)\,c')$

Properties of Heterogeneous Relations

A relation $R : B \leftrightarrow C$ is called:

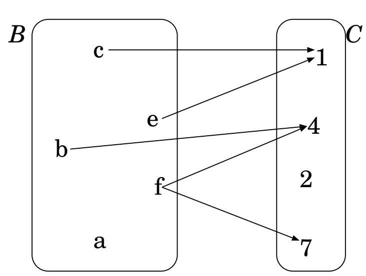
univalent determinate	$R^{\smile} ; R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b\,(R)\,c_1 \wedge b\,(R)\,c_2 \Rightarrow c_1 = c_2$
total	$\mathit{Dom}\ R = \mathbf{U}$ $\mathit{Dom}\ R = \llcorner B \lrcorner$ $\mathbb{I} \subseteq R ; R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b\,(R)\,c)$
injective	$R ; R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1\,(R)\,c \wedge b_2\,(R)\,c \Rightarrow b_1 = b_2$
surjective	$\mathit{Ran}\ R = \mathbf{U}$ $\mathit{Ran}\ R = \llcorner C \lrcorner$ $\mathbb{I} \subseteq R^{\smile} ; R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b\,(R)\,c)$
a **mapping**	iff it is univalent and total	
bijective	iff it is injective and surjective	

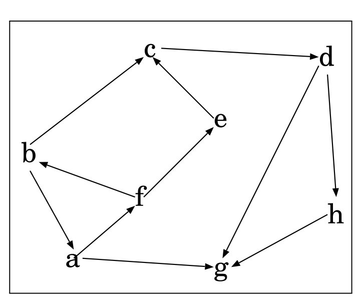
Univalent relations are also called **(partial) functions**.

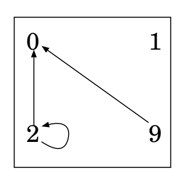
Mappings are also called **total functions**.

Properties of Heterogeneous Relations — Examples 1

univalent	$R^{\smile} ; R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b\,(R)\,c_1 \wedge b\,(R)\,c_2 \Rightarrow c_1 = c_2$
total	$\mathit{Dom}\ R = \mathbf{U}$ $\mathbb{I} \subseteq R ; R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b\,(R)\,c)$
a **mapping**	iff it is univalent and total	

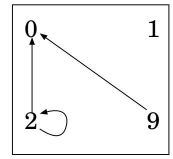


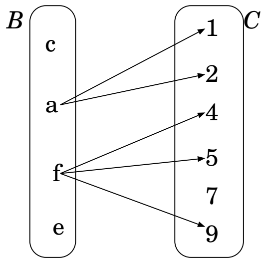


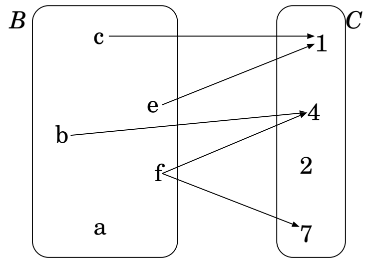


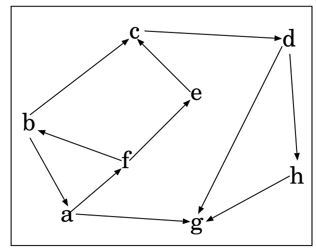
Properties of Heterogeneous Relations — Examples 2

injective	$R \,;\, R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 \llparenthesis R \rrparenthesis c \land b_2 \llparenthesis R \rrparenthesis c \Rightarrow b_1 = b_2$
surjective	$\mathit{Ran}\ R = \mathbf{U}$ $\mathbb{I} \subseteq R^{\smile} \,;\, R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b \llparenthesis R \rrparenthesis c)$
bijective	iff it is injective and surjective	









Function Types versus Sets of Univalent Relations

A relation $R : B \leftrightarrow C$ is called:

univalent	$R^{\smile} \,;\, R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b \llparenthesis R \rrparenthesis c_1 \land b \llparenthesis R \rrparenthesis c_2 \Rightarrow c_1 = c_2$
total	$\mathit{Dom}\ R = \mathbf{U}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b \llparenthesis R \rrparenthesis c)$
a **mapping**	iff it is univalent and total	

Univalent relations are also called **(partial) functions**.

Mappings are also called **total functions**.

— **These are of different type than functions of function type $B \to C$!**

The distinction corresponds to the way in which elements of the **Haskell** datatype *Data.Map.Map a b* are distinct from Haskell functions of type $a \to b$.

- A (set-theoretic) relation $R : B \leftrightarrow C$ **is** a set of pairs — "data"
- A function $f : B \to C$ is a different kind of entity — in Haskell, "computation".

  In most logics, including CalcCheck, if $f : B \to C$ and $b : B$, then $f\ b$ is **never undefined**.

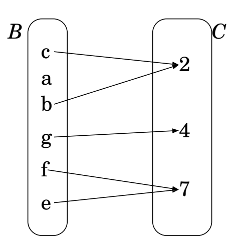
  (But may be **unspecified**, such as $\mathit{head}\ \epsilon$ in Ex7.3.)

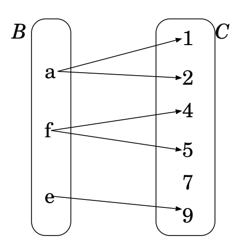
Properties of Heterogeneous Relations — Remarks

univalent	$R^{\smile} \,;\, R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b \llparenthesis R \rrparenthesis c_1 \land b \llparenthesis R \rrparenthesis c_2 \Rightarrow c_1 = c_2$
surjective	$R^{\smile} \,;\, R \supseteq \mathbb{I}$	$\forall\, c : C \bullet (\exists\, b : B \bullet b \llparenthesis R \rrparenthesis c)$
total	$R \,;\, R^{\smile} \supseteq \mathbb{I}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b \llparenthesis R \rrparenthesis c)$
injective	$R \,;\, R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 \llparenthesis R \rrparenthesis c \land b_2 \llparenthesis R \rrparenthesis c \Rightarrow b_1 = b_2$

- R is univalent and surjective
  iff $R^{\smile} \,;\, R = \mathbb{I}$
  iff $R^{\smile}$ is a left-inverse of R

- R is total and injective
  iff $R \,;\, R^{\smile} = \mathbb{I}$
  iff $R^{\smile}$ is a right-inverse of R





Properties of Heterogeneous Relations — Notes

univalent	$R^{\smile} ⨾ R$	$\subseteq$	$\mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b \,(R)\, c_1 \land b \,(R)\, c_2 \Rightarrow c_1 = c_2$
surjective	$R^{\smile} ⨾ R$	$\supseteq$	$\mathbb{I}$	$\forall\, c : C \bullet (\exists\, b : B \bullet b \,(R)\, c)$
total	$R ⨾ R^{\smile}$	$\supseteq$	$\mathbb{I}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b \,(R)\, c)$
injective	$R ⨾ R^{\smile}$	$\subseteq$	$\mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 \,(R)\, c \land b_2 \,(R)\, c \Rightarrow b_1 = b_2$

All these properties are defined for arbitrary relations! (Not only for functions!)

- R is univalent and surjective
  iff $R^{\smile} ⨾ R = \mathbb{I}$
  iff $R^{\smile}$ is a left-inverse of R
- R is total and injective
  iff $R ⨾ R^{\smile} = \mathbb{I}$
  iff $R^{\smile}$ is a right-inverse of R

It is convenient to have abbreviations, for example:

f is a partial function from X to Y:	$f \in X \nrightarrow Y$	
f is an injective mapping from X to Y:	$f \in X \rightarrowtail Y$	$\longrightarrow$ Z arrows!
f is a partial surjection from X to Y:	$f \in X \twoheadrightarrow Y$ (partial)	

The Z Specification Notation

- Mathematical notation intended for software specification
  Used for requirements contracts with customers who would be given a two-page "Z Reference Card"
- Very influential in Formal Methods; ISO-standardised
- Two parts:
  - Z **is** a typed set theory in first-order predicate logic
    — very close to the logic and set theory you are using in CALCCHECK
    — except that in Z:
    - types **are** maximal sets
    - sets can be used in variable declarations: $\forall\, x : S \mid \ldots \bullet \ldots$,
      — which makes quantifier reasoning harder.
    - functions **are** univalent relations

    (CALCCHECK and Haskell are type theories with embedded typed set theories.)
  - "Schemas" modelling of states and state transitions
- Avenue ⟶ Resources ⟶ Links ⟶ Z Specification Notation

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-01

Z Operators and Arrows, Ghosts. . .

The Z Specification Notation

- Mathematical notation intended for software specification

  Used for requirements contracts with customers who would be given a two-page "Z Reference Card"
- Very influential in Formal Methods; ISO-standardised
- Two parts:
  - Z **is** a typed set theory in first-order predicate logic

    — very close to the logic and set theory you are using in CALCCHECK

    — except that in Z:
    - types **are** maximal sets
    - sets can be used in variable declarations: $\forall\, x : S \;|\; \ldots \bullet \ldots$,

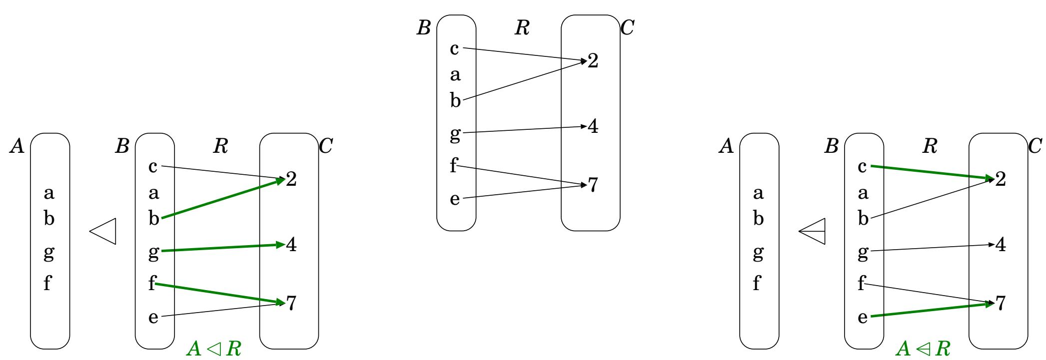
      — which makes quantifier reasoning harder.
    - functions **are** univalent relations

    (CALCCHECK and Haskell are type theories with embedded typed set theories.)
  - "Schemas" modelling of states and state transitions
- Avenue ⟶ Resources ⟶ Links ⟶ Z Specification Notation

More Z Symbols: Domain-Restriction and -Antirestriction

Given types t_1, t_2 : Type, a sets A : **set** t_1, and a relation $R : t_1 \leftrightarrow t_2$:

- **Domain restriction**: $A \lhd R \;=\; R \cap (A \times \mathbf{U})$
- **Domain antirestriction**: $A \ntriangleleft R \;=\; R - (A \times \mathbf{U}) \;=\; {\sim}A \lhd R \;=\; R \cap ({\sim}A \times \mathbf{U})$



More Z Symbols: Domain- and Range-Restriction and -Antirestriction

Given types t_1, t_2 : Type, sets A : **set** t_1 and B : **set** t_2, and relation $R : t_1 \leftrightarrow t_2$:

- **Domain restriction**: $A \lhd R \;=\; R \cap (A \times \mathbf{U})$
- **Domain antirestriction**: $A \ntriangleleft R \;=\; R - (A \times \mathbf{U}) \;=\; R \cap ({\sim}A \times \mathbf{U})$
- **Range restriction**: $R \rhd B \;=\; R \cap (\mathbf{U} \times B)$
- **Range antirestriction**: $R \ntriangleright B \;=\; R - (\mathbf{U} \times B) \;=\; R \cap (\mathbf{U} \times {\sim}B)$

$$B \mathbin{;} (\{Jun\} \times \mathbf{U}) \;\cap\; (C \mathbin{;} C^{\smile}) \;\subseteq\; \mathbb{I}$$

$\equiv$ ⟨ Domain- and range restriction properties ⟩

$$Dom(B \rhd \{Jun\}) \lhd (C \mathbin{;} C^{\smile}) \;\subseteq\; \mathbb{I}$$

Still no quantifiers, and no x, y of element type — but not only relations, also sets!

(The abstract version of this is called **Peirce algebra**, after Charles Sanders Peirce.)

Also in Z: Relational Image

Given types t_1, t_2 : Type, sets A : **set** t_1 and B : **set** t_2, and relations $R, S : t_1 \leftrightarrow t_2$:

- **Relational image**: $R (\!| A |\!) \quad = \quad \mathit{Ran}(A \lhd R)$

  "Relational image of set A under relation R"

  Notation as "generalised function application"...

$$B \fatsemi (\{\mathit{Jun}\} \times \mathbf{U}) \quad \cap \quad (C \fatsemi C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

$\equiv \langle$ Domain- and range restriction properties $\rangle$

$$\mathit{Dom}(B \rhd \{\mathit{Jun}\}) \lhd (C \fatsemi C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

$\equiv \langle$ Relational image $\rangle$

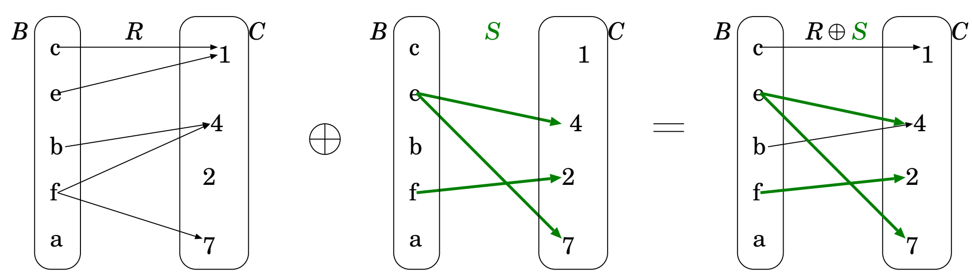
$$(B^{\smile} (\!| \{\mathit{Jun}\} |\!)) \lhd (C \fatsemi C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

Also in Z: Relation Overriding $\oplus$

Given types t_1, t_2 : Type, sets A : **set** t_1 and B : **set** t_2, and relations $R, S : t_1 \leftrightarrow t_2$:

- **Relation overriding**: $R \oplus S \quad = \quad (\mathit{Dom}\ S \ntriangleleft R) \cup S$

  "Updating R exactly where S relates with anything"



In the relation $\quad C \oplus \{\langle \mathit{Aos}, \mathit{Jun} \rangle\} \quad$, Aos called only Jun.

Function Sets — Z Definition and Description [Spivey 1992]

In Z, $X \leftrightarrow Y = \mathbb{P}(X \times Y)$, and $x \mapsto y = (x, y)$ is an abbreviation for pairs.

⇸	–	Partial functions
→	–	Total functions
⤔	–	Partial injections
↣	–	Total injections
⤀	–	Partial surjections
↠	–	Total surjections
⤖	–	Bijections

$$X ⇸ Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$$

$$X \rightarrow Y == \{ f : X ⇸ Y \mid \mathrm{dom}\, f = X \}$$

$$X ⤔ Y == \{ f : X ⇸ Y \mid (\forall x_1, x_2 : \mathrm{dom}\, f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \}$$

$$X \rightarrowtail Y == (X ⤔ Y) \cap (X \rightarrow Y)$$

$$X ⤀ Y == \{ f : X ⇸ Y \mid \mathrm{ran}\, f = Y \}$$

$$X \twoheadrightarrow Y == (X ⤀ Y) \cap (X \rightarrow Y)$$

$$X ⤖ Y == (X \twoheadrightarrow Y) \cap (X \rightarrowtail Y)$$

If X and Y are sets, $X ⇸ Y$ is the set of partial functions from X to Y. These are relations which relate each member x of X to at most one member of Y. This member of Y, if it exists, is written $f(x)$. The set $X \rightarrow Y$ is the set of total functions from X to Y. These are partial functions whose domain is the whole of X; they relate each member of X to exactly one member of Y.

Function Sets — Z Definition and Laws (1) [Spivey 1992]

In Z, $X \leftrightarrow Y = \mathbb{P}(X \times Y)$, and $x \mapsto y = (x, y)$ is an abbreviation for pairs, and $S \circ R = R \mathbin{⨾} S$.

$$X \pfun Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$$

$$X \longrightarrow Y == \{ f : X \pfun Y \mid \operatorname{dom} f = X \}$$

$$X \pinj Y == \{ f : X \pfun Y \mid (\forall x_1, x_2 : \operatorname{dom} f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \}$$

$$X \rightarrowtail Y == (X \pinj Y) \cap (X \longrightarrow Y)$$

Laws:

$$f \in X \pfun Y \Leftrightarrow f \circ f^\sim = \operatorname{id}(\operatorname{ran} f)$$

$$f \in X \pinj Y \Leftrightarrow f \in X \pfun Y \land f^\sim \in Y \pfun X$$
$$f \in X \rightarrowtail Y \Leftrightarrow f \in X \longrightarrow Y \land f^\sim \in Y \pfun X$$

$$f \in X \pinj Y \Rightarrow f(\!|S \cap T|\!) = f(\!|S|\!) \cap f(\!|T|\!)$$

Function Sets — Z Definition and Laws [Spivey 1992]

In Z, $X \leftrightarrow Y = \mathbb{P}(X \times Y)$, and $x \mapsto y = (x, y)$ is an abbreviation for pairs, and $S \circ R = R \mathbin{⨾} S$.

$$X \pfun Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$$

$$X \longrightarrow Y == \{ f : X \pfun Y \mid \operatorname{dom} f = X \}$$

$$X \psurj Y == \{ f : X \pfun Y \mid \operatorname{ran} f = Y \}$$

$$X \twoheadrightarrow Y == (X \psurj Y) \cap (X \longrightarrow Y)$$

$$X \bij Y == (X \twoheadrightarrow Y) \cap (X \rightarrowtail Y)$$

Laws:

$$f \in X \bij Y \Leftrightarrow f \in X \longrightarrow Y \land f^\sim \in Y \longrightarrow X$$
$$f \in X \psurj Y \Rightarrow f \circ f^\sim = \operatorname{id} Y$$

Totality and Surjectivity for Relations Between Sets

Recall: A relation $R : t_1 \leftrightarrow t_2$ is called:

total	$\mathit{Dom}\ R = \mathbf{U}$ $\mathit{Dom}\ R = \llcorner t_1 \lrcorner$ $\mathbb{I} \subseteq R \mathbin{⨾} R^\smile$	$\forall\, b : t_1 \bullet (\exists\, c : t_2 \bullet b\,(R)\,c)$
surjective	$\mathit{Ran}\ R = \mathbf{U}$ $\mathit{Ran}\ R = \llcorner t_2 \lrcorner$ $\mathbb{I} \subseteq R^\smile \mathbin{⨾} R$	$\forall\, c : t_2 \bullet (\exists\, b : t_1 \bullet b\,(R)\,c)$

A relation R with $R \in B \leftrightarrow C$ is called:

total on B	$\mathit{Dom}\ R = B$ $\operatorname{id} B \subseteq R \mathbin{⨾} R^\smile$	$\forall\, b \mid b \in B \bullet (\exists\, c \mid c \in C \bullet b\,(R)\,c)$
surjective onto C	$\mathit{Ran}\ R = C$ $\operatorname{id} C \subseteq R^\smile \mathbin{⨾} R$	$\forall\, c \mid c \in C \bullet (\exists\, b \mid b \in B \bullet b\,(R)\,c)$

Note: If $B \neq \mathbf{U}$, then no relation in $B \leftrightarrow C$ is total.

Z Function Sets in CalcCheck

For two sets $X : \textbf{set}\ t_1$ and $Y : \textbf{set}\ t_2$, we define the following **function sets**:

CalcCheck				Z
$f \in X \mathrel{-\!\!\circ\!\!\rightarrow} Y$	`\tfun`	total function	$Dom\ f = X \wedge f^\smile ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X \rightarrow Y$
$f \in X ⇸ Y$	`\pfun`	partial function	$Dom\ f \subseteq X \wedge f^\smile ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X ⇸ Y$
$f \in X \rightarrowtail Y$	`\tinj`	total injection	$f ⨾ f^\smile = \mathrm{id}\ X \wedge f^\smile ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X \rightarrowtail Y$
$f \in X ⤔ Y$	`\pinj`	partial injection	$f ⨾ f^\smile \subseteq \mathrm{id}\ X \wedge f^\smile ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X ⤔ Y$
$f \in X \twoheadrightarrow Y$	`\tsurj`	total surjection	$Dom\ f = X \wedge f^\smile ⨾ f = \mathrm{id}\ Y$	$f \in X \twoheadrightarrow Y$
$f \in X ⤀ Y$	`\psurj`	partial surjection	$Dom\ f \subseteq X \wedge f^\smile ⨾ f = \mathrm{id}\ Y$	$f \in X ⤀ Y$
$f \in X ⤖ Y$	`\tbij`	total bijection	$f ⨾ f^\smile = \mathrm{id}\ X \wedge f^\smile ⨾ f = \mathrm{id}\ Y$	$f \in X ⤖ Y$
$f \in X ⤗ Y$	`\pbij`	partial bijection	$f ⨾ f^\smile \subseteq \mathrm{id}\ X \wedge f^\smile ⨾ f = \mathrm{id}\ Y$	

Counting ...

Let X and Y be finite sets with $\#\, X = a$ and $\#\, Y = b$:

- $\#\,(X \times Y)\ =$ **?** — pairs
- $\#\,(X \leftrightarrow Y)\ =\ \#\,(\mathbb{P}\,(X \times Y))\ =$ **?** — relations
- $\#\,(X \mathrel{-\!\!\circ\!\!\rightarrow} Y)\ =$ **?** — total functions
- $\#\,(X ⇸ Y)\ =$ **?** — partial functions
- $\#\,(X ⤖ X)\ =$ **?** — homogeneous total bijections
- $\#\,(X ⤖ Y)\ =$ **?** — total bijections
- $\#\,(X \rightarrowtail Y)\ =$ **?** — total injections
- $\#\,(X ⤗ Y)\ =$ **?** — partial bijections
- $\#\,(X ⤔ Y)\ =$ **?** — partial injections
- $\#\,(X \twoheadrightarrow Y)\ =$ **?** — total surjections
- $\#\,\{\, S \mid S \subseteq Y \wedge \#\, S = k \,\}\ =$ **?** — k-combinations of Y

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-01

Part 2: Correctness Variations: Ghost Variables

Recall: The "While" Rule

The constituents of a while loop "while B do C od" are:

- The **loop condition** $B : \mathbb{B}$
- The **(loop) body** C : Cmd

The conventional **while rule** allows to infer only correctness statements for while loops that are in the shape of the conclusion of this inference rule, involving an **invariant** condition $Q : \mathbb{B}$:

```
              `B ∧ Q  ⇒[ C ]  Q`
 ⊢───────────────────────────────────────────
        `Q  ⇒[ while B do C od ]  ¬ B ∧ Q`
```

This rule reads:

- If you can prove that execution of the loop body C starting in states satisfying the loop condition B **preserves** the invariant Q,
- then you have proof that the whole loop also preserves the invariant Q, and in addition establishes the negation of the loop condition.

Recall: The "While" Rule — Induction for Partial Correctness

```
              `B ∧ Q  ⇒[ C ]  Q`
 ⊢───────────────────────────────────────────
        `Q  ⇒[ while B do C od ]  ¬ B ∧ Q`
```

The invariant will need to hold

- immediately before the loop starts,
- after each execution of the loop body,
- and therefore also after the loop ends.

The invariant will typically mention all variables that are changed by the loop, and explain how they are related.

Frequent pattern: Generalised postcondition using the negated loop condition

Recall: Using the "While" Rule

Theorem "While-example":

```
    Pre
  ⇒[ INIT ;
      while B
        do
          C
        od ;
      FINAL
    ]
    Post
```

Proof:

```
    Pre    ······Precondition
  ⇒[ INIT ] ⟨ ? ⟩
    Q      ······Invariant
  ⇒[ while B do
         C
       od ]  ⟨ "While" with subproof:
             B ∧ Q    ······Loop condition and invariant
           ⇒[ C ] ⟨ ? ⟩
             Q        ······Invariant
    ⟩
    ¬ B ∧ Q  ······Negated loop condition, and invariant
  ⇒[ FINAL ] ⟨ ? ⟩
    Post    ······Postcondition
```

Using the "While" Rule — Closer Look

$$\frac{\vdash\ `B \wedge Q \ \Rightarrow[\ C\ ]\ Q`}{`Q \ \Rightarrow[\ \text{while } B \text{ do } C \text{ od }]\ \neg B \wedge Q`}$$

```
⋮
  Q          ▪▪▪▪▪▪ Invariant
⇒[ while B do
       C
    od ]  ⟨ "While" with subproof:
           B ∧ Q    ▪▪▪▪▪▪ Loop condition and invariant
        ⇒[ C ] ⟨ ? ⟩
           Q       ▪▪▪▪▪▪ Invariant
   ⟩
  ¬ B ∧ Q  ▪▪▪▪▪▪ Negated loop condition, and invariant
⋮
```

Exercise 7.3: Correctness of a Program Containing a `while`-Loop

Theorem "Correctness of `elem`":

```
     true
  ⇒[ xs := xs₀ ;
     b := false ;
     while xs ≠ ϵ do
          if head xs = x
          then b := true
          else skip
          fi ;
          xs := tail xs
     od
  ]
  (b ≡ x ∈ xs₀) ▪▪▪▪▪▪Parentheses!
```

Proof:

```
     true
  ⇒[ xs := xs₀ ;
       b := false
     ]     ⟨ "Initialisation for `elem`" ⟩
     (∃ us • (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us))
  ⇒[ while xs ≠ ϵ do
          if head xs = x
          then b := true
          else skip
          fi ;
          xs := tail xs
       od
     ]     ⟨ "While" with "Invariant for `elem`" ⟩
     ¬ (xs ≠ ϵ) ∧ (∃ us • (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us))
  ⇒        ⟨ "Postcondition for `elem`" ⟩
     (b ≡ x ∈ xs₀)
```

Invariant involves quantifier: Good for practice with quantifier reasoning...

Easier to Prove than Exercise 7.3: With Ghost Variable — Ex9.1

Theorem "Correctness of `elem`":

```
     true
  ⇒[ xs := xs₀ ;
     us := ϵ ;       ▪▪▪▪▪▪Ghost variable: Does not influence program flow or result
     b := false ;
     ▪▪▪▪▪▪Invariant: (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us)
     while xs ≠ ϵ do
          if head xs = x then b := true else skip fi ;
          us := us ▹ head xs ;          ▪▪▪▪▪▪Ghost assignment
          xs := tail xs
     od
  ]
  (b ≡ x ∈ xs₀)          ▪▪▪▪▪▪Parentheses needed because of precedences!
```

"Ghost variables" can make proofs easier: They can be used to keep track of values that are important for **understanding** the logic of the program.

With language support for "ghost variables", they are compiled away, to avoid run-time cost.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-05

Relation-Algebraic Calculational Proofs

Plan for Today

- Relation-algebraic calculational proofs — "abstract relation algebra"

Relation-algebraic proof ...

- ... is what you started in the fill-in-the-blanks questions of H12
- ... will be the main topic of Exercises 9.*
- ... will be on Midterm 2

  (in addition to predicate logic reasoning, in particular about relations in set theory, etc. ...)
- ... is easier than quantifier reasoning

Recall: Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall x, y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y)$
$R \subseteq S$	$\equiv$	$(\forall x, y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y)$
$u \llparenthesis \{\} \rrparenthesis v$	$\equiv$	false
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis \sim S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \Rightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\ A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^{\smile} \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \fatsemi S \rrparenthesis v$	$\equiv$	$(\exists x \bullet u \llparenthesis R \rrparenthesis x \wedge x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \backslash S \rrparenthesis v$	$\equiv$	$(\forall x \bullet x \llparenthesis R \rrparenthesis u \Rightarrow x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall x \bullet v \llparenthesis R \rrparenthesis x \Rightarrow u \llparenthesis S \rrparenthesis x)$

Using Extensionality/Inclusion and the Translation Table, you Proved:

All subexpressions have $\mathbb{B}$ or _↔_ types!

Equations of relational expressions: **Relation algebra**

(Inclusions "are" equations: $R \subseteq S \;\equiv\; R \cup S = S$)

Theorem "Self-inverse of $\breve{}$": $R\breve{}\,\breve{} = R$
Theorem "Converse of ∩": $(R \cap S)\breve{} = R\breve{} \cap S\breve{}$
Theorem "Converse of ⨾": $(R ⨾ S)\breve{} = S\breve{} ⨾ R\breve{}$
Theorem "Converse of $\mathbb{I}$": $\mathbb{I}\breve{} = \mathbb{I}$
Theorem "Isotonicity of $\breve{}$": $R \subseteq S \;\equiv\; R\breve{} \subseteq S\breve{}$
Theorem "Converse of ∪": $(R \cup S)\breve{} = R\breve{} \cup S\breve{}$
Theorem "Distributivity of ⨾ over ∪": $Q ⨾ (R \cup S) = Q ⨾ R \cup Q ⨾ S$
Theorem "Sub-distributivity of ⨾ over ∩": $Q ⨾ (R \cap S) \subseteq Q ⨾ R \cap Q ⨾ S$
Theorem "Left-identity of ⨾" "Identity of ⨾": $\mathbb{I} ⨾ R = R$
Theorem "Right-identity of ⨾" "Identity of ⨾": $R ⨾ \mathbb{I} = R$
Theorem "Associativity of ⨾": $(Q ⨾ R) ⨾ S = Q ⨾ (R ⨾ S)$
Theorem "Distributivity of ⨾ over ∪": $(Q \cup R) ⨾ S = Q ⨾ S \cup R ⨾ S$
Theorem "Sub-distributivity of ⨾ over ∩": $(Q \cap R) ⨾ S \subseteq Q ⨾ S \cap R ⨾ S$
Theorem "Monotonicity of ⨾": $Q \subseteq R \Rightarrow Q ⨾ S \subseteq R ⨾ S$
Theorem "Converse of {}": $\{\}\breve{} = \{\}$
Theorem "Co-difunctionality" "Hesitation": $R \subseteq R ⨾ R\breve{} ⨾ R$
Theorem "Modal rule": $(Q ⨾ R) \cap S \subseteq Q ⨾ (R \cap Q\breve{} ⨾ S)$
Theorem "Dedekind rule": $(Q ⨾ R) \cap S \subseteq (Q \cap S ⨾ R\breve{}) ⨾ (R \cap Q\breve{} ⨾ S)$
Theorem "Schröder": $Q ⨾ R \subseteq S \;\equiv\; \sim S ⨾ R\breve{} \subseteq \sim Q$

Relation Algebra — Overview of Important Operatioons and Laws

- For any two types B and C, on the type $B \leftrightarrow C$ of **relations between B and C** we have the ordering $\subseteq$ with:
  - binary minima _∩_ and maxima _∪_ (which are monotonic)
  - least relation {} and largest ("universal") relation **U**
  - complement operation ~_ such that $R \cap \sim R = \{\}$ and $R \cup \sim R = \mathbf{U}$
  - relative pseudo-complement $R \Rightarrow S = \sim R \cup S$
- The composition operation _⨾_
  - is defined on any two relations $R : B \leftrightarrow C_1$ and $S : C_2 \leftrightarrow D$ iff $C_1 = C_2$
  - is associative, monotonic, and has identities $\mathbb{I}$
  - distributes over union: $Q ⨾(R \cup S) = Q ⨾ R \cup Q ⨾ S$
- The converse operation $\_\breve{}$
  - maps relation $R : B \leftrightarrow C$ to $R\breve{} : C \leftrightarrow B$
  - is self-inverse ($R\breve{}\,\breve{} = R$) and monotonic
  - is contravariant wrt. composition: $(R ⨾ S)\breve{} = S\breve{} ⨾ R\breve{}$
- The Dedekind rule holds: $Q ⨾ R \cap S \subseteq (Q \cap S ⨾ R\breve{}) ⨾ (R \cap Q\breve{} ⨾ S)$
- The Schröder equivalences hold:
  $Q ⨾ R \subseteq S \;\equiv\; Q\breve{} ⨾ \sim S \subseteq \sim R$ and $Q ⨾ R \subseteq S \;\equiv\; \sim S ⨾ R\breve{} \subseteq \sim Q$
- ⨾ has left-residuals $S \,/\, R = \sim(\sim S ⨾ R\breve{})$ and right-residuals $Q \setminus S = \sim(Q\breve{} ⨾ \sim S)$

Recall: Monotonicity of Relation Composition

Relation composition is monotonic in both arguments:

$$Q \subseteq R \;\Rightarrow\; Q ⨾ S \subseteq R ⨾ S$$
$$Q \subseteq R \;\Rightarrow\; P ⨾ Q \subseteq P ⨾ R$$

We could prove this via **"Relation inclusion"** *and* **"For any"**, *but we don't need to:*

Assume $Q \subseteq R$, which by (11.45) is equivalent to $Q \cup R = R$:

Proving $Q ⨾ S \subseteq R ⨾ S$:

$R ⨾ S$
= ⟨ Assumption $Q \cup R = R$ ⟩
$(Q \cup R) ⨾ S$
= ⟨ (14.23) Distributivity of ⨾ over ∪ ⟩
$Q ⨾ S \cup R ⨾ S$
⊇ ⟨ (11.31) Strengthening $S \subseteq S \cup T$ ⟩
$Q ⨾ S$

Relation-Algebraic Proof of Sub-Distributivity

Use set-algebraic properties and **Monotonicity of ⨾:** $Q \subseteq R \Rightarrow P ⨾ Q \subseteq P ⨾ R$

to prove: **Subdistributivity of ⨾ over ∩:** $Q ⨾ (R \cap S) \subseteq (Q ⨾ R) \cap (Q ⨾ S)$

$Q ⨾ (R \cap S)$

= ⟨ Idempotence of ∩ (11.35) ⟩

$(Q ⨾ (R \cap S)) \cap (Q ⨾ (R \cap S))$

⊆ ⟨ **Mon. of ∩ with Mon. of ⨾ with** Weakening $X \cap Y \subseteq X$ ⟩

$(Q ⨾ (R \cap S)) \cap (Q ⨾ S)$

⊆ ⟨ **Mon. of ∩ with Mon. of ⨾ with** Weakening $X \cap Y \subseteq X$
— without two-sided monotonicity,
separate ⊆-steps are needed in CALCCHECK! ⟩

$(Q ⨾ R) \cap (Q ⨾ S)$

Recall: Properties of Heterogeneous Relations

A relation $R : B \leftrightarrow C$ is called:

univalent determinate	$R^{\smile} ⨾ R \subseteq \mathbb{I}$	$\forall b, c_1, c_2 \bullet b (R) c_1 \land b (R) c_2 \Rightarrow c_1 = c_2$
total	$Dom\ R = B$ $\mathbb{I} \subseteq R ⨾ R^{\smile}$	$\forall b : B \bullet (\exists c : C \bullet b (R) c)$
injective	$R ⨾ R^{\smile} \subseteq \mathbb{I}$	$\forall b_1, b_2, c \bullet b_1 (R) c \land b_2 (R) c \Rightarrow b_1 = b_2$
surjective	$Ran\ R = C$ $\mathbb{I} \subseteq R^{\smile} ⨾ R$	$\forall c : C \bullet (\exists b : B \bullet b (R) c)$
a **mapping**	iff it is univalent and total	
bijective	iff it is injective and surjective	

Univalent relations are also called **(partial) functions**.

Mappings are also called **total functions**.

For Univalent Relations, Sub-distributivity turns into Distributivity

If $F : A \leftrightarrow B$ is univalent, then $F ⨾ (R \cap S) = (F ⨾ R) \cap (F ⨾ S)$

Proof: From sub-distributivity we have ⊆; because of antisymmetry of ⊆ (11.57) we only need to show ⊇:

Assume that F is univalent, that is, $F^{\smile} ⨾ F \subseteq \mathbb{I}$

$(F ⨾ R) \cap (F ⨾ S)$

⊆ ⟨ **"Modal rule"** $Q ⨾ R \cap S \subseteq Q ⨾ (R \cap Q^{\smile} ⨾ S)$ ⟩

$F ⨾ (R \cap (F^{\smile} ⨾ F ⨾ S))$

⊆ ⟨ **"Mon. of ⨾" with "Mon. of ∩" with "Mon. of ⨾" with** assumption `$F^{\smile} ⨾ F \subseteq \mathbb{I}$` ⟩

$F ⨾ (R \cap (\mathbb{I} ⨾ S))$

= ⟨ "Identity of ⨾" ⟩

$F ⨾ (R \cap S)$

Composition with Univalent Distributes over Intersection: In Diagrams

$(F ⨟ R) \cap (F ⨟ S)$

$\subseteq$ ⟨ **“Modal rule”** $Q ⨟ R \cap S \quad \subseteq \quad Q ⨟ (R \cap Q^{\smile} ⨟ S)$ ⟩

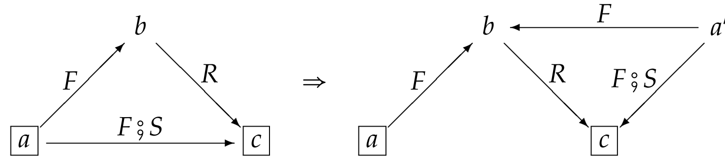
$F ⨟ (R \cap (F^{\smile} ⨟ F ⨟ S))$

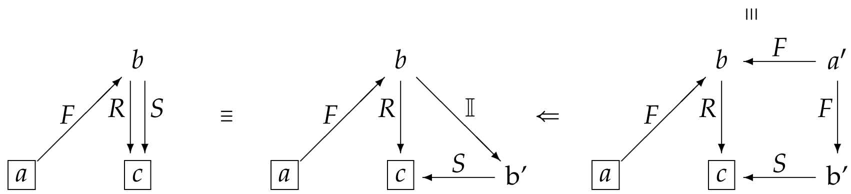
$\subseteq$ ⟨ **“Mon. of ⨟” with “Mon. of ∩” with “Mon. of ⨟” with** assumption `$F^{\smile} ⨟ F \subseteq \mathbb{I}$` ⟩

$F ⨟ (R \cap (\mathbb{I} ⨟ S))$

$=$ ⟨ “Identity of ⨟” ⟩

$F ⨟ (R \cap S)$





New Keywords: Monotonicity and Antitonicity

If $F : A \leftrightarrow B$ is univalent, then $F ⨟ (R \cap S) = (F ⨟ R) \cap (F ⨟ S)$

Proof: From sub-distributivity we have $\subseteq$; because of antisymmetry of $\subseteq$ (11.57) we only need to show $\supseteq$:

Assume that F is univalent, that is, $F^{\smile} ⨟ F \subseteq \mathbb{I}$

$(F ⨟ R) \cap (F ⨟ S)$

$\subseteq$ ⟨ **“Modal rule”** $Q ⨟ R \cap S \quad \subseteq \quad Q ⨟ (R \cap Q^{\smile} ⨟ S)$ ⟩

$F ⨟ (R \cap (F^{\smile} ⨟ F ⨟ S))$

$\subseteq$ ⟨ **Monotonicity with** assumption `$F^{\smile} ⨟ F \subseteq \mathbb{I}$` ⟩

$F ⨟ (R \cap (\mathbb{I} ⨟ S))$

$=$ ⟨ “Identity of ⨟” ⟩

$F ⨟ (R \cap S)$

Inverses are Defined from Composition and Identities

Definition: Let B and C be types, and $f : B \leftrightarrow C$ be a relation.

An **inverse of** f is a relation $g : C \leftrightarrow B$ such that $f ⨟ g = \mathbb{I}$ and $g ⨟ f = \mathbb{I}$.

Theorems:

- f has an inverse iff f is a **bijective mapping**.
- The inverse of a bijective mapping f is its converse $f^{\smile}$.

Note:

“Inverse” should always be defined this way, based on an associative composition with identities.

In such a context, if f has an inverse, it is also called an **isomorphism**.

(Ad-hoc “definitions of inverse” produce a moral proof obligation of the inverse properties.
Without these, one runs the risk of inducing strange theories…)

In particular: Converse of relations does **in general** **not** produce inverses.

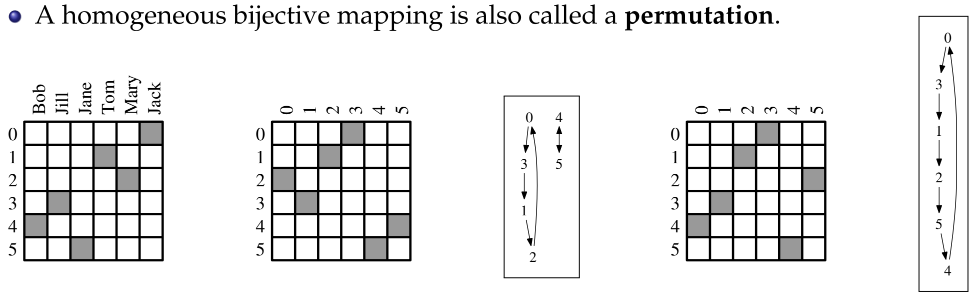
Inverses of Total Functions — Between Sets

We write "$f \in S_1 \mathrel{-\!\!\!\!\circ\!\!\!\!\rightarrow} S_2$" for "$f$ is a mapping fron S_1 to S_2" — $Dom\ f = S_1 \ \wedge\ f^\smile ⨾ f \subseteq \mathrm{id}\ S_2$

(14.43) **Definition:** Let f with $f \in S_1 \mathrel{-\!\!\!\!\circ\!\!\!\!\rightarrow} S_2$ be a **mapping** from S_1 to S_2.
An **inverse of** f is a mapping g from S_2 to S_1 such that $f ⨾ g = \mathrm{id}\ S_1$ and $g ⨾ f = \mathrm{id}\ S_2$.

Still:

- f has an inverse iff f is a bijective mapping.
- The inverse of a bijective mapping f is its converse $f^\smile$.
- A homogeneous bijective mapping is also called a **permutation**.

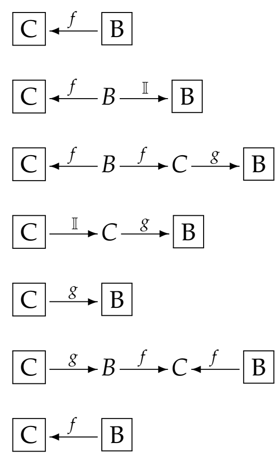


Inverses of Total Functions — Between Types

(14.43t) **Definition:** Let B and C be types, and $f : B \leftrightarrow C$ be a **mapping**.
An **inverse of** f is a mapping $g : C \leftrightarrow B$ such that $f ⨾ g = \mathbb{I} = \mathrm{id}\ \llcorner B \lrcorner$ and $g ⨾ f = \mathbb{I} = \mathrm{id}\ \llcorner C \lrcorner$.

Theorem: If g is an inverse of a mapping $f : B \to C$, then $g = f^\smile$.
Proof: (Using antisymmetry of $\subseteq$)

$f^\smile$
$=$ ⟨ Identity of ⨾ ⟩
$f^\smile ⨾ \mathbb{I}$
$=$ ⟨ g is an inverse of f ⟩
$f^\smile ⨾ f ⨾ g$
$\subseteq$ ⟨ **Mon. of ⨾ with** f is univalent, that is, $f^\smile ⨾ f \subseteq \mathbb{I}$ ⟩
$\mathbb{I} ⨾ g$
$=$ ⟨ Identity of ⨾ ⟩
g
$\subseteq$ ⟨ Identity of ⨾, **Mon. of ⨾ with** f is total, that is, $\mathbb{I} \subseteq f ⨾ f^\smile$ ⟩
$g ⨾ f ⨾ f^\smile$
$=$ ⟨ g is an inverse of f; Identity of ⨾ ⟩
$f^\smile$



Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-07

Relation-Algebraic Calculational Proofs (ctd.)

Recall: Properties of Homogeneous Relations

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \,(R)\, b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall\, b : B \bullet \neg(b \,(R)\, b))$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : B \bullet b \,(R)\, c \equiv c \,(R)\, b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall\, b, c \bullet b \,(R)\, c \wedge c \,(R)\, b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall\, b, c : B \bullet b \,(R)\, c \Rightarrow \neg(c \,(R)\, b))$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \wedge c \,(R)\, d \Rightarrow b \,(R)\, d)$

R is an **equivalence (relation) on** B iff it is reflexive, transitive, and symmetric. (E.g., $=$, $\equiv$)

R is a **(partial) order on** B
iff it is reflexive, transitive, and antisymmetric.
(E.g., $\leq$, $\geq$, $\subseteq$, $\supseteq$, $|$)

R is a **strict-order on** B
iff it is irreflexive, transitive, and asymmetric.
(E.g., $<$, $>$, $\subset$, $\supset$)

Homogeneous Relation Properties are Preserved by Converse

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \,(R)\, b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall\, b : B \bullet \neg(b \,(R)\, b))$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : B \bullet b \,(R)\, c \equiv c \,(R)\, b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall\, b, c \bullet b \,(R)\, c \wedge c \,(R)\, b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall\, b, c : B \bullet b \,(R)\, c \Rightarrow \neg(c \,(R)\, b))$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \,(R)\, d \Rightarrow b \,(R)\, d)$
idempotent	$R \fatsemi R = R$	

Theorem: If $R : B \leftrightarrow B$ is reflexive/irreflexive/symmetric/antisymmetric/asymmetric/transitive/idempotent, then $R^{\smile}$ has that property, too.

Proof: Reflexivity:

$R^{\smile}$
$\supseteq$ ⟨ **Mon.** $^{\smile}$ **with** Reflexivity of R ⟩
$\mathbb{I}^{\smile}$
$=$ ⟨ Symmetry of $\mathbb{I}$ ⟩
$\mathbb{I}$

Transitivity:

$R^{\smile} \fatsemi R^{\smile}$
$=$ ⟨ Converse of $\fatsemi$ ⟩
$(R \fatsemi R)^{\smile}$
$\subseteq$ ⟨ **Mon.** $^{\smile}$ **with** Trans. of R ⟩
$R^{\smile}$

Reflexive and Transitive Implies Idempotent

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \,(R)\, b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \,(R)\, d \Rightarrow b \,(R)\, d)$
idempotent	$R \fatsemi R = R$	

Theorem: If $R : B \leftrightarrow B$ is reflexive and transitive, then it is also idempotent.

Reflexive and Transitive Implies Idempotent — Direct Approach

reflexive	$\mathbb{I} \subseteq R$
transitive	$R ⨾ R \subseteq R$
idempotent	$R ⨾ R = R$

Theorem "Idempotency from reflexive and transitive":
reflexive R ⇒ transitive R ⇒ idempotent R
Proof:
Assuming \`reflexive $R$\`, \`transitive $R$\`:
idempotent R
≡ ⟨ "Definition of idempotency" ⟩
$R ⨾ R = R$
≡ ⟨ "Mutual inclusion" ⟩
$R ⨾ R \subseteq R \wedge R \subseteq R ⨾ R$
≡ ⟨ "Definition of transitivity", assumption \`transitive $R$\`, "Identity of ∧" ⟩
$R \subseteq R ⨾ R$
≡ ⟨ "Identity of ⨾" ⟩
$R ⨾ \mathbb{I} \subseteq R ⨾ R$
⇐⟨ "Monotonicity of ⨾" ⟩
$\mathbb{I} \subseteq R$
≡ ⟨ Assumption \`reflexive $R$\` with "Definition of reflexivity" ⟩
true

Reflexive and Transitive Implies Idempotent — "and using with"

reflexive	$\mathbb{I} \subseteq R$
transitive	$R ⨾ R \subseteq R$
idempotent	$R ⨾ R = R$

Theorem "Idempotency from reflexive and transitive":
reflexive R ⇒ transitive R ⇒ idempotent R
Proof:
Assuming \`reflexive $R$\` **and using with** "Definition of reflexivity",
\`transitive $R$\` **and using with** "Definition of transitivity":
idempotent R
≡ ⟨ "Definition of idempotency" ⟩
$R ⨾ R = R$
≡ ⟨ "Mutual inclusion" ⟩
$R ⨾ R \subseteq R \wedge R \subseteq R ⨾ R$
≡ ⟨ Assumption \`transitive $R$\`, "Identity of ∧" ⟩
$R \subseteq R ⨾ R$
≡ ⟨ "Identity of ⨾" ⟩
$R ⨾ \mathbb{I} \subseteq R ⨾ R$
⇐⟨ "Monotonicity of ⨾" ⟩
$\mathbb{I} \subseteq R$
≡ ⟨ Assumption \`reflexive $R$\` ⟩
true

Reflexive and Transitive Implies Idempotent — Semi-formal

reflexive	$\mathbb{I} \subseteq R$	$(\forall b : B \bullet b (R) b)$
transitive	$R ⨾ R \subseteq R$	$(\forall b, c, d \bullet b (R) c (R) d \Rightarrow b (R) d)$
idempotent	$R ⨾ R = R$	

Theorem: If $R : B \leftrightarrow B$ is reflexive and transitive, then it is also idempotent.

Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R ⨾ R$:

R
= ⟨ Identity of ⨾ ⟩
$R ⨾ \mathbb{I}$
⊆ ⟨ **Mon. ⨾ with** Reflexivity of R ⟩
$R ⨾ R$

Reflexive and Transitive Implies Idempotent — Cyclic $\subseteq$-chain Proving \` = \`

reflexive	$\mathbb{I}$	$\subseteq$	R
transitive	$R ⨾ R$	$\subseteq$	R
idempotent	$R ⨾ R$	$=$	R

Theorem "Idempotency from reflexive and transitive":
reflexive R $\Rightarrow$ transitive R $\Rightarrow$ idempotent R

Proof:

Assuming \`reflexive $R$\` **and using with** "Definition of reflexivity",
\`transitive $R$\` **and using with** "Definition of transitivity":

Using "Definition of idempotency":

Subproof for \`$R ⨾ R = R$\`:

$R ⨾ R$

$\subseteq$ ⟨ Assumption \`transitive $R$\` ⟩

R

$=$ ⟨ "Identity of ⨾" ⟩

$R ⨾ \mathbb{I}$

$\subseteq$ ⟨ "Monotonicity of ⨾" with assumption \`reflexive $R$\` ⟩

$R ⨾ R$

Using cyclic $\subseteq$-chains to prove equalities requires activation of antisymmetry of $\subseteq$.

Most Homogeneous Relation Properties are Preserved by Intersection

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R ⨾ R$	$\subseteq$	R
idempotent	$R ⨾ R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric/antisymmetric/asymmetric/transitive, then $R \cap S$ has that property, too.

Proof: Reflexivity:

$R \cap S$

$\supseteq$ ⟨ Mon. of $\cap$ with Refl. S ⟩

$R \cap \mathbb{I}$

$\supseteq$ ⟨ Mon. of $\cap$ with Refl. R ⟩

$\mathbb{I} \cap \mathbb{I}$

$=$ ⟨ Idempotence of $\cap$ ⟩

$\mathbb{I}$

Transitivity:

$(R \cap S) ⨾ (R \cap S)$

$\subseteq$ ⟨ Sub-distributivity of ⨾ over $\cap$ ⟩

$(R ⨾ R) \cap (R ⨾ S) \cap (S ⨾ R) \cap (S ⨾ S)$

$\subseteq$ ⟨ Weakening $X \cap Y \subseteq X$ ⟩

$(R ⨾ R) \cap (S ⨾ S)$

$\subseteq$ ⟨ Mon. $\cap$ with transitivity of R and S ⟩

$R \cap S$

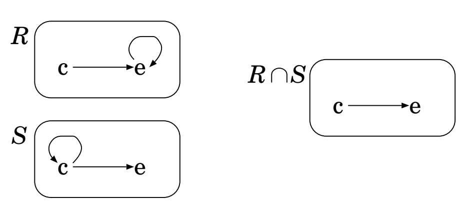
Most Homogeneous Relaton Properties are Preserved by Intersection

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R ⨾ R$	$\subseteq$	R
idempotent	$R ⨾ R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric/antisymmetric/asymmetric/transitive, then $R \cap S$ has that property, too.

Counter-example for preservation of idempotence:



Some Homogeneous Relation Properties are Preserved by Union

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric, then $R \cup S$ has that property, too.

Proof:

Reflexivity:

$$\begin{array}{ll} & \mathbb{I} \\ \subseteq & \langle \text{ Reflexivity of } R \text{ } \rangle \\ & R \\ \subseteq & \langle \text{ Weakening } X \subseteq X \cup Y \text{ } \rangle \\ & R \cup S \end{array}$$

Irreflexivity:

$$\begin{array}{ll} & \mathbb{I} \cap (R \cup S) \\ = & \langle \text{ Distributivity of } \cap \text{ over } \cup \text{ } \rangle \\ & (\mathbb{I} \cap R) \cup (\mathbb{I} \cap S) \\ = & \langle \text{ Irreflexivity of } R \text{ and } S \text{ } \rangle \\ & \{\} \cup \{\} \\ = & \langle \text{ Idempotence of } \cup \text{ } \rangle \\ & \{\} \end{array}$$

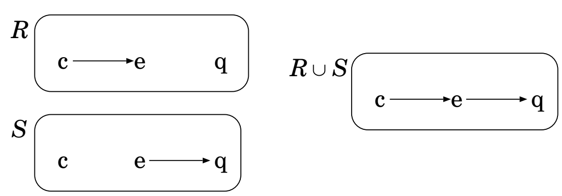
Some Homogeneous Relation Properties are Preserved by Union

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric, then $R \cup S$ has that property, too.

Counter-example for preservation of transitivity:



Weaker Formulation of Symmetry

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

For proving symmetry of $R, S : B \leftrightarrow B$, it is sufficient to prove $R^{\smile} \subseteq R$.

In other words:

Theorem: If $R^{\smile} \subseteq R$, then $R^{\smile} = R$.

Proof: By mutual inclusion, we only need to show $R \subseteq R^{\smile}$:

$$\begin{array}{ll} & R \\ = & \langle \text{ Self-inverse of converse } \rangle \\ & (R^{\smile})^{\smile} \\ \subseteq & \langle \text{ Mon. of } {}^{\smile} \text{ with Assumption } R^{\smile} \subseteq R \text{ } \rangle \\ & R^{\smile} \end{array}$$

Symmetric and Transitive Implies Idempotent

symmetric	$R^{\smile} \;=\; R$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
transitive	$R \fatsemi R \;\subseteq\; R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R \;=\; R$	

Modal rule:
$Q \fatsemi R \cap S \quad \subseteq \quad Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$

Theorem: A symmetric and transitive $R : B \leftrightarrow B$ is also idempotent.
Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R \fatsemi R$:

R
$=$ ⟨ Idempotence of $\cap$, Identity of $\fatsemi$ ⟩
$R \fatsemi \mathbb{I} \cap R$
$\subseteq$ ⟨ **Modal rule** $\quad Q \fatsemi R \cap S \quad \subseteq \quad Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$ ⟩
$R \fatsemi (\mathbb{I} \cap R^{\smile} \fatsemi R)$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ ⟩
$R \fatsemi R^{\smile} \fatsemi R$
$=$ ⟨ Symmetry of R ⟩
$R \fatsemi R \fatsemi R$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Transitivity of R ⟩
$R \fatsemi R$

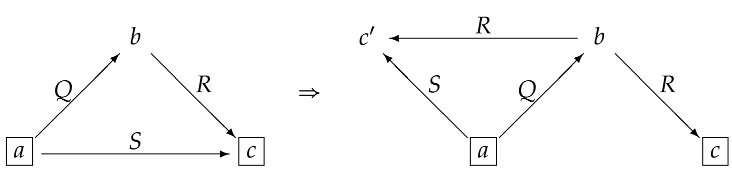
Symmetric and Transitive Implies Idempotent

symmetric	$R^{\smile} \;=\; R$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
transitive	$R \fatsemi R \;\subseteq\; R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R \;=\; R$	

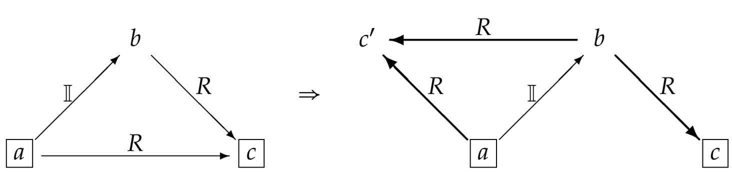
Theorem: A symmetric and transitive $R : B \leftrightarrow B$ is also idempotent.
Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R \fatsemi R$:

R
$=$ ⟨ Idempotence of $\cap$, Identity of $\fatsemi$ ⟩
$\mathbb{I} \fatsemi R \cap R$
$\subseteq$ ⟨ **Modal rule** $\quad Q \fatsemi R \cap S \quad \subseteq \quad (Q \cap S \fatsemi R^{\smile}) \fatsemi R)$ ⟩
$(\mathbb{I} \cap R \fatsemi R^{\smile}) \fatsemi R$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ ⟩
$R \fatsemi R^{\smile} \fatsemi R$
$=$ ⟨ Symmetry of R ⟩
$R \fatsemi R \fatsemi R$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Transitivity of R ⟩
$R \fatsemi R$

Modal Rule for "Symmetric and Transitive Implies Idempotent"



$\mathbb{I} \fatsemi R \cap R$
$\subseteq$ ⟨ **Modal rule** $\quad Q \fatsemi R \cap S \quad \subseteq \quad (Q \cap S \fatsemi R^{\smile}) \fatsemi R)$ ⟩
$(\mathbb{I} \cap R \fatsemi R^{\smile}) \fatsemi R$



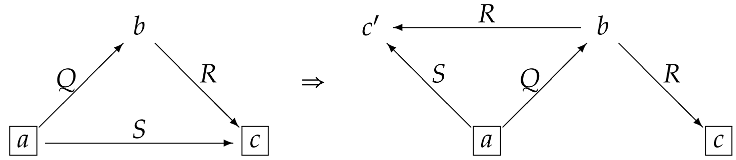
Modal Rules— Converse as Over-Approximation of Inverse

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \;\subseteq\; Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$$
$$Q \fatsemi R \cap S \;\subseteq\; (Q \cap S \fatsemi R^{\smile}) \fatsemi R$$

Useful to "**make information available locally**" (Q is replaced with $Q \cap S \fatsemi R^{\smile}$ for use in further proof steps.

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \Rightarrow$
$\quad (\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c')$

Modal Rules modulo Inclusion via Intersection

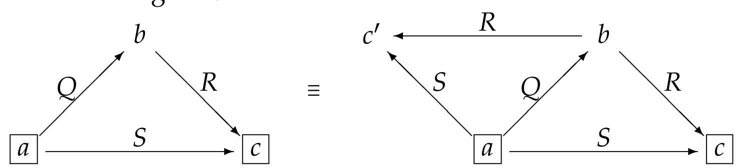
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \;\subseteq\; Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$$
$$Q \fatsemi R \cap S \;\subseteq\; (Q \cap S \fatsemi R^{\smile}) \fatsemi R$$

Equivalently, using $M \subseteq N \;\equiv\; M = M \cap N$ etc.:

$$Q \fatsemi R \cap S \;=\; Q \fatsemi (R \cap Q^{\smile} \fatsemi S) \cap S$$
$$Q \fatsemi R \cap S \;=\; (Q \cap S \fatsemi R^{\smile}) \fatsemi R \cap S$$

In **constraint** diagrams:



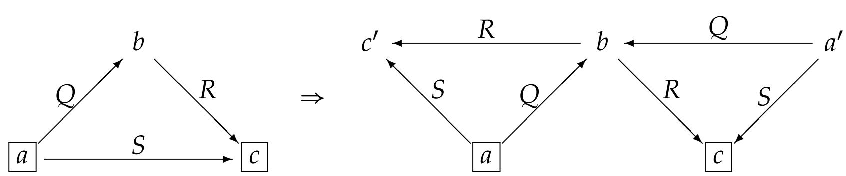
$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \equiv$
$\equiv \quad (\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c' \wedge b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c)$

Modal Rules and Dedekind Rule

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \;\subseteq\; Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$$
$$Q \fatsemi R \cap S \;\subseteq\; (Q \cap S \fatsemi R^{\smile}) \fatsemi R$$

Equivalent: **Dedekind Rule:** $Q \fatsemi R \cap S \;\subseteq\; (Q \cap S \fatsemi R^{\smile}) \fatsemi (R \cap Q^{\smile} \fatsemi S)$



Dedekind Rule modulo Inclusion via Intersection

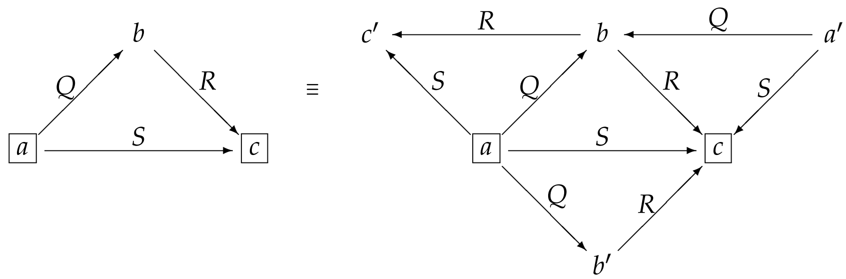
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Equivalent: **Dedekind Rule:** $$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$

Equivalently, via $M \subseteq N \;\equiv\; M = M \cap N$:

$$Q \mathbin{;} R \cap S \;=\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S) \cap (S \cap Q \mathbin{;} R)$$



Modal Rules and Dedekind Rule: Summary with Sharp Versions

For all $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

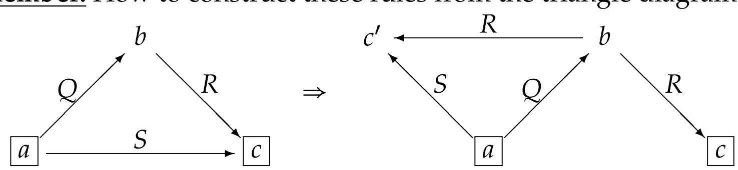
Modal rules:
$$Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Modal rules (sharp versions):
$$Q \mathbin{;} R \cap S \;=\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S) \cap S$$
$$Q \mathbin{;} R \cap S \;=\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \cap S$$

Dedekind: $$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
Dedekind (sharp version): $$Q \mathbin{;} R \cap S \;=\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S) \cap S$$

Proofs: Exercise!

Remember: How to construct these rules from the triangle diagram set-up!



Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

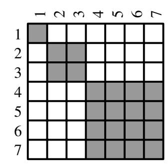
2024-11-08

Part 1: Equivalence Relations

Recall: Equivalence Relations

Recall: A (homogeneous) relation $R : t \leftrightarrow t$ is called:

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : t \bullet b \,(R)\, b)$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : t \bullet b \,(R)\, c \equiv c \,(R)\, b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \,(R)\, d \Rightarrow b \,(R)\, d)$
idempotent	$R \fatsemi R = R$	
equivalence	$\mathbb{I} \subseteq R = R \fatsemi R = R^{\smile}$	reflexive, transitive, symmetric

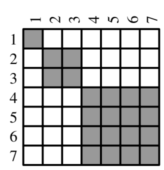


Equivalence Relations on a Set

Recall: $B \leftrightarrow B = \mathbb{P}\,(B \times B)$ is the **set** of relations on the **set** B.

Given a **set** B, a (homogeneous) relation R with $R \in B \leftrightarrow B$ is called:

reflexive on B	$\mathrm{id}\, B \subseteq R$	$(\forall\, b \mid b \in B \bullet b \,(R)\, b)$
symmetric	$R^{\smile} = R$	$(\forall\, b, c \bullet b \,(R)\, c \equiv c \,(R)\, b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \,(R)\, d \Rightarrow b \,(R)\, d)$
idempotent	$R \fatsemi R = R$	
equivalence on B	$\mathrm{id}\, B \subseteq R = R \fatsemi R = R^{\smile}$ $\wedge \quad R \subseteq B \times B$	reflexive on B, transitive, symmetric restricted to B



— is an equivalence on $\{1, 2, 3, 4, 5, 6, 7\}$

— is **not** reflexive on $\mathbb{Z}$ or `int`

Note: If $B \neq \mathbf{U}$, then no R from $B \leftrightarrow B$ is reflexive in the sense of "$\mathbb{I} \subseteq R$"!

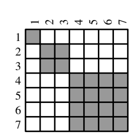
Equivalence Classes, Partitions

Definition (14.34): Let Ξ be an equivalence relation on B. Then $[b]_\Xi$. the **equivalence class of** b, is the subset of elements of B that are equivalent (under Ξ) to b:

$$x \in [b]_\Xi \quad \equiv \quad x \,(\Xi)\, b \qquad \text{Equivalently:} \qquad [b]_\Xi = \Xi\,(\!|\,\{b\}\,|\!)$$

Theorem: For an equivalence relation Ξ on B, the set $B|_\Xi = \{\, b \mid b \in B \bullet \Xi\,(\!|\,\{b\}\,|\!)\,\}$ of equivalence classes of Ξ is a partition of B.

$$\{\, \{1\},\ \{2, 3\},\ \{4, 5, 6, 7\}\, \}$$



Definition (11.76): If $\;T : \mathbf{set}\ t\;$ and $\;S : \mathbf{set}\ (\mathbf{set}\ t)$, then:

$$\begin{aligned} S \text{ is a } \textbf{partition of } T \quad \equiv \quad & (\forall u, v \mid u \in S \wedge v \in S \wedge u \neq v \bullet u \cap v = \{\}) \\ \wedge \quad & (\bigcup u \mid u \in S \bullet u) = T \end{aligned}$$

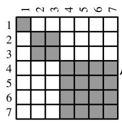
Theorem: $B|\_$ is a bijective mapping
between equivalence relations on B and partitions of B.

The partition view can be useful for **implementing** equivalence relations.

Equivalence Quotients

For an equivalence relation Ξ on B, the set $B|_\Xi = \{\ b : B\ \bullet\ [b]_\Xi\ \}$ of equivalence classes of Ξ is also called **quotient of B via Ξ**.

The mapping $\chi = \{\ b\ \mid\ b \in B\ \bullet\ \langle b, [b]_\Xi \rangle\ \}$ is the **quotient projection**.

For $\Xi =$ , $\chi \in \{1,2,3,4,5,6,7\} \twoheadrightarrow \{\ \{1\},\ \{2,3\},\ \{4,5,6,7\}\ \}$

$$\chi = \{1 \mapsto \{1\},\ 2 \mapsto \{2,3\},\ 3 \mapsto \{2,3\},\ 4 \mapsto \{4,5,6,7\},\ 5 \mapsto \{4,5,6,7\},\ 6 \mapsto \{4,5,6,7\},\ 7 \mapsto \{4,5,6,7\}\ \}$$

χ satisfies:

- $\chi\breve{}\ ; \chi = \mathrm{id}\ B|_\Xi$ — univalent, and surjective onto $B|_\Xi$
- $\chi ; \chi\breve{} = \Xi$ — therefore total on B, since Ξ is reflexive on B

The quotient together with the quotient projection is **determined uniquely up to isomorphism** by these two properties…

Specification of Quotient Projections Up To Isomorphism

For an equivalence relation Ξ on B, consider:

- the **quotient set** $B|_\Xi = \{\ b\ \mid\ b \in B\ \bullet\ [b]_\Xi \}$
- the **quotient projection** $\chi \in B \twoheadrightarrow B|_\Xi$ with $\chi = \{\ b\ \mid\ b \in B\ \bullet\ \langle b, [b]_\Xi \rangle\ \}$

Then we have $\chi\breve{}\ ; \chi = \mathrm{id}\ B|_\Xi$ and $\chi ; \chi\breve{} = \Xi$.

$\chi \in \{1,2,3,4,5,6,7\} \twoheadrightarrow \{\ \{1\},\ \{2,3\},\ \{4,5,6,7\}\ \}$
$\gamma \in \{1,2,3,4,5,6,7\} \twoheadrightarrow \{\ 'a', 'b', 'c'\}$

$$\chi = \{1 \mapsto \{1\},\ 2 \mapsto \{2,3\},\ 3 \mapsto \{2,3\},\ 4 \mapsto \{4,5,6,7\},\ 5 \mapsto \{4,5,6,7\},\ 6 \mapsto \{4,5,6,7\},\ 7 \mapsto \{4,5,6,7\}\ \}$$

$$\gamma = \{1 \mapsto 'a',\ 2 \mapsto 'b',\ 3 \mapsto 'b',\ 4 \mapsto 'c',\ 5 \mapsto 'c',\ 6 \mapsto 'c',\ 7 \mapsto 'c'\ \}$$

Also consider:

- an "alternate quotient set candidate" Q
- an "alternate quotient projection candidate" $\gamma : B \leftrightarrow Q$ satisfying $\gamma\breve{}\ ; \gamma = \mathrm{id}\ Q$ and $\gamma ; \gamma\breve{} = \Xi$

Then $\varphi = \chi\breve{}\ ; \gamma$ is an isomorphism between $B|_\Xi$ and Q:

- $\varphi ; \varphi\breve{} = \chi\breve{}\ ; \gamma ; \gamma\breve{}\ ; \chi = \chi\breve{}\ ; \Xi ; \chi = \chi\breve{}\ ; \chi ; \chi\breve{}\ ; \chi = \mathrm{id}\ B|_\Xi ; \mathrm{id}\ B|_\Xi = \mathrm{id}\ B|_\Xi$ — total and injective
- $\varphi\breve{}\ ; \varphi = \gamma\breve{}\ ; \chi ; \chi\breve{}\ ; \gamma = \gamma\breve{}\ ; \Xi ; \gamma = \gamma\breve{}\ ; \gamma ; \gamma\breve{}\ ; \gamma = \mathrm{id}\ Q ; \mathrm{id}\ Q = \mathrm{id}\ Q$ — univalent and surjective

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-08

Part 2: Relational Formalisation of Graph Properties

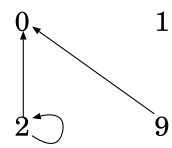
Recall: Simple Graphs

A **simple graph** (N, E) is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$, the element pairs of which are called "edges".

Example: $G_1 = (\{2, 0, 1, 9\}, \{\langle 2, 0\rangle, \langle 9, 0\rangle, \langle 2, 2\rangle\})$

Graphs are normally visualised via **graph drawings**:

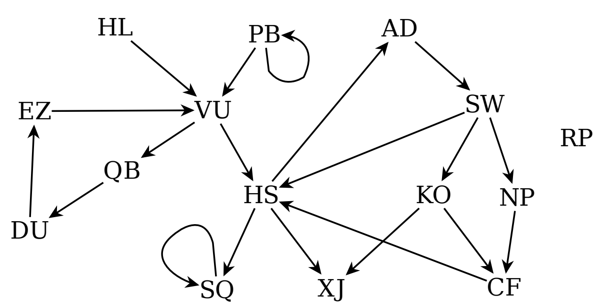


Simple graphs are exactly relations!

Reasoning with relations is reasoning about graphs!

Simple Reachability Statements in Graph $G = (V, E)$

- No edge ends at node s
  $s \notin \mathit{Ran}\ E$ or $s \in {\sim}(\mathit{Ran}\ E)$ — s is called a **source** of G
- No edge starts at node s
  $s \notin \mathit{Dom}\ E$ or $s \in {\sim}(\mathit{Dom}\ E)$ — s is called a **sink** of G
- Node n_2 is reachable from node n_1 via a three-edge path
  $n_1 \llparenthesis E \fatsemi E \fatsemi E \rrparenthesis n_2$



Simple Reachability Statements in Graph $G_{\mathbb{N}} = (\llcorner \mathbb{N} \lrcorner, \ulcorner \mathsf{suc} \urcorner)$

- No edge ends at node 0
  $0 \notin \mathit{Ran}\ \ulcorner \mathsf{suc} \urcorner$ or $0 \in {\sim}(\mathit{Ran}\ \ulcorner \mathsf{suc} \urcorner)$ — 0 is a **source** of $G_{\mathbb{N}}$

  0 is the only source of $G_{\mathbb{N}}$: ${\sim}(\mathit{Ran}\ \ulcorner \mathsf{suc} \urcorner) = \{0\}$

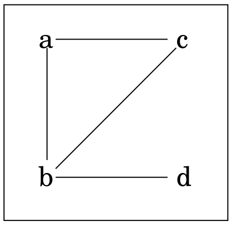
- s is a sink iff no edge starts at node s
  $s \notin \mathit{Dom}\ \ulcorner \mathsf{suc} \urcorner$ or $s \in {\sim}(\mathit{Dom}\ \ulcorner \mathsf{suc} \urcorner)$

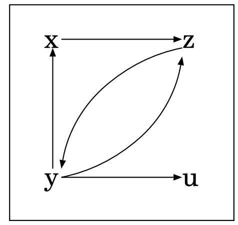
  $G_{\mathbb{N}}$ has no sinks: $\mathit{Dom}\ \ulcorner \mathsf{suc} \urcorner = \llcorner \mathbb{N} \lrcorner$ or ${\sim}(\mathit{Dom}\ \ulcorner \mathsf{suc} \urcorner) = \{\}$

- Node 5 is reachable from node 2 via a three-edge path:
  $2 \llparenthesis \ulcorner \mathsf{suc} \urcorner \fatsemi \ulcorner \mathsf{suc} \urcorner \fatsemi \ulcorner \mathsf{suc} \urcorner \rrparenthesis 5$

$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow 6 \longrightarrow 7 \longrightarrow \ldots$$

Directed versus Undirected Graphs





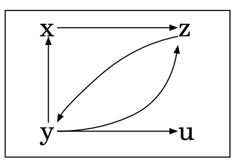
- Edges in simple undirected graphs can be considered as "unordered pairs" (two-element sets, or one-to-two-element sets)
- The **associated relation** of an undirected graph relates two nodes iff there is an edge between them
- **The associated relation of an undirected graph is always symmetric**
- In a **simple** graph, no two edges have the same source and the same target. (No "parallel edges".)
- Relations directly represent simple **directed** graphs.

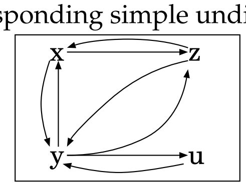
Symmetric Closure

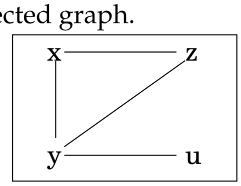
Relation $Q : B \leftrightarrow B$ is the **symmetric closure** of $R : B \leftrightarrow B$
iff Q is the smallest symmetric relation containing R,
or, equivalently, iff
- $R \subseteq Q$
- $Q = Q^{\smile}$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P = P^{\smile} \bullet Q \subseteq P)$

Theorem: The symmetric closure of $R : B \leftrightarrow B$ is $R \cup R^{\smile}$.

Fact: If R represents a simple directed graph, then the symmetric closure of R is the associated relation of the corresponding simple undirected graph.







We may draw a symmetric siumple graph as an undirected graph.
(Implicitly representing two symmetric arrows by a single edge without arrow tips.)

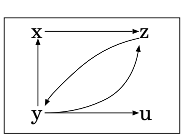
Reflexive Closure

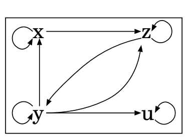
Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$
iff Q is the smallest reflexive relation containing R,

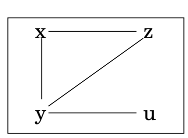
or, equivalently, iff
- $R \subseteq Q$
- $\mathbb{I} \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

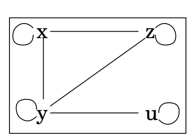
Theorem: The reflexive closure of $R : B \leftrightarrow B$ is $R \cup \mathbb{I}$.

Fact: If R represents a graph, then the reflexive closure of R "ensures that each node has a loop edge".









Transitive Closure

Relation $Q : B \leftrightarrow B$ is the **transitive closure** of $R : B \leftrightarrow B$
iff Q is the smallest transitive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $Q ⨾ Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge P ⨾ P \subseteq P \bullet Q \subseteq P)$

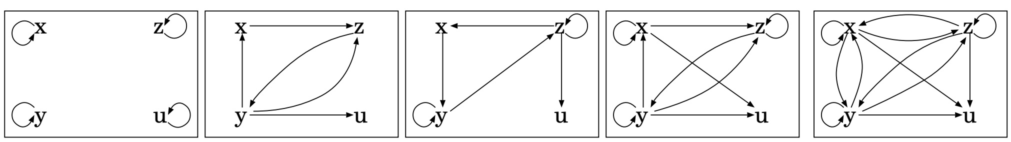
Definition: The transitive closure of $R : B \leftrightarrow B$ is written R^+.

Theorem: $R^+ = (\bigcap P \mid R \subseteq P \wedge P ⨾ P \subseteq P \bullet P)$.

Transitive Closure via Powers

Powers of a homogeneous relation $R : B \leftrightarrow B$:

- $R^0 = \mathbb{I}$
- $R^1 = R$
- $R^{n+1} = R^n ⨾ R$
- $R^2 = R ⨾ R$
- $R^3 = R ⨾ R ⨾ R$
- $R^4 = R ⨾ R ⨾ R ⨾ R$
- R^i is reachability via exactly i many R-steps



R^0 R^1 R^2 R^3 R^+

Theorem: $R^+ = (\bigcup i : \mathbb{N} \mid i > 0 \bullet R^i)$

This means:

- $R^+ = R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Transitive closure R^+ is reachability via at least one R-step

Reflexive Transitive Closure

$Q : B \leftrightarrow B$ is the **reflexive transitive closure** of $R : B \leftrightarrow B$
iff Q is the smallest reflexive transitive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q \wedge Q ⨾ Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P ⨾ P \subseteq P \bullet Q \subseteq P)$

Definition: The reflexive transitive closure of R is written R^*.

Theorem: $R^* = (\bigcap P \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P ⨾ P \subseteq P \bullet P)$.

Theorem: $R^* = (\bigcup i : \mathbb{N} \bullet R^i)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

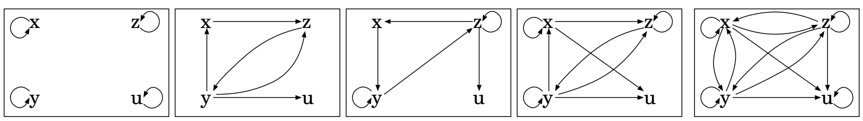
Wolfram Kahl

2024-11-12

Reachability, Closures

Reachability: Transitive and Reflexive Transitive Closure via Powers

- R^i is reachability via exactly i many R-steps



R^0 R^1 R^2 R^3 R^*

- $R^+ = (\bigcup i : \mathbb{N} \mid i > 0 \bullet R^i)$
- $R^+ = R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Transitive closure R^+ is reachability via at least one R-step

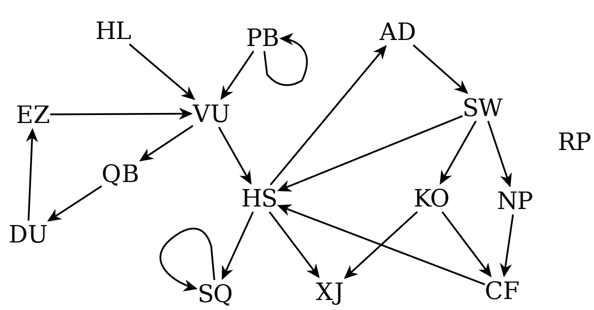
---

- $R^* = (\bigcup i : \mathbb{N} \bullet R^i)$
- $R^* = \mathbb{I} \cup R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Reflexive transitive closure R^*
  is reachability via any number of R-steps

- Variants of the **Warshall algorithm** calculate these closures in cubic time.

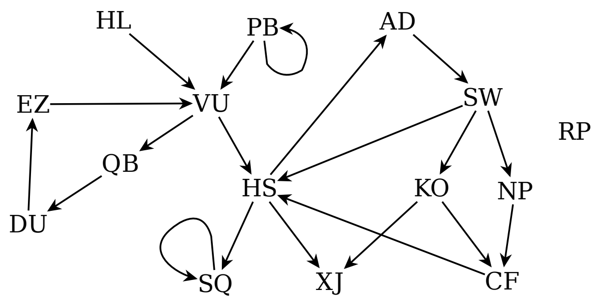
Reachability in graph $G = (V, E)$ — 1 (ctd.)

- No edge ends at node s
  $s \notin \mathit{Ran}\ E$ or $s \in {\sim}(\mathit{Ran}\ E)$ — s is called a **source** of G
- No edge starts at node s
  $s \notin \mathit{Dom}\ E$ or $s \in {\sim}(\mathit{Dom}\ E)$ — s is called a **sink** of G
- Node n_2 is reachable from node n_1 via a three-edge path
  $n_1 \llparenthesis E^3 \rrparenthesis n_2$ or $n_1 \llparenthesis E ; E ; E \rrparenthesis n_2$
- Node y is **reachable** from node x
  $x \llparenthesis E^* \rrparenthesis y$ — **reachability**



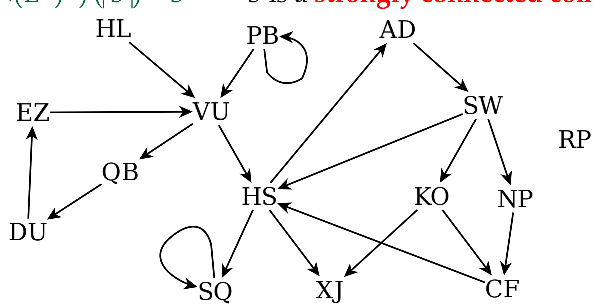
Reachability in graph $G = (V, E)$ — 2

- Node y is **reachable** from node x
  $x \mathrel{(E^*)} y$ — **reachability**
- Every node is reachable from node r
  $\{r\} \times V \subseteq E^*$ or $E^* \llparenthesis \{r\} \rrparenthesis = V$ — r is called a **root** of G
- Node y is **reachable via a non-empty path** from node x: $x \mathrel{(E^+)} y$
- Nodes x lies on a cycle: $x \mathrel{(E^+)} x$ or $x \mathrel{(E^+ \cap \mathbb{I})} x$ or $x \in Dom(E^+ \cap \mathbb{I})$



Reachability in graph $G = (V, E)$ — 3

- From every node, each node is reachable
  $V \times V \subseteq E^*$ — G is **strongly connected**
- From every node, each node is reachable by traversing edges in either direction
  $V \times V \subseteq (E \cup E^{\smile})^*$ — G is **connected**
- Nodes n_1 and n_2 reachable from each other both ways
  $n_1 \mathrel{(E^* \cap (E^*)^{\smile})} n_2$ — n_1 and n_2 are **strongly connected**
- S is an equivalence class of strong connectedness between nodes
  $S \times S \subseteq E^* \wedge (E^* \cap (E^*)^{\smile}) \llparenthesis S \rrparenthesis = S$ — S is a **strongly connected component (SCC)** of G

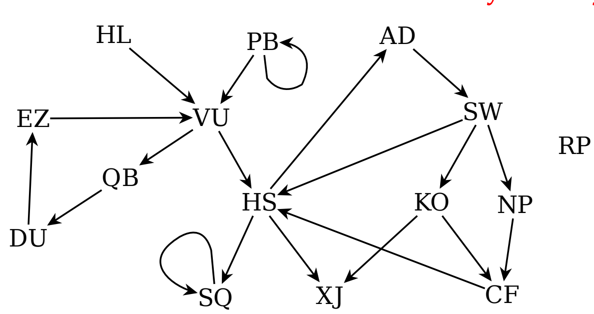


Reachability in graph $G = (V, E)$ — 4

- A node n is said to "lie on a cycle" if there is a non-empty path from n to n

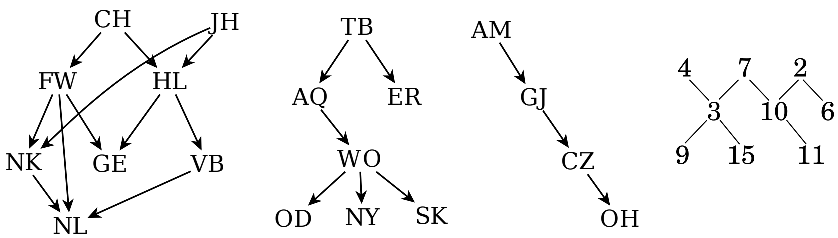
  $cycleNodes := Dom(E^+ \cap \mathbb{I})$

- No node lies on a cycle
  $Dom(E^+ \cap \mathbb{I}) = \{\}$
  $E^+ \cap \mathbb{I} = \{\}$
  E^+ is irreflexive — G is called **acyclic** or **cycle-free** or a **DAG**



Reachability in graph $G = (V, E)$ — 5 — DAGs

- No node lies on a cycle: $E^+ \cap \mathbb{I} = \{\}$ — G is a **directed acyclic graph**, or **DAG**
- Each node has at most one predecessor: $E \mathbin{;} E^{\smile} \subseteq \mathbb{I}$ or E is injective
  — if G is also acyclic, then G is called a **(directed) forest**
- Every node is reachable from node r
  $\{r\} \times V \subseteq E^*$ — if G is also a forest, then G is called a **(directed) tree**, and r is its **root**
- For undirected graphs: A tree is a graph where for each pair of nodes there is exactly one path connecting them.
  — **graph-theoretic tree concept**



Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-12

Part 2: Closures Generalised

Recall: Reflexive Closure

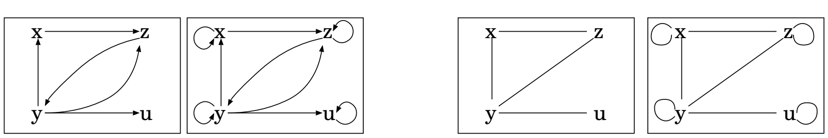
Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$
iff Q is the smallest reflexive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \land \mathbb{I} \subseteq P \bullet Q \subseteq P)$

Theorem: The reflexive closure of $R : B \leftrightarrow B$ is $R \cup \mathbb{I}$.

Fact: If R represents a graph, then the reflexive closure of R "ensures that each node has a loop edge".



Reflexive Closure Operator `reflClos`

Axiom "Definition of `reflClos` ": $\mathsf{reflClos}\ R \ = \ R \cup \mathbb{I}$

Theorem "Closure properties of `reflClos`: Expanding ":
$R \subseteq \mathsf{reflClos}\ R$
Proof:

?

Theorem "Closure properties of `reflClos`: Reflexivity ":
$\mathsf{reflexive}\ (\mathsf{reflClos}\ R)$
Proof:

?

Theorem "Closure properties of `reflClos`: Minimality ":
$R \subseteq S \ \wedge \ \mathsf{reflexive}\ S \ \Rightarrow \ \mathsf{reflClos}\ R \subseteq S$
Proof:

?

> Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$ iff Q is the smallest reflexive relation containing R, or, equivalently, iff
> - $R \subseteq Q$
> - $\mathbb{I} \subseteq Q$
> - $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

Closures

Let *pred* (for "predicate") be a property on relations, i.e., for some type B and C:

$$pred \ : \ (B \leftrightarrow C) \to \mathbb{B}$$

Relation $Q : B \leftrightarrow C$ is the ***pred*-closure** of $R : B \leftrightarrow C$ iff

- Q is the smallest relation
- that contains R
- and has property *pred*

or, equivalently, iff

- $R \subseteq Q$
- $pred\ Q$
- $(\forall P : B \leftrightarrow C \mid R \subseteq P \wedge pred\ P \bullet Q \subseteq P)$

> Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$ iff Q is the smallest reflexive relation containing R, or, equivalently, iff
> - $R \subseteq Q$
> - $\mathbb{I} \subseteq Q$
> - $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

(For some properties, closures are not defined, or not always defined.)

Formalising General Relation Closures

Let *pred* (for "predicate") be a property on relations, i.e.: $pred \ : \ (B \leftrightarrow C) \to \mathbb{B}$

Relation $Q : B \leftrightarrow C$ is the ***pred*-closure** of $R : B \leftrightarrow C$ iff

- Q is the smallest relation that contains R and has property *pred*,

or, equivalently, iff

- $R \subseteq Q$ and $pred\ Q$ and $(\forall P : B \leftrightarrow C \mid R \subseteq P \wedge pred\ P \bullet Q \subseteq P)$

General Relation Closures in Ref9.5:

Precedence 50 **for:** *_is_closure – of _*
Conjunctional: *_is_closure – of _*
Declaration: *_is_closure – of _* :
$(A \leftrightarrow B) \to ((A \leftrightarrow B) \to \mathbb{B}) \to (A \leftrightarrow B) \to \mathbb{B}$

Axiom "Relation closure ":
Q **is** *pred* closure-of R
$\equiv \ R \subseteq Q \ \wedge \ pred\ Q \ \wedge \ (\forall\ P \ \bullet \ R \subseteq P \wedge pred\ P \Rightarrow Q \subseteq P)$

Theorem “Well-definedness of `reflClos` ”:

Declaration: $\_is\_closure-of\_$:
$(A \leftrightarrow B) \rightarrow ((A \leftrightarrow B) \rightarrow \mathbb{B}) \rightarrow (A \leftrightarrow B) \rightarrow \mathbb{B}$

Axiom “Relation closure ”:
Q **is** *pred* closure-of R
$\equiv R \subseteq Q \wedge pred\ Q \wedge (\forall P \bullet R \subseteq P \wedge pred\ P \Rightarrow Q \subseteq P)$

Theorem “Well-definedness of `reflClos` ”:
reflClos R **is** reflexive closure-of R
Proof:
By “Relation closure ”
with “Closure properties of `reflClos`: Expanding ”
and “Closure properties of `reflClos`: Reflexivity ”
and “Closure properties of `reflClos`: Minimality ”

Theorem “Well-definedness of `reflClos` ”:

Declaration: $\_is\_closure-of\_$:
$(A \leftrightarrow B) \rightarrow ((A \leftrightarrow B) \rightarrow \mathbb{B}) \rightarrow (A \leftrightarrow B) \rightarrow \mathbb{B}$

Axiom “Relation closure ”:
Q **is** *pred* closure-of R
$\equiv R \subseteq Q \wedge pred\ Q \wedge (\forall P \bullet R \subseteq P \wedge pred\ P \Rightarrow Q \subseteq P)$

Theorem “Well-definedness of `reflClos` ”:
reflClos R **is** reflexive closure-of R
Proof:
Using “Relation closure ”:
Subproof for `$R \subseteq$ reflClos $R$`:
?
Subproof for `reflexive (reflClos $R$)`:
?
Subproof for `$\forall P \bullet R \subseteq P \wedge$ reflexive $P \Rightarrow$ reflClos $R \subseteq P$`:
For any `$P$`:
Assuming `$R \subseteq P$`, `reflexive $P$`:
?

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-12

Part 3: More Quantification Calculations: Change of Dummy

Changing the Quantified Domain

$$(\sum i \mid 2 \le i < 10 \bullet i^2)$$
$= \langle$ (8.22) with `(_+_ 2) hasAnInverse` $\rangle$
$$(\sum k \mid 0 \le k < 8 \bullet (k+2)^2)$$

(8.22) **Change of dummy:** Provided f has an inverse and $\neg occurs(`y', `R, P')$
(that is, "y is fresh"), then:

$$(\star x \mid R \bullet P) \quad = \quad (\star y \mid R[x := f\ y] \bullet P[x := f\ y])$$

Above: $f\ y \ = \ 2 + y$ and $f^{-1}\ x \ = \ x - 2$

A function f **has an inverse** f^{-1} iff $x = f\ y \quad \equiv \quad y = f^{-1}\ x$

Inverses of Functions from Function Types

LADM for (8.22): "A function f **has an inverse** f^{-1} iff $x = f\ y \equiv y = f^{-1}\ x$"

This is not a definition of a new inverse concept ...

... but a theorem about the proper inverse concept for functions between types:

- Equality of functions can be proven via "Function extensionqality":
  Axiom "Function extensionality axiom": $(\forall x \bullet f\ x = g\ x) \Rightarrow f = g$
- Composition is conventional mathematical function composition $\_\circ\_$ (read "after"):
  Declaration: $\_\circ\_ : (B \to C) \to (A \to B) \to (A \to C)$
  Axiom "Function composition": $(g \circ f)\ x = g\ (f\ x)$
  Theorem "Associativity of $\circ$": $h \circ (g \circ f) = (h \circ g) \circ f$
- This composition has identities at every type:
  Declaration: $\mathsf{Id} : A \to A$
  Axiom "Identity function": $\mathsf{Id}\ x = x$
  Theorem "Identity of $\circ$": $\mathsf{Id} \circ f = f = f \circ \mathsf{Id}$
- This gives rise to the conventional inverse concept:
  Declaration: $\_isInverseOf\_ : (B \to A) \to (A \to B) \to \mathbb{B}$
  Axiom "Inverse function": $g\ \mathsf{isInverseOf}\ f \equiv g \circ f = \mathsf{Id} \wedge f \circ g = \mathsf{Id}$
- ... and we can prove:
  Theorem "Inverse function connection": $g\ \mathsf{isInverseOf}\ f \equiv (\forall x \bullet \forall y \bullet y = f\ x \equiv x = g\ y)$

Assume f has an inverse and $\neg occurs(`y', `x, R, P')$

$$(\star y \mid R[x := f\ y] \bullet P[x := f\ y])$$
$= \langle$ (8.14) One-point rule: $\neg occurs(`x', `f\ y')$ $\rangle$
$$(\star y \mid R[x := f\ y] \bullet (\star x \mid x = f\ y \bullet P))$$
$= \langle$ (8.20) Nesting: $\neg occurs(`x', `R[x := f\ y]')$, Dummy permutation $\rangle$
$$(\star x, y \mid R[x := f\ y] \wedge x = f\ y \bullet P)$$
$= \langle$ (3.84a) Replacement $(e = f) \wedge E[z := e] \equiv (e = f) \wedge E[z := f]$ $\rangle$
$$(\star x, y \mid R[x := x] \wedge x = f\ y \bullet P)$$
$= \langle$ $R[x := x] = R$; (8.20) Nesting: $\neg occurs(`y', `R')$ $\rangle$
$$(\star x \mid R \bullet (\star y \mid x = f\ y \bullet P))$$
$= \langle$ Assumption "Inverse" `$\forall x, y \bullet x = f\ y \equiv y = f^{-1}\ x$` $\rangle$
$$(\star x \mid R \bullet (\star y \mid y = f^{-1}\ x \bullet P))$$
$= \langle$ (8.14) One-point rule: $\neg occurs(`y', `f^{-1}\ x')$ $\rangle$
$$(\star x \mid R \bullet P[y := f^{-1}\ x])$$
$= \langle$ Textual substitution, $\neg occurs(`y', `P')$ $\rangle$
$$(\star x \mid R \bullet P)$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-14

Part 1: Functions, Change of Dummy

Changing the Quantified Domain

$(\sum i \mid 2 \leq i < 10 \bullet i^2)$

$= \langle$ (8.22) with `(_+_ 2) hasAnInverse` $\rangle$

$(\sum k \mid 0 \leq k < 8 \bullet (k+2)^2)$

(8.22) **Change of dummy:** Provided f has an inverse and $\neg occurs(`y', `R, P')$ (that is, "y is fresh"), then:

$$(\star x \mid R \bullet P) \quad = \quad (\star y \mid R[x := f\, y] \bullet P[x := f\, y])$$

Above: $f\, y = 2 + y$ and $f^{-1}\, x = x - 2$

A function f **has an inverse** f^{-1} iff $x = f\, y \quad \equiv \quad y = f^{-1}\, x$

Recall: Inverses of Functions from Function Types

LADM for (8.22): "A function f **has an inverse** f^{-1} iff $x = f\, y \equiv y = f^{-1}\, x$"

This is not a definition of a new inverse concept ...

... but a theorem about the proper inverse concept for functions between types:

- Equality of functions can be proven via "Function extensionqality":
  Axiom "Function extensionality axiom": $(\forall x \bullet f\, x = g\, x) \Rightarrow f = g$
- Composition is conventional mathematical function composition $\_\circ\_$ (read "after"):
  Declaration: $\_\circ\_ : (B \to C) \to (A \to B) \to (A \to C)$
  Axiom "Function composition": $(g \circ f)\, x = g\, (f\, x)$
  Theorem "Associativity of $\circ$": $h \circ (g \circ f) = (h \circ g) \circ f$
- This composition has identities at every type:
  Declaration: $\mathsf{Id} : A \to A$
  Axiom "Identity function": $\mathsf{Id}\, x = x$
  Theorem "Identity of $\circ$": $\mathsf{Id} \circ f = f = f \circ \mathsf{Id}$
- This gives rise to the conventional inverse concept:
  Declaration: $\_isInverseOf\_ : (B \to A) \to (A \to B) \to \mathbb{B}$
  Axiom "Inverse function": $g \text{ isInverseOf } f \equiv g \circ f = \mathsf{Id} \wedge f \circ g = \mathsf{Id}$
- ... and we can prove:
  Theorem "Inverse function connection": $g \text{ isInverseOf } f \equiv (\forall x \bullet \forall y \bullet y = f\, x \equiv x = g\, y)$

Some More "Prelude" Functions and Some of Their Properties

Declaration: flip : $(A \to B \to C) \to (B \to A \to C)$
Axiom "flip": flip $f\ y\ x = f\ x\ y$

Declaration: curry : $(\langle A, B \rangle \to C) \to (A \to B \to C)$
Declaration: uncurry : $(A \to B \to C) \to (\langle A, B \rangle \to C)$

Axiom "curry": curry $g\ x\ y = g\ \langle x, y \rangle$
Axiom "uncurry": uncurry $f\ \langle x, y \rangle = f\ x\ y$

Theorem "curry∘uncurry": curry (uncurry f) $= f$

Declaration: swap : $\langle A, B \rangle \to \langle B, A \rangle$
Axiom "swap": swap $\langle x, y \rangle = \langle y, x \rangle$

Theorem "flip∘curry": flip (curry f) $=$ curry ($f \circ$ swap)

How to Prove that flip is Self-inverse?

Declaration: flip : $(A \to B \to C) \to (B \to A \to C)$
Axiom "flip": flip $f\ y\ x = f\ x\ y$

Theorem "Self-inverse `flip`": flip (flip f) $= f$
Proof:

flip (flip f) $x\ y$
= ⟨ "flip" ⟩
flip $f\ y\ x$
= ⟨ "flip" ⟩
$f\ x\ y$

Not a proof!

The missing piece:

Theorem "Function extensionality": $f = g \equiv \forall x \bullet f\ x = g\ x$

Proving that flip is Self-inverse

Declaration: flip : $(A \to B \to C) \to (B \to A \to C)$
Axiom "flip": flip $f\ y\ x = f\ x\ y$

Theorem "Function extensionality": $f = g \equiv \forall x \bullet f\ x = g\ x$

Theorem "Self-inverse `flip`": flip (flip f) $= f$
Proof:
Using "Function extensionality":
Subproof for `$\forall x \bullet$ flip (flip $f$) $x = f\ x$`:
For any `$x$`:
Using "Function extensionality":
For any `$y$`:
flip (flip f) $x\ y$
= ⟨ "flip" ⟩
flip $f\ y\ x$
= ⟨ "flip" ⟩
$f\ x\ y$

More Conveniently Proving that flip is Self-inverse

Declaration: flip : $(A \to B \to C) \to (B \to A \to C)$
Axiom "flip": flip $f\ y\ x\ =\ f\ x\ y$

Theorem "Function extensionality": $f = g \ \equiv\ \forall\, x \bullet f\, x = g\, x$

Theorem "Function extensionality 2": $f = g \ \equiv\ \forall\, x, y \bullet f\, x\, y = g\, x\, y$
Proof:
  By "Function extensionality", "Nesting for ∀"

Theorem "Self-inverse \`flip\`": flip (flip f) $= f$
Proof:
  Using "Function extensionality 2":
    For any \`$x$, $y$\`:
      flip (flip f) $x\ y$
    = ⟨ "flip" ⟩
      flip $f\ y\ x$
    = ⟨ "flip" ⟩
      $f\ x\ y$

Assume f has an inverse and $\neg occurs(`y', `x, R, P')$

$(\star\, y \mid R[x := f\ y] \bullet P[x := f\ y])$
= ⟨ (8.14) One-point rule: $\neg occurs(`x', `f\ y')$ ⟩
$(\star\, y \mid R[x := f\ y] \bullet (\star\, x \mid x = f\ y \bullet P))$
= ⟨ (8.20) Nesting: $\neg occurs(`x', `R[x := f\ y]')$, Dummy permutation ⟩
$(\star\, x, y \mid R[x := f\ y] \wedge x = f\ y \bullet P)$
= ⟨ (3.84a) Replacement $(e = f) \wedge E[z := e] \ \equiv\ (e = f) \wedge E[z := f]$ ⟩
$(\star\, x, y \mid R[x := x] \wedge x = f\ y \bullet P)$
= ⟨ $R[x := x] = R$; (8.20) Nesting: $\neg occurs(`y', `R')$ ⟩
$(\star\, x \mid R \bullet (\star\, y \mid x = f\ y \bullet P))$
= ⟨ Assumption "Inverse" \`$\forall x, y \bullet x = f\ y \ \equiv\ y = f^{-1}\, x$\` ⟩
$(\star\, x \mid R \bullet (\star\, y \mid y = f^{-1}\, x \bullet P))$
= ⟨ (8.14) One-point rule: $\neg occurs(`y', `f^{-1}\, x')$ ⟩
$(\star\, x \mid R \bullet P[y := f^{-1}\, x])$
= ⟨ Textual substitution, $\neg occurs(`y', `P')$ ⟩
$(\star\, x \mid R \bullet P)$

Changing the Quantified Domain — $occurs(`y', `x')$

In LADM:

(8.22) **Change of dummy:** Provided f has an inverse and $\neg occurs(`y', `R, P')$,

$$(\star\, x \mid R \bullet P) \quad = \quad (\star\, y \mid R[x := f\ y] \bullet P[x := f\ y])$$

We might have that $occurs(`y', `x')$.

(Note that x and y are metavariables for variables!)

Then x is the same variable as y, and $\neg occurs(`x', `R, P')$.

Therefore $R[x := f\ y] = R$ and $P[x := f\ y] = P$.

So the theorem's consequence becomes trivial:

$$(\star\, x \mid R \bullet P) \quad = \quad (\star\, x \mid R \bullet P)$$

So (8.22) as stated in LADM is valid, but the proof covers only the case $\neg occurs(`y', `x')$.

Changing the Quantified Domain — Variants — see Ref. 4.2

Theorem (8.22) "Change of dummy in ⋆":

$$\forall f \bullet \forall g \bullet (\forall x \bullet \forall y \bullet x = f\,y \equiv y = g\,x) \Rightarrow ((\star x \mid R \bullet P) = (\star y \mid R[x := f\,y] \bullet P[x := f\,y]))$$

Theorem (8.22.1) "Change of dummy in ⋆ ⁻ variant":

$$(\forall x \bullet \forall y \bullet x = f\,y \Rightarrow y = g\,x) \Rightarrow ((\star x \mid R \wedge x = f\,(g\,x) \bullet P) = (\star y \mid R[x := f\,y] \bullet P[x := f\,y]))$$

Theorem (8.22.3) "Change of restricted dummy in ⋆":

$$\forall f \bullet \forall g \bullet (\forall x \mid R \bullet (\forall y \bullet x = f\,y \equiv y = g\,x)) \Rightarrow ((\star x \mid R \bullet P) = (\star y \mid R[x := f\,y] \bullet P[x := f\,y]))$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-14

Part 2: Kleene Algebra

Recall: Reflexive Transitive Closure

$Q : B \leftrightarrow B$ is the **reflexive transitive closure** of $R : B \leftrightarrow B$
iff Q is the smallest reflexive transitive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q \wedge Q ⨾ Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P ⨾ P \subseteq P \bullet Q \subseteq P)$

Definition: The reflexive transitive closure of R is written R^*.

Theorem: $R^* = (\bigcap P \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P ⨾ P \subseteq P \bullet P)$.

Theorem: $R^* = (\bigcup i : \mathbb{N} \bullet R^i)$

- R^i is reachability via exactly i many R-steps
- Reflexive transitive closure R^* is reachability via any number of R-steps
- Transitive closure $R^+ = (\bigcup i : \mathbb{N} \mid i > 0 \bullet R^i)$ is reachability via at least one R-step

Kleene Algebra

The transitive and reflexive-transitive closure operators satisfy many useful algebraic properties, e.g.:

- $(R^*)^{\smile} = (R^{\smile})^*$ $\qquad (R^+)^{\smile} = (R^{\smile})^+$
- $R^* = \mathbb{I} \cup R \cup R^* ⨾ R^*$
- $(R \cup S)^* = (R^* ⨾ S)^* ⨾ R^*$ — **Remember this!**
- $(R \cup S)^+ = R^+ \cup (R^* ⨾ S)^+ ⨾ R^*$
- $R^* \cup S^* \subseteq (R \cup S)^*$

On can prove such properties via reasoning about arbitrary unions $\bigcup$ of relation powers — see Ex10.2 ...

One can also derive these properties from a simple axiomatisation starting from $\subseteq$. ⨾, $\mathbb{I}$, $\cup$:

Axiom (KA.1) "Definition of *": $R^* = \mathbb{I} \cup R \cup R^* ⨾ R^*$

Axiom (KA.2) "Left-induction for *": $R ⨾ S \subseteq S \Rightarrow R^* ⨾ S \subseteq S$

Axiom (KA.3) "Right-induction for *": $Q ⨾ R \subseteq Q \Rightarrow Q ⨾ R^* \subseteq Q$

Axiom (KA.4) "Definition of $^+$": $R^+ = R ⨾ R^*$

Kleene Algebra — Example for Using the Induction Axioms

"Left-ind. *": $R ⨾ S \subseteq S \Rightarrow R^* ⨾ S \subseteq S$ $\qquad$ "Right-ind. *": $Q ⨾ R \subseteq Q \Rightarrow Q ⨾ R^* \subseteq Q$

Theorem (KA.14) "Shuffle *": $R ⨾ R^* = R^* ⨾ R$

Proof:

$R ⨾ R^*$

$\subseteq$ ⟨ "Identity of ⨾", "Monotonicity of ⨾" with "Reflexivity of *" ⟩

$R^* ⨾ R ⨾ R^*$

$\subseteq$ ⟨ "Right-induction for *" with \`$Q := R^* ⨾ R$\` and subproof:

$\quad R^* ⨾ R ⨾ R$

$\quad\subseteq$ ⟨ Monotonicity with "* increases", "⨾-idempotency of *" ⟩

$\quad R^* ⨾ R$

⟩

$R^* ⨾ R$

$\subseteq$ ⟨ "Identity of ⨾", "Monotonicity of ⨾" with "Reflexivity of *" ⟩

$R^* ⨾ R ⨾ R^*$

$\subseteq$ ⟨ "Left-induction for *" with \`$S := R ⨾ R^*$\` and subproof:

$\quad R ⨾ R ⨾ R^*$

$\quad\subseteq$ ⟨ Monotonicity with "* increases", "⨾-idempotency of *" ⟩

$\quad R ⨾ R^*$

⟩

$R ⨾ R^*$

Kleene Algebra — Not Only Relations: Formal Languages

Definition: A **word** over "alphabet" A is a sequence of elements of A.

Definition: A **formal language** over "alphabet" A is a set of words over A.

Interpret:

- $\mathbb{I}$ as the language containing only the empty word
- $\cup$ as language union
- ⨾ as **language concatenation**: $L_1 ⨾ L_2 = \{ u, v \mid u \in L_1 \wedge v \in L_2 \bullet u \frown v \}$
- $\_^*$ as **language iteration**: $L^* = (\bigcup i : \mathbb{N} \bullet L^i)$

Then:

- Formal languages over A form a Kleene algebra.
- Regular languages over A form a Kleene algebra.

  (A regular language is generated by a regular grammar, and accepted by a finite automaton — COMPSCI 2AC3.)
- Each regular language over A is denoted by a Kleene algebra expressions built from only $\mathbb{I}$, and the one-letter-word languages $\{a \triangleleft \epsilon\}$ for letters $a \in A$ as constants.

Kleene Algebra — Not Only Relations: Control Flow Semantics

Definition: A **trace** is a sequence of commands,

Interpret:

- $\mathbb{I}$ as the singleton trace set containing the empty trace
- $\cup$ as trace set union
- $⨟$ as trace set concatenation
- $\_^*$ as trace set iteration

Then:

- Kleene algebra can be used for reasoning about traces (possible executions) of imperative programs
- Kleene algebra provides semantics for control flow

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-15

Part 1: Bags/Multisets

"Multisets" or "Bags" — LADM Section 11.7

A **bag** (or **multiset**) is "like a set, but each element can occur any (finite) number of times".
Bag comprehension and enumeration: Written as for sets, but with delimiters ⦃ and ⦄.
Sets versus bags example:

$$\{ x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x \} = \{4,1,0\} = \{0,1,4\} = \{0,0,0,1,1,4\}$$

$$⦃ x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x ⦄ = ⦃4,1,0,1,4⦄ = ⦃0,1,1,4,4⦄ \neq ⦃0,1,4⦄$$

The operator $\_\#\_ : t \to Bag\ t \to \mathbb{N}$ counts the number of occurrences of an element in a bag:

$$1 \# ⦃0,0,0,1,1,4⦄ = 2$$

Bag extensionality and **bag inclusion** are defined via all occurrence counts:

$$B = C \quad \equiv \quad (\forall\ x \bullet x \# B = x \# C) \qquad B \subseteq C \quad \equiv \quad (\forall\ x \bullet x \# B \le x \# C)$$

Bag operations:

$$x \# (B \cup C) = (x \# B) + (x \# C)$$

$$x \# (B \cap C) = (x \# B) \downarrow (x \# C)$$

$$x \# (B - C) = (x \# B) - (x \# C)$$

Bag Product and Bag Reconstitution

Recall: A **bag** is "like a set, but each element can occur any (finite) number of times".

$$⦃ x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x ⦄ = ⦃4,1,0,1,4⦄ = ⦃0,1,1,4,4⦄ \ne ⦃0,1,4⦄$$

$\_\#\_ : t \to Bag\ t \to \mathbb{N}$ counts the number of occurrences: $1 \# ⦃0,0,0,1,1,4⦄ = 2$

$\_\sqsubseteq\_ : t \to Bag\ t \to \mathbb{B}$ is membership, with $x \sqsubseteq B \equiv x \# B \ne 0$: $1 \sqsubseteq ⦃0,0,0,1,1,4⦄ \equiv true$

Calculate: $⦃x \mid x \sqsubseteq ⦃0,0,0,1,1,4⦄⦄ =$ **?**

Define $bagProd : Bag\ \mathbb{N} \to \mathbb{N}$ such that: $bagProd\ ⦃ e_1, e_2, \ldots, e_n ⦄ = e_1 \cdot e_2 \cdot \ldots \cdot e_n$

e.g., $bagProd\ ⦃ 2,2,3,3,5 ⦄ = 180$

- Easy with exponentiation $\_**\_$: $bagProd\ B = \prod$ **?**
- Without exponentiation: **?**

Related question: For sets, we have (11.5): $S = \{x \mid x \in S \bullet x\}$

What is the corresponding theorem for bags?

Bag reconstitution: $B = ⦃$ **?** $\mid$ **?** $\bullet$ **?** $⦄$

Pigeonhole Principle — LADM section 16.4

The pigeonhole principle is usually stated as follows.

(16.43) If more than n pigeons are placed in n holes, at least one hole will contain more than one pigeon.

Assume:

- $S : Bag\ \mathbb{R}$ is a bag of real numbers
- $\mathsf{av}\ S$ is the average of the elements of S
- $\mathsf{max}\ S$ is the maximum of the elements of S

Reformulating the pigeonhole principle: (16.44) $\mathsf{av}\ S > 1 \Rightarrow \mathsf{max}\ S > 1$

Generalising:

(16.45) Pigeonhole principle:

If $S : Bag\ \mathbb{R}$ is non-empty, then: $\mathsf{av}\ S \le \mathsf{max}\ S$

Stronger on integers:

(16.46) Pigeonhole principle:

If $S : Bag\ \mathbb{Z}$ is non-empty, then: $\lceil \mathsf{av}\ S \rceil \le \mathsf{max}\ S$

Generalised Pigeonhole Principle — Application

(16.46) Pigeonhole principle: If $S : Bag\ \mathbb{Z}$ is non-empty, then $\lceil \mathsf{av}\ S \rceil \le \mathsf{max}\ S$

(16.47) Example: In a room of eight people, at least two of them have birthdays on the same day of the week.

Proof: Let bag S contain, for each day of the week, the number of people in the room whose birthday is on that day. The number of people is 8 and the number of days is 7.

$$S = ⦃ d : Weekday \bullet \# \{ p \mid p\ \mathsf{inRoom}\ r_0 \wedge p\ \mathsf{HasBirthdayOnA}\ d \} ⦄$$

Then:

$\mathsf{max}\ S$

$\ge$ ⟨ Pigeonhole principle (16.46) — S contains integers ⟩

$\lceil \mathsf{av}\ S \rceil$

$=$ ⟨ S has 7 values that sum to 8 ⟩

$\lceil 8/7 \rceil$

$=$ ⟨ Definition of ceiling ⟩

2

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-15

Part 2: Programming with Arrays

Modelling Arrays as Partial Functions

Precedence 100 **for:** $\_ \nrightarrow \_$
Associating to the right: $\_ \nrightarrow \_$
Declaration: $\_ \nrightarrow \_ \ :\ \mathsf{set}\,A \rightarrow \mathsf{set}\,B \rightarrow \mathsf{set}\,(A \leftrightarrow B)$ — type "\tfun" for $\nrightarrow$

Axiom "Definition of $\nrightarrow$":
$X \nrightarrow Y \ = \ \{ f \mid f\breve{}\ ⨾\ f \subseteq \mathsf{id}\,Y \wedge \mathsf{Dom}\,f = X \}$

Useful for the domain of arrays:

Precedence 100 **for:** $\_..\_$
Non-associating: $\_..\_$
Declaration: $\_..\_ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathsf{set}\,\mathbb{N}$ ······type: \..
Axiom "Definition of ..": $m .. n \ = \ \{ i \mid m \leq i \leq n \}$
Theorem "Membership in ..": $i \in m .. n \ \equiv \ m \leq i \leq n$
Theorem "Membership in 0 ..": $i \in 0 .. n \ \equiv \ i \leq n$

Array access: `a[i]` $\Longrightarrow$ $a\,@\,i$
Array update: `a[i] := E` $\Longrightarrow$ $a := a \oplus \{ \langle i, E \rangle \}$

Swapping Two Elements of an Array: Specification

$$i \leq k \geq j \wedge \mathsf{xs} = xs_0 \in (0 .. k) \nrightarrow \llcorner \mathbb{N} \lrcorner$$
$$\Rightarrow[$$
$$\quad Swap$$
$$]$$
$$\mathsf{xs} = xs_0 \oplus \{ \langle i, xs_0\,@\,j \rangle, \langle j, xs_0\,@\,i \rangle \}$$

Swapping Two Elements of an Array: Implementation

$z := xs[i]$;
$xs[i] := xs[j]$;
$xs[j] := z$

Theorem "Array swap":

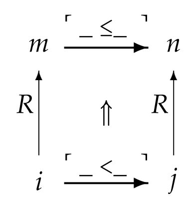
$$i \le k \ge j \ \wedge\ \mathsf{xs} = xs_0 \in (0 \,..\, k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$$
$$\Rightarrow[\ z := \mathsf{xs} \ @\ i\ ;$$
$$\mathsf{xs} := \mathsf{xs} \oplus \{ \langle i, \mathsf{xs} \ @\ j \rangle \}\ ;$$
$$\mathsf{xs} := \mathsf{xs} \oplus \{ \langle j, z \rangle \}$$
$$]$$
$$\mathsf{xs} = xs_0 \oplus \{ \langle i, xs_0 \ @\ j \rangle, \langle j, xs_0 \ @\ i \rangle \}$$

Sortedness

Declaration: sorted : $(\mathbb{N} \leftrightarrow \mathbb{N}) \to \mathbb{B}$

Axiom "Definition of `sorted`":

$$\mathsf{sorted}\ R \ \equiv\ R^{\smile} \,;\, \ulcorner \_<\_ \urcorner \,;\, R \ \subseteq\ \ulcorner \_\le\_ \urcorner$$



Note: No assumption that R is univalent or contiguous!

Theorem "Sortedness":

$$\mathsf{sorted}\ R \equiv \forall\, i \bullet \forall\, j \mid i < j \bullet$$
$$\forall\, m \bullet \forall\, n \mid i \,(\!|\, R \,|\!)\, m \wedge j \,(\!|\, R \,|\!)\, n \bullet m \le n$$

Theorem "Sortedness of functions": univalent A

$$\Rightarrow (\mathsf{sorted}\ A \equiv \forall\, i \bullet \forall\, j \mid \{i, j\} \subseteq \mathsf{Dom}\ A \wedge i < j \bullet A \ @\ i \le A \ @\ j)$$

Specification of Sorting — First Attempt

$$\mathsf{xs} \in (0 \,..\, k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$$
$$\Rightarrow[\ \mathsf{SORT}$$
$$]$$
$$\mathsf{xs} \in (0 \,..\, k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \quad \wedge \quad \mathsf{sorted\ xs}$$

Theorem "Sorting 0":

$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$[$p := 0$;
while $p \neq k + 1$ do
xs := xs $\oplus$ { ⟨ p, 42 ⟩ } ;
$p := p + 1$
od
]
$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner \wedge$ sorted xs

Proof:

$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$⟨ ? ⟩
$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner \wedge$ Ran $((0 .. 0) \lhd \mathsf{xs}) = \{ \mathsf{xs} @ 0 \}$
$\Rightarrow$[$p := 0$] ⟨ "Assignment" with substitution ⟩
$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner \wedge$ Ran $((0 .. p) \lhd \mathsf{xs}) = \{ \mathsf{xs} @ 0 \}$
$\Rightarrow$[while $p \neq k + 1$ do xs := xs $\oplus$ { ⟨ p, 42 ⟩ } ; $p := p + 1$ od
] ⟨ "While" with subproof:
?
⟩
$\neg (p \neq k + 1) \wedge \mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner \wedge$ Ran $((0 .. p) \lhd \mathsf{xs}) = \{ \mathsf{xs} @ 0 \}$
$\Rightarrow$⟨ ? ⟩
$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner \wedge$ sorted xs

A Program Satisfying the Sorting Specification from the Previous Slide:

```
p := 0 ;
while p ≠ k + 1 do
    xs[p] := 42 ;
    p := p + 1
```

Permutation-based Specification of Sorting

$xs_0 = \mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$[SORT
]
$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner \wedge$ sorted xs
$\wedge (\exists f \mid f \in (0 .. k) \twoheadrightarrowtail (0 .. k) \bullet \mathsf{xs} = f \mathbin{;} xs_0)$

- You have some experience with ∃-quantifications in invariants in Ex7.3…
- Moving f into a ghost variable would make verification easier here as well.

Bag-based Specification of Sorting

$xs_0 = \mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$[SORT
]
$\mathsf{xs} \in (0 .. k) \multimap\!\!\!\rightarrow \llcorner \mathbb{N} \lrcorner \wedge$ sorted xs
$\wedge \, ⦃ p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p ⦄ = ⦃ p \mid p \in xs_0 \bullet \mathsf{snd}\, p ⦄$

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

xs_0 = xs ∈ (0 .. k) ⤀ ⌞ℕ⌟
⇒[while true do
  xs := xs ⊕ { ⟨ 0, 42 ⟩ }
 od
]
xs ∈ (0 .. k) ⤀ ⌞ℕ⌟ ∧ sorted xs
 ∧ ⦃ p ❙ p ∈ xs • snd p ⦄ = ⦃ p ❙ p ∈ xs_0 • snd p ⦄

```
while true do
    xs[0] := 42
```

Proof structure?

You need to be able to write down the proof structure without help, e.g., in M2!

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

xs_0 = xs ∈ (0 .. k) ⤀ ⌞ℕ⌟
⇒[while true do
  xs := xs ⊕ { ⟨ 0, 42 ⟩ }
 od
]
xs ∈ (0 .. k) ⤀ ⌞ℕ⌟ ∧ sorted xs
 ∧ ⦃ p ❙ p ∈ xs • snd p ⦄ = ⦃ p ❙ p ∈ xs_0 • snd p ⦄

```
while true do
    xs[0] := 42
```

Proof:
xs_0 = xs ∈ (0 .. k) ⤀ ⌞ℕ⌟
⇒⟨ ? ⟩
?
⇒[while true do xs := xs ⊕ { ⟨ 0, 42 ⟩ } od
] ⟨ "While" with subproof:
  ?
  ⇒[xs := xs ⊕ { ⟨ 0, 42 ⟩ }]
   ⟨ ? ⟩
  ?
⟩
?
⇒⟨ ? ⟩
xs ∈ (0 .. k) ⤀ ⌞ℕ⌟ ∧ sorted xs
 ∧ ⦃ p ❙ p ∈ xs • snd p ⦄ = ⦃ p ❙ p ∈ xs_0 • snd p ⦄

Where do we flag the invariant?

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

xs_0 = xs ∈ (0 .. k) ⤀ ⌞ℕ⌟
⇒[while true do
  xs := xs ⊕ { ⟨ 0, 42 ⟩ }
 od
]
xs ∈ (0 .. k) ⤀ ⌞ℕ⌟ ∧ sorted xs
 ∧ ⦃ p ❙ p ∈ xs • snd p ⦄ = ⦃ p ❙ p ∈ xs_0 • snd p ⦄

```
while true do
    xs[0] := 42
```

Proof:
xs_0 = xs ∈ (0 .. k) ⤀ ⌞ℕ⌟
⇒⟨ ? ⟩
Q — **Invariant**
⇒[while true do xs := xs ⊕ { ⟨ 0, 42 ⟩ } od
] ⟨ "While" with subproof:
  ?
  ⇒[xs := xs ⊕ { ⟨ 0, 42 ⟩ }]
   ⟨ ? ⟩
  ?
⟩
?
⇒⟨ ? ⟩
xs ∈ (0 .. k) ⤀ ⌞ℕ⌟ ∧ sorted xs
 ∧ ⦃ p ❙ p ∈ xs • snd p ⦄ = ⦃ p ❙ p ∈ xs_0 • snd p ⦄

Which other conditions ere determined by the invariant?

A Verified Sorting Algorithm

Theorem "Sorting 0′":

$xs_0 = \mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$

$\Rightarrow$[while true do
  xs := xs $\oplus$ { ⟨ 0, 42 ⟩ }
 od
]

$\mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge$ sorted xs
$\wedge \ \lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \rbag = \lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \rbag$

```
while true do
    xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$⟨ ? ⟩
Q — Invariant
$\Rightarrow$[while true do xs := xs $\oplus$ { ⟨ 0, 42 ⟩ } od
] ⟨ "While" with subproof:
  true $\wedge$ Q
  $\Rightarrow$[xs := xs $\oplus$ { ⟨ 0, 42 ⟩ }]
  ⟨ ? ⟩
  Q
⟩
$\neg$ true $\wedge$ Q
$\Rightarrow$⟨ ? ⟩
$\mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge$ sorted xs
$\wedge \ \lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \rbag = \lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \rbag$

Can we already complete some proof obligations now, without even fixing the invariant?

A Verified Sorting Algorithm

Theorem "Sorting 0′":

$xs_0 = \mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$

$\Rightarrow$[while true do
  xs := xs $\oplus$ { ⟨ 0, 42 ⟩ }
 od
]

$\mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge$ sorted xs
$\wedge \ \lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \rbag = \lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \rbag$

```
while true do
    xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$⟨ ? ⟩
Q — Invariant
$\Rightarrow$[while true do xs := xs $\oplus$ { ⟨ 0, 42 ⟩ } od
] ⟨ "While" with subproof:
  true $\wedge$ Q
  $\Rightarrow$[xs := xs $\oplus$ { ⟨ 0, 42 ⟩ }]
  ⟨ ? ⟩
  Q
⟩
$\neg$ true $\wedge$ Q
$\Rightarrow$⟨ "Definition of \`false\`", "Zero of $\wedge$", "ex falso quodlibet" ⟩
$\mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge$ sorted xs
$\wedge \ \lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \rbag = \lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \rbag$

How can we choose the invariant to make the remaining proof obligations easy?

A Verified Sorting Algorithm

Theorem "Sorting 0′":

$xs_0 = \mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$

$\Rightarrow$[while true do
  xs := xs $\oplus$ { ⟨ 0, 42 ⟩ }
 od
]

$\mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge$ sorted xs
$\wedge \ \lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \rbag = \lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \rbag$

```
while true do
    xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$⟨ "Right-zero of $\Rightarrow$" ⟩
true — Invariant
$\Rightarrow$[while true do xs := xs $\oplus$ { ⟨ 0, 42 ⟩ } od
] ⟨ "While" with subproof:
  true $\wedge$ true
  $\Rightarrow$[xs := xs $\oplus$ { ⟨ 0, 42 ⟩ }]
  ⟨ "Idempotency of $\wedge$", "Assignment" with substitution ⟩
  true
⟩
$\neg$ true $\wedge$ true
$\Rightarrow$⟨ "Contradiction", "ex falso quodlibet" ⟩
$\mathsf{xs} \in (0 \mathinner{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge$ sorted xs
$\wedge \ \lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \rbag = \lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \rbag$

This program has herewith been proven partially correct with respect to our sorting algorithm specification.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-19

Program Correctness Statements $P \Rightarrow[\ C\ ]\ Q$ and Their Meaning

In Exercise 6.6 you proved:

Theorem "Adding$_2$":

$m = m_0 \wedge n = n_0$
$\Rightarrow[$ while $m \neq 0$
do
$m := m - 1$;
$n := n + 1$
od
$]$
$n = m_0 + n_0$

- What does this correctness statement imply for start states satisfying

$$m = m_0 = -3 \quad \wedge \quad n = n_0 = 3 \ ?$$

Answer:

"This program then only terminates in states satisfying $n = 0$."

- What does this program "do" when started in such a state?

H14: Domain and Range Relation-algebraically

- In the abstract relation-algebraic setting, we are only dealing with **relation types** $A \leftrightarrow B$
- No set types, and therefore no direct way to express *Dom*, $\lhd$, $(\!|\_|\!)$, etc.
- One candidate for "relations representing sets" are subidentities, $q \subseteq \mathbb{I}$
- In set theory, id A is a relation that can just serve as a representation of set A
- id allows us to define $\lhd$:
  Theorem (14.237) "Domain restriction via ⨾": $A \lhd R = \text{id}\ A ⨾ R$
- In the abstract relation-algebraic setting, the role of the operation
  $Dom : (A \leftrightarrow B) \rightarrow set\ A$
  is taken by the new operation
  $\text{dom} : (A \leftrightarrow B) \rightarrow (A \leftrightarrow A)$
  $\text{dom}\ R = R ⨾ R^{\smile} \cap \mathbb{I}$

  taking each relation R to the subidentity relation representing the set *Dom* R
- In set theory:
  $\text{dom}\ R = \text{id}\ (\text{Dom}\ R)$

$\Longrightarrow$ Ref11.2, Ref11.3, H14

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-19

Relational Semantics: Partial Correctness

Formalising Partial Correctness — Syntax Types

So far, we have been using the **dynamic logic** notation:

$$P \Rightarrow\!\![\ C\ ]\ Q$$

with its **partial correctness meaning:**

> If command C is started in a state in which the **precondition** P holds
> then it will terminate **only** in a state in which the **postcondition** Q holds.

What are P, Q, C?

- P and Q are some kind of Boolean expressions — of type Expr𝔹
- C is a command — of type Cmd
- We also need expression e for assignment RHSs, "$x := e$" — of type ExprV

The Programming Language: Expressions and Commands

The types Cmd, ExprV, and Expr𝔹 are abstract syntax tree (AST) types

Declaration: *ExprV*, Expr𝔹 : Type
Declaration: *Var′* : Var → ExprV
Declaration: *Int′* : ℤ → ExprV
Declaration: _ +′ _ : ExprV → ExprV → ExprV
Declaration: *true′*, *false′* : Expr𝔹
Declaration: ¬′_ : Expr𝔹 → Expr𝔹
Declaration: _ ∧′ _ : Expr𝔹 → Expr𝔹 → Expr𝔹
Declaration: _ =′ _ : ExprV → ExprV → Expr𝔹

Declaration: Cmd : Type
Declaration: _;_ : Cmd → Cmd → Cmd
Declaration: _: =_ : Var → ExprV → Cmd
Declaration: if_then_else_fi : Expr𝔹 → Cmd → Cmd → Cmd
Declaration: while_do_od : Expr𝔹 → Cmd → Cmd

Formalising Partial Correctness — Semantics Types

So far, we have been using the **dynamic logic** notation:

$$P \Rightarrow [\, C \,]\, Q$$

with its **partial correctness meaning:**

> If command C is started in a state in which the **precondition** P holds
> then it will terminate **only** in a state in which the **postcondition** Q holds.

What does "state" mean? "starts"? "holds"? "terminates"? . . .

- States assign variable to values
- here we simply model states as function — of type $\mathsf{Var} \to \mathsf{Value}$
- "P holds in state s": semantics of Boolean expressions: $\mathsf{sat} : \mathsf{Expr}\mathbb{B} \to \mathsf{set\ State}$
  ($s \in \mathsf{sat}\ P$ iff "condition P is **sat**isfied in state s")
  (Alternatively, start from $\mathsf{eval}\mathbb{B} : \mathsf{State} \to \mathsf{Expr}\mathbb{B} \to \mathbb{B}$ and define $\mathsf{sat}\ P = \{ s \mid \mathsf{eval}\mathbb{B}\ s\ P \}$)

Types for Semantics of Expressions and Commands

What does "state" mean? "holds"? . . .

Imperative programs, such as Cmd, transform a State that assigns values to variables.

Declaration: $\mathsf{Var} : \mathsf{Type}$ — variables
Declaration: $\mathsf{Value} : \mathsf{Type}$ — storable values
Declaration: $\mathsf{State} : \mathsf{Type}$

Axiom "Definition of `State` ": $\mathsf{State} = \mathsf{Var} \to \mathsf{Value}$

Declaration: $\mathsf{eval} : \mathsf{State} \to \mathsf{ExprV} \to \mathsf{Value}$ — value expression semantics
Declaration: $\mathsf{sat} : \mathsf{Expr}\mathbb{B} \to \mathsf{set\ State}$ — Boolean expression semantics

Declaration: $\_ \oplus' \_ : (A \to B) \to \langle A , B \rangle \to (A \to B)$ — state update
Axiom "Definition of function override":

$$(x = z \Rightarrow (f \oplus' \langle x, y \rangle)\, z = y)$$
$$\land\ (x \neq z \Rightarrow (f \oplus' \langle x, y \rangle)\, z = f\, z)$$

Semantics of Commands

What does "starts" mean? "terminates"? . . .

Program execution induces a **state transformation relation**.

Declaration: $[\![\_]\!] : \mathsf{Cmd} \to (\mathsf{State} \leftrightarrow \mathsf{State})$

$s_1\ (\!([\![C]\!])\!)\ s_2$ iff "when started in state s_1, command C can terminate in state s_2".

Inductive definition of $[\![\_]\!]$ over the structure of *Cmd*:

Axiom "Semantics of :=": $[\![x := e]\!] = \{ s : \mathsf{State} \bullet \langle s , s \oplus' \langle x , \mathsf{eval}\ s\ e \rangle \rangle \}$
Axiom "Semantics of **;**": $[\![C_1 \,;\, C_2]\!] = [\![C_1]\!] \,⨾\, [\![C_2]\!]$
Axiom "Semantics of `if` ":

$$[\![\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{fi}]\!] = (\mathsf{sat}\ B \lhd [\![C_1]\!]) \cup (\mathsf{sat}\ B \mathbin{⩤} [\![C_2]\!])$$

Axiom "Semantics of `while` ":

$$[\![\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}]\!] = (\mathsf{sat}\ B \lhd [\![C]\!])^{*} \mathbin{⩥} \mathsf{sat}\ B$$

Formalising Partial Correctness

So far, we have been using the **dynamic logic** notation:

$$P \Rightarrow[\ C\ ]\ Q$$

with its **partial correctness meaning:**

> If command C is started in a state in which the **precondition** P holds
> then it will terminate **only** in a state in which the **postcondition** Q holds.

Declaration: $\_ \Rightarrow[\_]\_ : \mathsf{Expr}\mathbb{B} \to \mathsf{Cmd} \to \mathsf{Expr}\mathbb{B} \to \mathbb{B}$

Axiom "Partial Correctness":

$$(P \Rightarrow[\ C\ ]\ Q) \quad \equiv \quad [\![C]\!]\ (\!|\ \mathsf{sat}\ P\ |\!) \subseteq \mathsf{sat}\ Q$$

Theorem "Partial Correctness":

$$(P \Rightarrow[\ C\ ]\ Q) \quad \equiv \quad \forall\ s_1, s_2 \bullet s_1 \in \mathsf{sat}\ P \wedge s_1\ (\ [\![C]\!]\ )\ s_2 \Rightarrow s_2 \in \mathsf{sat}\ Q$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-21

Relational Semantics: Partial Correctness

How to Finish this Hoare Logic Proof for Arbitrary Loop Body C ?

Theorem "while true": $P \Rightarrow[\ \mathsf{while\ true\ do}\ C\ \mathsf{od}\ ]\ Q$

Proof:

```
    P              ▪▪▪▪▪▪Precondition
  ⇒ ⟨ "Right-zero of ⇒" ⟩
    true           ▪▪▪▪▪▪Invariant
  ⇒[ while true do C od ] ⟨ "While" with subproof:
        true ∧ true ▪▪▪▪▪▪Loop condition and invariant
      ≡ ⟨ "Identity of ∧" ⟩
        true
      ⇒[ C ] ⟨ ? ⟩
        true       ▪▪▪▪▪▪Invariant
    ⟩
    ¬ true ∧ true  ▪▪▪▪▪▪Negated loop condition, and invariant
  ⇒⟨ "Contradiction", "ex falso quodlibet" ⟩
    Q              ▪▪▪▪▪▪Postcondition
```

Separation of Concerns...

Derived inference rule "while true":

$$\frac{\vdash \ `\mathsf{true} \Rrightarrow[\, C \,]\ \mathsf{true}`}{`P \Rrightarrow[\ \mathsf{while\ true\ do}\ C\ \mathsf{od}\ ]\ Q`}$$

Proof:

Assuming "Inv" `true ⇛[ C ] true`:

P ▪▪▪▪▪▪Precondition
⇒ ⟨ "Right-zero of ⇒" ⟩
true ▪▪▪▪▪▪Invariant
⇛[while true do C od] ⟨ "While" with subproof:
  true ∧ true ▪▪▪▪▪▪Loop cond. and inv.
  ≡ ⟨ "Identity of ∧" ⟩
  true
  ⇛[C] ⟨ Assumption "Inv" ⟩
  true ▪▪▪▪▪▪Invariant
⟩
¬ true ∧ true ▪▪▪▪▪▪Negated loop cond. and inv.
⇛⟨ "Contradiction", "ex falso quodlibet" ⟩
Q ▪▪▪▪▪▪Postcondition

How to prove `true ⇛[ C ] true` ?
(Or even X ⇛[C] true ?)

- By structural induction over $C : Cmd$, using the command correctness proof rules ("Hoare logic")
- *Or:* Using the definition of ⇛[_] in terms of the semantics ⟦_⟧

Recall: Types for Semantics of Expressions and Commands

What does "state" mean? "holds"? ...

Imperative programs, such as Cmd, transform a State that assigns values to variables.

Declaration: Var : Type — variables
Declaration: Value : Type — storable values
Declaration: State : Type

Axiom "Definition of `State`": State = Var → Value

Declaration: evalV : State → ExprV → Value — value expression semantics
Declaration: sat : Exprℬ → set State — Boolean expression semantics

Declaration: $\_\oplus'\_ : (A \to B) \to \langle A , B \rangle \to (A \to B)$ — state update
Axiom "Definition of function override":
$$(x = z \Rightarrow (f \oplus' \langle x, y \rangle)\, z = y)$$
$$\wedge\ (x \neq z \Rightarrow (f \oplus' \langle x, y \rangle)\, z = f\, z)$$

Recall: Semantics of Commands

What does "starts" mean? "terminates"? ...

Program execution induces a **state transformation relation**.

Declaration: ⟦_⟧ : Cmd → (State ↔ State)

s_1 ⦇ ⟦ C ⟧ ⦈ s_2 iff "when started in state s_1, command C can terminate in state s_2".

Inductive definition of ⟦_⟧ over the structure of *Cmd*:

Axiom "Semantics of :=": ⟦ $x := e$ ⟧ = { s : State • ⟨ s , $s \oplus' \langle x , \mathsf{evalV}\ s\ e \rangle$ ⟩ }
Axiom "Semantics of ;": ⟦ C_1 ; C_2 ⟧ = ⟦ C_1 ⟧ ⨾ ⟦ C_2 ⟧
Axiom "Semantics of `if`":
⟦ if B then C_1 else C_2 fi ⟧ = (sat B ◁ ⟦ C_1 ⟧) ∪ (sat B ⩤ ⟦ C_2 ⟧)

Axiom "Semantics of `while`":
⟦ while B do C od ⟧ = (sat B ◁ ⟦ C ⟧)* ⩥ sat B

Formalising Partial Correctness

So far, we have been using the **dynamic logic** notation:

$$P \Rrightarrow[\, C \,]\, Q$$

with its **partial correctness meaning:**

> If command C is started in a state in which the **precondition** P holds then it will terminate **only** in a state in which the **postcondition** Q holds.

Declaration: $\_ \Rrightarrow[\_]\_ : \mathsf{Expr}\mathbb{B} \to \mathsf{Cmd} \to \mathsf{Expr}\mathbb{B} \to \mathbb{B}$

Axiom "Partial Correctness":

$$(P \Rrightarrow[\, C \,]\, Q) \quad \equiv \quad [\![C]\!] \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, Q$$

Theorem "Partial Correctness":

$$(P \Rrightarrow[\, C \,]\, Q) \quad \equiv \quad \forall s_1, s_2 \bullet s_1 \in \mathsf{sat}\, P \wedge s_1 \,(\, [\![C]\!] \,)\, s_2 \Rightarrow s_2 \in \mathsf{sat}\, Q$$

Proving "Postcondition `true`" is now Easy

Declaration: $\_ \Rrightarrow[\_]\_ : \mathsf{Expr}\mathbb{B} \to \mathsf{Cmd} \to \mathsf{Expr}\mathbb{B} \to \mathbb{B}$

Axiom "Partial Correctness": $(P \Rrightarrow[\, C \,]\, Q) \equiv [\![C]\!] \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, Q$

Theorem "Postcondition `true`" "Right-zero of $\Rrightarrow[\_]$":

$P \Rrightarrow[\, C \,]\, \mathit{true}'$

Proof:

$P \Rrightarrow[\, C \,]\, \mathit{true}'$

$\equiv$ ⟨ "Partial correctness" ⟩

$[\![C]\!] \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, \mathit{true}'$

$\equiv$ ⟨ "sat true′" ⟩

$[\![C]\!] \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathbf{U}$

— **This is** "Universal set is greatest"

Partial Correctness: "Terminate Only in States Satisfying Postcondition"

Axiom "Partial Correctness": $(P \Rrightarrow[\, C \,]\, Q) \equiv [\![C]\!] \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, Q$

Axiom "Semantics of `while`": $[\![\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}]\!] = (\mathsf{sat}\, B \lhd [\![C]\!])^{*} \rhd \mathsf{sat}\, B$

Theorem "Partial correctness of `while true`": $P \Rrightarrow[\, \mathsf{while}\ \mathit{true}'\ \mathsf{do}\ C\ \mathsf{od} \,]\, Q$

Proof:

$P \Rrightarrow[\, \mathsf{while}\ \mathit{true}'\ \mathsf{do}\ C\ \mathsf{od} \,]\, Q$

$\equiv$ ⟨ "Partial correctness" ⟩

$[\![\mathsf{while}\ \mathit{true}'\ \mathsf{do}\ C\ \mathsf{od}]\!] \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, Q$

$\equiv$ ⟨ "Semantics of `while`" ⟩

$((\mathsf{sat}\, \mathit{true}' \lhd [\![C]\!])^{*} \rhd \mathsf{sat}\, \mathit{true}') \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, Q$

$\equiv$ ⟨ "sat true′" ⟩

$((\mathbf{U} \lhd [\![C]\!])^{*} \rhd \mathbf{U}) \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, Q$

$\equiv$ ⟨ "$\rhd \mathbf{U}$" ⟩

$\{\} \,(\!|\, \mathsf{sat}\, P \,|\!)\, \subseteq \mathsf{sat}\, Q$

$\equiv$ ⟨ "Relational image under {}" ⟩

$\{\} \subseteq \mathsf{sat}\, Q$ — **This is** "Empty set is least"

That is:

Any "while *true*" loop is partially correct with respect to any pre-post-condition specification.

Soundness of the Inference Rules for Correctness

Since partial correctness statements ($P \Rightarrow\!\![\; C \;]\; Q$) are now defined via the relational semantics, we can prove **soundness** of the Hoare logic proof rules by deriving them, e.g.:

Derived inference rule "Sequence":

$$\vdash \frac{`P \Rightarrow\!\![\; C_1 \;]\; Q`,\quad `Q \Rightarrow\!\![\; C_2 \;]\; R`}{`P \Rightarrow\!\![\; C_1 ; C_2 \;]\; R`}$$

Proof:

Assuming (C_1) $`P \Rightarrow\!\![\; C_1 \;]\; Q`$ **and using with** "Partial correctness",
(C_2) $`Q \Rightarrow\!\![\; C_2 \;]\; R`$ **and using with** "Partial correctness":

$P \Rightarrow\!\![\; C_1 ; C_2 \;]\; R$
$\equiv$ ⟨ "Partial correctness" ⟩
$[\![C_1 ; C_2]\!]\ (\!|\ \mathsf{sat}\ P\ |\!)\ \subseteq\ \mathsf{sat}\ R$
$\equiv$ ⟨ "Semantics of ;", "Relational image of ⨾" ⟩
$[\![C_2]\!]\ (\!|\ [\![C_1]\!]\ (\!|\ \mathsf{sat}\ P\ |\!)\ |\!)\ \subseteq\ \mathsf{sat}\ R$
$\Leftarrow$ ⟨ Antitonicity with assumption (C_1) ⟩
$[\![C_2]\!]\ (\!|\ \mathsf{sat}\ Q\ |\!)\ \subseteq\ \mathsf{sat}\ R$
$\equiv$ ⟨ Assumption (C_2) ⟩
true

Soundness of the Inference Rules for Correctness (ctd.)

Derived inference rule "Conditional":

$$\vdash \frac{`B \wedge' P \Rightarrow\!\![\; C_1 \;]\; Q`,\quad `\neg' B \wedge' P \Rightarrow\!\![\; C_2 \;]\; Q`}{`P \Rightarrow\!\![\; \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{fi} \;]\; Q`}$$

Derived inference rule "While":

$$\vdash \frac{`B \wedge' Q \Rightarrow\!\![\; C \;]\; Q`}{`Q \Rightarrow\!\![\; \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od} \;]\; \neg' B \wedge' Q`}$$

"Operational Semantics", "Axiomatic Semantics"

For a command $C : Cmd$, we introduced it relational semantics $[\![C]\!] : State \leftrightarrow State$.

This semantics only captures the **terminating behaviours** of C, in the shape of an "input-output relation".

This is also called **"big-step operational semantics"**, or **"natural semantics"**.

"Small-step operational semantics" maps C to a relation of type $State \leftrightarrow (State^* \cup State^\infty)$:

- Each start state s_0 is related to all possible execution sequences starting from s_0.
- All intermediate states (after each assignment) are recorded.
- Non-terminating behaviours give rise to infinite state sequences.
- Terminating behaviours give rise to finite sequences $s_0, \ldots, s_n$, with $s_0 \,(\![\![C]\!]\!)\, s_n$
  — this is either a proof obligation, or a way to define $[\![C]\!]$.

"Axiomatic semantics" is the set of correctness statements ($P \Rightarrow\!\![\; C \;]\; Q$) that can be derived about C in a "Hoare logic" inference system of the kind we have used.

As seen on the previous slides, such an inference system can (and should!) be justified against the operational semantics.

— More in COMPSCI 3MI3!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-22

Total Correctness

Precondition-Postcondition Specifications

- Program correctness statement in LADM (and much current use): "Hoare triple":

  $$\{ P \} C \{ Q \}$$

  Meaning (LADM ch. 10): "Total correctness":
  If command C is started in a state in which the **precondition** P holds
  then it **will terminate** in a state in which the **postcondition** Q holds.

- So far, we have been using the **dynamic logic** notation:

  $$P \Rightarrow [\, C \,]\, Q$$

  with its **partial correctness meaning:**
  If command C is started in a state in which the **precondition** P holds
  then it will terminate **only in states** in which the **postcondition** Q holds.

Differences between partial and total correctness:

Total correctness forbids commands that do not terminate (properly):

- Infinite loops
- Commands that crash — evaluating "undefined" expressions

Undefined Behaviours in C

- Spatial memory safety violations — **int** $a[5]$; **int** $k = a[6]$;
- Temporal memory safety violations — $free(p)$; $k = *p$;
- Integer overflow — $k = maxint + 2$; $m = minint - 3$;
- Strict aliasing violations
- Alignment violations
- Unsequenced modifications — *printf*("%d␣%d", a++, a++);
- Data races
- Loops that neither perform I/O nor terminate

Homework 3 Lemma 5

In Homework 3, you proved, for variables x and y of type $\mathbb{Z}$:

Lemma (5):

$$
\begin{array}{l}
\quad p = p_0 \wedge q = q_0 \\
\Rightarrow[\ p := p + q\ ; \\
\quad q := p - q\ ; \\
\quad p := p - q \\
\quad] \\
\quad p = q_0 \wedge q = p_0
\end{array}
$$

The proof typically used “Subtraction”, “Unary minus”, “Identity of +”, and (implicitly) “Associativity of +”.

What Do These C Program Fragments Do?

Let `p` and `q` be variables of type **int**.

```
p = p + q;
q = p - q;
p = p - q;
```

- **int** overflow is undefined behaviour!
- (Going below `minint` is still called “integer overflow”.)
- this swap “works” only if none of the operations overflows

Let `k` and `n` be variables of type **unsigned int**.

```
k = k + n;
n = k - n;
k = k - n;
```

- **unsigned int** has “wrap-around arithmetic” (e.g., modulo 2^{64}) – **totally defined**
- this swap “works”

Let `c` and `d` be variables of type **double**.

```
c = c + d;
d = c - d;
c = c - d;
```

- “+” at floating-point types is not even associative...
- floating-point arithmetic is hard to reason about...

Recall: Total Correctness

- Program correctness statement in LADM (and much current use): “Hoare triple”:

  $$\{ P \}\ C\ \{ Q \}$$

  Meaning (LADM ch. 10): “<u>Total correctness</u>”:

  If command C is started in a state in which the **precondition** P holds then it **will terminate** in a state in which the **postcondition** Q holds.

Differences between partial and total correctness:

Total correctness forbids commands that do not terminate (properly):

- Infinite loops
- Commands that crash — evaluating “undefined” expressions

What difference does this make for the rules of Hoare logic?

Rules That Work for Both Partial and Total Correctness

Sequential composition:

$$\vdash \frac{\texttt{`}P \Rrightarrow[\ C_1\ ]\ Q\texttt{`},\quad \texttt{`}Q \Rrightarrow[\ C_2\ ]\ R\texttt{`}}{\texttt{`}P \Rrightarrow[\ C_1\ ;\ C_2\ ]\ R\texttt{`}}$$

Strengthening the precondition:

$$\vdash \frac{\texttt{`}P_1 \Rightarrow P_2\texttt{`},\quad \texttt{`}P_2 \Rrightarrow[\ C\ ]\ Q\texttt{`}}{\texttt{`}P_1 \Rrightarrow[\ C\ ]\ Q\texttt{`}}$$

Weakening the postcondition:

$$\vdash \frac{\texttt{`}P \Rrightarrow[\ C\ ]\ Q_1\texttt{`},\quad \texttt{`}Q_1 \Rightarrow Q_2\texttt{`}}{\texttt{`}P \Rrightarrow[\ C\ ]\ Q_2\texttt{`}}$$

Total Correctness Rule for Assignment

Assignment ":=": Two characters; type ":="

Substitution "≔": One Unicode character; type "\:="

Used so far: **Dynamic Logic Partial Correctness Assignment Axiom:**

$$Q[x \coloneqq E] \quad \Rrightarrow[\ x := E\ ] \quad Q$$

LADM Total Correctness Assignment Axiom (10.1):

$$\{\ \mathit{dom}\ \text{‘}E\text{’} \land Q[x \coloneqq E]\ \} \quad x := E \quad \{\ Q\ \}$$

For each *programming-language* expression E, the predicate
dom ‘E’
is satisfied exactly in the states in which E is defined.
(*dom* is a ***meta-function*** taking expressions to Boolean conditions.)

Examples:

- *dom* ‘*sqrt* $(x\ /\ y)$’ $\equiv$ $y \neq 0 \land x\ /\ y \geq 0$
- *dom* ‘a @ i’ $\equiv$ $i \in \mathit{Dom}\ a$
- For int-variables i and j:
  dom ‘$i + j$’ $\equiv$ minint $\leq to\mathbb{Z}\ i\ +\ to\mathbb{Z}\ j \leq$ maxint

Conditional Rule

Each evaluation of an expression E needs to be guarded by a precondition *dom* ‘E’:

$$\frac{\{\ B \land P\ \}\quad C_1\quad \{\ Q\ \} \qquad\qquad \{\ \neg B \land P\ \}\quad C_2\quad \{\ Q\ \}}{\{\ \mathit{dom}\ \text{‘}B\text{’} \land P\ \}\quad \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi}\quad \{\ Q\ \}}$$

“While” Rule

So far for partial correctness:

$$\vdash \frac{`B \wedge Q \Rightarrow[\ C\ ]\ Q`}{`Q \Rightarrow[\ \text{while } B \text{ do } C \text{ od}\ ]\ \neg B \wedge Q`}$$

Now **two** additional ingredients (besides B and C):

- **Invariant:** $Q : \mathbb{B}$ — as before, ensuring functional correctness
- **Variant** (or “bound function”): $T : \mathbb{Z}$ — ensuring termination

$$\frac{\{ B \wedge Q \}\ C\ \{ dom\ `B' \wedge Q \} \qquad \{ B \wedge Q \wedge T = t_0 \}\ C\ \{ T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ dom\ `B' \wedge Q \}\ \ \text{while } B \text{ do } C \text{ od}\ \ \{ \neg B \wedge Q \}}$$

In each iteration:

- The invariant Q is preserved.
- The loop condition B can be evaluated again.
- The variant T decreases.

Termination: The relation $<$ on the subset $\{t : \mathbb{Z} \mid t > 0\}$ is well-founded.

“Merged” While Rule

Now **two** additional ingredients:

- **Invariant:** $Q : \mathbb{B}$ — as before, ensuring functional correctness
- **Variant** (or “bound function”): $T : \mathbb{Z}$ — ensuring termination

$$\frac{\{ B \wedge Q \wedge T = t_0 \}\ \ C\ \ \{ dom\ `B' \wedge Q \wedge T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ dom\ `B' \wedge Q \}\ \ \text{while } B \text{ do } C \text{ od}\ \ \{ \neg B \wedge Q \}}$$

provided $\neg occurs(`t_0', `B, C, Q, T')$

In each iteration:

- The invariant Q is preserved.
- The loop condition B can be evaluated again.
- The variant T decreases.

Relation-Algebraic Total and Partial Correctness

- Program correctness statement in LADM (and much current use): “Hoare triple”:

  $$\{ P \}\ C\ \{ Q \}$$

  Meaning (LADM ch. 10): “Total correctness”:
  If command C is started in a state in which the **precondition** P holds
  then it **will terminate** in a state in which the **postcondition** Q holds.

  Axiom “Total Correctness”:

  $$(P \Rightarrow[\langle C \rangle]\ Q) \quad \equiv \quad [\![C]\!]\ (\!|\ \text{sat } P\ |\!) \subseteq \text{sat } Q \quad \wedge \quad \text{sat } P \subseteq \text{Dom } [\![C]\!]$$

  (So far not modelling “undefined” expressions, only non-termination.)

- So far, we have been using the **dynamic logic** notation:

  $$P \Rightarrow[\ C\ ]\ Q$$

  with its **partial correctness meaning:**
  If command C is started in a state in which the **precondition** P holds
  then it will terminate **only** in a state in which the **postcondition** Q holds.

  Axiom “Partial Correctness”:

  $$(P \Rightarrow[\ C\ ]\ Q) \quad \equiv \quad [\![C]\!]\ (\!|\ \text{sat } P\ |\!) \subseteq \text{sat } Q$$

Total and Partial Correctness in Predicate Logic

- Program correctness statement in LADM (and much current use): "Hoare triple":

$$\{ P \}\ C\ \{ Q \}$$

Meaning (LADM ch. 10): "Total correctness":
If command C is started in a state in which the **precondition** P holds
then it **will terminate** in a state in which the **postcondition** Q holds.

Theorem "Total Correctness":
$(P \Rightarrow [\langle C \rangle] Q)$
$\equiv (\forall s_1, s_2 \bullet s_1 \in \mathsf{sat}\, P \wedge s_1 \,(\![\![C]\!]\!)\, s_2 \Rightarrow s_2 \in \mathsf{sat}\, Q)$
$\wedge (\forall s_1 \mid s_1 \in \mathsf{sat}\, P \bullet \exists s_2 \mid s_1 \,(\![\![C]\!]\!)\, s_2 \bullet s_2 \in \mathsf{sat}\, Q)$

- So far, we have been using the **dynamic logic** notation:

$$P \Rightarrow [\, C \,]\, Q$$

with its **partial correctness meaning:**
If command C is started in a state in which the **precondition** P holds
then it will terminate **only** in a state in which the **postcondition** Q holds.

Theorem "Partial Correctness":
$(P \Rightarrow [\, C \,]\, Q)$
$\equiv \forall s_1, s_2 \bullet s_1 \in \mathsf{sat}\, P \wedge s_1 \,(\![\![C]\!]\!)\, s_2 \Rightarrow s_2 \in \mathsf{sat}\, Q$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-26

Temporal Logic: PLTL

Fast Version of Syntax and Semantics of Propositional Logic in Ex11.3

- Given: A set $\mathcal{E}$ of **expressions** $e_1, e_2, \ldots$ (for example: "$x + 5$", "$3 \cdot (y + 2)$"
- An **atomic proposition** in Ex11.3 is an equation "$e_1 = e_2$", for example, "$2 \cdot x + 5 = 89$"
- A **formula** $\varphi, \psi, \ldots$ is (an abstract syntax tree) generated by the following "grammar" (informal):

$$\varphi \quad ::= \quad e_1 = e_2 \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

- A **state** is a function $\alpha : \mathcal{V} \to \mathbb{Z}$
- The semantics of propositional formula φ is the function $[\![\varphi]\!] : (\mathcal{V} \to \mathbb{Z}) \to \mathbb{B}$ that maps each state α to a truth value, the "value of φ in α":

$$[\![e_1 = e_2]\!]\ \alpha \quad = \quad ([\![e_1]\!]\ \alpha = [\![e_2]\!]\ \alpha)$$
$$[\![\neg\varphi]\!]\ \alpha \quad = \quad \neg([\![\varphi]\!]\ \alpha)$$
$$[\![\varphi \wedge \psi]\!]\ \alpha \quad = \quad [\![\varphi]\!]\ \alpha \wedge [\![\psi]\!]\ \alpha$$

$\wedge$: Formula constructor; $\wedge$: Boolean operator

- α **satisfies** φ iff $[\![\varphi]\!]\ \alpha = \mathit{true}$; this is also written: $\alpha \models \varphi$
- φ **is valid** iff $(\forall \alpha \bullet [\![\varphi]\!]\ \alpha = \mathit{true})$; this is also written: $\models \varphi$

Syntax and Semantics of Traditional Propositional Logic

- Given: A type $\mathcal{P}$ of **proposition symbols** $p, q, \ldots$ to be used as atomic propositions
- A **propositional formula** $\varphi, \psi, \ldots$ is (an abstract syntax tree) generated by the following "grammar" (informal):

  $\varphi ::= \mathsf{T} \mid \mathsf{F} \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi$
- A **state** is a function $\alpha : \mathcal{P} \to \mathbb{B}$
- The semantics of propositional formula φ is the function

  $[\![\varphi]\!] : (\mathcal{P} \to \mathbb{B}) \to \mathbb{B}$

  that maps each state α to a truth value, the "value of φ in α":

  $$\begin{aligned} [\![\mathsf{T}]\!]\ \alpha &= \mathit{true} \\ [\![\neg\varphi]\!]\ \alpha &= \neg([\![\varphi]\!]\ \alpha) \\ [\![\varphi \wedge \psi]\!]\ \alpha &= [\![\varphi]\!]\ \alpha \wedge [\![\psi]\!]\ \alpha \end{aligned}$$
- α **satisfies** φ iff $[\![\varphi]\!]\ \alpha = \mathit{true}$; this is also written: $\alpha \models \varphi$
- φ **is valid** iff $(\forall \alpha \bullet [\![\varphi]\!]\ \alpha = \mathit{true})$; this is also written: $\models \varphi$

Syntax and Semantics of Propositional Logic — Applications

- Define a (Haskell) datatype for propositional formule: *data PropForm p = ...*
- Write functions that takes each formula to its disjunctive/conjunctive normal form

toCNF, toDNF :: *PropForm p* → *PropForm p*

  Use CalcCheck to prove that your implementations are correct
- Define the semantics as an evaluation function

evalPropForm :: *PropForm p* → *State p* → **Bool**
- Define a representation of truth tables
- Write a truth table generation fucntion
- Write a validity checker using truth tables

validPropForm :: *PropForm p* → **Bool**
- Write a satisfiability checker using truth tables

satPropForm :: *PropForm p* → **Maybe** (*State p*)
- Look up the DPLL algorithm and write a more efficient satisfiability solver

Syntax and Semantics of Predicate Logic

- Given: A **vocabulary/signature** Σ consisting of
  - a countably infinite set $\mathcal{V}$ of **variable symbols** $v, v_1, v_2, \ldots$
  - a countable set of **function symbols** $f, g, \ldots$ (with arity information) — *fact*, _+_, 42
  - a countable set of **predicate symbols** $p, q, \ldots$ (with arity information) — *odd*, _=_, _>_
- A **term** t, t_1, t_2 is (an abstract syntax tree) generated by the following "grammar":

  $t ::= v \mid f(t_1, \ldots, t_n)$ — "$\mathit{fact}(5)$", "42", "$x + 2$"
- A **predicate-logic/first-order-logic formula** $\varphi, \psi, \ldots$ is (an abstract syntax tree) generated by the following "grammar":

  $\varphi ::= p(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid (\forall v \bullet \varphi) \mid (\exists v \bullet \varphi)$
- An **interpretation of Σ, also called a "Σ-structure"**, $\mathcal{A}$, consists of
  - a **domain** D
  - a mapping that maps each n-ary function symbol f to a function $f^{\mathcal{A}} : D^n \to D$
  - a mapping that maps each n-ary predicate symbol p to a function $p^{\mathcal{A}} : D^n \to \mathbb{B}$
- A **variable assignment** for $\mathcal{A}$ is a function $\alpha : \mathcal{V} \to D$
- Semantics of terms: $[\![t]\!]_{\mathcal{A}} : (\mathcal{V} \to D) \to D$
- Semantics of formulae: $[\![\varphi]\!]_{\mathcal{A}} : (\mathcal{V} \to D) \to \mathbb{B}$; we write "$\mathcal{A}, \alpha \models \varphi$" for $[\![\varphi]\!]_{\mathcal{A}}\ \alpha = \mathit{true}$
- ... ⟶ **RSD chapters 3, 4**

Intended Infinite Program Executions

- Even simple imperative programming languages have programs that do not terminate — **while** *true* **do** ...
- Not all programs are expected to terminate:
  - Operating systems
  - Bank databases
  - Online shops
- Pre-postcondition specifications are useless for programs that are expected to not terminate!
- Different patterns of specification are used for such systems:
  - Each request will generate a response
  - The ledger is always balanced
  - Shipping commands are sent to the warehouse only after payment is confirmed
- Central concept: **Time**
- System behaviour: Different states at different time points
- Plausible abstraction: Discrete time, with time points taken from $\mathbb{N}$
- **Infinite state sequences**: Functions of type $\mathbb{N} \rightarrow \mathsf{State}$

How to Reason About Infinite State Sequences?

- **Infinite state sequences**: Functions of type $\mathbb{N} \rightarrow \mathsf{State}$
- Specification example sketches in predicate logic:
  - $\forall\, t_0, rId, d_{in} \;\; | \;\; request(rId, d_{in}, t_0)$
    $\bullet\; \exists\, t_1, d_{out} \;\; | \;\; t_0 < t_1 \quad \bullet\; response(rId, d_{out}, t_1)$
    $\wedge\; appropriate(d_{out}, d_{in})$
  - $\forall\, t \;\bullet\; (\sum\, a : Account \;\bullet\; balance\ a\ t) = 0$
  - ...
- **Lots of quantification about time points!**
- **Quantification about time points follows relatively few patterns!**
- **Temporal logics** **"internalise"** these time point quantification patterns and allow to express them **without bound variables for time points**.

Important Temporal Modalities

- **Quantification about time points follows relatively few patterns!**
- **Temporal logics** **"internalise"** these time point quantification patterns and allow to express them **without bound variables for time points**.

Consider the following timeline:

$x=3, y=5 \longrightarrow x=2, y=5 \longrightarrow x=1, y=7 \longrightarrow x=2, y=7 \longrightarrow x=5, y=8 \longrightarrow x=3, y=8 \longrightarrow x=0, y=7 \longrightarrow \cdots$

We have:

- $F\ (y = 3 \cdot x + 1)$ —- "eventually $(y = 3 \cdot x + 1)$"; "at some time in the future, $(y = 3 \cdot x + 1)$"
- $G\ (y > x)$ —- "always $(y > x)$". "at all times in the future, $(y > x)$"
- $(x < 4)\ U\ (y = 8)$ —- "$(x < 4)$ until $(y = 8)$"
- $X\ (x = 2)$ —- "in the next state, $(y = 2)$"
- $(x = 3)$ —- "(in the current state,) $(x = 3)$"

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL)

- Given: A set A of **atomic propositions** $p, q, \ldots$
- A **PLTL formula** $\varphi, \psi, \ldots$ is (an abstract syntax tree) generated by the following "grammar" (informal):

  $$\varphi ::= \mathsf{T} \mid \mathsf{F} \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid F\,\varphi \mid G\,\varphi \mid X\,\varphi \mid \varphi\, U\, \psi$$
- A **state** associates a truth value with each atom: $\mathsf{State} \;=\; A \to \mathbb{B}$
- A **time line** α associates a state with each time point — for simplicity, we use $\mathbb{N}$ for time points:

  $$\alpha \;:\; \mathbb{N} \to A \to \mathbb{B}$$
- Given an LTL formula φ and a time line α, the semantics of φ in α, written "$[\![\, \varphi \,]\!]\ \alpha$", is a function that associates with each time point $t \;:\; \mathbb{N}$ the truth value "$[\![\, \varphi \,]\!]\ \alpha\ t$":

  Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 1

$[\![\, \varphi \,]\!]\ \alpha\ t = \mathit{true}$ iff LTL formula φ holds in time line $\alpha \;:\; \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

An atomic proposition p is true at time t iff the time line contains, at time t, a state in which p is true:

"Semantics of LTL atoms": $[\![\, {}^{\backprime}p \,]\!]\ \alpha\ t \;\equiv\; \alpha\ t\ p$

"Semantics of LTL $\neg$": $[\![\, \neg' \varphi \,]\!]\ \alpha\ t \;\equiv\; \neg\, [\![\, \varphi \,]\!]\ \alpha\ t$

"Semantics of LTL $\wedge$": $[\![\, \varphi \wedge' \psi \,]\!]\ \alpha\ t \;\equiv\; [\![\, \varphi \,]\!]\ \alpha\ t \wedge [\![\, \psi \,]\!]\ \alpha\ t$

"Semantics of LTL $\vee$": $[\![\, \varphi \vee' \psi \,]\!]\ \alpha\ t \;\equiv\; [\![\, \varphi \,]\!]\ \alpha\ t \vee [\![\, \psi \,]\!]\ \alpha\ t$

"Semantics of LTL $\Rightarrow$": $[\![\, \varphi \Rightarrow' \psi \,]\!]\ \alpha\ t \;\equiv\; [\![\, \varphi \,]\!]\ \alpha\ t \Rightarrow [\![\, \psi \,]\!]\ \alpha\ t$

- $[\![\, p \,]\!]\ \alpha\ 0 \;=\; ?$
- $[\![\, p \,]\!]\ \alpha\ 3 \;=\; ?$
- $[\![\, q \,]\!]\ \alpha\ 0 \;=\; ?$
- $[\![\, p \wedge q \,]\!]\ \alpha\ 0 \;=\; ?$
- $[\![\, p \vee \neg q \,]\!]\ \alpha\ 3 \;=\; ?$
- $[\![\, q \Rightarrow r \,]\!]\ \alpha\ 42 \;=\; ?$

$\alpha \;=$

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, ...	√		√	√
7, 17, 27, ...	√	√		
8, 18, 28, ...	√		√	
9, 19, 29, ...	√	√	√	
10, 20, 30, ...	√		√	
11, 21, 31, ...	√	√		
12, 22, 32, ...	√		√	
13, 23, 33, ...	√	√		
14, 24, 34, ...	√		√	
15, 25, 35, ...	√	√		

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 2

$[\![\, \varphi \,]\!]\ \alpha\ t = \mathit{true}$ iff LTL formula φ holds in time line $\alpha \;:\; \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$F\,\varphi$ is true at time t if φ is true at some time $t' \geq t$:

"Semantics of `$F$`":

$$[\![\, F\,\varphi \,]\!]\ \alpha\ t \;\equiv\; \exists\, t' : \mathbb{N} \;\mid\; t \leq t' \;\bullet\; [\![\, \varphi \,]\!]\ \alpha\ t'$$

$G\,\varphi$ is true at time t if φ is true at all times $t' \geq t$.

"Semantics of `$G$`":

$$[\![\, G\,\varphi \,]\!]\ \alpha\ t \;\equiv\; \forall\, t' : \mathbb{N} \;\mid\; t \leq t' \;\bullet\; [\![\, \varphi \,]\!]\ \alpha\ t'$$

- $[\![\, G\,p \,]\!]\ \alpha\ 0 \;=\; ?$
- $[\![\, G\,p \,]\!]\ \alpha\ 5 \;=\; ?$
- $[\![\, F\,q \,]\!]\ \alpha\ 0 \;=\; ?$
- $[\![\, F\,s \,]\!]\ \alpha\ 7 \;=\; ?$
- $[\![\, F\,\neg p \,]\!]\ \alpha\ 0 \;=\; ?$
- $[\![\, F\,\neg p \,]\!]\ \alpha\ 100 \;=\; ?$

$\alpha \;=$

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, ...	√		√	√
7, 17, 27, ...	√	√		
8, 18, 28, ...	√		√	
9, 19, 29, ...	√	√	√	
10, 20, 30, ...	√		√	
11, 21, 31, ...	√	√		
12, 22, 32, ...	√		√	
13, 23, 33, ...	√	√		
14, 24, 34, ...	√		√	
15, 25, 35, ...	√	√		

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 3

$[\![\varphi]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$X\ \varphi$ is true at time t iff φ is true at time $t + 1$:

"Semantics of `$X$`":

$$[\![X\ \varphi]\!]\ \alpha\ t \ \equiv\ [\![\varphi]\!]\ \alpha\ (\mathsf{suc}\ t)$$

- $[\![X\ p]\!]\ \alpha\ 0 = ?$
- $[\![X\ q]\!]\ \alpha\ 0 = ?$
- $[\![q \wedge X\ r]\!]\ \alpha\ 1 = ?$
- $[\![G\ F\ (q \wedge X\ r)]\!]\ \alpha\ 0 = ?$
- $[\![F\ (s \wedge X\ s)]\!]\ \alpha\ 0 = ?$
- $[\![F\ (s \wedge X\ s)]\!]\ \alpha\ 10 = ?$
- $[\![G\ (q \equiv X\ r)]\!]\ \alpha\ 12 = ?$
- $[\![G\ F\ (q \equiv X\ r)]\!]\ \alpha\ 12 = ?$

$\alpha =$

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, ...	√		√	√
7, 17, 27, ...	√	√		
8, 18, 28, ...	√		√	
9, 19, 29, ...	√	√	√	
10, 20, 30, ...	√		√	
11, 21, 31, ...	√	√		
12, 22, 32, ...	√		√	
13, 23, 33, ...	√	√		
14, 24, 34, ...	√		√	
15, 25, 35, ...	√	√		

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 4

$[\![\varphi]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$\varphi\ U\ \psi$ is true at time t if ψ is true at some time $t' \geq t$, and for all times t'' such that $t \leq t'' < t'$, φ is true.

Axiom "Semantics of `$U$`": ▪▪▪▪▪▪"until"

$$\begin{aligned} & [\![\varphi\ U\ \psi]\!]\ \alpha\ t \\ \equiv\ & \exists\, t' : \mathbb{N} \mid t \leq t' \bullet [\![\psi]\!]\ \alpha\ t' \wedge \forall\, t'' : \mathbb{N} \mid t \leq t'' < t' \bullet [\![\varphi]\!]\ \alpha\ t'' \end{aligned}$$

- $[\![p\ U\ q]\!]\ \alpha\ 0 = ?$
- $[\![p\ U\ s]\!]\ \alpha\ 0 = ?$
- $[\![\neg s\ U\ \neg p]\!]\ \alpha\ 0 = ?$
- $[\![p\ U\ (q \wedge r)]\!]\ \alpha\ 42 = ?$
- $[\![p\ U\ (q \wedge s)]\!]\ \alpha\ 42 = ?$
- $[\![(p \vee r)\ U\ s]\!]\ \alpha\ 1 = ?$

$\alpha =$

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, ...	√		√	√
7, 17, 27, ...	√	√		
8, 18, 28, ...	√		√	
9, 19, 29, ...	√	√	√	
10, 20, 30, ...	√		√	
11, 21, 31, ...	√	√		
12, 22, 32, ...	√		√	
13, 23, 33, ...	√	√		
14, 24, 34, ...	√		√	
15, 25, 35, ...	√	√		

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-28

More About Temporal Logics, Model Checking

Temporal Logics for Specification of Reactive and Distributed Systems

- **Reactive Systems**: No clear input-output relation
  - Operating systems
  - Embedded systems
  - Network protocols
- Specification techniques: **Temporal logics**
  - Rich choice of temporal logics — multiple classification criteria
  - Some important logics are (polynomial-time) decidable — **Model checking**
- Applications: Safety- and liveness properties
  - **Safety property:** "Something bad will never happen"
  - **Liveness property:** "Something good will eventually happen"
- Application area: Concurrent systems, protocols, ...

Modal Logics

- Original philosophical motivation: Express different **modalities**:

  The proposition "Napoleon was victorious at Waterloo"
  - is false in this world,
  - but could be true in another world.
- Typical modal operators:
  - "**possibly**": $\diamond p$ — "it is imaginable that p holds" "diamond p"
  - "**necessarily**": $\Box p$ — "it is not imaginable that p doesn't hold" "box p"
- Kripke (1963): "**possible world semantics**" (orig. Kanger 1957)

Temporal Logics

- Prior (1955): **Tense Logic —** notation still customary today
  - instead of $\diamond p$ now temporally: $\mathbf{F}\, p$ — "p will eventually be true"
  - instead of $\Box p$ now temporally: $\mathbf{G}\, p$ — "p will always be true"
- Dynamic Logic [Pratt 1976 (originally developed for Hoare logic in course notes 1974)]:
  - Parameterised box modality: $[\, A\,]\varphi$ means "after performing action A, the condition φ will always hold"
  - Useful for pre-/post-condition correctness statements: $P \Rightarrow ([\, C\,]\, Q)$
- Pnueli (1977): "**The Temporal Logic of Programs**":

  Argues for using temporal logics as tool for specification and verification, in particular for **reactive systems** such as operating systems and network protocols
- Two kinds of applications: Temporal logics are used
  - in software technology, to let the world reason about programs
  - in AI, to let programs reason about the world

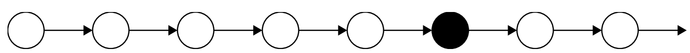
Different Treatments of Time

- **Future Only versus Also Past**
  - Philosophiscal approaches: Past at least as important as future
  - Software: Frequently only future
  - Past operators are frequently useful in compositional specifications.
- **Discrete Time versus Continuous Time**
  - Continuous (or dense) time first considered in philosophy
  - Possible application in real time systems
- **Time Points versus Time Intervals**
  - Some properties are easier to formulate using intervals.
- The following distinction is mainly semantic, but also reflected in syntax:
  - **Linear Time**: At any point only **one** possible future
  - **Branching Time**: At any point **multiple** possible futures

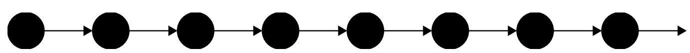
  Both approaches are used in software technology

Temporal Operators of Propositional Linear-Time Temporal Logic (PLTL)

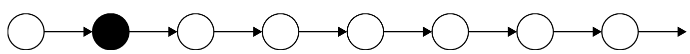
- $\mathbf{F}\,p$ — "eventually p"



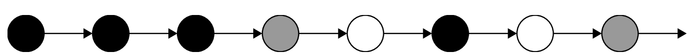
- $\mathbf{G}\,p$ — "always p"



- $\mathbf{X}\,p$ — "in the next state p"



- $p\ \mathbf{U}\ q$ — "eventually q, and until then p" (**until**)



Propositional Linear-Time Temporal Logic — Syntax

Definition: The set of formulae of **propositional linear-time temporal logic** is the smallest set generated by the following rules:

- every atomic proposition $P : AP$ is a formula;
- if p and q are formulae, then $p \wedge q$ and $\neg p$ are formulae, too;
- if p and q are formulae, then $p\ \mathbf{U}\ q$ and $\mathbf{X}\,p$ formulae, too.

Abbreviations:

$p \vee q$	$:\equiv$	$\neg(\neg p \wedge \neg q)$	$\mathbf{F}\,p$	$:\equiv$	$\mathit{true}\ \mathbf{U}\ p$	
$p \Rightarrow q$	$:\equiv$	$\neg p \vee q$	$\mathbf{G}\,p$	$:\equiv$	$\neg \mathbf{F}\,\neg p$	
$p \Leftrightarrow q$	$:\equiv$	$(p \Rightarrow q) \wedge (q \Rightarrow p)$	$\mathbf{F}^{\infty}\,p$	$:\equiv$	$\mathbf{G}\,\mathbf{F}\,p$	— **"infinitely often"**
true	$:\equiv$	$p \vee \neg p$	$\mathbf{G}^{\infty}\,p$	$:\equiv$	$\mathbf{F}\,\mathbf{G}\,p$	— **"almost everywhere"**
false	$:\equiv$	$\neg \mathit{true}$	$p\ \mathbf{B}\ q$	$:\equiv$	$\neg((\neg p)\ \mathbf{U}\ q)$	— "p before q"

Semantics of the Temporal Modalities in PLTL

$[\![\,\varphi\,]\!]\;\alpha\;t = \mathit{true}$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t

Declaration: $[\![\_]\!] : \text{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

- $F\ \varphi$ is true at time t if φ is true at some time $t' \geq t$.
- $G\ \varphi$ is true at time t if φ is true at all times $t' \geq t$.
- $X\ \varphi$ is true at time t iff φ is true at time $t+1$.
- $\varphi\ U\ \psi$ is true at time t if ψ is true at some time $t' \geq t$, and for all times t'' such that $t \leq t'' < t'$, φ is true.

α =

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, ...	√		√	√
7, 17, 27, ...	√	√		
8, 18, 28, ...	√		√	
9, 19, 29, ...	√	√	√	
10, 20, 30, ...	√		√	
11, 21, 31, ...	√	√		
12, 22, 32, ...	√		√	
13, 23, 33, ...	√	√		
14, 24, 34, ...	√		√	
15, 25, 35, ...	√	√		

- $[\![\,\neg s\ U \neg p\,]\!]\ \alpha\ 0\ =\ ?$
- $[\![\,p\ U\ (q \wedge r)\,]\!]\ \alpha\ 42\ =\ ?$
- $[\![\,p\ U\ (q \wedge s)\,]\!]\ \alpha\ 42\ =\ ?$
- $[\![\,(p \vee r)\ U\ s\,]\!]\ \alpha\ 1\ =\ ?$

Important Valid Formulae

$\models \mathbf{G}\,\neg p \Leftrightarrow \neg \mathbf{F}\,p$ $\quad\models \mathbf{G}^{\infty}\,\neg p \Leftrightarrow \neg \mathbf{F}^{\infty}\,p$ $\quad\models \mathbf{X}\,\neg p \Leftrightarrow \neg \mathbf{X}\,p$

$\models \mathbf{F}\,\neg p \Leftrightarrow \neg \mathbf{G}\,p$ $\quad\models \mathbf{F}^{\infty}\,\neg p \Leftrightarrow \neg \mathbf{G}^{\infty}\,p$ $\quad\models ((\neg p)\ \mathbf{U}\ q) \Leftrightarrow \neg(p\ \mathbf{B}\ q)$

Idempotencies

$\models \mathbf{F}\,\mathbf{F}\,p \Leftrightarrow \mathbf{F}\,p$

$\models \mathbf{G}\,\mathbf{G}\,p \Leftrightarrow \mathbf{G}\,p$

$\models \mathbf{F}^{\infty}\,\mathbf{F}^{\infty}\,p \Leftrightarrow \mathbf{F}^{\infty}\,p$

$\models \mathbf{G}^{\infty}\,\mathbf{G}^{\infty}\,p \Leftrightarrow \mathbf{G}^{\infty}\,p$

Implications

$\models p \Rightarrow \mathbf{F}\,p$ $\quad\models \mathbf{G}\,p \Rightarrow p$

$\models \mathbf{X}\,p \Rightarrow \mathbf{F}\,p$ $\quad\models \mathbf{G}\,p \Rightarrow \mathbf{X}\,p$

$\models \mathbf{G}\,p \Rightarrow \mathbf{F}\,p$ $\quad\models \mathbf{G}\,p \Rightarrow \mathbf{X}\,\mathbf{G}\,p$

$\models p\ \mathbf{U}\ q \Rightarrow \mathbf{F}\,q$ $\quad\models \mathbf{G}^{\infty}\,q \Rightarrow \mathbf{F}^{\infty}\,q$

$\models \mathbf{X}\,\mathbf{F}\,p \Leftrightarrow \mathbf{F}\,\mathbf{X}\,p$ $\quad\models \mathbf{X}\,\mathbf{G}\,p \Leftrightarrow \mathbf{G}\,\mathbf{X}\,p$ $\quad\models ((\mathbf{X}\,p)\ \mathbf{U}\ (\mathbf{X}\,q)) \Leftrightarrow \mathbf{X}\,(p\ \mathbf{U}\ q)$

$$\models\quad \mathbf{F}^{\infty}\,p \Leftrightarrow \mathbf{X}\,\mathbf{F}^{\infty}\,p \Leftrightarrow \mathbf{F}\,\mathbf{F}^{\infty}\,p \Leftrightarrow \mathbf{G}\,\mathbf{F}^{\infty}\,p \Leftrightarrow \mathbf{F}^{\infty}\,\mathbf{F}^{\infty}\,p \Leftrightarrow \mathbf{G}^{\infty}\,\mathbf{F}^{\infty}\,p$$

$$\models\quad \mathbf{G}^{\infty}\,p \Leftrightarrow \mathbf{X}\,\mathbf{G}^{\infty}\,p \Leftrightarrow \mathbf{F}\,\mathbf{G}^{\infty}\,p \Leftrightarrow \mathbf{G}\,\mathbf{G}^{\infty}\,p \Leftrightarrow \mathbf{F}^{\infty}\,\mathbf{G}^{\infty}\,p \Leftrightarrow \mathbf{G}^{\infty}\,\mathbf{G}^{\infty}\,p$$

(considering $\Leftrightarrow$ to be conjunctional)

Interplay between Junctors and Temporal Operators

$\models \mathbf{F}\,(p \vee q) \Leftrightarrow (\mathbf{F}\,p \vee \mathbf{F}\,q)$ $\quad\models \mathbf{G}\,(p \wedge q) \Leftrightarrow (\mathbf{G}\,p \wedge \mathbf{G}\,q)$

$\models \mathbf{F}^{\infty}\,(p \vee q) \Leftrightarrow (\mathbf{F}^{\infty}\,p \vee \mathbf{F}^{\infty}\,q)$ $\quad\models \mathbf{G}^{\infty}\,(p \wedge q) \Leftrightarrow (\mathbf{G}^{\infty}\,p \wedge \mathbf{G}^{\infty}\,q)$

$\models p\ \mathbf{U}\ (q \vee r) \Leftrightarrow (p\ \mathbf{U}\ q \vee p\ \mathbf{U}\ r)$ $\quad\models (p \wedge q)\ \mathbf{U}\ r \Leftrightarrow (p\ \mathbf{U}\ r \wedge q\ \mathbf{U}\ r)$

$\models \mathbf{X}\,(p \vee q) \Leftrightarrow (\mathbf{X}\,p \vee \mathbf{X}\,q)$ $\quad\models \mathbf{X}\,(p \Rightarrow q) \Leftrightarrow (\mathbf{X}\,p \Rightarrow \mathbf{X}\,q)$

$\models \mathbf{X}\,(p \wedge q) \Leftrightarrow (\mathbf{X}\,p \wedge \mathbf{X}\,q)$ $\quad\models \mathbf{X}\,(p \Leftrightarrow q) \Leftrightarrow (\mathbf{X}\,p \Leftrightarrow \mathbf{X}\,q)$

$\models (\mathbf{G}\,p \vee \mathbf{G}\,q) \Rightarrow \mathbf{G}\,(p \vee q)$ $\quad\models \mathbf{F}\,(p \wedge q) \Rightarrow \mathbf{F}\,p \wedge \mathbf{F}\,q$

$\models (\mathbf{G}^{\infty}\,p \vee \mathbf{G}^{\infty}\,q) \Rightarrow \mathbf{G}^{\infty}\,(p \vee q)$ $\quad\models \mathbf{F}^{\infty}\,(p \wedge q) \Rightarrow \mathbf{F}^{\infty}\,p \wedge \mathbf{F}^{\infty}\,q$

$\models ((p\ \mathbf{U}\ r) \vee (q\ \mathbf{U}\ r)) \Rightarrow ((p \vee q)\ \mathbf{U}\ r)$ $\quad\models (p\ \mathbf{U}\ (q \wedge r)) \Rightarrow ((p\ \mathbf{U}\ q) \wedge (p\ \mathbf{U}\ r))$

Monotonicity and Fixpoint Characterisations

$\models \mathbf{G}\,(p \Rightarrow q) \Rightarrow (\mathbf{F}\,p \Rightarrow \mathbf{F}\,q)$

$\models \mathbf{G}\,(p \Rightarrow q) \Rightarrow (\mathbf{G}\,p \Rightarrow \mathbf{G}\,q)$

$\models \mathbf{G}\,(p \Rightarrow q) \Rightarrow ((p\,\mathbf{U}\,r) \Rightarrow (q\,\mathbf{U}\,r))$

$\models \mathbf{G}\,(p \Rightarrow q) \Rightarrow (\mathbf{X}\,p \Rightarrow \mathbf{X}\,q)$

$\models \mathbf{G}\,(p \Rightarrow q) \Rightarrow (\mathbf{F}^{\infty}\,p \Rightarrow \mathbf{F}^{\infty}\,q)$

$\models \mathbf{G}\,(p \Rightarrow q) \Rightarrow (\mathbf{G}^{\infty}\,p \Rightarrow \mathbf{G}^{\infty}\,q)$

$\models \mathbf{G}\,(p \Rightarrow q) \Rightarrow ((r\,\mathbf{U}\,p) \Rightarrow (r\,\mathbf{U}\,q))$

Fixpoint Characterisations:

$\models \mathbf{F}\,p \Leftrightarrow p \vee \mathbf{X}\,\mathbf{F}\,p$

$\models \mathbf{G}\,p \Leftrightarrow p \wedge \mathbf{X}\,\mathbf{G}\,p$

$\models (p\,\mathbf{U}\,q) \Leftrightarrow q \vee (p \wedge \mathbf{X}\,(p\,\mathbf{U}\,q))$

$\models (p\,\mathbf{B}\;q) \Leftrightarrow \neg q \wedge (p \vee \mathbf{X}\,(p\,\mathbf{B}\;q))$

Variants of the Basic Temporal Operators

- $p\,\mathbf{U}\,q$, until now, is known as "**strong until**":
  There is a future state q, and until then p.
- Alternative notations: $p\,\mathbf{U}_s\,q$ or $p\,\mathbf{U}_{\exists}\,q$.
- **Weak until** $p\,\mathbf{U}_w\,q$ or $p\,\mathbf{U}_{\forall}\,q$:
  p holds as long as q does not hold — if necessary, forever.
- $x \models p\,\mathbf{U}_{\forall}\,q$ iff for all $j : \mathbb{N}$ we have $x^j \models p$ as far as for all $k \leq j$ we have $x^k \models \neg q$.

We have:

- $\models p\,\mathbf{U}_{\exists}\,q \Leftrightarrow p\,\mathbf{U}_{\forall}\,q \wedge \mathbf{F}\,q$
- $\models p\,\mathbf{U}_{\forall}\,q \Leftrightarrow (p\,\mathbf{U}_{\exists}\,q \vee \mathbf{G}\,p) \Leftrightarrow (p\,\mathbf{U}_{\exists}\,q \vee \mathbf{G}\,(p \wedge \neg q))$

Past

Until now, all operators are future-related — explicitly:

- $\mathbf{F}^{+}\,p$ — "in the future, eventually p"
- $\mathbf{G}^{+}\,p$ — "in the future, always p"
- $\mathbf{X}^{+}\,p$ — "in the next state p"
- $p\,\mathbf{U}^{+}\,q$ — "in the future, eventually q, and until then p"

Purely future-oriented propositional linear-time temporal logic —
Propositional Linear-time Temporal Logic / Future: PLTLF

Corresponding past-oriented operators (originally P, H, and S for **since**):

- $\mathbf{F}^{-}\,p$ — "in the past at some point p"
- $\mathbf{G}^{-}\,p$ — "in the past, always p"
- $p\,\mathbf{U}^{-}\,q$ — "in the past at some point q, and since then p"
- $\mathbf{X}^{-}_{\exists}\,p$ — "in the previous state we had p"

Logic only with past-oriented operators: PLTLP; with both: PLTLB.

Safety

- Safety properties: “nothing bad happens”
- Safety properties are **invariance properties:**
  Every finite prefix of the execution satisfies the invariance condition
- In PLTLB: initially equivalent to **G** p for a past formula p:
  “nothing bad has happened until now” must always be true.
- Examples Safety Properties:
  - **Partial correctness** wrt. precondition φ and postcondition ψ:
    If a program (with start label l_0 and halting label l_h) starts executing in a state satisfying the precondition φ and terminates, the the terminating state satisfies the postcondition ψ:

$$\text{at}l_0 \land \varphi \Rightarrow \mathbf{G}\ (\text{at}l_h \Rightarrow \psi)$$

  - **Mutual Exclusion**: **G** $(\neg(\text{atCS}_1 \land \text{atCS}_2))$
  - **Deadlock-freeness**: **G** $(\text{enabled}_1 \lor \ldots \lor \text{enabled}_m)$

Liveness

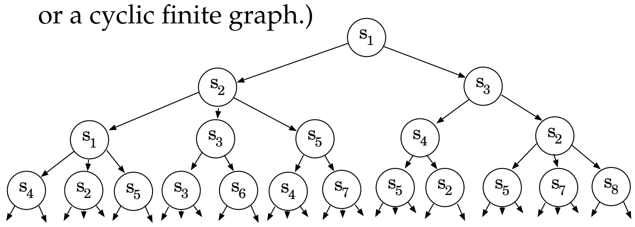
- **Liveness**: “Something good will still happen (often enough)”
- p is an “**invincible**” past formula iff every finite sequence x has a finite extension x' such that p holds in the last state of x':

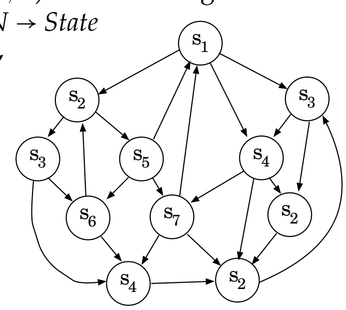
$$[\![p]\!]\ x'\ (\mathit{length} x') \equiv \mathit{true}$$

- A **pure liveness property** is a PLTLB formula that is initially equivalent to a formula **F** p, **G F** p or **F G** p, where p is an invincible past formula
- If p is a pure liveness property, then every finite sequence x can be extended to a finite or infinite sequence x' such that $(x', 0) \models p$
- **Temporal implication G** $(p \Rightarrow \mathbf{F}\ q)$ (where p and q are past formulae) is a generic liveness property

Propositional Branching-time Temporal Logic

- Semantic setting: A **branching-time structure** is a graph (N, E) with total edge relation E (“no sinks”) and a node labelling with states $L : N \to \mathit{State}$
  (This can in particular be an infinite “commputational tree” or a cyclic finite graph.)

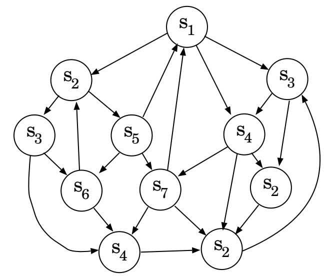




- “Path quantification”:
  - E φ means “φ holds in some future”, “φ holds on some infinite path”
  - A φ means “φ holds in all possible futures”, “φ holds on all infinite paths”
- “Computational Tree Logic” CTL: Path quantifiers (E , A) and temporal modalities (**F** , **G** , **U** , **X**) only occur together.

CTL Specification Patterns

- "Path quantification":
  - E φ means "φ holds in some future", "φ holds on some infinite path"
  - A φ means "φ holds in all possible futures", "φ holds on all infinite paths"
- "Computational Tree Logic" CTL: Path quantifiers (E , A) and temporal modalities (**F** , **G** , **U** , **X**) only occur together.



- Example CTL Specifications:
  - E **F** (started ∧ ¬ready)
  - A **G** (requested ⇒ A **F** acknowledged)
  - A **G** (A **F** enabled)
  - A **F** (A **G** deadlock)
  - A **G** (E **F** restart)
  - A **G** (floor = 2 ∧ direction = up ∧ ButtonPressed5 ⇒ A [direction = up **U** floor = 5])
  - A **G** (floor = 3 ∧ idle ∧ door = closed ⇒ E **G** (floor = 3 ∧ idle ∧ door = closed))

Small Models Theorem for CTL

Theorem: Let p_0 be a CTL formula of length n. Then the following statements are equivalent:

- p_0 is satisfiable,that is, there is a branching-time structure in which p_0 holds, that is, a **model** of p_0
- p_0 has an infinite tree model with finite branching degree in $\mathcal{O}(n)$.
- p_0 **has a finite model of size** $\leq n \cdot 2^n$**.**

Why is this useful? — **Synthesis of correct-by-construction automata!** (For satisfiable specifications. . .)

But:

Theorem: The satisfiability test for CTL is DEXPTIME complete.

Model Checking

The **Model Checking Problem**:

$$M \overset{?}{\vDash} p$$

I.e., is a given finite structure M a model for a given temporal logic formula p?

I.e., does a given **implementation** M satisfy the given temporal logic **specification** p?

- The model checking problem for propositional temporal logics is **decidable**.
- The model checking problem for PLTL(F,X) is PSPACE-complete.
- The model checking problem for PLTL(F) ist NP-complete.
- The model checking problem for CTL* is PSPACE-complete.
- The model checking problem for CTL is solvable in **deterministic polynomial time**.

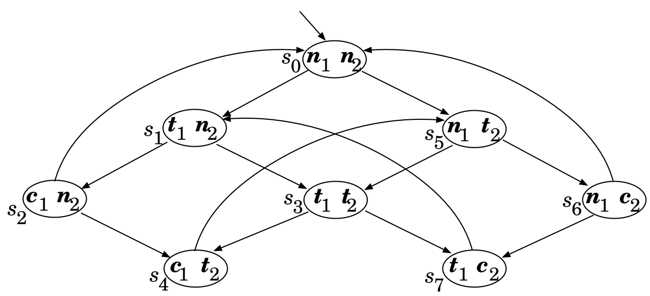
A CTL Model Checker: SMV

- Developed since 1992 at Carnegie Mellon University
- OBDD-based symbolic model checking for CTL
- Finite datatypes: Booleans, enumeration types, finite arrays
- Model description: Arbitrary propositional-logic formulae allowed
- Safe model description: Parallel assignments
- Original motivation: **hardware description**

```
MODULE main
VAR
  request : boolean;
  status  : {ready, busy};
ASSIGN
  init(status) := ready;
  next(status) :=
      case
        request : busy;
        1 : {ready, busy};
      esac;
SPEC
  AG(request → AF status=busy)
```

SMV Example from [Huth, Ryan]: Mutual Exclusion

Two processes, each with three states: "n": non-critical, "t": trying, "c": critical.
First protocol:



Safety	$\Phi_1 :\equiv \mathsf{A}\,\mathbf{G}\,\neg(c_1 \wedge c_2)$
Liveness	$\Phi_2 :\equiv \mathsf{A}\,\mathbf{G}\,(t_1 \Rightarrow \mathsf{A}\,\mathbf{F}\,c_1)$
Non-blocking	$\Phi_3 :\equiv \mathsf{A}\,\mathbf{G}\,(n_1 \Rightarrow \mathsf{E}\,\mathbf{X}\,t_1)$
No strict sequencing	$\Phi_4 :\equiv \mathsf{E}\,\mathbf{F}\,(c_1 \wedge \mathsf{E}\,[c_1\,\mathbf{U}\,(\neg c_1 \wedge \mathsf{E}\,[\neg c_2\,\mathbf{U}\,c_1])])$

First Translation into SMV Input Language

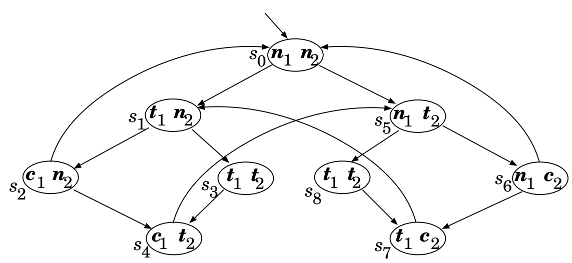
```
MODULE main
VAR
  p1 : {n, t, c};
  p2 : {n, t, c};
ASSIGN
  init(p1) := n;
  init(p2) := n;
TRANS
  (next(p2) = p2 & ((p1 = n → next(p1) = t) &
                    (p1 = t → next(p1) = c) &
                    (p1 = c → next(p1) = n))) |
  (next(p1) = p1 & ((p2 = n → next(p2) = t) &
                    (p2 = t → next(p2) = c) &
                    (p2 = c → next(p2) = n)))
TRANS next(p1) = c → next(p2) ≠ c

SPEC  AG !(p1=c & p2=c)
SPEC  AG (p1=t → AF p1=c)
SPEC  AG (p1=n → EX p1=t)
SPEC  EF (p1=c & E[p1=c U (p1≠c & E[ p2≠c  U p1=c])])
```

SMV Output

```
-- specification  AG (!(p1 = c & p2 = c))  is  true
-- specification  AG (p1 = t → AF p1 = c)  is  false
-- as demonstrated by the following execution sequence
state 1.1:
p1 = n,  p2 = n

-- loop starts here --
state 1.2:
p1 = t

state 1.3:
p2 = t

state 1.4:
p2 = c

state 1.5:
p2 = n

-- specification  AG (p1 = n → EX p1 = t)  is  true
-- specification  EF (p1 = c & E(p1 = c U (p1 ≠ c & E(p2 ...  is  true
```

Mutual Exclusion — continued

Safety	$\Phi_1 :\equiv \mathsf{A}\,\mathbf{G}\,\neg(c_1 \wedge c_2)$
Liveness	$\Phi_2 :\equiv \mathsf{A}\,\mathbf{G}\,(t_1 \Rightarrow \mathsf{A}\,\mathbf{F}\,c_1)$
Non-blocking	$\Phi_3 :\equiv \mathsf{A}\,\mathbf{G}\,(n_1 \Rightarrow \mathsf{E}\,\mathbf{X}\,t_1)$
No strict sequencing	$\Phi_4 :\equiv \mathsf{E}\,\mathbf{F}\,(c_1 \wedge \mathsf{E}\,[c_1\,\mathbf{U}\,(\neg c_1 \wedge \mathsf{E}\,[\neg c_2\,\mathbf{U}\,c_1])])$



That can even be synthesised from the specification!

Two Different Model Concepts

	Logic	Toys
Think:	"implementation satisfies specification"	"model airplane"
Context:	a vocabulary / signature / API declaration	(air transport) domain knowledge
What is a model of X?	a structure/implementation that satisfies specification X (useful where an implementation of X is needed)	some smaller/simpler/more-abstract version of airplane/system X "looks like X, may or may not fly like X"
Important derived concepts	**Model checking**	"Model-driven engineering" (MDE)

Reading More about Temporal Logics

- E. Allen Emerson: **Temporal and Modal Logic**, pages 995–1072 of Jan van Leeuwen (ed.): **Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics**, Elsevier Science Publishers B. V., 1990
  `https://doi.org/10.1016/B978-0-444-88074-1.50021-4`

  Thode Library Bookstacks: QA 76 .H279 1990

  "Post-print"? linked on Wikipedia:
  `https://profs.info.uaic.ro/~masalagiu/pub/handbook3.pdf`

- Michael R. A. Huth and Mark D. Ryan: **Logic in Computer Science, Modelling and Reasoning about Systems, 2nd edition**, Cambridge University Press 2004,

  Thode Library Bookstacks: QA 76.9 .L63H88 2004

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-11-29

Part 1: Frama-C and ACSL

Frama-C: `https://www.frama-c.com/`

> Frama-C is an open-source extensible and collaborative platform dedicated to source-code analysis of C software. The Frama-C analyzers assist you in various source-code-related activities, from the navigation through unfamiliar projects up to the certification of critical software.

- Platform with multiple plug-ins
- Plug-in for **total correctness** proofs: WP
- Specification language: ACSL "ANSI C Specification Language"
  - Similar to JML
  - Based on first-order predicate logic
  - Not all ACSL features are currently supported by Frama-C and WP
- 2024 Book: "Guide to Software Verification with Frama-C: Core Components, Usages, and Applications" `https://link.springer.com/book/10.1007/978-3-031-55608-1`
- WP tutorial: `https://allan-blanchard.fr/publis/frama-c-wp-tutorial-en.pdf`

Frama-C and ACSL — https://www.frama-c.com/

Frama-C: An industrially-used framework for C code analysis and verification

- Delegates "simple" proofs to external tools, mostly **S**atisfiability-**M**odulo-**T**heories solvers (e.g., Z3)
- Practical Program Proof = Verification Condition Generation (VCG) + SMT checking

ACSL: ANSI-**C S**pecification **L**anguage

- Similar to the JML — **J**ava **M**odelling **L**anguage
- But Java is more complex:
  Statements that can raise exceptions need additional postconditions for those.
- ACSL "is" standard first-order predicate logic in C syntax.
- ACSL allows definition of inductive datatypes
  — natural abstractions for specification, but rather clumsy in ACSL

  — From discrete math to C: **A big gap to bridge!**

ACSL Function Contracts

Overall program correctness is based on **function contracts**, mainly:

- "**requires**": Procedure call precondition
- "**assigns**": Global variables that may be updated
  (Much more economical that having pre- and post-conditions $x = x_0$ for each global variable x that must **not** be assigned)
- "**ensures**": Procedure call postcondition
  May refer to **result** for the return value.

Contracts of exported functions are part of the module interface, and therefore should be in the module interface file (`*.h`).

all_zeros.h:

```
/*@ requires  n ≥ 0  ∧  \valid(t + (0.. n−1));
    assigns   \nothing;
    ensures   \result ≠ 0  ⇔  (∀ integer j; 0 ≤ j < n ⇒ t[j] ≡ 0);
*/
int all_zeros(int *t, int n);
```

ACSL Loop Annotations

Support infrastructure for the total-correctness **While** rule:

$$\frac{\{\, B \wedge Q \wedge T = t_0 \,\} \quad C \quad \{\, dom\ `B' \wedge Q \wedge T < t_0 \,\} \qquad\qquad B \wedge Q \Rightarrow T > 0}{\{\, dom\ `B' \wedge Q \,\} \quad \text{while } B \text{ do } C \text{ od} \quad \{\, \neg B \wedge Q \,\}}$$

"**loop invariant** Q": Property always true in the following loop

- true at loop entry, at each loop iteration, at loop exit
- usually contains a generalisation of the post-condition
- may need to contain additional "sanity" conditions

"**loop assigns** *footprint*": What may be assigned to within the loop

"**loop variant** T": To prove termination:

- Integer metric T that is **strictly decreasing** at each iteration and **bounded** by 0

all_zeros

all_zeros.c:

```
/*@ requires n ≥ 0 ∧ \valid(t + (0.. n−1));
    assigns \nothing;
    ensures \result ≠ 0 ⇔ (∀ integer j; 0 ≤ j < n ⇒ t[j] ≡ 0);
*/
int all_zeros(int *t, int n) {
  int k=0;
  /*@ loop invariant 0 ≤ k ≤ n;
      loop invariant ∀ integer j; 0 ≤ j < k ⇒ t[j] ≡ 0;
      loop assigns k;
      loop variant n − k;
  */
  while(k < n){
    if (t[k] ≠ 0)
      return 0;
    k++;
  }
  return 1;
}
```

findMax Attempt 1

findMax1.c:

```
/*@ requires n > 0;
    requires \valid(a + (0 .. n − 1));
    ensures ∀ integer i ; 0 ≤ i < n ⇒ \result ≥ a[i];
    ensures ∃ integer i ; 0 ≤ i < n ⇒ \result ≡ a[i];
*/
int findMax(int n, int a[]) {
  int i;
  /*@ loop invariant ∀ integer j ; 0 ≤ j < i ⇒ a[j] ≡ 0;
      loop invariant 0 ≤ i ≤ n;
      loop variant n − i;
  */
  for( i = 0; i < n; i++) a[i] = 0;
  return 0;
}
```

```
frama-c-gui -wp findMax1.c
frama-c -wp findMax1.c
frama-c-gui -wp -wp-rte findMax1.c
frama-c -wp -wp-rte findMax1.c
```

"RTE": Run-time exceptions (include undefined behaviour)

The *findMax* Attempt 1a

findMax1a.c:

```
/*@ requires n > 0;
    requires \valid(a + (0 .. n − 1));
    ensures ∀ integer i ; 0 ≤ i < n ⇒ \result ≥ a[i];
    ensures ∃ integer i ; 0 ≤ i < n ⇒ \result ≡ a[i];
*/
int findMax(int n, int a[]) {
  int i;
  /*@ loop invariant ∀ integer j ; 0 ≤ j < i ⇒ a[j] ≡ 0;
      loop invariant 0 ≤ i ≤ n;
      loop assigns i, a[0 .. n − 1];
      loop variant n − i;
  */
  for( i = 0; i < n; i++) a[i] = 0;
  return 0;
}
```

Reconsidering the *findMax* Specification

```
/*@ requires n ≥1;
    requires \valid_read(a + (0 .. n − 1));
    ensures ∀ integer i; 0 ≤ i < n ⇒ a[i] ≤ \result;
    ensures ∃ integer i; 0 ≤ i < n ∧ a[i] ≡ \result;
    assigns \nothing;
 */
int findMax(int n, int a []);
```

- "**requires** **valid_read**(a + (0 .. n − 1))" is necessary for array access (pointer dereference)
- "**assigns** **nothing**" documents that *findMax* must not have memory side-effects
- What if we wish to replace "**requires** $n \geq 1$" with "**requires** $n \geq 0$"?

  "**ensures** ∃ **integer** i; $0 \leq i < n \wedge a[i] \equiv$ **result**" would be unsatisfiable for "$n \equiv 0$"!

  A different specification for that case is needed: *findMax* then has two distict **behaviours**, that can be specified separately:

"ACSL by Example": The *max_element* Algorithm — Specification

max_element.h:

```
#include "typedefs.h"
/*@ requires valid:    \valid_read(a + (0..n−1));
    assigns            \nothing;
    ensures result:   0 ≤ \result ≤ n;

    behavior empty:
      assumes          n ≡ 0;
      assigns          \nothing;
      ensures result:  \result ≡ 0;
    behavior not_empty:
      assumes          0 < n;
      assigns          \nothing;
      ensures result:  0 ≤ \result < n;
      ensures upper:   ∀ integer i; 0 ≤ i < n        ⇒ a[i] ≤ a[\result];
      ensures first :  ∀ integer i; 0 ≤ i < \result  ⇒ a[i] < a[\result];

    complete behaviors; disjoint behaviors;
*/
size_type max_element(const value_type* a, size_type n);
```

"ACSL by Example": The *max_element* Algorithm — Implementation

max_element.c:

```
#include "max_element.h"

size_type max_element(const value_type* a, size_type n)
{ if (0u < n) {
    size_type max = 0u;
    /*@ loop invariant bound: 0 ≤ i ≤ n;
        loop invariant max:   0 ≤ max < n;
        loop invariant upper: ∀ integer k; 0 ≤ k < i   ⇒ a[k] ≤ a[max];
        loop invariant first : ∀ integer k; 0 ≤ k < max ⇒ a[k] < a[max];
        loop assigns max, i;
        loop variant n−i;
    */
    for (size_type i = 1u; i < n; i++) {
      if (a[max] < a[i]) { max = i; }
    }
    return max;
  }
  return n;
}
```

"ACSL By Example" — Conventions

`SizeValueTypes.h`:

```
#ifndef SIZEVALUETYPES

typedef int value_type;
typedef unsigned int size_type;
typedef int bool;
#define false 0
#define true 1

#define SIZEVALUETYPES
#endif
```

`IsValidRange.h`:

```
#ifndef ISVALIDRANGE

#include "SizeValueTypes.h"
/*@ predicate IsValidRange(value_type* a, integer n)
      = (0 ≤ n) ∧ \valid(a+(0.. n−1));
*/
```

BISLs — See Also…

"BISL": "Behavioural Interface Specification Language"

- **ACSL** — supported by Frama-C
- **JML:** The Java Modeling Language `https://www.cs.ucf.edu/~leavens/JML/`
  KeY: "The core feature of KeY is a theorem prover for Java Dynamic Logic based on a sequent calculus." `https://www.key-project.org/`
- **SPARK 2014** — version of Ada with verification support
  `http://www.adacore.com/about-spark`
- **Dafny:** "designed as a verification-aware programming language, requiring verification along with code development. [...] The general proof framework is that of Hoare logic." `https://dafny.org/`

- **Eiffel:** First programming language supporting "Design by Contract" (1986)
- **LiquidHaskell:** "refines Haskell's types with logical predicates that let you enforce important properties at compile time."
  `http://ucsd-progsys.github.io/liquidhaskell/`
- **Deal:** "A Python library for design by contract"
  `https://deal.readthedocs.io/basic/verification.html`

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-12-03

Part 1: Loop Variants (demonstrated in ACSL)

ACSL Loop Annotations

Recall the total correctness **While** rule:

$$\frac{\{\, B \wedge Q \,\}\ C\ \{\, dom\ \text{‘}B\text{’} \wedge Q \,\} \quad \{\, B \wedge Q \wedge T = t_0 \,\}\ C\ \{\, T < t_0 \,\} \quad B \wedge Q \Rightarrow T \geq 0}{\{\, dom\ \text{‘}B\text{’} \wedge Q \,\}\ \ \text{while } B \text{ do } C \text{ od}\ \ \{\, \neg B \wedge Q \,\}}$$

"**loop invariant** Q": Property "always" true in the following loop:

- true at loop entry, at each loop iteration, at loop exit
- usually contains a generalisation of the post-condition
- may need to contain additional "sanity" conditions

"**loop assigns** *footprint*": What may be assigned to within the loop

"**loop variant** T": To prove termination:

- Integer metric T that is **strictly decreasing** at each iteration and **bounded** by 0
- Conceptually, this establishes a well-founded relation on the states encountered at start and end of loop body executions.
  $s_1 \gneq s_2 \quad \equiv \quad [\![T]\!]\, s_1 > [\![T]\!]\, s_2$ — (using $[\![\_]\!]$ also for expression semantics evalV)
- **Any** expression T for which the premises can be proven is acceptable.
- Some expressions T may make these proofs easier than others…

Loop Variants 1

$$\frac{\{\, B \wedge Q \,\}\ C\ \{\, dom\ \text{‘}B\text{’} \wedge Q \,\} \quad \{\, B \wedge Q \wedge T = t_0 \,\}\ C\ \{\, T < t_0 \,\} \quad B \wedge Q \Rightarrow T \geq 0}{\{\, dom\ \text{‘}B\text{’} \wedge Q \,\}\ \ \text{while } B \text{ do } C \text{ od}\ \ \{\, \neg B \wedge Q \,\}}$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;   // `T`
   */
  while (i > 0)
  {
    i--;
  }
}
```

- T needs to be some upper bound for the "number of iterations still remaining"

Loop Variants 2

$$\frac{\{\, B \wedge Q \,\}\ C\ \{\, dom\ \text{‘}B\text{’} \wedge Q \,\} \quad \{\, B \wedge Q \wedge T = t_0 \,\}\ C\ \{\, T < t_0 \,\} \quad B \wedge Q \Rightarrow T \geq 0}{\{\, dom\ \text{‘}B\text{’} \wedge Q \,\}\ \ \text{while } B \text{ do } C \text{ od}\ \ \{\, \neg B \wedge Q \,\}}$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;   // `T`
   */
  while (i ≥ 0)
  {
    i--;
  }
}
```

ACSL only requires $B \wedge Q \Rightarrow T \geq 0$

ACSL def., section "Loop Variants":

"its value at the beginning of the iteration must be nonnegative."

Loop Variants 3

$$\frac{\{\, B \wedge Q \,\}\ C\ \{\, dom\ \text{`}B\text{'} \wedge Q \,\} \qquad \{\, B \wedge Q \wedge T = t_0 \,\}\ C\ \{\, T < t_0 \,\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{\, dom\ \text{`}B\text{'} \wedge Q \,\}\quad \text{while } B \text{ do } C \text{ od} \quad \{\, \neg B \wedge Q \,\}}$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;  // `T`
   */
  while (i ≥ −1)
  {
    i−−;
  }
}
```

[wp] [Alt–Ergo] Goal typed_f_loop_variant_positive : Timeout (Qed:1ms) (10s)

- We need $B \wedge Q \Rightarrow T \geq 0$!

Loop Variants 4

$$\frac{\{\, B \wedge Q \,\}\ C\ \{\, dom\ \text{`}B\text{'} \wedge Q \,\} \qquad \{\, B \wedge Q \wedge T = t_0 \,\}\ C\ \{\, T < t_0 \,\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{\, dom\ \text{`}B\text{'} \wedge Q \,\}\quad \text{while } B \text{ do } C \text{ od} \quad \{\, \neg B \wedge Q \,\}}$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;  // `T`   */
  while (i > 0) {
    if (i % 2 ≡ 0) { i−−; }
    else           { i = i − 3; }
  }
}
```

- T needs to be **some** upper bound for the "number of iterations still remaining"
- T does not need to be a tight upper bound!
- Simpler variants may have "faster proofs"

Loop Variants 5

$$\frac{\{\, B \wedge Q \,\}\ C\ \{\, dom\ \text{`}B\text{'} \wedge Q \,\} \qquad \{\, B \wedge Q \wedge T = t_0 \,\}\ C\ \{\, T < t_0 \,\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{\, dom\ \text{`}B\text{'} \wedge Q \,\}\quad \text{while } B \text{ do } C \text{ od} \quad \{\, \neg B \wedge Q \,\}}$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i / 2;  // `T`   */
  while (i > 0) {
    if (i % 2 ≡ 0) { i−−; }
    else           { i = i − 3; }
  }
}
```

- T needs to be **some** upper bound for the "number of iterations still remaining"
- T does not need to be a tight upper bound!
- More complex variants may have "slower proofs", or time-outs. . .

Loop Variants 6

$$\frac{\{\,B \wedge Q\,\}\; C\;\{\,dom\,\text{'}B\text{'} \wedge Q\,\} \qquad \{\,B \wedge Q \wedge T = t_0\,\}\; C\;\{\,T < t_0\,\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{\,dom\,\text{'}B\text{'} \wedge Q\,\} \quad \textsf{while}\ B\ \textsf{do}\ C\ \textsf{od} \quad \{\,\neg B \wedge Q\,\}}$$

```
#define N 1000
//@ assigns \nothing;
void f () {
  int i = 0;
  /*@ loop assigns i;
      loop variant N − i;   // `T`
   */
  while (i ≤ N)
  {
    i++;
  }
}
```

- T needs to be **decreasing**, even if your counters are increasing!

Loop Variants 7

$$\frac{\{\,B \wedge Q\,\}\; C\;\{\,dom\,\text{'}B\text{'} \wedge Q\,\} \qquad \{\,B \wedge Q \wedge T = t_0\,\}\; C\;\{\,T < t_0\,\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{\,dom\,\text{'}B\text{'} \wedge Q\,\} \quad \textsf{while}\ B\ \textsf{do}\ C\ \textsf{od} \quad \{\,\neg B \wedge Q\,\}}$$

```
//@ assigns \nothing;
void f () {
    int i = 100, k = 200;
  /*@ loop assigns i, k;
      loop variant i + k;   // `T`
   */
  while (i ≥ 0 ∧ k ≥ 0)
  {
      if( (i + k) % 2 ≡ 0 ) { i−−; }
      else                  { k−−; }
  }
}
```

- If your loop is not a "plain `for`-loop", several variables may be involved in the variant.

Loop Variants 8

$$\frac{\{\,B \wedge Q\,\}\; C\;\{\,dom\,\text{'}B\text{'} \wedge Q\,\} \qquad \{\,B \wedge Q \wedge T = t_0\,\}\; C\;\{\,T < t_0\,\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{\,dom\,\text{'}B\text{'} \wedge Q\,\} \quad \textsf{while}\ B\ \textsf{do}\ C\ \textsf{od} \quad \{\,\neg B \wedge Q\,\}}$$

```
//@ assigns \nothing;
void f () {
    int i = 0, k = 10;
  /*@ loop assigns    i, k;
      loop invariant 0 ≤ i ≤ k + 1 ∧ 0 ≤ k;
      loop variant   k ∗ (k + 1) + i;           // `T`
   */
  while (k > 0)
  {
      if( i > 0 ) { i−−; }
      else        { i = k; k−−; }
}}
```

- Invariants may be needed to contribute to provability of the variant.
- Finding appropriate variants can be tricky. . .

Loop Variants 9

$$\frac{\{\, B \wedge Q \,\}\ C\ \{\, dom\ `B' \wedge Q \,\} \qquad \{\, B \wedge Q \wedge T = t_0 \,\}\ C\ \{\, T < t_0 \,\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{\, dom\ `B' \wedge Q \,\} \quad \text{while } B \text{ do } C \text{ od} \quad \{\, \neg B \wedge Q \,\}}$$

```
//@ assigns \nothing;
void f () {
    int i = 0, k = 10;
  /*@ loop assigns     i, k;
      loop invariant  0 ≤ i ≤ (k + 1) ∗ (k + 1) ∧ 0 ≤ k;
      loop variant    k ∗ k ∗ (k + 1) + i;   // `T`
   */
  while (k > 0)
  {
      if( i > 0 ) { i−−; }
      else        { i = k ∗ k; k−−; }
  }
}
```

- ...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-12-03

Part 2: Graphs, Subgraphs, Lattices

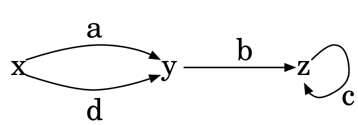
Graphs

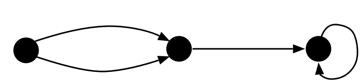
Definition: A **graph** is a tuple $\langle V, E, \mathsf{src}, \mathsf{trg} \rangle$ consisting of

- a set V of ***vertices*** or ***nodes***
- a set E of ***edges*** or ***arrows***
- a mapping $\mathsf{src} : E \twoheadrightarrow V$ that assigns each edge its ***source*** node
- a mapping $\mathsf{trg} : E \twoheadrightarrow V$ that assigns each edge its ***target*** node

Example graph:

$$\langle\ \{x,y,z\},\ \{a,b,c,d\},\ \{\langle a,x\rangle, \langle b,y\rangle, \langle c,z\rangle, \langle d,x\rangle\},\ \{\langle a,y\rangle, \langle b,z\rangle, \langle c,z\rangle, \langle d,y\rangle\}\ \rangle$$





Well-definedness condition expanded: $\quad Dom\ \mathsf{src} = E \wedge Ran\ src \subseteq V \wedge \text{univalent src}$
$\wedge\ Dom\ \mathsf{trg} = E \wedge Ran\ trg \subseteq V \wedge \text{univalent trg}$

A graph implementation may guarantee univalence via choice of data structure (e.g., array), leaving the other conditions still to be verified.

Subgraphs, Induced Subgraphs

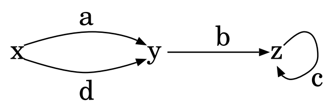
Definition: Let two graphs $G_1 = \langle V_1, E_1, \mathsf{src}_1, \mathsf{trg}_1 \rangle$ and $G_2 = \langle V_2, E_2, \mathsf{src}_2, \mathsf{trg}_2 \rangle$ be given.

- G_1 is called a ***subgraph*** of G_2 iff $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$ and $\mathsf{src}_1 \subseteq \mathsf{src}_2$ and $\mathsf{trg}_1 \subseteq \mathsf{trg}_2$.

Def. and Theorem: The subgraph relation is an order on graphs.

Def. and Theorem: Given a subset $V_0 \subseteq V$ of the vertex set of graph $G = \langle V, E, \mathsf{src}, \mathsf{trg} \rangle$, define E_0 and G_0 by:

- $E_0 = E \cap Dom\,(\mathsf{src} \rhd V_0) \cap Dom\,(\mathsf{trg} \rhd V_0)$
  $= E \cap \mathsf{src}^{\smile}\,(\!| V_0 |\!) \cap \mathsf{trg}^{\smile}\,(\!| V_0 |\!)$, the edges incident with **only** nodes in V_0
- $G_0 = \langle V_0,\ E_0,\ E_0 \lhd \mathsf{src},\ E_0 \lhd \mathsf{trg} \rangle$



but $a \in \mathsf{trg}^{\smile}\,(\!| \{y, z\} |\!)$, $a \notin \mathsf{src}^{\smile}\,(\!| \{y, z\} |\!)$

Then:
- G_0 is a <u>well-defined</u> graph.
- G_0 is a subgraph of G (by cobnstruction).
- G_0 is the largest subgraph of G with node set V_0.
- G_0 is called the ***subgraph of*** G ***induced by*** V_0.

Facts about Subgraphs

Definition: Let two graphs $G_1 = \langle V_1, E_1, \mathsf{src}_1, \mathsf{trg}_1 \rangle$ and $G_2 = \langle V_2, E_2, \mathsf{src}_2, \mathsf{trg}_2 \rangle$ be given.

- G_1 is called a ***subgraph*** of G_2 iff $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$ and $\mathsf{src}_1 \subseteq \mathsf{src}_2$ and $\mathsf{trg}_1 \subseteq \mathsf{trg}_2$.
- We write $\mathsf{Subgraph}_G$ for the set of all subgraphs of G.
- For a given graph G, we write $G_1 \sqsubseteq_G G_2$ if both G_1 and G_2 are subgraphs of G, and G_1 is a subgraph of G_2.

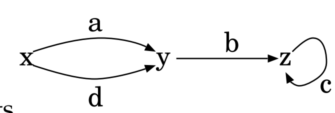
Theorem: $\sqsubseteq_G$ is an ordering on $\mathsf{Subgraph}_G$.

Theorem: $\sqsubseteq_G$ has greatest element $\top = G$ and least element $\bot = \langle \{\}, \{\}, \{\}, \{\} \rangle$.

Theorem: $\sqsubseteq_G$ has binary meets defined by intersection.

Theorem: $\sqsubseteq_G$ has binary joins defined by union.

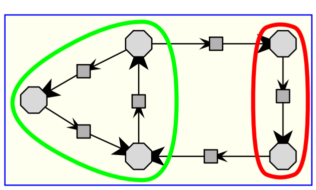
Theorem: $\sqsubseteq_G$ has pseudo-complements, but not complements.

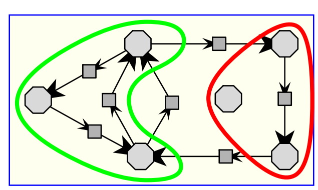


The subgraph induced by $\{y, z\}$ has the subgraph induced by $\{x\}$ as pseudo-complement, but their union is not the whole graph.

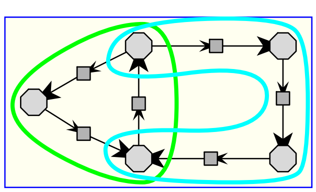
Pseudo- and Semi-Complements of a Subgraph

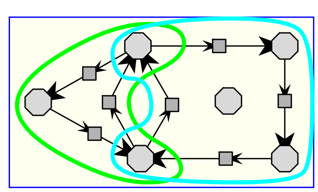
Pseudo-complement of S: The largest X such that $X \cap S = \bot$:





Semi-complement of S: The smallest X such that $X \cup S = \top$:





Joins and Meets

- Given an order $\sqsubseteq$, z is an "upper bound" of two elements x and y iff $x \sqsubseteq z \wedge y \sqsubseteq z$
- Given an order $\sqsubseteq$, the two elements x and y have j as "join" or "least upper bound" (lub), iff $\forall z \bullet j \sqsubseteq z \equiv x \sqsubseteq z \wedge y \sqsubseteq z$
- The order $\sqsubseteq$ "has binary joins" if for any two elements, there is a join — see "Characterisation of $\cup$" for the inclusion order $\subseteq$
- Given an order $\sqsubseteq$, the set S of elements has j as "join" or "least upper bound" (lub), iff $\forall z \bullet j \sqsubseteq z \equiv (\forall x \mid x \in S \bullet x \sqsubseteq z)$
- The order $\sqsubseteq$ "has arbitrary joins" if for any set of elements, there is a join — see "Characterisation of $\bigcup$"
- Given an order $\sqsubseteq$, the set S of elements has m as "meet" or "greatest lower bound" (glb), iff $\forall z \bullet z \sqsubseteq m \equiv (\forall x \mid x \in S \bullet z \sqsubseteq x)$
- The order $\sqsubseteq$ "has binary meets" if for any two-element set, there is a meet — see "Characterisation of $\cap$"
- The order $\sqsubseteq$ "has arbitrary meets" if for any set of elements, there is a meet.

Lattices

Definition: A **lattice** is a partial order with binary meets and joins.

Examples:

- For every graph G, its subgraphs, that is, $\langle \mathsf{Subgraph}_G, \sqsubseteq_G \rangle$ with $\sqcap_G$ and $\sqcup_G$
- $\langle \mathbb{Z}, \leq \rangle$ with $\downarrow$ and $\uparrow$
- $\langle \mathbb{Z}, \geq \rangle$ with $\uparrow$ and $\downarrow$
- $\langle \mathbb{N}, \leq \rangle$ with $\downarrow$ and $\uparrow$
- $\langle \mathbb{N}, | \rangle$ with gcd and lcm
- $\langle \wp A, \subseteq \rangle$ with $\cap$ and $\cup$
- Equivalence relations on A ordered wrt. $\subseteq$, with $\cap$ and $(E_1 \cup E_2)^*$

Algebraic Definition: A **lattice** $\langle A, \sqcap, \sqcup \rangle$ consists of a set A with two binary operations $\sqcap$, $\sqcup$ on A such that:

- $\sqcap$ and $\sqcup$ each are idempotent, symmetric, and associative
- The absorption laws hold: $x \sqcup (x \sqcap y) = x = x \sqcap (x \sqcup y)$

A **Boolean lattice** $\langle A, \sqcap, \sqcup, \bot, \top, \sim \rangle$ in addition has least and greatest elements $\bot$ and $\top$, and a unary **complement** operation $\sim$ satisfying $\sim x \sqcap x = \bot$ and $\sim x \sqcup x = \top$.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2024-12-05

Part 1: Graph Homomorphisms, Categories

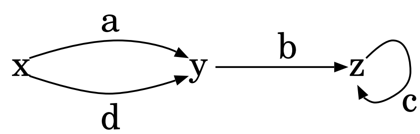
Recall: Graphs

Definition: A **graph** is a tuple $\langle V, E, \mathsf{src}, \mathsf{trg}\rangle$ consisting of

- a set V of ***vertices*** or ***nodes***
- a set E of ***edges*** or ***arrows***
- a mapping $\mathsf{src} : E \twoheadrightarrow V$ that assigns each edge its ***source*** node
- a mapping $\mathsf{trg} : E \twoheadrightarrow V$ that assigns each edge its ***target*** node

Example graph:

$$\langle\ \{x,y,z\},\ \{a,b,c,d\},\ \{\langle a,x\rangle,\langle b,y\rangle,\langle c,z\rangle,\langle d,x\rangle\},\ \{\langle a,y\rangle,\langle b,z\rangle,\langle c,z\rangle,\langle d,y\rangle\}\ \rangle$$



Graphs as Structures over Signature sigGraph

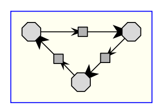
A **signature** is a tuple $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{R})$ consisting of

- a set $\mathcal{S}$ of **sorts**
- a set $\mathcal{F}$ of **function symbols** $f : s_1 \times \cdots \times s_n \to t$
- a set $\mathcal{R}$ of **relation symbols** $r : s_1 \times \cdots \times s_n \leftrightarrow t$

A Σ**-structure** $\mathcal{A}$ consists of:

- for every sort $s : \mathcal{S}$, a **carrier** $s^{\mathcal{A}}$, and
- for every function symbol $f : s_1 \times \cdots \times s_n \to t$ a **mapping** $f^{\mathcal{A}} : s_1^{\mathcal{A}} \times \cdots \times s_n^{\mathcal{A}} \to t^{\mathcal{A}}$.
- for every relation symbol $r : s_1 \times \cdots \times s_n \leftrightarrow t$ a **relation** $r^{\mathcal{A}} : s_1^{\mathcal{A}} \times \cdots \times s_n^{\mathcal{A}} \leftrightarrow t^{\mathcal{A}}$.

$\mathcal{G} :\equiv \langle$ **sorts:** $\mathcal{V}, \mathcal{E}$
ops: $src, trg : \mathcal{E} \to \mathcal{V}$
$\rangle$



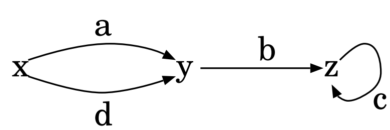
The signature graph of sigGraph: $\mathcal{E} \overset{src}{\underset{trg}{\rightrightarrows}} \mathcal{V}$

Signatures, as mathematical objects, are of a similar kind as graphs!

Different Kinds of Graphs as Structures for Different Signatures

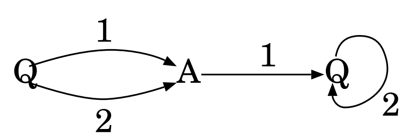
Graphs:

$\mathcal{G} :\equiv \langle$ **sorts:** $\mathcal{N}, \mathcal{E}$
ops: $src, trg : \mathcal{E} \to \mathcal{N}$
$\rangle$



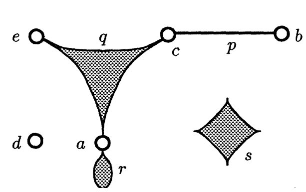
Node-labelled edge-weighted graphs:

$\mathcal{LWG} :\equiv \langle$ **sorts:** $\mathcal{N}, \mathcal{E}$
ops: $src, trg : \mathcal{E} \to \mathcal{N}$
$nLabel : \mathcal{N} \to \mathcal{L}$
$eWeight : \mathcal{E} \to \mathbb{N}$
$\rangle$



Undirected hypergraphs:

$\mathcal{HG} :\equiv \langle$ **sorts:** $\mathcal{N}, \mathcal{E}$
ops: $incident : \mathcal{E} \leftrightarrow \mathcal{N}$
$\rangle$



	a	b	c	d	e
p	0	1	1	0	0
q	1	0	1	0	1
r	1	0	0	0	0
s	0	0	0	0	0

M

Graph Homomorphisms

Definition: Let two graphs $G_1 = \langle V_1, E_1, \mathsf{src}_1, \mathsf{trg}_1 \rangle$ and $G_2 = \langle V_2, E_2, \mathsf{src}_2, \mathsf{trg}_2 \rangle$ be given.
A pair $\Phi = \langle \Phi_V, \Phi_E \rangle$ is called a **graph homomorphism from** G_1 **to** G_2 iff

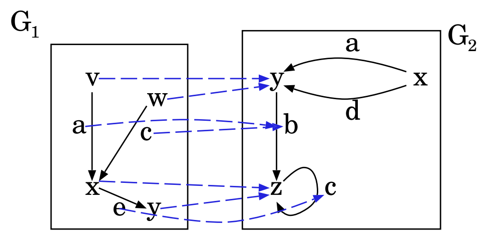
- $\Phi_V \in V_1 \multimap\!\!\rightarrow V_2$ and $\Phi_E \in E_1 \multimap\!\!\rightarrow E_2$
- $\Phi_E \fatsemi \mathsf{src}_2 = \mathsf{src}_1 \fatsemi \Phi_V$ and $\Phi_E \fatsemi \mathsf{trg}_2 = \mathsf{trg}_1 \fatsemi \Phi_V$

Homomorphisms are "structure-preserving mappings".

(Mappings; Total and univalent)

Graph homomorphisms can:

- Identify different structure elements
  — if not injective
- Not cover the target completely
  — if not surjective

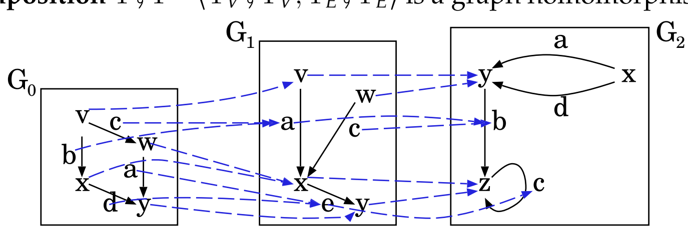


Graph Homomorphisms Compose

Definition: Let two graphs $G_1 = \langle V_1, E_1, \mathsf{src}_1, \mathsf{trg}_1 \rangle$ and $G_2 = \langle V_2, E_2, \mathsf{src}_2, \mathsf{trg}_2 \rangle$ be given.
A pair $\Phi = \langle \Phi_V, \Phi_E \rangle$ is called a **graph homomorphism from** G_1 **to** G_2 iff

- $\Phi_V \in V_1 \multimap\!\!\rightarrow V_2$ and $\Phi_E \in E_1 \multimap\!\!\rightarrow E_2$
- $\Phi_E \fatsemi \mathsf{src}_2 = \mathsf{src}_1 \fatsemi \Phi_V$ and $\Phi_E \fatsemi \mathsf{trg}_2 = \mathsf{trg}_1 \fatsemi \Phi_V$

Definition and theorem: Let three graphs G_0, G_1, and G_2 be given.
Let $\Phi = \langle \Phi_V, \Phi_E \rangle$ be a graph homomorphism from G_0 to G_1 and $\Psi = \langle \Psi_V, \Psi_E \rangle$ be a graph homomorphism from G_1 to G_2.
Then their **composition** $\Phi \fatsemi \Psi = \langle \Phi_V \fatsemi \Psi_V, \Phi_E \fatsemi \Psi_E \rangle$ is a graph homomorphism from G_0 to G_2.



Definition and theorem: The **identity graph homomorphism** $\mathbb{I} = \langle \mathrm{id}\, V, \mathrm{id}\, E \rangle$ is well-defined, and is "the" identity for graph homomorphism composition.

Graph Homomorphisms Compose — and Form a Category

Graph homomorphisms have

- source and target graphs,
- associative composition $\fatsemi$ of consecutive homomorphisms,
- identity homomorphisms $\mathbb{I}$ (satisfying the identity laws).

That is, graphs with graph homomorphisms form a **category**.

In particular:

- Ψ is an inverse of Φ iff $\Phi \fatsemi \Psi = \mathbb{I}$ and $\Psi \fatsemi \Phi = \mathbb{I}$.
- $\Phi = \langle \Phi_V, \Phi_E \rangle$ has an inverse iff it is bijective, that is, iff both Φ_V and Φ_E are bijective.
  The inverse of Φ is then $\langle \Phi_V{}^{\smile}, \Phi_E{}^{\smile} \rangle$.

(Category theory is the source of the words "functor", "monad", "arrow", etc. in the context of Haskell.)

Categories

A **category C** consists of:

- a collection of **objects**
- for every two objects $\mathcal{A}$ and $\mathcal{B}$ a **homset** containing **morphisms** $f : \mathcal{A} \to \mathcal{B}$
- associative **composition** "$\fatsemi$" of morphisms, defined for $\mathcal{A} \xrightarrow{f} \mathcal{B} \xrightarrow{g} \mathcal{C}$, with $(f \fatsemi g) : \mathcal{A} \to \mathcal{C}$
- for every object $\mathcal{A}$ an **identity** morphism $\mathbb{I}_{\mathcal{A}}$ which is both a right and left unit for composition.

In **category C**, morphisnm $g : \mathcal{B} \to \mathcal{A}$ is called **inverse of** $f : \mathcal{A} \to \mathcal{B}$ iff $f \fatsemi g = \mathbb{I}_{\mathcal{A}}$ and $g \fatsemi f = \mathbb{I}_{\mathcal{B}}$.

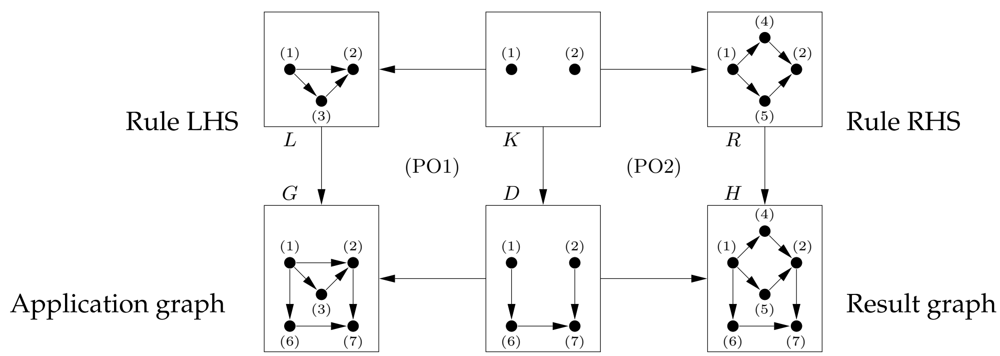
If f has an inverse, then f is called an **isomorphism** or just **iso**.

Example categories:
Sets with mappings
Sets with partial functions
Sets with relations
Graphs with graph homomorphisms
Lattices with lattice homomorphisms
Categories with functors

Categorial Graph Transformation

Graphs with graph homomorphisms form a **category** — category theory is **re-usable theory**!

Using category-theoretical concepts, various **graph transformation** mechanisms are defined; these are used for system modelling and model transformation.



Pushouts — A Typical Categorial "Universal Construction"

Pushouts can be seen as a generalisation of unions/joins:

Recall "Characterisation of $\cup$":

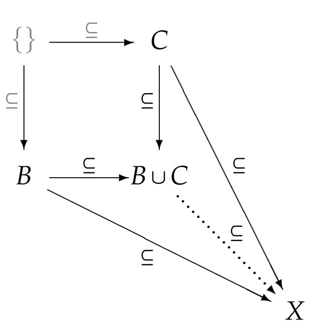
$B \cup C$ is **union** of sets B and C iff

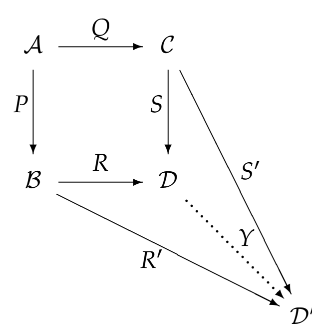
$\forall\, X \bullet\ B \subseteq X \wedge C \subseteq X \ \equiv\ B \cup C \subseteq X$

$\langle \xrightarrow{R} \mathcal{D} \xleftarrow{S} \rangle$ is **pushout** of span "$\mathcal{B} \xleftarrow{P} \mathcal{A} \xrightarrow{Q} \mathcal{C}$" iff

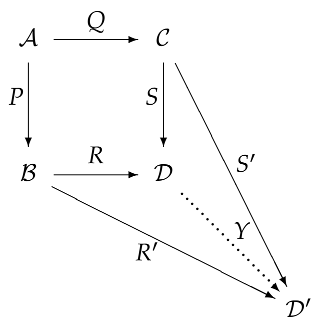
$P \fatsemi R = Q \fatsemi S \ \wedge\ \forall \langle \xrightarrow{R'} \mathcal{D}' \xleftarrow{S'} \rangle \mid P \fatsemi R' = Q \fatsemi S'$

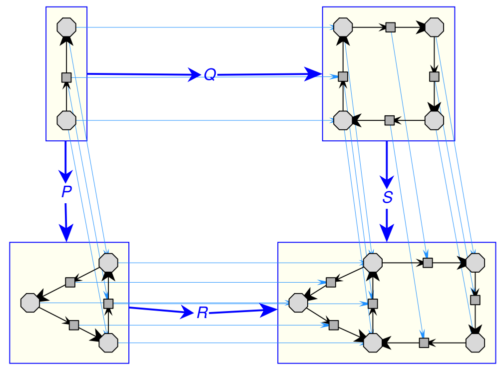
$\bullet\ \exists\, Y : D \to D' \bullet R \fatsemi Y = R' \wedge S \fatsemi Y = S'$





Pushouts of Graph Homomorphisms: "Gluing"





Such a pushout can be understood as:

gluing $\mathcal{B}$ and $\mathcal{C}$ together "along the interface $\xleftarrow{P} \mathcal{A} \xrightarrow{Q}$".

Double-Pushout Rewriting

Rule: $\mathcal{L} \xleftarrow{\Phi_L} \mathcal{G} \xrightarrow{\Phi_R} \mathcal{R}$

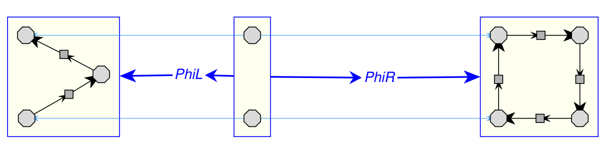
Redex:

$$\begin{array}{ccccc} \mathcal{L} & \xleftarrow{\Phi_L} & \mathcal{G} & \xrightarrow{\Phi_R} & \mathcal{R} \\ {\scriptstyle X_L}\downarrow & & & & \\ \mathcal{A} & & & & \end{array}$$

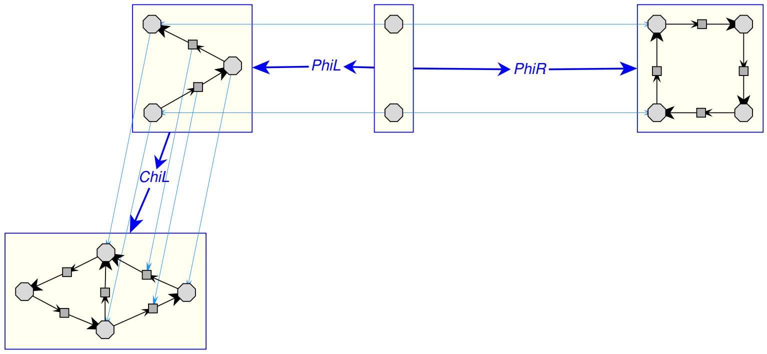
Rewriting step:

$$\begin{array}{ccccc} \mathcal{L} & \xleftarrow{\Phi_L} & \mathcal{G} & \xrightarrow{\Phi_R} & \mathcal{R} \\ {\scriptstyle X_L}\downarrow & \text{PO} & {\scriptstyle \Xi}\downarrow & \text{PO} & \downarrow{\scriptstyle X_R} \\ \mathcal{A} & \xleftarrow{\Psi_L} & \mathcal{H} & \xrightarrow{\Psi_R} & \mathcal{B} \end{array}$$

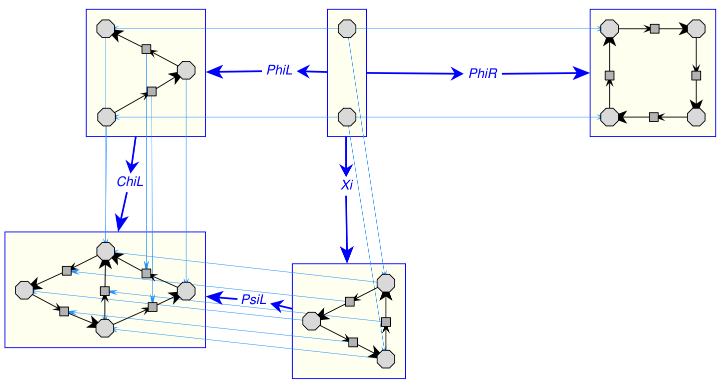
Example Double-Pushout Rewriting Step: Rule



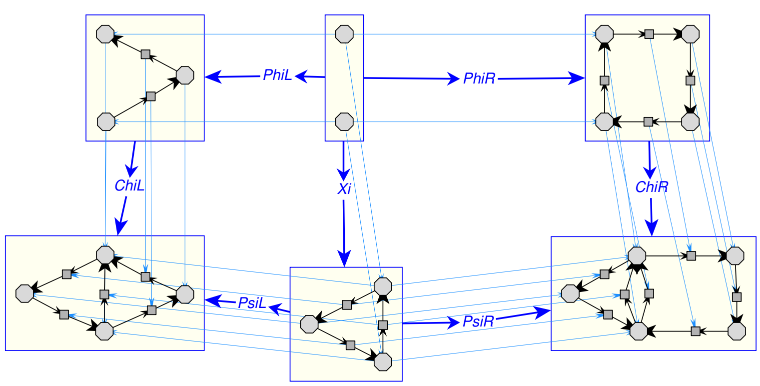
Example Double-Pushout Rewriting Step: Redex



Example Double-Pushout Rewriting Step: Host



Example Double-Pushout Rewriting Step: Result



The Power of Double-Pushout Rewriting

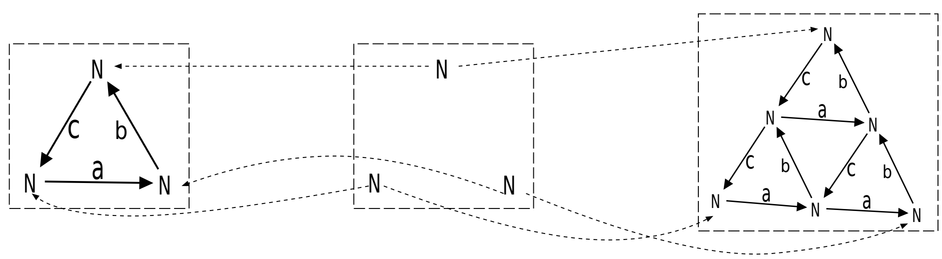
- **easy to understand**
- **easy to implement**
- can { delete / identify / add } **precisely specified** items
- **cannot** duplicate or delete **loosely specified** items
  — no **“subgraph variables”**

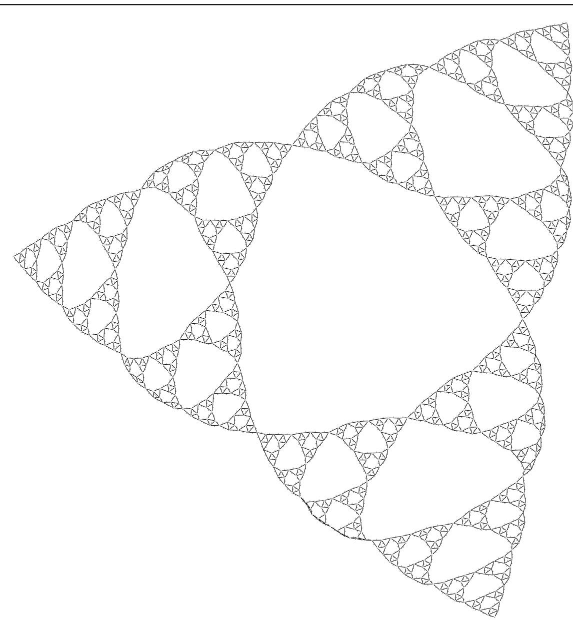
DPO graph rewriting is the most widely used graph transformation formalism.

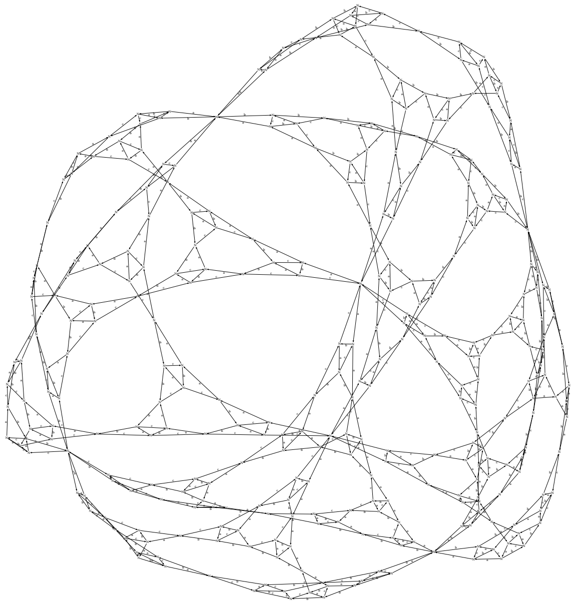
- Describing evolution/execution of systems modelled as graphs
- Defining model transformations (e.g., of UML diagrams) for system development

Categorial approaches are more likely to interact usefully with graph semantics than “node-label-controlled” etc. low-level graph transformation approaches.

DPO Rule for Generating Sierpinski Triangles







The Power of Gluing

- Gluing via pushouts (or more general colimits) works in many intersting categories
- A **component specification** consists of a signature and axioms
- Such component specifications form a category; specification homomorphism can structure complex specifications:

```
spec BinRel is
   sort E
   op _le_ : E, E → Boolean
   end-spec
```

$\longrightarrow$

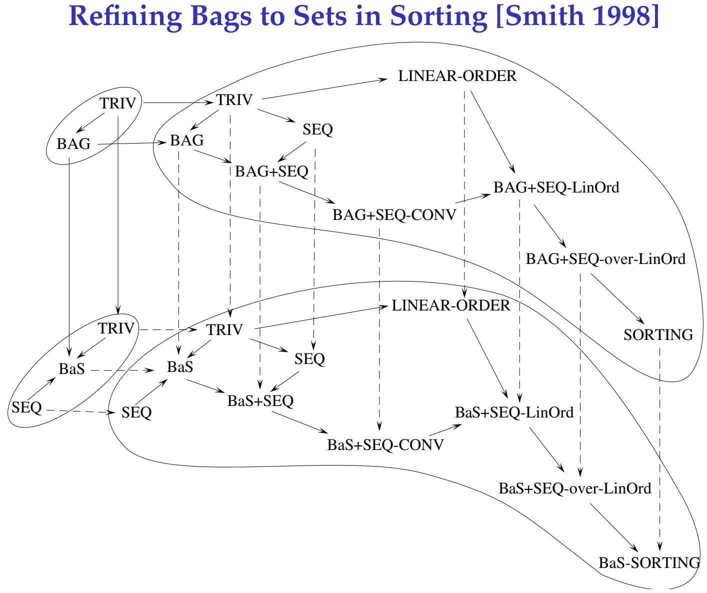
```
spec PreOrder is
   import BinRel
   axiom reflexivity is x le x
   axiom transitivity is
      x le y ∧ y le z ⟹ x le z
   end-spec
```

$\downarrow$

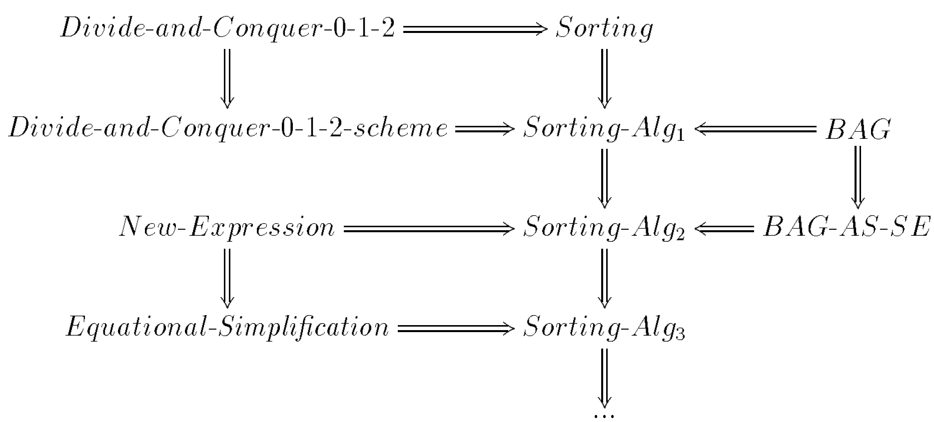
```
spec Antisymmetry is
   import BinRel
   axiom antisymmetry is
      x le y ∧ y le x ⟹ x = z
   end-spec
```

- Specification homomorphism can also be used for **refinement** —
  this method is used for **correct-by-construction software development**

Refining Bags to Sets in Sorting [Smith 1998]



… as One Step in Correct-By-Construction Algorithm Development [Smith 1998]



https://link.springer.com/chapter/10.1007/978-3-540-40007-3_17

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2024

Wolfram Kahl

2023-12-06

Part 2: Conclusion

Organisation

Extra TA office hours:

- Sunday, Dec. 8th, 1:00 to 4:00 p.m. — room TBA
- Monday, Dec. 9th, 1:00 to 4:00 p.m. — room TBA

The **final exam** covers the whole course. Expect questions that combine several topics.

- COMPSCI 2LC3 on Avenue and CalcCheck$_{\text{Web}}$ remains active throughout term 2.
- Collected lecture slides will be posted under “General”.
- Please fill in the course experience surveys for **all your courses!**

⟶ mcmaster.bluera.com/mcmaster

Proofs — (Simplified) Inference Rules —- See LADM p. 133, "Using Z" ch. 2&3

"Natural Deduction" — A Presentation of Logic for Mathematical Study of Logic

$$\frac{P \wedge Q}{P}\ \wedge\text{-Elim}_1 \qquad \frac{P \wedge Q}{Q}\ \wedge\text{-Elim}_2 \qquad \frac{\forall\, x \bullet P}{P[x := E]}\ \text{Instantiation } (\forall\text{-Elim})$$

$$\frac{P}{P \vee Q}\ \vee\text{-Intro}_1 \qquad \frac{Q}{P \vee Q}\ \vee\text{-Intro}_2 \qquad \frac{P[x := E]}{\exists\, x \bullet P}\ \exists\text{-Intro}$$

$$\frac{P \Rightarrow Q \qquad P}{Q}\ \Rightarrow\text{-Elim} \qquad \frac{P \qquad Q}{P \wedge Q}\ \wedge\text{-Intro} \qquad \frac{P}{\forall\, x \bullet P}\ \forall\text{-Intro (prov. } x \text{ not free in assumptions)}$$

$$\frac{\begin{array}{c}\ulcorner P \urcorner\\ \vdots\\ Q\end{array}}{P \Rightarrow Q}\ \Rightarrow\text{-Intro} \qquad \frac{P \vee Q \qquad \begin{array}{c}\ulcorner P \urcorner\\ \vdots\\ R\end{array} \qquad \begin{array}{c}\ulcorner Q \urcorner\\ \vdots\\ R\end{array}}{R}\ \vee\text{-Elim} \qquad \frac{(\exists x \bullet P) \qquad \begin{array}{c}\ulcorner P \urcorner\\ \vdots\\ R\end{array}}{R}\ \exists\text{-Elim (prov. } x \text{ not free in } R\text{, assumptions)}$$

About Natural Deduction

Example proof (using the inference rules as shown in Using Z):

$$\dfrac{\dfrac{\dfrac{\ulcorner \exists x : a \bullet p \Rightarrow q \urcorner^{[1]} \qquad \dfrac{\ulcorner x \in a \urcorner^{[3]} \qquad \dfrac{\ulcorner p \Rightarrow q \urcorner^{[3]} \qquad \dfrac{\ulcorner \forall x : a \bullet p \urcorner^{[2]} \qquad \ulcorner x \in a \urcorner^{[3]}}{p}\ \forall\text{-elim}}{q}\ \Rightarrow\text{-elim}}{\exists x : a \bullet q}\ \exists\text{-intro}}{\exists x : a \bullet q}\ \exists\text{-elim}^{[3]}}{(\forall x : a \bullet p) \Rightarrow (\exists x : a \bullet q)}\ \Rightarrow\text{-intro}^{[2]}}{(\exists x : a \bullet p \Rightarrow q) \quad \Rightarrow \quad ((\forall x : a \bullet p) \Rightarrow (\exists x : a \bullet q))}\ \Rightarrow\text{-intro}^{[1]}$$

- Each formula construction C has:
  - **Introduction rule(s):** How to prove a C-formula?
  - **Elimination rule(s):** How to use a C-formula to prove something else?
- Tactical theorem provers (Coq, Isabelle) provide methods to (virtually) construct such trees piecewise from all directions
- Several of the Natural Deduction inference rules correspond
  - to LADM Metatheorems or proof methods,
  - to CalcCheck proof structures.

Writing Proofs

- Natural deduction was designed as a variant of **sequent calculus** that closely corresponds to the "natural" way of reasoning used in traditional mathematics.
- As such, natural deduction rules constitute building blocks of proof strategies.
- Natural deduction inference trees are **not normally used for proof presentation.**
- CalcCheck structured proofs are **readable formalisations** of conventional informal proof presentation patterns.
- If you wish to write prose proofs, you still need to get the right proof structure first — **think CalcCheck!**
- For proofs, **informality as such is not a value.**
  Rigorous (informal) proofs (e.g. in LADM)
  strive to "make the eventual formalisation effort minimal".
- There is value to **readable proofs**, no matter whether formal or informal.
- There is value to **formal, machine-checkable proofs**,
  especially in the software context,
  where the world of mathematics is not watching.

Strive for readable formal proofs!

Proofs for Software

- **Partial correctness**: Verifying essential functionality
- **Total correctness**: Verifying also termination
- Absence of **r**un-**t**ime **e**rrors imposes additional preconditions on commands
- Termination is typically dealt with separately; it requires a **well-founded** "termination order".

These are supported by tools like Frama-C, VeriFast, Key, ...:

- Hoare calculus inference rules are turned into **Verification Condition Generation**
- Many simple verification conditions can be proved using SMT solvers (Satisfiability Modulo Theories) — Z3, veriT, ...
- More complex properties may need human assitance:
  Proof assistants: Isabelle, Coq, PVS, Agda, ...
- Pointer structures require an extension of Hoare logic: **Separation Logic**

Industry has more and more **formal methods jobs**!

- Legacy C/C++ code needs to be analysed for issues
- Legacy C/C++ code bases are still growing...

Mathematical Programming Languages

- **Software is a mathematical artefact**
- **Functional programming languages** and **logic programming languages** aim to make expression in mathematical manner easier
- Among reasonably-widespread programming languages.
  Haskell is "the most mathematical"
- **Dependently-typed logics** (e.g., Coq, Lean, PVS, Agda) make it possible to express mathematics in a more natural way than in first-order predicate logic:
  - For a matrix $M : \mathbb{R}^{3\times 4}$, the element access $M_{5,6}$ raises a **type error**
  - A simple graph (V, E) can consist of a **type** V and a relation $E : V \leftrightarrow V$.
- **Dependently-typed programming languages** (e.g., Agda, Idris)
  - contain dependently-typed logics — "proofs are programs, too"
  - make it possible to express functional specifications via the type system — "formulae as types": **Curry-Howard correspondence**
  - A program that has not been proven correct wrt. the stated specification does not even compile.

Continued Use of Logical Reasoning

- **COMPSCI 2AC3 Automata and Computability**
  — formal languages, grammars, finite automata, transition relations, Kleene algebra! acceptance predicates, ...
- **COMPSCI 2SD3 Concurrent Systems Design**
  —**correctness of concurrent programs, may use temporal logic**
- **COMPSCI 2DB3 Databases**
  — n-ary relations, relation**al** algebra; functional dependencies
- **COMPSCI 3MI3 Principles of Programming Languages**
  — Programming paradigms, including functional programming; mathematical understanding of prog. language constructs, semantics
- **COMPSCI&SFWRENG 3RA3 Software Requirements**
  — Capturing **precisely** what the customer wants, formalisation
- **COMPSCI 3EA3 Software and System Correctness**
  — Formal specifications, validation, verification
- **COMPSCI 4FP3 Advanced Functional Programming**

Concluding Remarks

- How do I find proofs? — There is no general recipe
- Proving is somewhat like doing puzzles — **practice helps**
- **Proofs** are especially **important for software** — and much care is needed!
- Be aware of **types**, both in programming, and in mathematics
- Be aware of **variable binding** — in quantification, local variables, formal parameters
- Strive to use **abstraction** to **avoid variable binding**
  — e.g., using relation algebra instead of predicate logic
- When designing **data representations**, **think mathematics**: **Subsets, relations, functions, injectivity**, . . .
- **Thinking mathematics in programming**
  is easiest in functional languages, e.g., **Haskell**, OCaml
- **Specify formally!** — **Design for provability!**
- **When doing software, think logics and discrete mathematics!**