Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-07

What is This Course About? *What Not?*

- Calendar description:

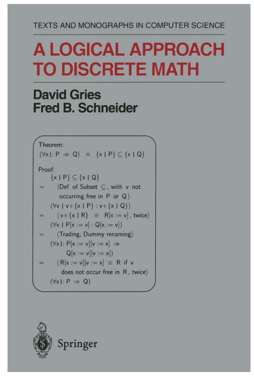
  > Introduction to logic and proof techniques for practical reasoning: propositional logic, predicate logic, structural induction; rigorous proofs in discrete mathematics and programming.

- *Calculus is the mathematics of **continuous** phenomena: physical sciences, traditional engineering — used for specifying bridges; used for justifying bridge designs.*
- **Discrete Mathematics** is
  - the math of data— **whether complex or big**
  - the math of reasoning— **logic**
  - the math of AI— **machine reasoning**
  - **used for specifying software**
- **Logical Reasoning** is
  - **used for justifying software designs**
  - **used for proving software implementations correct**
- *Advanced topic combining both: Cyber-physical systems (CPS)*

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
- *. . . skill development takes time and effort . . .*
- Introduction to **reasoning about (imperative) programs**
- Encounter 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)

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!**

  — 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) $\quad S = \{x \mid x \in S : x\}$.
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,

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

```
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
  - Tutorials have 20–40 students and use this approach, with students working on their computers

    — this still works 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 twice 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/CS2LC3/2021/`

  — check in case of Avenue and MSTeams outage!

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

— Read the Outline!

Rough Timeline

• Introduction to Calculational Reasoning	Parts of Chapters 1, 15	
Boolean Expressions and Propositional Logic	Chapters 1–5	≈ 4 weeks
Quantification, Predicate Logic, Sets	Chapters 8–9, 11	
• Induction, Sequences, Trees	Chapters 12–13	≈ 2 weeks
• Relations and Functions, Graphs	Chapters 14, 19	≈ 3 weeks
• Correctness of Imperative Programs	Chapter 10, other	≈ 3 weeks

Schedule

	Mon	Tue	Wed	Thu	Fri
9:30–				T4	
–11:20				T4	
12:30–13:20	Lecture			Lecture	T2, T3, “T5”
13:30–14:20		Lecture			T2, T3, “T5”
14:30–	T1				
–16:30	T1				

- **Lectures:** On MSTeams, recorded at source (not in MSTeams) — **attend!**, **take notes!**
- **Office hour:** For now, on MSTeams by appointment
- **2-hour Tutorials** (starting **Thursday, September 9**):
  - – Discuss student approaches to “Exercise” questions.
  - – “T5” (not on Mosaic) for not-in-person students, online, recorded
- **TA office hours**: TBA, on **“Course Help” channel** on MSTeams
- **Studying and Homework:** About 2–3 hours per lecture
  — **reading the textbook** , **writing proofs in CalcCheck$_{\text{Web}}$**

Grading

- **Homework**, from one lecture to the next — in total: **10%**
  - The weakest 2 or 3 homeworks are dropped (see outline)
  - MSAFs for homework are not processed
- **Roughly-weekly assignments** — in total: **16%**
  - 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:**
  - **15%** if not better than your final
  - **20%** if better than your final
    — in total at least: **30%**
    — in total up to: **40%**
  - Deferred midterms may be oral
  - Midterm weight will not be moved to final exam
- **Final** (closed book, 2.5 hours, **on CalcCheck$_{\text{Web}}$ / . . .**) **34%–44%**
  = **100%**
- Possible **bonus assignments** and other **bonus marks**
  — only count if you passed the course

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 <u>by hand on paper</u>
- **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 2021

Wolfram Kahl

2021-09-07

Part 2: Expressions and Calculations

The Answer

H1 Starting Point

```
Calculation:
    7 · 8
  =⟨ Fact `8 = 7 + 1` ⟩
    7 · (7 + 1)
  =⟨ Fact `7 = 10 - 3` ⟩
    (10 - 3) · (7 + 1)
  =⟨ “Distributivity of · over +” ⟩
    (10 - 3) · 7 + (10 - 3) · 1
  =⟨ “Distributivity of · over -” ⟩
    10 · 7 - 3 · 7 + 10 · 1 - 3 · 1
  =⟨ “Identity of ·” ⟩
    10 · 7 - 3 · 7 +   10   -   3
  =⟨ Fact `3 · 7 = 21` ⟩
    10 · 7 -    21  +   10   -   3
  =⟨ Fact `10 · 7 = 70` ⟩
    70     -    21  +   10   -   3
  =⟨ Fact `10 - 3 = 7` ⟩
    70     -    21  +          7
  =⟨ Fact `21 + 7 = 28` ⟩
    70     -         28
  =⟨ Fact `70 - 28 = 42` ⟩
    42
```

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 <u>as such</u> 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

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.

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:

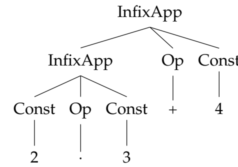
- **either** **some constant**
- **or** **some variable**
- **or** **some simpler expression** in parentheses
- **or** the application of **some unary prefix operator** to **some simpler expression**
- **or** the application of **some binary infix operator** to **two simpler expressions**

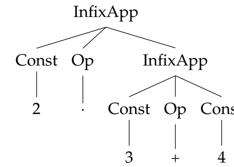
Why is this an expression?

$2 \cdot 3 + 4$

- 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 $+$.

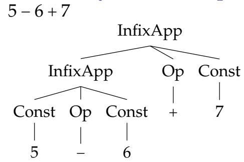
Table of Precedences

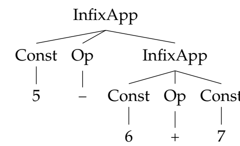
- $[x := e]$ (textual substitution) **(highest precedence)**
- . (function application)
- unary prefix operators $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot \quad / \quad \div \quad mod \quad gcd$
- $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \bullet$
- $\downarrow \quad \uparrow$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

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

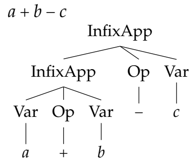
Why are these expressions? Which expressions are these?

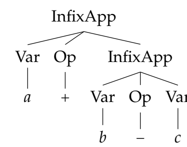
1. $5 - 6 + 7$





2. $a + b - c$





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 $\quad 9 - (5 - 2) \neq (9 - 5) - 2$

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

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

- 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 $\quad F \Rightarrow (T \Rightarrow F) \neq (F \Rightarrow T) \Rightarrow F$

An Equational Theory of Integers — Axioms (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)

Declaration: ℤ : Type
Declaration: _+_ : ℤ → (ℤ → ℤ)
— CalcCheck: Operator _+_: Associating to the left; precedence 100
Declaration: _·_ : ℤ → (ℤ → ℤ)
— CalcCheck: Operator _·_: Associating to the left; precedence 110
Axiom (15.1) (15.1a) "Associativity of +": (a + b) + c = a + (b + c)
Axiom (15.1) (15.1b) "Associativity of ·": (a · b) · c = a · (b · c)
Axiom (15.2) (15.2a) "Symmetry of +": a + b = b + a
Axiom (15.2) (15.2b) "Symmetry of ·": a · b = b · a
Axiom (15.3) "Additive identity" "Identity of +": 0 + a = a
Axiom (15.4) "Multiplicative identity" "Identity of ·": 1 · a = a
Axiom (15.5) "Distributivity" "Distributivity of · over +": a · (b + c) = a · b + a · c
Axiom (15.9) "Zero of ·": a · 0 = 0
Declaration: -_ : ℤ → ℤ
— CalcCheck: Operator -_: Non-associating; precedence 130
Declaration: _-_ : ℤ → ℤ
— CalcCheck: Operator _-_: Associating to the left; precedence 100
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 + $\quad 0 + a = a$	(15.13) Unary minus $\quad a + (-a) = 0$

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

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

The Answer

H1 Starting Point

```
Calculation:
    7 · 8
  =⟨ Fact `8 = 7 + 1` ⟩
    7 · (7 + 1)
  =⟨ Fact `7 = 10 - 3` ⟩
    (10 - 3) · (7 + 1)
  =⟨ "Distributivity of · over +" ⟩
    (10 - 3) · 7 + (10 - 3) · 1
  =⟨ "Distributivity of · over -" ⟩
    10 · 7 - 3 · 7 + 10 · 1 - 3 · 1
  =⟨ "Identity of ·" ⟩
    10 · 7 - 3 · 7 +   10   -   3
  =⟨ Fact `3 · 7 = 21` ⟩
    10 · 7 -   21  +   10   -   3
  =⟨ Fact `10 · 7 = 70` ⟩
    70     -   21  +   10   -   3
  =⟨ Fact `10 - 3 = 7` ⟩
    70     -   21  +        7
  =⟨ Fact `21 + 7 = 28` ⟩
    70     -       28
  =⟨ Fact `70 - 28 = 42` ⟩
    42
```

- Work through Homework 1
- Submit by 9 a.m. on Thursday, Sept. 9
- Get started working on Exercises 1.1, 1.2., 1.3
- Go to yor tutorial to continue working on Ex1 — bring your laptop!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-09

Part 1: Syntax of Mathematical Expressions

Mathematical Modelling

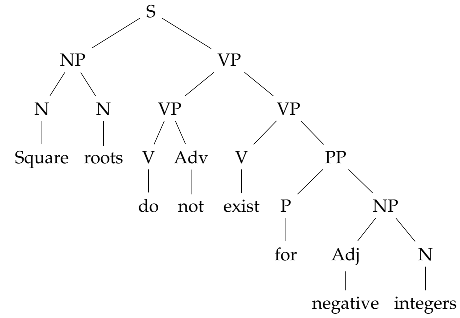
Textbook p. 2: How to specify an algorithm to compute b, an integer approximation to $\sqrt{n}$ for some integer n?

- Square roots do not exist for negative integers!
  Therefore, the algorithm must only be used for non-negative n.
  Precondition: $n \geq 0$
- To compute *an* approximation???
  42 is *an* approximation of $\sqrt{1000}$!
  "Reasonable" approximations (candidates for the *postcondition*):
  - $b^2 \leq n \leq (b+1)^2$
  - $abs(b^2 - n) \leq abs((b+1)^2 - n)$ and $abs(b^2 - n) \leq abs((b-1)^2 - n)$
  - $(b-1)^2 \leq n \leq b^2$

Now step back, and do "grammatical analysis"!

Natural-Language Grammatical Analysis: Sentence Structure Trees

Square roots do not exist for negative integers.



Mathematical Modelling uses Mathematical Expressions

Textbook p. 2: How to specify an algorithm to compute b, an integer approximation to $\sqrt{n}$ for some integer n?

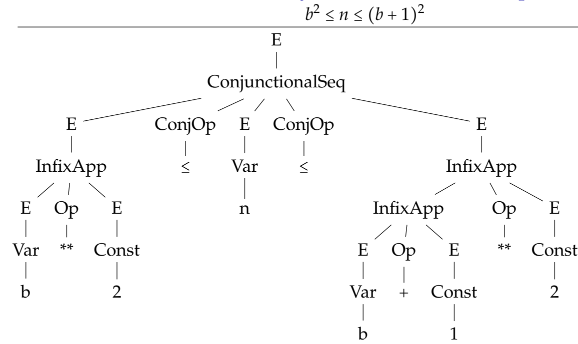
- Square roots do not exist for negative integers!
  Therefore, the algorithm must only be used for non-negative n.
  Precondition: $n > 0$
- To compute *an* approximation??? — 42 is *an* approximation of $\sqrt{1000}$!
  "Reasonable" approximations (candidates for the *postcondition*):
  - $b^2 \leq n \leq (b+1)^2$
  - $abs(b^2 - n) \leq abs((b+1)^2 - n)$ and $abs(b^2 - n) \leq abs((b-1)^2 - n)$
  - $(b-1)^2 \leq n \leq b^2$

Now step back, and do "grammatical analysis"!

- How is all that math put together?
- What are the different kinds of atoms ("words")?
- What are the different kinds of composite structures ("phrases")?
- What are the rules for analysis/synthesis of composite structures?

Grammatical Analysis for Mathematical Expression

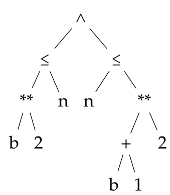
$b^2 \leq n \leq (b+1)^2$



Term Tree Presentation of Mathematical Expression

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

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



We write strings, but we think trees.

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

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.

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**

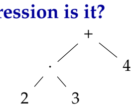
Why is this an expression?

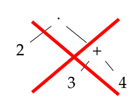
$2 \cdot 3 + 4$

- 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 $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot \quad / \quad \div \quad mod \quad gcd$
- $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \bullet$
- $\downarrow \quad \uparrow$
- $\#$
- $\triangleleft \quad \triangleright \quad \hat{}$
- $= \quad < \quad > \quad \in \quad \subset \quad \subseteq \quad \supset \quad \supseteq \quad |$ (conjunctional)
- $\vee \quad \wedge$
- $\Rightarrow \quad \Leftarrow$
- $\equiv$ **(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.

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 is no need to discuss whether we mean $(a + b) + c$ or $a + (b + c)$, because they are the same:

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

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

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

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

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

- 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 $\quad F \Rightarrow (T \Rightarrow F) \neq (F \Rightarrow T) \Rightarrow F$

Conjunctional Operators

Chains can involve different conjunctional operators:

$$
\begin{array}{ll}
 & 1 < i \leq j < 5 = k \\
\equiv & \langle \text{"Reflexivity of ="} \ `x = x` \quad \text{— conjunctional operators} \rangle \\
 & 1 < i \quad \wedge \quad i \leq j \quad \wedge \quad j < 5 \quad \wedge \quad 5 = k \\
\equiv & \langle \text{"Reflexivity of ="} \quad \text{—} \ \wedge \ \text{has lower precedence} \rangle \\
 & (1 < i) \quad \wedge \quad (i \leq j) \quad \wedge \quad (j < 5) \quad \wedge \quad (5 = k)
\end{array}
$$

$$
\begin{array}{ll}
 & x < 5 \in S \subseteq T \\
\equiv & \langle \text{"Reflexivity of ="} \quad \text{— conjunctional operators} \rangle \\
 & x < 5 \quad \wedge \quad 5 \in S \quad \wedge \quad S \subseteq T \\
\equiv & \langle \text{"Reflexivity of ="} \quad \text{—} \ \wedge \ \text{has lower precedence} \rangle \\
 & (x < 5) \quad \wedge \quad (5 \in S) \quad \wedge \quad (S \subseteq T)
\end{array}
$$

Remember this!!!

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 2021

Wolfram Kahl

2021-09-09

Part 2: Substitution

Plan for Part 2

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

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

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

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

  (The details of the underlying mechanisms, LADM 1.3, 1.5, are left to the next lecture.)

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).

Example 1:

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

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:

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

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:

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

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

$(x + y)[x := y - 3][y := z + 2]$

$=$ ⟨ "Reflexivity of =" — adding parentheses for clarity ⟩

$((x + y)[x := y - 3])[y := z + 2]$

$=$ ⟨ Substitution — performing inner substitution ⟩

$(((y - 3) + y))[y := z + 2]$

$=$ ⟨ Substitution — performing outer substitution ⟩

$((((z + 2) - 3) + (z + 2)))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$z + 2 - 3 + z + 2$

On CalcCheck$_{\text{Web}}$: **Exercise 2.2: *Substitutions***

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:

$(x + y)[x, y := y - 3, z + 2]$

$=$ ⟨ Substitution — performing substitution ⟩

$((y - 3) + (z + 2))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$y - 3 + z + 2$

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:

$(x + y)[x, y := y - 3, z + 2]$

$=$ ⟨ Substitution — performing substitution ⟩

$((y - 3) + (z + 2))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$y - 3 + z + 2$

Sequential Substitution:

$(x + y)[x := y - 3][y := z + 2]$

$=$ ⟨ "Reflexivity of =" — adding parentheses for clarity ⟩

$((x + y)[x := y - 3])[y := z + 2]$

$=$ ⟨ Substitution — performing inner substitution ⟩

$(((y - 3) + y))[y := z + 2]$

$=$ ⟨ Substitution — performing outer substitution ⟩

$((((z + 2) - 3) + (z + 2)))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$z + 2 - 3 + z + 2$

An Equational Theory of Integers — Axioms (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 $x := -(-a)$ ⟩ $\boxed{(0 + x = x)[x := -(-a)] \quad = \quad (0 + -(-a) = -(-a))}$

$0 \; + \; -(-a)$

$=$ ⟨ Unary minus (15.13y) with $y := a$ ⟩ $\boxed{(y + (-y) = 0)[y := a] \quad = \quad (a \; + \; (-a) = 0)}$

$a \; + \; (-a) \; + \; -(-a)$

$=$ ⟨ Unary minus (15.13y) with $y := -a$ ⟩ $\boxed{(y + (-y) = 0)[y := -a] \quad = \quad (-a \; + \; (-(-a)) = 0)}$

$a \; + \; 0$

$=$ ⟨ Identity of + (15.3x) with $x := a$ ⟩ $\boxed{(0 + x = x)[x := a)] \quad = \quad (0 + a = a}$

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 · (a + b)
  =⟨ “Distributivity of · over +” with `a, b, c ≔ - 1, a, b` ⟩
    - 1 · a + - 1 · b
  =⟨ (15.20) with `a ≔ a` ⟩
    - a  + - 1 · b
  =⟨ (15.20) with `a ≔ b` ⟩
    - a + - b
```

- Backquotes enclose math embedded in English. (Markdown convention)
- Substitution notation as in LADM: *variables* $:=$ *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.
- 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 2021

Wolfram Kahl

2021-09-13

Part 1: Foundations of Applying Equations in Context

Plan for Today — LADM 1.2–1.6

- **Anatomy of calculation** based on **Substitution:**
  - **Inference rule Substitution:** Justifies applying instances of theorems:

    $2 \cdot y \; + \; -(2 \cdot y)$

    $= \langle$ “Unary minus” $a \; + \; -a \; = \; 0$ with ‘$a \; := \; 2 \cdot y$’ $\rangle$

    0
  - **Inference rule Leibniz:** Justifies applying (instances of) **equational** theorems deeper inside expressions:

    $2 \cdot x + 3 \cdot (y \; - \; 5 \cdot (4 \cdot x + 7))$

    $= \langle$ “Subtraction” $a \; - \; b \; = \; a \; + \; -b$ with ‘$a, b \; := \; y, 5 \cdot (4 \cdot x + 7)$’ $\rangle$

    $2 \cdot x + 3 \cdot (y \; + \; -(5 \cdot (4 \cdot x + 7)))$
- **Reasoning about Assignment Commands in Imperative Programs**

  $$\{\ Q[x := E]\ \}\ x := E\ \{\ Q\ \}$$

  …and more inference rules!

What is an Inference Rule?

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

- **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
- 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:** $\dfrac{E}{E[x := R]}$

Example:

If $\quad a + 0 = a \quad$ is a theorem, $\qquad$ $\boxed{\text{“Identity of +”}}$

then $\; 3 \cdot b + 0 = 3 \cdot b \quad$ is also a theorem. $\qquad$ $\boxed{\text{“Identity of +” with ‘}a := 3 \cdot b\text{’}}$

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

Example:

$$\frac{z \;\geq\; x \uparrow y \quad \equiv \quad z \geq x \quad \wedge \quad z \geq y}{x + y \;\geq\; x \uparrow y \quad \equiv \quad x + y \geq x \quad \wedge \quad x + y \geq y}$$

Inference Rule Scheme: Substitution

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

Really an **inference rule scheme**:
works for **every combination** of

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

Example 1: $\qquad \dfrac{a + 0 = a}{3 \cdot b + 0 = 3 \cdot b}$

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

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

Inference Rule Scheme: Substitution

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

Really an **inference rule scheme**:
works for **every combination** of

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

Example 2: $\qquad \dfrac{a \cdot (b + c) = a \cdot b + a \cdot c}{(2 + x) \cdot (b + c) = (2 + x) \cdot b + (2 + x) \cdot c}$

If $\quad a \cdot (b + c) = a \cdot b + a \cdot c \quad$ is a theorem,
then $(2 + x) \cdot (b + c) = (2 + x) \cdot b + (2 + x) \cdot c \quad$ is also a theorem.

- expression E is $a \cdot (b + c) = a \cdot b + a \cdot c$
- the variable x substituted into is $\quad a$
- the substituted expression R is $\quad 2 + x$

Inference Rule Scheme: Substitution

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

Really an **inference rule scheme**:
works for **every combination** of

- expression E,
- <u>variable **list**</u> x, and
- **corresponding** <u>expression **list**</u> R.

Example:

If $\quad x + y = y + x \quad$ is a theorem,
then $\quad b + 3 = 3 + b \quad$ is also a theorem.

- expression E is $\quad x + y = y + x$
- variable list x is $\quad x, y$
- corresponding expression list R is $\quad b, 3$

Logical Definition of Equality

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

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

Two **inference rule schemes**:

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

— the rule of “replacing equals for equals”

Using Leibniz’ Rule in (15.21)

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

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

Prove: $\quad$ (15.21) $\quad (-a) \cdot b = a \cdot (-b)$

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

$\quad (-a) \cdot b$

$= \langle$ (15.20) — **via Leibniz (1.5) with** E **chosen as** $\; z \cdot b \; \rangle$

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

$= \langle$ Associativity (15.1) and Symmetry (15.2) of $\cdot$ $\rangle$

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

$= \langle$ (15.20) $\rangle$

$\quad a \cdot (-b)$

Using Leibniz together with Substitution in (15.21)

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

Prove: (15.21) $(-a) \cdot b = a \cdot (-b)$

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

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

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

Combining Leibniz' Rule with Substitution

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

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

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

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

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

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

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

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

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

Opportunity for Practice: Equational Theory of Integers — Axioms and Theorems

(15.1) **Associativity**	(15.2) **Symmetry**	(15.3) **Identity of** +
$(a + b) + c = a + (b + c)$	$a + b = b + a$	$0 + a = a$
$(a \cdot b) \cdot c = a \cdot (b \cdot c)$	$a \cdot b = b \cdot a$	$a + 0 = a$
(15.5) **Distributivity**	(15.4) **Identity of** $\cdot$	(15.13) **Unary minus**
$a \cdot (b + c) = a \cdot b + a \cdot c$	$1 \cdot a = a$	$a + (-a) = 0$
$(b + c) \cdot a = b \cdot a + c \cdot a$	$a \cdot 1 = a$	(15.14) **Subtraction** $a - b = a + (-b)$

(15.17) $-(-a) = a$ (15.22) $a \cdot (-b) = -(a \cdot b)$

(15.18) $-0 = 0$ (15.23) $(-a) \cdot (-b) = a \cdot b$

(15.20) $-a = -1 \cdot a$ (15.24) $a - 0 = a$

(15.19) $-(a + b) = -a + -b$ (15.25) $(a - b) + (c - d) = (a + c) - (b + d)$

(15.21) $(-a) \cdot b = a \cdot (-b)$ (15.25a) $a + (b - c) = (a + b) - c$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-13

Part 2: Correctness of Assignment Commands

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

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**:

$$
\begin{array}{lll}
& [\ (x, 5),\ (y, 6)\ ] & \text{`}x < y\text{` holds} \\
\leadsto & \langle \quad x := x + y \quad \rangle & \\
& [\ (x, 11),\ (y, 6)\ ] & \text{`}x < y\text{` does not hold} \\
\leadsto & \langle \quad y := x - y \quad \rangle & \\
& [\ (x, 11),\ (y, 5)\ ] & \text{`}x < y\text{` does not hold}
\end{array}
$$

- 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:** $\{\, Q[x := E] \,\}\ x \mathrel{\mathbf{:=}} E\ \{\, Q \,\}$ $\boxed{Q[x := E] \quad \Rightarrow[\, x \mathrel{\mathbf{:=}} E \,] \quad Q}$
- **Example:**
  - $(x = 5)[x := x + 1] \quad \Rightarrow[\, x \mathrel{\mathbf{:=}} x + 1 \,] \quad x = 5$
  - $(x + 1 = 5) \quad \Rightarrow[\, x \mathrel{\mathbf{:=}} x + 1 \,] \quad x = 5$
  -
    $$\begin{array}{ll} & x + 1 = 5 \\ \equiv & \langle \text{ Substitution } \rangle \\ & (x = 5)[x := x + 1] \\ \Rightarrow[\, x \mathrel{\mathbf{:=}} x + 1 \,] & \langle \text{ Assignment } \rangle \\ & x = 5 \end{array}$$
- **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):
  $P \Rightarrow[\, C \,]\ Q$
- **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:** $\{\, Q[x := E] \,\}\ x \mathrel{\mathbf{:=}} E\ \{\, Q \,\}$ $\boxed{Q[x := E] \quad \Rightarrow[\, x \mathrel{\mathbf{:=}} E \,] \quad Q}$
- **Longer example:**

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

Sequential Composition of Commands

```
Primitive inference rule “SEQ”:
  `{ P } C₁ { Q }` , `{ Q } C₂ { R }`
 ⊢─────────────────────────────────────
         `{ P } C₁ ; C₂ { R }`
```

```
Primitive inference rule “Sequence”:
   `P  ⇒[ C₁ ]  Q`,  `Q  ⇒[ C₂ ]  R`
 ⊢──────────────────────────────────────
     `P  ⇒[ C₁ ; C₂ ]  R`
```

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

$$\begin{array}{ll} & P \\ \Rightarrow[\, C_1 \,] & \langle \ldots \rangle \\ & Q \\ \Rightarrow[\, C_2 \,] & \langle \ldots \rangle \\ & R \end{array}$$

- No need to refer to this rule explicitly.

Example Proof for a Sequence of Assignments

Fact: $x = 5 \Rightarrow[\, (y := x + 1 \,;\, x := y + y) \,]\ x = 12$

Proof:

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

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;
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-14

Part 1: Boolean Expression

Plan for Today

- LADM Chapter 2: Boolean Expressions
  - Meaning of Boolean Operators
  - Equality versus Equivalence
  - Truth Tables
  - Satisfiability and Validity
  - Modeling English Propositions
- Starting with LADM Chapter 3: Propositional Calculus
  - Equivalence, Negation, Inequivalence

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”
- $\_\wedge\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — conjunction, “logical and”
- $\_\vee\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — disjunction, “logical or”
- $\_{\Rightarrow}\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — implication, “implies”, “if … then …”
- $\_{\equiv}\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — equivalence, “if and only if”, “iff”
- $\_{\not\equiv}\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — inequivalence, “exclusive or”

Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- $.$ (function application)
- unary prefix operators $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot \quad / \quad \div \quad \mathit{mod} \quad \mathit{gcd}$
- $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \bullet$
- $\downarrow \quad \uparrow$
- $\#$
- $\lhd \quad \rhd \quad \hat{}$
- $= \quad \neq \quad < \quad > \quad \in \quad \subset \quad \subseteq \quad \supset \quad \supseteq \quad |$ (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 $**$, $\lhd$, $\Rightarrow$, $\to$, 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.

Equality versus Equivalence

The operators $=$ (as Boolean operator) and $\equiv$

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

      $$\boxed{(p \equiv q \equiv r) \quad = \quad ((p \equiv q) \equiv r) \quad = \quad (p \equiv (q \equiv r))}$$

    - $=$ is **conjunctional**:

      $$\boxed{(x = y = z) \quad \equiv \quad ((x = y) \quad \wedge \quad (y = z))}$$

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

Evaluation of Boolean Expressions Using Truth Tables

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

			$\wedge$					$\not\equiv$ $\neq$	$\vee$	nor	$\equiv$ $=$		$\Leftarrow$		$\Rightarrow$	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

Alternative Presentation of Truth Tables

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

- Identify variables
- Identify subexpressions — **in doubt, add parentheses!**
- Enumerate possible states (of the variables)
- Evaluate (sub-)expressions in all states
  writing the result below the operator forming the subexpression

Validity and Satisfiability

- A boolean expression is **satisfied** in state s
  iff it evaluates to *true* in state s.
- A boolean expression is **valid**
  iff it is satisfied in every state.
- A valid boolean expression is called a **tautology**.
- A boolean expression is **satisfiable**
  iff there is a state in which it is satisfied.
- 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

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-14

Part 2: Propositional Calculus: $\equiv$, $\neg$, $\not\equiv$

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 $\equiv$**: $\boxed{((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))}$

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

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

$$p \equiv p \equiv q \equiv q$$
$$= \langle \text{(3.2) Symmetry of } \equiv \text{, with } p,\ q\ :=\ p,\ q \equiv q\ \rangle$$
$$p \equiv q \equiv q \equiv p \quad \text{— This is (3.2) Symmetry of } \equiv$$

Equivalence Axioms — Example Proof with Parentheses

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

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

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

$$p \equiv (p \equiv (q \equiv q))$$
$$\equiv \langle \text{(3.2) Symmetry of } \equiv \text{, with } p,\ q\ :=\ p,\ (q \equiv q)\ \rangle$$
$$p \equiv ((q \equiv q) \equiv p) \quad \text{— This is (3.2) Symmetry of } \equiv$$

Equivalence Axioms — Introducing *true*

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

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

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 $\equiv$**: $\boxed{true \equiv q \equiv q}$

Can be used as:

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

Equivalence Axioms, and Theorem (3.4)

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

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

(3.3) **Axiom, Identity of $\equiv$:** $\boxed{\mathit{true} \equiv q \equiv q}$

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

The least interesting theorem:

Proving (3.4) *true*:

$$
\begin{array}{ll}
& \mathit{true} \\
= & \langle \text{Identity of } \equiv \text{(3.3), with } q := \mathit{true} \rangle \\
& \mathit{true} \equiv \mathit{true} \\
= & \langle \text{Identity of } \equiv \text{(3.3), with } q := q \rangle \\
& \mathit{true} \equiv q \equiv q \quad \text{— This is Identity of } \equiv \text{(3.3)}
\end{array}
$$

Equivalence Axioms and Theorems

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

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

(3.3) **Axiom, Identity of $\equiv$:** $\boxed{\mathit{true} \equiv q \equiv q}$

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.

Negation Axioms

(3.8) **Axiom, Definition of *false*:** $\boxed{\mathit{false} \equiv \neg \mathit{true}}$

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

(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 $\not\equiv$:** $\boxed{(p \not\equiv q) \equiv \neg(p \equiv q)}$

(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 $\not\equiv$:**

$$
\begin{array}{lll}
& p \not\equiv q & \\
= & \langle \text{(3.10) Definition of } \not\equiv \rangle & \text{...... Unfold} \\
& \neg(p \equiv q) & \\
= & \langle \text{(3.2) Symmetry of } \equiv \rangle & \\
& \neg(q \equiv p) & \\
= & \langle \text{(3.10) Definition of } \not\equiv \rangle & \text{...... Fold} \\
& q \not\equiv p &
\end{array}
$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-16

Part 1: Propositional Calculus: $\neg$, $\not\equiv$, $\vee$

Plan for Today

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

Equivalence Axioms and Theorems

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

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

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 $\equiv$:** $\boxed{\mathit{true} \equiv q \equiv q}$

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 $P \equiv \mathit{true}$.

Negation Axioms

(3.8) **Axiom, Definition of *false*:** $\boxed{\mathit{false} \equiv \neg \mathit{true}}$

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

(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 $\not\equiv$:** $\boxed{(p \not\equiv q) \equiv \neg(p \equiv q)}$

Negation Axioms and Theorems

(3.8) **Axiom, Definition of *false*:** $\boxed{\mathit{false} \equiv \neg \mathit{true}}$

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

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

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*:** $\neg \mathit{false} \equiv \mathit{true}$

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

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

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 ∘ that is defined in terms of another, say •, expand the definition of ∘ to arrive at a formula that contains •; exploit properties of • to manipulate the formula, and then (possibly) reintroduce ∘ 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 $\not\equiv$:**

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

Disjunction Axioms

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

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

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

(3.27) **Axiom, Distributivity of $\vee$ over $\equiv$**:
$\boxed{p \vee (q \equiv r) \equiv p \vee q \equiv p \vee r}$

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

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**: $\boxed{p \vee \neg p}$

— 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) $p \vee true \equiv true$**:**

$p \vee true$
$\equiv \langle$ Identity of $\equiv$ (3.3) $\rangle$
$p \vee (q \equiv q)$
$\equiv \langle$ Distr. of $\vee$ over $\equiv$ (3.27) $\rangle$
$p \vee q \equiv p \vee q$
$\equiv \langle$ Identity of $\equiv$ (3.3) $\rangle$
$true$

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

$true$
$\equiv \langle$ Identity of $\equiv$ (3.3) $\rangle$
$p \vee p \equiv p \vee p$
$\equiv \langle$ Distr. of $\vee$ over $\equiv$ (3.27) $\rangle$
$p \vee (p \equiv p)$
$\equiv \langle$ Identity of $\equiv$ (3.3) $\rangle$
$p \vee true$

?

(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.

(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

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-16

Part 2: Propositional Calculus: $\wedge$

The Conjunction Axiom: The "Golden Rule"

(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:

- $p \wedge q \quad = \quad (p \equiv q \quad \equiv \quad p \vee q)$ — **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**: $p \wedge (p \vee q) \equiv p$

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

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

$p \vee (\neg p \wedge q) \equiv p \vee q$

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

$\neg(p \vee q) \equiv \neg p \wedge \neg q$

De Morgan's Laws

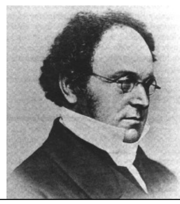
Prove: (3.47) **De Morgan**: $\neg(p \wedge q) \equiv \neg p \vee \neg q$

$\neg(p \vee q) \equiv \neg p \wedge \neg q$

Use, in particular:

(3.32t) $t \vee u \equiv t \vee \neg u \equiv t$

(3.35t) **Axiom, Golden rule:** $t \wedge u \equiv t \equiv u \equiv t \vee u$



Theorems Relating $\wedge$ and $\equiv$

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

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

(3.50) Strong Modus Ponens: $p \wedge (q \equiv p) \equiv p \wedge q$

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

Alternative Definitions of $\equiv$ and $\not\equiv$

(3.52) **Definition of** $\equiv$: $p \equiv q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$

(3.53) **Definition of** $\not\equiv$: $p \not\equiv q \equiv (\neg p \wedge q) \vee (p \wedge \neg q)$

Ladies or Tigers: First Case, Formalisation, Long S_2

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.

$R1L$	:=	There is a lady in room 1	S_1	$\equiv$	$R1L \wedge R2T$
$R2T$	:=	There is a tiger in room 2	S_2	$\equiv$	$(R1L \vee \neg R2T) \wedge (\neg R1L \vee R2T)$

$S_1 \not\equiv S_2$

Ladies or Tigers: First Case, Long S_2, Solution

$R1L$	:=	There is a lady in room 1	S_1	$\equiv$	$R1L \wedge R2T$
$R2T$	:=	There is a tiger in room 2	S_2	$\equiv$	$(R1L \vee \neg R2T) \wedge (\neg R1L \vee R2T)$

$$\begin{array}{ll}
 & S_1 \not\equiv S_2 \\
= & \langle \text{Def. } S_1, S_2 \rangle \\
 & (R1L \wedge R2T) \not\equiv ((R1L \vee \neg R2T) \wedge (\neg R1L \vee R2T)) \\
= & \langle \text{(3.14) } p \not\equiv q \equiv \neg p \equiv q\text{, (3.35) Golden Rule} \rangle \\
 & \neg(R1L \wedge R2T) \equiv R1L \vee \neg R2T \equiv \neg R1L \vee R2T \equiv R1L \vee \neg R2T \vee \neg R1L \vee R2T \\
= & \langle \text{(3.28) Excluded Middle, (3.29) Zero of } \vee \rangle \\
 & \neg(R1L \wedge R2T) \equiv R1L \vee \neg R2T \equiv \neg R1L \vee R2T \equiv \mathit{true} \\
= & \langle \text{(3.47) De Morgan, (3.3) Identity of } \equiv \rangle \\
 & \neg R1L \vee \neg R2T \equiv R1L \vee \neg R2T \equiv \neg R1L \vee R2T \\
= & \langle \text{(3.32) } p \vee q \equiv p \vee \neg q \equiv p \rangle \\
 & \neg R2T \equiv \neg R1L \vee R2T \\
= & \langle \text{(3.32) } p \vee q \equiv p \vee \neg q \equiv p \rangle \\
 & \neg R2T \equiv \neg R1L \vee \neg R2T \equiv \neg R1L \\
= & \langle \text{(3.35) Golden Rule} \rangle \\
 & \neg R1L \wedge \neg R2T \\
= & \langle R1T = \neg R1L \text{ and } R2L = \neg R2T \rangle \\
 & R1T \wedge R2L
\end{array}$$

Raymond Smullyan posed many puzzles about an island that has two kinds of inhabitants:

- **knights, who always tell the truth**, and
- **knaves**, who always lie.

You encounter two people A and B.
What are A and B if

1. A says "We are both knaves."?
2. A says "At least one of us is a knave."?
3. A says "If I am a knight, then so is B."?
4. **A says "We are of the same type."?**
5. A says "B is a knight" and

   B says "The two of us are opposite types."?

Explanation: $A_H \equiv \boxed{A \text{ is a knight}}$

Axiom schema "Knighthood": $\boxed{A \text{ says "}X\text{"}} \equiv A_H \equiv X$

You encounter two people A and B. What are A and B if

4. A says "We are of the same type."?

$$\begin{array}{ll}
 & \boxed{A \text{ says "}A_H \equiv B_H\text{"}} \\
\equiv & \langle \text{"Knighthood"} \rangle \\
 & A_H \quad \equiv \quad (A_H \equiv B_H) \\
\equiv & \langle \text{(3.3) Associativity of } \equiv \rangle \\
 & A_H \quad \equiv \quad A_H \quad \equiv \quad B_H \\
\equiv & \langle \text{(3.2) Symmetry of } \equiv\text{: } p \equiv q \equiv q \equiv p \rangle \\
 & B_H
\end{array}$$

You encounter two people A and B. What are A and B if

4. A says "We are of the same type."?

Explanation: $A_V \equiv \boxed{A \text{ is a knave}}$

Axiom schema "Knavehood": $\boxed{A \text{ says } X} \equiv A_V \equiv \neg X$

$$\begin{array}{ll}
 & \boxed{A \text{ says } (A_V \equiv B_V)} \equiv A_V \equiv \neg(A_V \equiv B_V) \qquad \text{— This is "Knavehood"} \\
\equiv & \langle \text{(3.9) } \neg(p \equiv q) \equiv \neg p \equiv q \rangle \\
 & \boxed{A \text{ says } (A_V \equiv B_V)} \quad \equiv \quad A_V \quad \equiv \quad A_V \quad \equiv \quad \neg B_V \\
\equiv & \langle \text{(3.2) Symmetry of } \equiv\text{: } p \equiv q \equiv q \equiv p \rangle \\
 & \boxed{A \text{ says } (A_V \equiv B_V)} \quad \equiv \quad \neg B_V
\end{array}$$

Avoid Repetition in Proofs!

(3.22) **Principle:** Structure proofs to avoid repeating the same subexpression on many lines.

Textbook, p. 48

You encounter two people A and B. What are A and B if

- A says "We are of the same type."?

Explanation: $A_V \equiv \boxed{A \text{ is a knave}}$

Axiom schema "Knavehood": $\boxed{A \text{ says } X} \quad \equiv \quad A_V \equiv \neg X$

$$
\begin{array}{ll}
& \boxed{A \text{ says } (A_V \equiv B_V)} \\
\equiv & \langle \text{"Knavehood"} \rangle \\
& A_V \quad \equiv \quad \neg(A_V \equiv B_V) \\
\equiv & \langle (3.9)\ \neg(p \equiv q) \ \equiv\ \neg p \equiv q \rangle \\
& A_V \quad \equiv \quad A_V \quad \equiv \quad \neg B_V \\
\equiv & \langle (3.2) \text{ Symmetry of } \equiv\text{: } p \equiv q \equiv q \equiv p \rangle \\
& \neg B_V
\end{array}
$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-20

Part 1: Natural Numbers, Natural Induction

Plan for Today

- Natural Numbers and **Induction**
- **Continuing Propositional Calculus (LADM Chapter 3)**
  - (Conjunction)
  - Implication

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!

What is a natural number?

How is the set ℕ of all natural numbers defined?

(Without referring to the integers)

(From first principles…)

Natural Numbers — ℕ

- The set of all **natural numbers** is written 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 "$suc\ 3$"
  - …

Natural Numbers — Rigorous 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 equal **if and only if** they are constructed in the same way.
  Example: $suc\ suc\ suc\ 0 \quad \neq \quad suc\ suc\ suc\ suc\ 0$

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

---

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.

Induction principle for the natural numbers:

- if $P(0)$ — If P holds for 0
- and if $P(m)$ implies $P(suc\ m)$, — and whenever P holds for m, it also holds for $suc\ m$,
- then for all $m : ℕ$ 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 $P[m := suc\ m]$ from P, — and whenever P holds for m, it also holds for $suc\ m$,
- 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 $P[m := suc\ m]$
        using **Induction hypothesis** P

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

Factorial — Inductive 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 only equal if constructed in the same way.

$\mathbb{N}$ is an inductively-defined set.

The factorial operator "_!" on $\mathbb{N}$ 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:
  $$(suc\ n)\,! = (suc\ n) \cdot (n\,!)$$

_! is an inductively-defined function.

Proving properties about inductively-defined functions on $\mathbb{N}$ frequently requires use of the induction principle for $\mathbb{N}$.

Even Natural Numbers — Inductive Definition

- The predicates even and odd are declared as Boolean-valued **functions**:
  ```
  Declaration: even, odd : ℕ → 𝔹
  ```
- Function application of function f to argument a is written as **juxtaposition**: $f\ a$
- The definitions provided in Homework 5.1 are **inductive definitions**:
  ```
  Axiom "Zero is even":                even 0
  Axiom "Even successor (direct)":     even (suc n) ≡ ¬ (even n)
  ```

even is an inductively-defined function.

Why does this define even for all possible arguments?
Because:

- *even* takes **one** argument of type $\mathbb{N}$
- This argument is **always** either 0, or $suc\ k$ for some **smaller** $k : \mathbb{N}$
- Each clause covers one case completely.
- The second clause "builds up" the domain of definition of *even* from smaller to larger n.

Proving "Odd is not even"

```
Theorem "Odd is not even":  odd n ≡ ¬ (even n)
Proof:
  By induction on `n : ℕ`:
    Base case:
        odd 0
      ≡⟨ ? ⟩
        ¬ (even 0)
    Induction step:
        odd (suc n)
      ≡⟨ ? ⟩
        ¬ even (suc n)
```

```
"Zero is even":                even 0
"Even successor (direct)":     even (suc n) ≡ ¬ (even n)
```

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 $P[m := suc\ m]$
        using **Induction hypothesis** P

Natural Number Addition — Inductive 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 only equal if constructed in the same way.

$\mathbb{N}$ is an inductively-defined set.

Addition on $\mathbb{N}$ 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: $(suc\ n) + q = suc\ (n + q)$

_+_ is an inductively-defined function.

Proving "Right-Identity of +"

```
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
```

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 $P[m := suc\ m]$
        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".

Proving "Right-Identity of +" — With Details

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

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-20

Part 2: Propositional Calculus: (∧), ⇒

The Conjunction Axiom: The "Golden Rule"

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

Can be used as:

- $p \land q \quad = \quad (p \equiv q \quad \equiv \quad p \lor q)$ — **Definition of ∧**
- $(p \equiv q) \quad = \quad (p \land q \quad \equiv \quad p \lor q)$
- ...

Theorems:

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

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

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

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

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

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

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

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**: $p \wedge (p \vee q) \equiv p$
$p \vee (p \wedge q) \equiv p$

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

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

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)** $p \wedge q \equiv p \wedge \neg q \equiv \neg p$

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

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

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

Alternative Definitions of $\equiv$ and $\not\equiv$

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

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

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**: $\boxed{p \Leftarrow q \equiv q \Rightarrow p}$

Rewriting Implication:

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

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**: $\boxed{p \wedge q \equiv p \equiv q \equiv p \vee q}$

Can be used as:

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

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

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

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 Theorems 2

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

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

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

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

Implication Theorems 3

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

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

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

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

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

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-21

Part 1: CalcCheck-checked Mystery Steps

Plan for Today

- **Continuing Propositional Calculus (LADM Chapter 3)**
  - CalcCheck-checked mystery steps
  - Implication, continued
  - Implication as an Order, Order Relations, Order Concepts

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

What Equivalences/Equalities are in the Golden Rule?

$p \wedge q \quad\equiv\quad p \equiv q$ **is not a consequence of (3.35) Golden rule!**

$p \wedge q \quad\equiv\quad p \vee q$ **is not a consequence of (3.35) Golden rule!**

Equality versus Equivalence

The operators $=$ (as Boolean operator) and $\equiv$

- have the **same meaning** (represent the same function),
- but **are used with different notational conventions:**
  - different precedences ($\equiv$ has lowest)
  - different **chaining behaviour**:
    - $\equiv$ is associative: $\boxed{(p \equiv q \equiv r) \quad=\quad ((p \equiv q) \equiv r) \quad=\quad (p \equiv (q \equiv r))}$
    - $=$ is **conjunctional**: $\boxed{(p = q = r) \quad\equiv\quad ((p = q) \quad\wedge\quad (q = r))}$

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

What Equivalences/Equalities are in the Golden Rule?

$p \wedge q \quad\equiv\quad p \equiv q$ **is not a consequence of (3.35) Golden rule!**

$p \wedge q \quad\equiv\quad p \vee q$ **is not a consequence of (3.35) Golden rule!**

Equality versus Equivalence — in other words

- Writing $p = q = r$ is the same as writing $(p = q) \wedge (q = r)$
- Writing $p \equiv q \equiv r$ is the same as writing $p \equiv (q \equiv r)$
  and the same as writing $(p \equiv q) \equiv r$
- Writing $p \equiv q \equiv r$ can also be seen to be
  the same as writing $p = (q = r)$
  and the same as writing $(p = q) = r$
  — **but only for Boolean expression** p, q, r

How?

$$
\begin{array}{ll}
& p \wedge p \\
\equiv & \langle \text{(3.35) Golden rule } p \wedge q \equiv p \equiv q \equiv p \vee q \rangle \\
& p \vee p \\
\equiv & \langle \text{(3.26) Idempotency of } \vee \rangle \\
& p
\end{array}
$$

?

How can the Golden rule have been applied here?

(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:

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

Three Steps!

$$
\begin{array}{ll}
& p \wedge p \\
\equiv & \langle \text{(3.35) Golden rule } (p \wedge q) = (p \equiv q \equiv p \vee q) \rangle \\
& p \equiv p \equiv p \vee p \\
\equiv & \langle \text{Adding parentheses} \rangle \\
& p \equiv (p \equiv p \vee p) \\
\equiv & \langle \text{(3.35) Golden rule } (p \wedge q \equiv p) = (q \equiv p \vee q) \rangle \\
& p \equiv (p \equiv p \wedge p) \\
\equiv & \langle \text{Removing parentheses} \rangle \\
& p \equiv p \equiv p \wedge p \\
\equiv & \langle \text{(3.35) Golden rule } (p \wedge q \equiv p \equiv q) = (p \vee q) \rangle \\
& p \vee p \\
\equiv & \langle \text{(3.26) Idempotency of } \vee \rangle \\
& p
\end{array}
$$

!

CalcCheck-checked Mystery Steps

```
Calculation:
    true ≡ p ≡ ¬ p
  ≡⟨ (3.15) `¬ p ≡ p ≡ false` ⟩
    false
```

?

```
Calculation:
    p ≡ ¬ q ≡ p ∨ q
  ≡⟨ (3.32) ⟩
    ¬ p ∨ ¬ q
```

?

- **If you don't understand it, don't submit it!**
  (Understand the precise way in which the rule has been applied!)
- **If you encounter such "mystery steps", report!**
  (E.g. in MSTeams channels)
- When reporting such cases, or asking questions about CalcCheck,
  in particular when writing e-mails,

  include (plain UTF8) text, not images!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-21

Part 2: Implication

Implication Theorems 4

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

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

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

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

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

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 $-$: $\quad (x+y)-z \;=\; x+(y-z)$
- $-$ **is not** mutually associative with $+$: $\quad (5-2)+3 \neq 5-(2+3)$

Some Property Names (ctd.)

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 idempotent**": $x \odot x \;=\; x$
- "**$\square$ is a left-unit (or left-identity) of $\odot$**": $\square \odot x \;=\; x$
- "**$\square$ is a right-unit (or right-identity) of $\odot$**": $x \odot \square \;=\; x$
- "**$\square$ is a unit/identity of $\odot$**": $\square \odot x \;=\; x \;=\; x \odot \square$
- "**$\square$ is a left-zero of $\odot$**": $\square \odot x \;=\; \square$
- "**$\square$ is a right-zero of $\odot$**": $x \odot \square \;=\; \square$
- "**$\square$ is a zero of $\odot$**": $\square \odot x \;=\; \square \;=\; x \odot \square$
- "**$\odot$ 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

Implication Theorems 5

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

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

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

Do not be discouraged by the number of theorems. You do not have to memorize them all. It will suffice to become familiar with them and how they are organized, so you can find the ones you need when developing a proof. The more practice you have using the theorems, the more they will become your formal friends, who serve you in your mathematical work.

LADM p. 42

Some Important Implication Theorems

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.

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

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

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

(3.74) **Definition of $\neg$ from $\Rightarrow$:** $p \Rightarrow false \;\equiv\; \neg p$

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

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

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

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

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-21

Part 3: Implication as Order, Order Relations

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: $k \mid n \;\equiv\; ((lcm(k,n) \;=\; n)$

(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: $k \mid n \;\equiv\; ((gcd(k,n) \;=\; k)$

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**.
- We have seen:
  - $\_{=}\_ : T \rightarrow T \rightarrow \mathbb{B}$
  - $\_{=}\_ : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B}$
  - $\_{=}\_ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B}$
  - $\_{\leq}\_ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B}$
  - $\_{\equiv}\_ : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$
  - $\_{\Rightarrow}\_ : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$

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:
  - $\_{=}\_ : T \rightarrow T \rightarrow \mathbb{B}$
  - $\_{\leq}\_ : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B}$
  - $\_{\geq}\_ : \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B}$
  - $\_{\leq}\_ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B}$
  - $\_{\geq}\_ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B}$
  - $\_{\mid}\_ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B}$
  - $\_{\equiv}\_ : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$
  - $\_{\Rightarrow}\_ : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$
  - $\_{\subseteq}\_ : \textbf{set}\ T \rightarrow \textbf{set}\ T \rightarrow \mathbb{B}$

Implication Theorems 6

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

(3.80b) **Reflexivity wrt. Equivalence:** $(p \equiv q) \Rightarrow (p \Rightarrow q)$

(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:** $(p \Rightarrow q) \wedge (q \equiv r) \;\Rightarrow\; (p \Rightarrow r)$

Some Order-Related Concepts

An order $\_\le\_$ on T may have (or may not have):

- a **least element** denoted b: A constant b such that $b \le x$ is valid
  - $\_\le\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$ has no least element
  - $\_\le\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has least element 0
  - $\_\ge\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has no least element
  - $\_|\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has least element 1
- a **greatest element** denoted t: A constant t such that $x \le t$ is valid
  - $\_\le\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has no greatest element
  - $\_\ge\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has greatest element 0
  - $\_|\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has greatest element 0
- have **binary maxima**: An operation $\_\sqcup\_$ such that $x \sqcup y$ is the least element that is at least x and also at least y
- have **binary minima**: An operation $\_\sqcap\_$ such that $x \sqcap y$ is the greatest element that is at most x and also at most y

Monotonicity, Isotonicity, Antitonicity

- Let $\_\le\_$ 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 ∧:** $(p \Rightarrow q) \;\Rightarrow\; p \wedge r \;\Rightarrow\; q \wedge r$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-23

Part 1: Leibniz as Axiom, Replacement Theorems

Plan for Today

- **Continuing Propositional Calculus (LADM Chapter 3)**
  - Leibniz as axiom, and “Replacement” theorems
- Sum and Product Quantification
  - (approaching LADM chapter 8)
  - Quantification expansion
- (Next week: LADM chapter 4, and then chapters 8 and 9.)

Announcement for the CS Society: Hiring Year Reps

The CS Society is hiring year reps and applications are out now! Here is the link to the google form: `https://forms.gle/Z6fPPCcbCvb6G5ECA`

The form is also on our discord: `https://discord.com/invite/gwgrkgb`

As a CS Society Year Representative, you will be responsible for:

- Actively communicating with CS students to determine ways to improve student life.
- Attending weekly meetings, and relaying important information to your peers.
- Attending (some super fun) events, or planning some of your own!
- Being the voice for your year!

Applications are due on Friday, 24th September 2021.

Leibniz’s Rule as an Axiom

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

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

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**

Leibniz’s Rule as an Axiom — Examples

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

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

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!

— Allmost all applications are via derived **“Replacement”** theorems

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

(3.84a) **“Replacement”:** $(e = f) \wedge P[z := e] \;\equiv\; (e = f) \wedge P[z := f]$

(3.84b) **“Replacement”:** $(e = f) \Rightarrow P[z := e] \;\equiv\; (e = f) \Rightarrow P[z := f]$

(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”:** $(e = f) \wedge P[z := e] \;\equiv\; (e = f) \wedge P[z := f]$

Textbook-style application:

$$\begin{array}{ll} & k = n + 1 \quad \wedge \quad k \cdot (n - 1) = n^2 - 1 \\ = & \langle \text{(3.84a) \textbf{“Replacement”}} \rangle \\ & k = n + 1 \quad \wedge \quad (n + 1) \cdot (n - 1) = n^2 - 1 \end{array}$$

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

$$\begin{array}{ll} & k = n + 1 \quad \wedge \quad k \cdot (n - 1) = n \cdot n - 1 \\ = & \langle \text{Substitution} \rangle \\ & k = n + 1 \quad \wedge \quad (z \cdot (n - 1) = n \cdot n - 1)[z := k] \\ = & \langle \text{(3.84a) \textbf{“Replacement”}} \rangle \\ & k = n + 1 \quad \wedge \quad (z \cdot (n - 1) = n \cdot n - 1)[z := n + 1] \\ = & \langle \text{Substitution} \rangle \\ & k = n + 1 \quad \wedge \quad (n + 1) \cdot (n - 1) = n \cdot n - 1 \end{array}$$

Some Replacements

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

$\equiv \langle\ \ ?\ \ \rangle$

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

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

$\equiv \langle\ \ ?\ \ \rangle$

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

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

$\equiv \langle\ \ ?\ \ \rangle$

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

Replacements 1 & 2

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

$\equiv \langle$ (3.51) **"Replacement"** $(p \equiv q) \wedge (r \equiv p) \ \equiv\ (p \equiv q) \wedge (r \equiv q)$ $\rangle$

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

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

$\equiv \langle$ Substitution $\rangle$

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

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

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

Replacements 1 & 2 in CalcCheck

```
Calculation:
    ((x > f 5) ≡ (y < g 7)) ∧ ((f x ≤ g y) ≡ (x > f 5))
  ≡⟨ “Replacement” ⟩
    ((x > f 5) ≡ (y < g 7)) ∧ ((f x ≤ g y) ≡ (y < g 7))

Calculation:
    ((f 5) = (g y)) ∧ ((f x ≤ g y) ≡ (x > f 5))
  ≡⟨ Substitution ⟩
    ((f 5) = (g y)) ∧ ((f x ≤ g y) ≡ (x > z))[z ≔ f 5]
  ≡⟨ “Replacement”, Substitution ⟩
    ((f 5) = (g y)) ∧ ((f x ≤ g y) ≡ (x > g y))
```

Replacement 3

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

$\equiv \langle$ Substitution $\rangle$

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

$\equiv \langle$ (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 $\rangle$

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

In CalcCheck, ≡ does not match =!

Explicit conversions using "Definition of ≡" are necessary.

Leibniz's Rule Axiom, and Further Replacement Rules

Axiom scheme (E can be any expression; z can be of any type):

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

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

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

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

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

(Below, p and z are **of type** $\mathbb{B}$)

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

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.

"Prove the following theorems (**without using Shannon or the proof method of case analysis by Shannon**), ..."

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-23

Part 2: $\sum$ and $\prod$ Quantification,

Quantification expansion

Counting Integral Points

How many integral points are in the triangle

$$\begin{array}{ccc} (0,n) & & \\ | & \diagdown & \\ (0,0) & — & (n,0) \end{array}$$

?

$\sum_{x=0}^{n} (n - x + 1)$

$= \langle$ Summing 1 values $\rangle$

$\sum_{x=0}^{n} (\sum_{y=0}^{n-x} 1)$

$= \langle$ Switch to LADM notation $\rangle$

$(\sum x \mid 0 \le x \le n \bullet (\sum y \mid 0 \le y \le n - x \bullet 1))$

$= \langle$ Nesting $\rangle$

$(\sum x, y \mid 0 \le x \le n \wedge 0 \le y \le n - x \bullet 1)$

$= \langle$ Isotonicity of + $\rangle$

$(\sum x, y \mid 0 \le x \le n \wedge x \le x + y \le n \bullet 1)$

$= \langle$ Def. of ⇒ (3.60) with Transitivity of ≤ $\rangle$

$(\sum x, y \mid 0 \le x \le x + y \le n \bullet 1)$

$= \langle$ Switching to ℕ, and 0 is the least natural number $\rangle$

$(\sum x, y : \mathbb{N} \mid x + y \le n \bullet 1)$

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"

  $\mathit{factorial} : \mathbb{N} \to \mathbb{N}$

  $\mathit{factorial}\ n \;=\; (\prod k : \mathbb{N} \mid 0 < k \le n \bullet k)$

- "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:

$$(\sum i \mid 1 \le i \le n : e) \qquad \text{or} \qquad (+ i \mid 1 \le i \le n : e)$$

We use a variant with a "spot" "•" instead of the colon ":" and only use "big" operators:

$$(\sum i \mid 1 \le i \le n \bullet e)$$

Reasons for using this kind of **linear** quantification notation:

- Clearly delimited introduction of **quantified variables** (**dummies**)
- **Arbitrary** Boolean expressions can define the **range**
  $(\sum i \mid 1 \le i \le 7 \wedge \mathit{even}\ i \bullet i) = 2 + 4 + 6$
- The notation extends easily to multiple quantified variables:
  $(\sum i, j : \mathbb{Z} \mid 1 \le i < j \le 4 \bullet i/j) = 1/2 + 1/3 + 1/4 + 2/3 + 2/4 + 3/4$

Expanding Sum and Product Quantification

Sum quantification ($\sum$) is **"addition (+) of arbitrarily many terms"**:

$(\sum i \mid 5 \le i < 9 \bullet i \cdot (i + 1))$

$= \langle$ Quantification expansion $\rangle$

$(i \cdot (i+1))[i := 5] \;+\; (i \cdot (i+1))[i := 6] \;+\; (i \cdot (i+1))[i := 7] \;+\; (i \cdot (i+1))[i := 8]$

$= \langle$ Substitution $\rangle$

$5 \cdot (5 + 1) \;+\; 6 \cdot (6 + 1) \;+\; 7 \cdot (7 + 1) \;+\; 8 \cdot (8 + 1)$

Product quantification ($\prod$) is **"multiplication ($\cdot$) of arbitrarily many factors"**:

$(\prod i \mid 0 \le i < 3 \bullet 5 \cdot i + 1)$

$= \langle$ Quantification expansion $\rangle$

$(5 \cdot i + 1)[i := 0] \;\cdot\; (5 \cdot i + 1)[i := 1] \;\cdot\; (5 \cdot i + 1)[i := 2]$

$= \langle$ Substitution $\rangle$

$(5 \cdot 0 + 1) \;\cdot\; (5 \cdot 1 + 1) \;\cdot\; (5 \cdot 2 + 1)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-23

Part 3: Natural Induction — Recap.

Proving "Even double"

```
Theorem "Even double": even (n + n)
Proof:
  By induction on `n : ℕ`:
    Base case:
        even (0 + 0)
      ≡⟨ ? ⟩
        ?
    Induction step:
        even (suc n + suc n)
      ≡⟨ ? ⟩
        ?
```

```
"Zero is even":                 even 0
"Even successor (direct)":      even (suc n) ≡ ¬ (even n)
"Definition of + for 0":               0 + n = n
"Definition of + for `suc`":    (suc m) + n = suc (m + n)
```

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 $P[m := \mathit{suc}\ m]$
        using **Induction hypothesis** P

Proving "Even double"

```
Theorem "Even double": even (n + n)
Proof:
  By induction on `n : ℕ`:
    Base case:
        even (0 + 0)
      ≡⟨ "Definition of + for 0" ⟩
        even 0
      ≡⟨ "Zero is even" ⟩
        true
    Induction step:
        even (suc n + suc n)
      ≡⟨ "Definition of + for `suc`" ⟩
        even (suc (n + suc n))
      ≡⟨ "Even successor" ⟩
        odd (n + suc n)
      ≡⟨ "Adding the successor" ⟩
        odd (suc (n + n))
      ≡⟨ "Odd successor" ⟩
        even (n + n)
      ≡⟨ Induction hypothesis ⟩
        true
```

```
"Zero is even":                 even 0
"Even successor (direct)":      even (suc n) ≡ ¬ (even n)
"Definition of + for 0":               0 + n = n
"Definition of + for `suc`":    (suc m) + n = suc (m + n)
```

Proving "Even double" — Using "— This is …"

```
Theorem "Even double": even (n + n)
Proof:
  By induction on `n : ℕ`:
    Base case:
        even (0 + 0)
      ≡⟨ "Definition of + for 0" ⟩
        even 0             — This is "Zero is even"
    Induction step:
        even (suc n + suc n)
      ≡⟨ "Definition of + for `suc`" ⟩
        even (suc (n + suc n))
      ≡⟨ "Even successor" ⟩
        odd (n + suc n)
      ≡⟨ "Adding the successor" ⟩
        odd (suc (n + n))
      ≡⟨ "Odd successor" ⟩
        even (n + n)       — This is induction hypothesis
```

Proving "Even double" — With Explicit Details

```
Theorem "Even double": even (n + n)
Proof:
  By induction on `n : ℕ`:
    Base case `even (0 + 0)`:
        even (0 + 0)
      ≡⟨ "Definition of + for 0" ⟩
        even 0             — This is "Zero is even"
    Induction step `even (suc n + suc n)`:
        even (suc n + suc n)
      ≡⟨ "Definition of + for `suc`" ⟩
        even (suc (n + suc n))
      ≡⟨ "Even successor" ⟩
        odd (n + suc n)
      ≡⟨ "Adding the successor" ⟩
        odd (suc (n + n))
      ≡⟨ "Odd successor" ⟩
        even (n + n)
      — This is induction hypothesis `even (n + n)`
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-27

Part 1: Transitivity Calculations, Monotonicity

Plan for Today

- LADM Chapter 4: "Relaxing the Proof Style" — Introducing Structured Proofs
  - extending the calculational proof format to transitive operators
  - Monotonicity
  - Resolving antecedents of used implications using `with`

?

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

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

Calculational Proof Format

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

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

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

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

$$E_0 = E_3$$

Calculational Proof Format

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

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

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

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

$$E_0 \leq E_3$$

Calculational Proof Format

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

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

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

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

$$E_0 \leq E_3$$

Calculational Proof Format

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

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

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

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

$$E_0 \Rightarrow E_3$$

Calculational Proof Format

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

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 Proof Format

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

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 only for equality

Example Application of "Monotonicity of –"

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

---

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

```
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) $\dfrac{P \Rightarrow Q \qquad P}{Q}$ $\Rightarrow$-Elim $\qquad \dfrac{f : A \to B \qquad x : A}{(f\ x) : B}$ 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 –": $a \le b \;\Rightarrow\; a - c \le b - c$

```
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 **antitone in the second argument**:
  $x \le y \quad \Rightarrow \quad z - y \le z - x$ is a theorem

---

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

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

Multiplication on ℕ is Monotonic…

Calculation:
42
$=$ ⟨ Evaluation ⟩
$6 \cdot 7$
$=$ ⟨ Evaluation ⟩
$(10 - 4) \cdot (12 - 5)$
$\le$ ⟨ "Monotonicity of $\cdot$" with
  "Antitonicity of –" with Fact \`$3 \le 4$\`
⟩
$(10 - 3) \cdot (12 - 5)$
$\le$ ⟨ "Monotonicity of $\cdot$" with
  "Antitonicity of –" with Fact \`$4 \le 5$\`
⟩
$(10 - 3) \cdot (12 - 4)$
$=$ ⟨ Evaluation ⟩
$7 \cdot 8$

with$_2$ Works Also With ≡ — Example Using "Isotonicity of +"

- $\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is isotone in the first argument:
  $x \le y \quad \equiv \quad x + z \le y + z$ 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 +"
```

Lectures, Homework, Exercises, Assignments

- **Lectures** iuntroduce new material
  Just like in in-person lectures, you can raise your hand and ask questions
- **Homework** takes up the new material from the lecture.
  Intended for "hands-on reading" Intended for reading and practicing for **retaining**
- **Exercises** are discussed (selectively) in tutorials
  — after possible homework covering that new material
- **Assignments** follow on after exercises have been discussed in tutorials.
  (While there are assignments, most homework will be short.)
- **You always need everything that came before!**

How would you do Homework without CalcCheck?

Seen on the "Course Help" channel:

Calculation:
$\sum k, n : \mathbb{N} \mid 3 \le k < 5 \wedge 4 \le n < 6 \bullet k \cdot n$
$=$ ⟨ Quantification expansion ⟩
$(k \cdot n)\,[k, n := 3, 4]$
$+ (k \cdot n)\,[k, n := 4, 5]$
$=$ ⟨ Substitution, Evaluation ⟩
32

Without CalcCheck, probably:

- This looks good enough; submit.
- Notice lost marks when the homework is returned.

With CalcCheck:

- Notice that there is a problem right away.
- Alternatives:
  1. Work towards figuring out the problem. (This may involve asking on "Course Help"…)
  2. Decide that this is good enough for submitting — *pen-and-paper compatibility mode*
  3. Anything in-between…

It is OK to submit homework/assignments that are not 100% correct!

What Was The Problem Anyways?

From H6.3: $\sum k, n : \mathbb{N} \mid 3 \le k < 5 \wedge 4 \le n < 6 \bullet k \cdot n$

For each state for all quantified variables, where that state satisfies the range predicate, add up the corresponding substitution instance of the body.

The states for k, n satisfying the range predicate $3 \le k < 5 \wedge 4 \le n < 6$ are:

- $[\langle k, 3\rangle, \langle n, 4\rangle]$
- $[\langle k, 3\rangle, \langle n, 5\rangle]$
- $[\langle k, 4\rangle, \langle n, 4\rangle]$
- $[\langle k, 4\rangle, \langle n, 5\rangle]$

…corresponding substitution instances of the body:

Calculation:
$\sum k, n : \mathbb{N} \mid 3 \le k < 5 \wedge 4 \le n < 6 \bullet k \cdot n$
$=$ ⟨ Quantification expansion ⟩
$(k \cdot n)\,[k, n := 3, 4]$
$+ (k \cdot n)\,[k, n := 3, 5]$
$+ (k \cdot n)\,[k, n := 4, 4]$
$+ (k \cdot n)\,[k, n := 4, 5]$
$=$ ⟨ Substitution, Evaluation ⟩
63

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-27

Part 2: Subproofs, Assuming, …

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:

```
Calculation:
    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: $\boxed{p \Rightarrow r}$

Because:

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

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

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

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

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

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 such 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)

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

$$
\begin{array}{ll}
 & q \Rightarrow p \\
\equiv & \langle \text{(3.59) Definition of implication} \rangle \\
 & \neg q \vee p \\
\Leftarrow & \langle \text{(3.76a) Strenghtening — used as } p \vee q \Leftarrow p \rangle \\
 & p
\end{array}
$$

In CalcCheck, if the **converse property** is not **activated**, then $\Leftarrow$ is a separate operator requiring explicit conversion:

```
Theorem (4.1): p ⇒ (q ⇒ p)
Proof:
    q ⇒ p
  ≡⟨ “Definition of ⇒” (3.59) ⟩
    ¬ q ∨ p
  ⇐⟨ “Strengthening” (3.76a), “Definition of ⇐” ⟩
    p
```

(4.1) Implicitly Using "Consequence"

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

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

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

Recall: Weakening/Strengthening Theorems

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

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

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

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

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

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-28

Part 1: Assuming the Antecedent

Plan for Today

- **Textbook Chapter 4: "Relaxing the Proof Style"** — **New Proof Structures**
  - Proving implications: **Assuming** the antecedent
  - Proving **By cases**
  - **Using** theorems as proof methods
    - Proof by Contrapositive
    - Proof by Mutual Implication
- **Universal and Existential Quantification**

(4.2) Left-Monotonicity of $\vee$

$$\boxed{(p \Rightarrow q) \;\Rightarrow\; (p \vee r \Rightarrow q \vee r)}$$

$$
\begin{array}{ll}
 & p \vee r \Rightarrow q \vee r \\
\equiv & \langle \text{(3.57) Definition of } \Rightarrow \quad p \Rightarrow q \;\equiv\; p \vee q \;\equiv\; q \rangle \\
 & p \vee r \vee q \vee r \quad \equiv \quad q \vee r \\
\equiv & \langle \text{(3.26) Idempotency of } \vee \rangle \\
 & p \vee q \vee r \quad \equiv \quad q \vee r \\
\equiv & \langle \text{(3.27) Distributivity of } \vee \text{ over } \equiv \rangle \\
 & (p \vee q \quad \equiv \quad q) \vee r \\
\equiv & \langle \text{(3.57) Definition of } \Rightarrow \quad p \Rightarrow q \;\equiv\; p \vee q \;\equiv\; q \rangle \\
 & (p \Rightarrow q) \vee r \\
\Leftarrow & \langle \text{(3.76a) Strengthening } p \Rightarrow p \vee q \rangle \\
 & p \Rightarrow q
\end{array}
$$

(4.3) Left-Monotonicity of $\wedge$

Proving (4.3) $(p \Rightarrow q) \;\Rightarrow\; p \wedge r \;\Rightarrow\; q \wedge r$**:**

$$
\begin{array}{ll}
 & p \wedge r \;\Rightarrow\; q \wedge r \\
\equiv & \langle \text{(3.60) Definition of } \Rightarrow \rangle \\
 & p \wedge r \wedge q \wedge r \equiv p \wedge r \\
\equiv & \langle \text{(3.38) Idempotency of } \wedge \rangle \\
 & (p \wedge q) \wedge r \equiv p \wedge r \\
\equiv & \langle \text{(3.49) Semi-distributivity of } \wedge \rangle \\
 & ((p \wedge q) \equiv p) \wedge r \equiv r \\
\equiv & \langle \text{(3.60) Definition of } \Rightarrow \rangle \\
 & (p \Rightarrow q) \wedge r \equiv r \\
\equiv & \langle \text{(3.60) Definition of } \Rightarrow \rangle \\
 & r \Rightarrow (p \Rightarrow q) \\
\Leftarrow & \langle \text{(4.1) } p \Rightarrow (q \Rightarrow p) \rangle \\
 & p \Rightarrow q
\end{array}
$$

Proving Implications...

How to prove the following?

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

"We have been doing this via Leibniz (1.5)... ..."

- One of the "Replacement" theorems of the "Leibniz as Axiom" section can help.
- It may be nicer to turn this into a situation where the inference rule Leibniz (1.5) can be used again...

Assuming the Antecedent:

```
Lemma “=-Congruence of +”:  b = c  ⇒  a + b = a + c
Proof:
  Assuming `b = c`:
      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 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 LADM Chapter 3!

Inference Rule for Proving Implications: ⇒-Introduction

One way to prove $P \Rightarrow Q$:

> **Assuming** \`$P$\`:
>
> > *Proof of* Q
> > *possibly using:* Assumption \`$P$\`

(And $\boxed{\textbf{Assuming } `P`: \ldots}$ 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):

$$\dfrac{\begin{matrix}\ulcorner P \urcorner \\ \vdots \\ Q\end{matrix}}{P \Rightarrow Q}\ \Rightarrow\text{-Intro} \qquad\qquad \dfrac{\begin{matrix}\ulcorner x : A \urcorner \\ \vdots \\ e : B\end{matrix}}{(\lambda x : A \bullet e) : A \to B}\ \lambda\text{-Abstraction}$$

Proving and Using Implication Theorems: `Assuming` and `with`$_2$

"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 \`$a \neq 0$\` , \`$b \neq 0$\`**:**

$a \cdot b \neq 0$
≡⟨ "Definition of ≠" ⟩
$\neg (a \cdot b = 0)$
≡⟨ "Zero of ·" ⟩
$\neg (a \cdot b = a \cdot 0)$
≡⟨ "Cancellation of ·" with Assumption \`$a \neq 0$\` ⟩
$\neg (b = 0)$
≡⟨ "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)

(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

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

(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 ⟩
$q \wedge r$

```
Theorem (4.3) “Left-monotonicity of ∧”: (p ⇒ q) ⇒ (p ∧ r ⇒ q ∧ r)
Proof:
  Assuming `p ⇒ q`:
      p ∧ r
    ≡⟨ Assumption `p ⇒ q` with “Definition of ⇒” (3.60) ⟩
      p ∧ q ∧ r
    ⇒⟨ “Weakening” ⟩
      q ∧ r
```

Assuming ... and using with ...

(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 ⟩
$q \wedge r$

```
Theorem (4.3) “Left-monotonicity of ∧” “Monotonicity of ∧”:
    (p ⇒ q) ⇒ ((p ∧ r) ⇒ (q ∧ r))
Proof:
  Assuming `p ⇒ q` and using with “Definition of ⇒” (3.60):
      p ∧ r
    ≡⟨ Assumption `p ⇒ q` ⟩
      p ∧ q ∧ r
    ⇒⟨ “Weakening” (3.76b) ⟩
      q ∧ r
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-28

Part 2: Case Analysis and Other Proof Methods

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 $R \Rightarrow S$)

LADM Case Analysis Example: (4.2) $(p \Rightarrow q) \Rightarrow p \vee r \Rightarrow q \vee r$

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) ⟩
$q \vee r$

CalcCheck By cases with "Zero or successor of predecessor": $n = 0 \vee n = suc\ (pred\ n)$

```
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`:
        m - 0 = m
      ≡⟨ Assumption `m = 0` ⟩
        0 - 0 = 0
      — This is "Subtraction from zero"
    Case `m = suc (pred m)`:
        m - 0
      =⟨ Assumption `m = suc (pred m)` ⟩
        (suc (pred m)) - 0
      =⟨ "Subtraction of zero from successor" ⟩
        suc (pred m)
      =⟨ Assumption `m = suc (pred m)` ⟩
        m
```

Case Analysis with Calculation for "Completeness:" . . .

```
By cases: `pos b`, `¬ pos b`
  Completeness:
      pos b ∨ ¬ pos b
    ≡⟨ "Excluded Middle" ⟩
      true
  Case `pos b`:
    By (15.31a) with Assumption `pos b`
```

- After "**Completeness:**" goes a proof for the disjunction of all cases listed after "**By cases:**"
- This can be any kind of proof.
- Inside the "**Case '*p*':**" block, you may use "**Assumption '*p*'**"

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 $x + y \geq 2 \quad \Rightarrow \quad x \geq 1 \vee y \geq 1$:

$$
\begin{array}{ll}
 & \neg(x \geq 1 \vee y \geq 1) \\
\equiv & \langle \text{ De Morgan (3.47) } \rangle \\
 & \neg(x \geq 1) \wedge \neg(y \geq 1) \\
\equiv & \langle \text{ Def. } \geq \text{ (15.39) with Trichotomy (15.44) } \rangle \\
 & x < 1 \wedge y < 1 \\
\Rightarrow & \langle \text{ Monotonicity of + (15.42) } \rangle \\
 & x + y < 1 + 1 \\
\equiv & \langle \text{ Def. 2 } \rangle \\
 & x + y < 2 \\
\equiv & \langle \text{ Def. } \geq \text{ (15.39) with Trichotomy (15.44) } \rangle \\
 & \neg(x + y \geq 2)
\end{array}
$$

Proof by Contrapositive in CalcCheck — `Using`

```
Theorem "Example for use of Contrapositive": x + y ≥ 2 ⇒ x ≥ 1 ∨ y ≥ 1
Proof:
  Using "Contrapositive":
    Subproof for `¬ (x ≥ 1 ∨ y ≥ 1) ⇒ ¬ (x + y ≥2)`:
        ¬ (x ≥ 1 ∨ y ≥ 1)
      ≡⟨ "De Morgan" ⟩
        ¬ (x ≥ 1) ∧ ¬ (y ≥ 1)
      ≡⟨ "Complement of <" with (3.14) ⟩
        x < 1 ∧ y < 1
      ⇒⟨ "<-Monotonicity of +" ⟩
        x + y < 1 + 1
      ≡⟨ Evaluation ⟩
        x + y < 2
      ≡⟨ "Complement of <" with (3.14) ⟩
        ¬ (x + y ≥ 2)
```

- "**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. . .

Proof by Mutual Implication — `Using`

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

```
Theorem (15.44A) "Trichotomy — A":
    a < b  ≡  a = b  ≡  a > b
Proof:
  Using "Mutual implication":
    Subproof for `a = b  ⇒  (a < b  ≡  a > b)`:
      Assuming `a = b`:
          a < b
        ≡⟨ "Converse of <", Assumption `a = b` ⟩
          a > b
    Subproof for `(a < b  ≡  a > b) ⇒ a = b`:
        a < b  ≡  a > b
      ≡⟨ "Definition of <", "Definition of >" ⟩
        pos (b - a) ≡ pos (a - b)
      ≡⟨ (15.17), (15.19), "Subtraction" ⟩
        pos (b - a) ≡ pos (- (b - a))
      ⇒⟨ (15.33c) ⟩
        b - a = 0
      ≡⟨ "Cancellation of +" ⟩
        b - a + a = 0 + a
      ≡⟨ "Identity of +", "Subtraction", "Unary minus" ⟩
        a = b
```

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"

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.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-28

Part 3: Universal and Existential Quantification

Formalise:

- Distributivity of addition over multiplication does not hold.

$k + (m \cdot n) \; \neq \; (k + m) \cdot (k + n)$???

Universal and Existential Quantification

$(\forall x \bullet P)$

- "For all x, we have P"

$(\forall x \mid R \bullet P)$

- "For all x with R, we have P"

$(\exists x \bullet P)$

- "There exists an x such that P (holds)"
- "For some x, we have P"

$(\exists x \mid R \bullet P)$

- "There exists an x with R such that P (holds)"
- "For some x with R, we have P"

Formalise:

- Distributivity of addition over multiplication does not hold.

$(k + (m \cdot n) \; \neq \; (k + m) \cdot (k + n))[k, m, n := ?, ?, ?]$

$\exists\, k, m, n : \mathbb{N} \bullet k + (m \cdot n) \; \neq \; (k + m) \cdot (k + n)$

$\exists\, k, m, n : \mathbb{N} \bullet \neg(k + (m \cdot n) \; = \; (k + m) \cdot (k + n))$

$\neg(\forall\, k, m, n : \mathbb{N} \bullet k + (m \cdot n) \; = \; (k + m) \cdot (k + n))$

Expanding Universal and Existential Quantification

Universal quantification ($\forall$) is **"conjunction ($\wedge$) with arbitrarily many conjuncts"**:

$$(\forall\, i \mid 1 \le i < 3 \bullet i \cdot d \ne 6)$$

$=$ ⟨ Quantification expansion, substitution ⟩

$$1 \cdot d \ne 6 \quad \wedge \quad 2 \cdot d \ne 6$$

Existential quantification ($\exists$) is **"disjunction ($\vee$) with arbitrarily many disjuncts"**:

$$(\exists\, i \mid 0 \le i < 21 \bullet b[i] = 0)$$

$=$ ⟨ Quantification expansion, substitution ⟩

$$b[0] = 0 \quad \vee \quad b[1] = 0 \quad \vee \quad \ldots \quad \vee \quad b[20] = 0$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-30

Part 1: More About the Natural Numbers

Plan for Today

- More About the Natural Numbers
- More About Command Correctness

- **Next Week: Quantification**

Midterm 1, Tuesday Oct. 5, 13:30–14:20, ONLINE

The main emphasis of M1 will be on:

- Propositional calculus, LADM chapter 3 (Ex2.7, Ex3.2, Ex3.3, H4, H5.2, Ex3.4, A2.1, H6.1, (Ex4.2))
- Natural numbers and induction proofs (H5.1, Ex3.5, A2.2, Ex4.1, Ex4.6)

Additionally, the following can occur in M1:

- Correctness proofs (H3.1, Ex2.6, A1.3, H8)
- Quantification expansion (H6.2, H6.3, Ex4.5)
- Monotonicity (H7, Ex4.3)
- Integers (Ex2, A1.1, A1.2)

— (No promise that the will be a correctness proof on M1.)
— (No promise that the won't be a correctness proof on M1.)

Topics can be combined.

Multiple-choice questions can occur.

M1 will be written without proof checking (but with syntax checking).

- Limited to things you are expected to confidently get right.

FYI: I never answer "How many questions will there be on the test?".

The Predecessor Function pred on $\mathbb{N}$

The "predecessor function" pred is total; since zero has no predecessor, it maps 0 to 0.

> **Declaration**: pred : $\mathbb{N} \to \mathbb{N}$
> **Axiom** "Predecessor of zero": pred 0 $=$ 0
> **Axiom** "Predecessor of successor": pred (suc n) $=$ n

Whe 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 $m$)\`:

Defining (Monus) Subtraction Inductively

```
Axiom "Subtraction from zero":                  0 -       n  =  0
Axiom "Subtraction of zero from successor":      (suc m) -       0  =  suc m
Axiom "Subtraction of successor from successor": (suc m) - (suc n)  =  m - n
```

Note: $\boxed{2 - 5 = 0}$

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 $suc\ k$ 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.

Defining Subtraction Inductively Using Three Clauses

```
Axiom "Subtraction from zero":                  0 -       n  =  0
Axiom "Subtraction of zero from successor":      (suc m) -       0  =  suc m
Axiom "Subtraction of successor from successor": (suc m) - (suc n)  =  m - n
```

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

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

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-09-30

Part 2: More Command Correctness

Partial Correctness for Pre-Postcondition 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:** $\{\, Q[x := E] \,\}\ x := E\ \{\, Q \,\}$ $\qquad$ $Q[x := E] \Rightarrow [\, x := E \,]\, Q$
- **Sequential composition:**

```
Primitive inference rule "Sequence":
     `P  ⇒[ C₁ ]  Q`,  `Q  ⇒[ C₂ ]  R`
  ⊢———————————————————————————————————
        `P  ⇒[ C₁ ; C₂ ]  R`
```

Command Sequences

Axiom "Assignment": $P[x := E] \Rightarrow [\, x := E \,]\, P$

Primitive inference rule "Sequence":

$$\vdash \frac{`P \Rightarrow [\, C_1 \,]\, Q`, \quad `Q \Rightarrow [\, C_2 \,]\, R`}{`P \Rightarrow [\, C_1 ; C_2 \,]\, R`}$$

Fact: $x = 5 \quad \Rightarrow[\ y := x + 1 ; x := y + y\ ]\ \ x = 12$

Proof:

$x = 5$
$\equiv \langle$ "Cancellation of +" $\rangle$
$x + 1 = 5 + 1$
$\equiv \langle$ Fact `$5 + 1 = 6$` $\rangle$
$x + 1 = 6$
$\equiv \langle$ Substitution $\rangle$
$(y = 6)[y := x + 1]$
$\Rightarrow[\ y := x + 1\ ]\ \langle$ "Assignment" $\rangle$
$y = 6$
$\equiv \langle$ "Cancellation of ·" with Fact `$2 \neq 0$` $\rangle$
$2 \cdot y = 2 \cdot 6$
$\equiv \langle$ Evaluation $\rangle$
$(1 + 1) \cdot y = 12$
$\equiv \langle$ "Distributivity of · over +" $\rangle$
$1 \cdot y + 1 \cdot y = 12$
$\equiv \langle$ "Identity of ·" $\rangle$
$y + y = 12$
$\equiv \langle$ Substitution $\rangle$
$(x = 12)[x := y + y]$

Using converse operator for backward presentation:

$\_[\_] \Leftarrow \_$

Fact: $x = 5 \Rightarrow[\ (y := x + 1 ; x := y + y)\ ]\ x = 12$

Proof:

$x = 12$
$[\ x := y + y\ ] \Leftarrow \langle$ "Assignment" with Substitution $\rangle$
$y + y = 12$
$\equiv \langle$ "Identity of ·" $\rangle$
$1 \cdot y + 1 \cdot y = 12$
$\equiv \langle$ "Distributivity of · over +" $\rangle$
$(1 + 1) \cdot y = 12$
$\equiv \langle$ Evaluation $\rangle$
$2 \cdot y = 2 \cdot 6$
$\equiv \langle$ "Cancellation of ·" with Fact `$2 \neq 0$` $\rangle$
$y = 6$
$[\ y := x + 1\ ] \Leftarrow \langle$ "Assignment" with Substitution $\rangle$
$x + 1 = 6$
$\equiv \langle$ Fact `$5 + 1 = 6$` $\rangle$
$x + 1 = 5 + 1$
$\equiv \langle$ "Cancellation of +" $\rangle$
$x = 5$

Transitivity Rules for Calculational Command Correctness Reasoning

Primitive inference rule "Sequence":

$$\vdash \frac{`P \Rightarrow[\ C_1\ ]\ Q`, \qquad `Q \Rightarrow[\ C_2\ ]\ R`}{`P \Rightarrow[\ C_1 ; C_2\ ]\ R`}$$

Strengthening the precondition:

$$\vdash \frac{`P_1 \Rightarrow P_2`, \quad `P_2 \Rightarrow[\ C\ ]\ Q`}{`P_1 \Rightarrow[\ C\ ]\ Q`}$$

Weakening the postcondition:

$$\vdash \frac{`P \Rightarrow[\ C\ ]\ Q_1`, \quad `Q_1 \Rightarrow Q_2`}{`P \Rightarrow[\ C\ ]\ Q_2`}$$

P
$\Rightarrow[\ C_1\ ]\ \langle \ldots \rangle$
Q
$\Rightarrow \quad \langle \ldots \rangle$
Q'
$\Rightarrow[\ C_2\ ]\ \langle \ldots \rangle$
R

- Activated as transitivity rules
- Therefore used implicitly in calculations, e.g., proving $P \Rightarrow[\ C_1 ⨾ C_2\ ]\ R$ to the right

---

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":

$$\vdash \frac{`B \wedge P \Rightarrow[\ C_1\ ]\ Q`, \qquad `\neg B \wedge P \Rightarrow[\ C_2\ ]\ Q`}{`P \Rightarrow[\ \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi}\ ]\ Q`}$$

```
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
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-04

Part 1: Order on Integers via Positivity

Plan for Today

- Order on Integers via Positivity (LADM chapter 15, pp. 307–308)
  $\Longrightarrow$ Opportunities for structured proofs
- Quantification laws at the example of $\sum$
- **Thursday:** General quantification, LADM chapter 8

Midterm 1, Tuesday Oct. 5, 13:30–14:20, ONLINE

The main emphasis of M1 will be on:

- Propositional calculus, LADM chapter 3 (Ex2.7, Ex3.2, Ex3.3, H4, H5.2, Ex3.4, A2.1, H6.1, (Ex4.2))
- Natural numbers and induction proofs (H5.1, Ex3.5, A2.2, Ex4.1, Ex4.6)

Additionally, the following can occur in M1:

- Correctness proofs (H3.1, Ex2.6, A1.3, H8)
- Quantification expansion (H6.2, H6.3, Ex4.5)
- Monotonicity (H7, Ex4.3)
- Integers (Ex2, A1.1, A1.2)

— (No promise that the will be a correctness proof on M1.)
— (No promise that the won't be a correctness proof on M1.)

Topics can be combined.

Multiple-choice questions can occur.

M1 will be written without proof checking (but with syntax checking).

- Limited to things you are expected to confidently get right.

FYI: I never answer "How many questions will there be on the test?".

LADM Theory of Integers — Positivity and Ordering

(15.30) **Axiom, Addition in** *pos*: $\quad pos\ a \wedge pos\ b \Rightarrow pos\ (a + b)$

(15.31) **Axiom, Multiplication in** *pos*: $\quad pos\ a \wedge pos\ b \Rightarrow pos\ (a \cdot b)$

(15.32) **Axiom:** $\quad \neg\, pos\ 0$

(15.33) **Axiom:** $\quad b \neq 0 \quad \Rightarrow \quad (pos\ b \ \equiv\ \neg pos\ (-b))$

(15.34) **Positivity of Squares:** $\quad b \neq 0 \quad \Rightarrow \quad pos\ (b \cdot b)$

(15.35) $\quad pos\ a \quad \Rightarrow \quad (pos\ b \quad \equiv \quad pos\ (a \cdot b))$

(15.36) **Axiom, Less:** $\quad a < b \quad \equiv \quad pos\ (b - a)$

(15.37) **Axiom, Greater:** $\quad a > b \quad \equiv \quad pos\ (a - b)$

(15.38) **Axiom, At most:** $\quad a \leq b \quad \equiv \quad a < b \vee a = b$

(15.39) **Axiom, At least:** $\quad a \geq b \quad \equiv \quad a > b \vee a = b$

(15.40) **Positive elements:** $\quad pos\ b \quad \equiv \quad 0 < b$

LADM Theory of Integers — Ordering Properties

(15.41) **Transitivity:** (a) $a < b \;\wedge\; b < c \;\Rightarrow\; a < c$
(b) $a \le b \;\wedge\; b < c \;\Rightarrow\; a < c$
(c) $a < b \;\wedge\; b \le c \;\Rightarrow\; a < c$
(d) $a \le b \;\wedge\; b \le c \;\Rightarrow\; a \le c$

(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 ≤:** $a \le b \;\wedge\; a \ge b \;\equiv\; a = b$

(15.46) **Reflexivity of ≤:** $a \le a$

Structured Proof Example from LADM

Theorems for *pos*
(15.34) $b \ne 0 \Rightarrow pos(b \cdot b)$

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 $\neg pos.b \wedge b \ne 0$. We have the following.

$pos(b \cdot b)$
= ⟨(15.23), with $a, b := b, b$⟩
$pos((-b) \cdot (-b))$
⇐ ⟨Multiplication (15.31)⟩
$pos(-b) \wedge pos(-b)$
= ⟨Idempotency of ∧ (3.38)⟩
$pos(-b)$
= ⟨Double negation (3.12) —note that $b \ne 0$; (15.33)⟩
$\neg pos.b$ —the case under consideration

The Same Proof in CalcCheck

Theorem (15.34) “Positivity of squares”: $b \ne 0 \Rightarrow$ pos $(b \cdot b)$
Proof:
  Assuming \`$b \ne 0$\`:
    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 $(b \cdot b)$
        ≡ ⟨ (15.23) \`$-a \cdot -b = a \cdot b$\` ⟩
          pos $((-b) \cdot (-b))$
        ⇐ ⟨ “Positivity under ·” (15.31) ⟩
          pos $(-b)$ ∧ pos $(-b)$
        ≡ ⟨ “Idempotency of ∧”, “Double negation” ⟩
          ¬ ¬ pos $(-b)$
        ≡ ⟨ “Positivity under unary minus” (15.33) with assumption \`$b \ne 0$\` ⟩
          ¬ pos b **— This is** assumption \`¬ pos $b$\`

Case Analysis with Calculation for “Completeness:” . . .

By cases: \`pos b\`, \`¬ pos b\`
  Completeness:
      pos b ∨ ¬ pos b
    ≡⟨ “Excluded Middle” ⟩
      true
  Case \`pos b\`**:**
    By (15.31a) with Assumption \`pos b\`

- After “**Completeness:**” goes a proof for the disjunction of all cases listed after “**By cases:**”
- This can be any kind of proof.
- Inside the “**Case ‘*p*’:**” block, you may use “**Assumption ‘*p*’**”

The CalcCheck Language — Calculational Proofs on Steroids

- LADM emphasises use of axioms and theorems in calculations over other inference rules

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!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-04

Part 2: Quantification, Variable Binding

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:

$$(\sum i \mid 1 \le i \le n : e) \qquad \text{or} \qquad (+ i \mid 1 \le i \le n : e)$$

We use a variant with a “spot” “•” instead of the colon “:” and only use “big” operators:

$$(\sum i \mid 1 \le i \le n \bullet e)$$

Reasons for using this kind of **linear** quantification notation:

- Clearly delimited introduction of **quantified variables** (**dummies**)
- **Arbitrary** Boolean expressions can define the **range**
  $(\sum i \mid 1 \le i \le 7 \wedge even\ i \bullet i) = 2 + 4 + 6$
- The notation extends easily to multiple quantified variables:
  $(\sum i, j : \mathbb{Z} \mid 1 \le i < j \le 4 \bullet i/j) = 1/2 + 1/3 + 1/4 + 2/3 + 2/4 + 3/4$

Formalise:

- 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!

```
Theorem “Odd-number sum”:
    (∑ i : ℕ ❙ i < n • suc i + i) = n · 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)
      =⟨ ? ⟩



      =⟨ ? ⟩
        0 · 0
    Induction step:
        (∑ i : ℕ ❙ i < suc n • suc i + i)
      =⟨ ? ⟩




      =⟨ ? ⟩
        suc n · suc n
```

Empty Range Axioms

(8.13) **Axiom, Empty Range**:

$$(\sum x \mid false \bullet E) \quad = \quad 0$$

$$(\prod x \mid false \bullet E) \quad = \quad 1$$

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
```

Disjoint Range Split (LADM)

(8.16) **Axiom, Range Split:**

$$(\Sigma x \mid Q \vee R \bullet P) = (\Sigma x \mid Q \bullet P) + (\Sigma x \mid R \bullet P)$$

provided $Q \wedge R = \mathit{false}$ and each sum is defined.

(8.16) **Axiom, Range Split:**

$$(\Pi x \mid Q \vee R \bullet P) = (\Pi x \mid Q \bullet P) \cdot (\Pi x \mid R \bullet P)$$

provided $Q \wedge R = \mathit{false}$ and each product is defined.

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

Axioms for One-element Ranges

(8.14) **Axiom, One-point Rule:** Provided $\neg occurs(‘x’, ‘D’)$,

$$(\Sigma x \mid x = D \bullet E) = E[x := D]$$
$$(\prod x \mid x = D \bullet E) = E[x := D]$$
$$(\forall x \mid x = D \bullet P) = P[x := D]$$
$$(\exists x \mid x = D \bullet P) = P[x := D]$$

Example:

$$\begin{aligned}
& (\Sigma\, i : \mathbb{N} \bullet 5 + 2 \cdot i < 7 \mid 5 + 7 \cdot i) \\
= \; & \langle \ldots \rangle \\
& (\Sigma\, i : \mathbb{N} \bullet i = 0 \mid 5 + 7 \cdot i) \\
= \; & \langle \text{One-point rule} \rangle \\
& (5 + 7 \cdot i)[i := 0] \\
= \; & \langle \text{Substitution} \rangle \\
& 5 + 7 \cdot 0
\end{aligned}$$

Variable Binding is Everywhere!

- Calculus: $f(y) = \int_0^1 x^2 y^2 dx$
- Imperative Programming (here C):

```
int f(int x)
{
  int q;
  q = x * x;
  return 2 * q;
}
```

- Functional Programming (here Haskell):

```
f x = let q = x * x in 2 * q
```

The $\neg occurs$ Proviso for the One-point Rule

(8.14) **Axiom, One-point Rule for $\sum$:** Provided $\neg occurs(‘x’, ‘E’)$,

$$(\Sigma x \mid x = E \bullet P) = P[x := E]$$

(8.14) **Axiom, One-point Rule for $\prod$:** Provided $\neg occurs(‘x’, ‘E’)$,

$$(\prod x \mid x = E \bullet P) = P[x := E]$$

Examples:

- $(\Sigma x \mid x = 1 \bullet x \cdot y) \quad = \quad 1 \cdot y$
- $(\prod x \mid x = y + 1 \bullet x \cdot x) \quad = \quad (y + 1) \cdot (y + 1)$

Counterexamples:

- $(\Sigma x \mid x = x + 1 \bullet x)$ **?** $x + 1$ — **"=" not valid!**
- $(\prod x \mid x = 2 \cdot x \bullet y + x)$ **?** $y + 2 \cdot x$ — **"=" not valid!**

Manipulating Ranges

(8.23) **Theorem Split off term**: For $n : \mathbb{N}$ and dummies $i : \mathbb{N}$,

$$(\Sigma\, i \mid 0 \le i < n + 1 \bullet P) = (\Sigma\, i \mid 0 \le i < n \bullet P) + P[i := n]$$
$$(\Sigma\, i \mid 0 \le i < n + 1 \bullet P) = P[i := 0] + (\Sigma\, i \mid 0 < i < n + 1 \bullet P)$$

- 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:

$m \le n \quad \Rightarrow$
$\qquad (\Sigma\, i \mid m \le i < n + 1 \bullet P) = (\Sigma\, i \mid m \le i < n \bullet P) + P[i := n]$

Theorem: Split off term from bottom:

$m \le n \quad \Rightarrow$
$(\Sigma\, i \mid m \le i < n + 1 \bullet P) = P[i := m] + (\Sigma\, i \mid m + 1 \le i < n + 1 \bullet P)$

Proving Split-off Term

(8.16) **Axiom, Range Split:**

$$(\Sigma x \mid Q \vee R \bullet P) = (\Sigma x \mid Q \bullet P) + (\Sigma x \mid R \bullet P)$$

provided $Q \wedge R = \mathit{false}$ and each sum is defined.

```
Theorem "Split off term" "Split off term at top":
    (∑ i : ℕ ❙ i < suc n • E) = (∑ i : ℕ ❙ i < n • E) + E[i ≔ n]
```

- Use range split first
  $\Longrightarrow$ Need to transform the range expression $i < suc\ n$ into an appropriate disjunction
- The second range will have one element
  $\Longrightarrow$ The second sum has range $i = n$
  $\Longrightarrow$ The second sum disappears via the **one-point rule**

Bound / Free Variable Occurrences

$(\Sigma\, i : \mathbb{N} \mid i < x \bullet i + 1) = 10$ — example expression

Is this true or false? In which states?

We have: $\quad (\Sigma\, i : \mathbb{N} \mid i < x \bullet i + 1) = 10 \quad \equiv \quad x = 4$

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>:

$$(\Sigma\, i : \mathbb{N} \mid i < x \bullet i + 1) = (\Sigma\, j : \mathbb{N} \mid j < x \bullet j + 1)$$

- **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.: $\quad 3 \cdot i + (\Sigma\, i : \mathbb{N} \mid i < x \bullet 2 \cdot i)$
- The variable declarations after the quantification operator
  may be called **binding occurrences**.

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.

$occurs(‘i’, ‘5 \cdot i’)$ ✓

$occurs(‘i’, ‘0 \cdot i’)$ ✓

$occurs(‘i’, ‘5 \cdot k’)$ ×

$occurs(‘i’, ‘(\Sigma\, i \mid 0 \le i < k \bullet n^i)’)$ ×

$occurs(‘n’, ‘(\Sigma\, i \mid 0 \le i < k \bullet n^i)’)$ ✓

$occurs(‘i, n’, ‘(\Sigma\, i \mid 0 \le i < k \bullet n^i)’)$ ✓

$occurs(‘i, n’, ‘(\Sigma\, i, n \mid 1 \le i \cdot n \le k \bullet n^i)’)$ ×

Textual Substitution Revisited

Let E and R be expressions and let x be a variable. **Original definition:**

> We write: $\quad E[x := R] \quad$ or $\quad E^x_R$
> 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 $\neg occurs(‘y’, ‘x, F’)$,

$$(\Sigma\, y \mid R \bullet P)[x := F] = (\Sigma\, y \mid R[x := F] \bullet P[x := F])$$

(8.11) is part of the Substitution keyword in CalcCheck.

Substitution Examples

(8.11) Provided $\neg occurs(\text{'}y\text{'}, \text{'}x, F\text{'})$,

$$(\sum y \mid R \bullet P)[x := F] \quad = \quad (\sum y \mid R[x := F] \bullet P[x := F])$$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[y := y + z]$
  $=$ ⟨ substitution ⟩
  $(\sum x \mid 1 \le x \le 2 \bullet y + z)$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[y := y + x]$
  $=$ ⟨ (8.21) Variable renaming ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y)[y := y + x]$
  $=$ ⟨ substitution ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y + x)$

Substitution Examples (ctd.)

(8.11) Provided $\neg occurs(\text{'}y\text{'}, \text{'}x, F\text{'})$,

$$(\sum y \mid R \bullet P)[x := F] \quad = \quad (\sum y \mid R[x := F] \bullet P[x := F])$$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[x := y + x]$
  $=$ ⟨ (8.21) Variable renaming ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y)[x := y + x]$
  $=$ ⟨ Substitution ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y)$
  $=$ ⟨ (8.21) Variable renaming ⟩
  $(\sum x \mid 1 \le x \le 2 \bullet y)$

(8.11f) Provided $\neg occurs(\text{'}x\text{'}, \text{'}E\text{'})$,

$$E[x := F] \quad = \quad E$$

Renaming of Bound Variables

(8.21) **Axiom, Dummy renaming** (α-conversion):

$$(\sum x \mid R \bullet P) \quad = \quad (\sum y \mid R[x := y] \bullet P[x := y])$$

provided $\neg occurs(\text{'}y\text{'}, \text{'}R, P\text{'})$.

$(\sum i \mid 0 \le i < k \bullet n^i)$
$=$ ⟨ Dummy renaming (8.21), $\neg occurs(\text{'}j\text{'}, \text{'}0 \le i < k,\ n^i\text{'})$ ⟩
$(\sum j \mid 0 \le j < k \bullet n^j)$

$(\sum i \mid 0 \le i < k \bullet n^i)$
? ⟨ Dummy renaming (8.21) ⟩ ×
$(\sum k \mid 0 \le k < k \bullet n^k)$

In CalcCheck, renaming of bound variables is part of "Reflexivity of =",
but can also be mentioned explicitly.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-07

Part 1: with$_3$: Rewriting Theorems Using Equations

Plan for Today

- with$_3$: Rewriting Theorems Using Equations
- **General Quantification (LADM chapter 8) — Variable Binding**
- **Predicate Logic 1**:
  Axioms and Theorems about Universal and Existential Quantification
  (LADM chapter 9)

`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
- "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.

Using hi_1: sp_1 sp_2	is essentially syntactic sugar for:	By hi_1 with sp_1 and sp_2

`with`$_2$: Conditional Rewriting

ThmA `with` *ThmB* and *ThmB*$_2$ …

- If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$:
  - 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 '$m + n \ne 0$'
  when trying to prove $(m + n) \cdot (n + 2) = (m + n) \cdot 5 \cdot k$:
  - "Cancellation of $\cdot$" is: $c \ne 0 \Rightarrow (c \cdot a = c \cdot b \equiv a = b)$
  - 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 $\sigma = [a, b, c := (n + 2), (5 \cdot k), (m + n)]$
  - $(c \ne 0)\sigma$ is $(m + n) \ne 0$
    and can be proven by "Assumption '$m + n \ne 0$'"
  - The goal is rewritten to $(a = b)\sigma$, that is, $(n + 2) = 5 \cdot k$.

`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 ⇒ q` `` with (3.60) `` `p ⇒ q ≡ p ∧ q ≡ 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: $p \quad \equiv \quad p \wedge q$
  which can be used for the rewrite: $p \quad \mapsto \quad p \wedge q$

Theorem (4.3) "Left-monotonicity of $\wedge$": $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
Proof:
Assuming `` `p ⇒ q` ``:
$p \wedge r$
$\equiv$ ⟨ Assumption `` `p ⇒ q` `` with "Definition of $\Rightarrow$ from $\wedge$" ⟩
$p \wedge q \wedge r$
$\Rightarrow$ ⟨ "Weakening" ⟩
$q \wedge r$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-07

Part 2: General Quantification

Quantification Examples

$(\sum i \mid 0 \le i < 4 \bullet i \cdot 8)$
$=$ ⟨ Quantification expansion, substitution ⟩
$0 \cdot 8 + 1 \cdot 8 + 2 \cdot 8 + 3 \cdot 8$

$(\prod i \mid 0 \le i < 3 \bullet i + (i + 1))$
$=$ ⟨ Quantification expansion, substitution ⟩
$(0 + 1) \cdot (1 + 2) \cdot (2 + 3)$

$(\forall i \mid 1 \le i < 3 \bullet i \cdot d \ne 6)$
$=$ ⟨ Quantification expansion, substitution ⟩
$1 \cdot d \ne 6 \wedge 2 \cdot d \ne 6$

$(\exists i \mid 0 \le i < 21 \bullet b\ i = 0)$
$=$ ⟨ Quantification expansion, substitution ⟩
$b\ 0 = 0 \vee b\ 1 = 0 \vee \boxed{\ldots} \vee b\ 20 = 0$

General Quantification

It works not only for +, ∧, ∨ …

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:

$$(\star\, x : T_1, y : T_2 \mid R \bullet P)$$

- $R : \mathbb{B}$ is the **range** of the quantification
- $P : T$ is the **body** of the quantification
- P 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: $(\star\, x : T_1, y : T_2 \mid R \bullet P)$

- $\_\vee\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.24), associative (3.25), and has *false* as identity (3.30) — the "big operator" for $\vee$ is $\exists$":
  $(\exists\, k : \mathbb{N} \mid k > 0 \bullet k \cdot k < k + 1)$
- $\_\wedge\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.36), associative (3.27), and has *true* as identity (3.39) — the "big operator" for $\wedge$ 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 $\sum$":
  $(\sum n : \mathbb{Z} \mid 0 < n < 100 \wedge \mathit{prime}\ n \bullet n \cdot n)$

Trivial Range Axioms

(8.13) **Axiom, Empty Range** (where u is the identity of $\star$):

$$(\star\, x \mid \mathit{false} \bullet P) \quad = \quad u$$

$$(\forall\, x \mid \mathit{false} \bullet P) \quad = \quad \mathit{true}$$

$$(\exists\, x \mid \mathit{false} \bullet P) \quad = \quad \mathit{false}$$

$$(\sum x \mid \mathit{false} \bullet P) \quad = \quad 0$$

$$(\prod x \mid \mathit{false} \bullet P) \quad = \quad 1$$

(8.14) **Axiom, One-point Rule:** Provided $\neg \mathit{occurs}(‘x’, ‘E’)$,

$$(\star x \mid x = E \bullet P) \quad = \quad P[x := E]$$

Manipulating Ranges

(8.23) **Theorem Split off term**: For $n : \mathbb{N}$ and dummies $i : \mathbb{N}$,

$$(\star\, i \mid 0 \le i < n+1 \bullet P) \quad = \quad (\star\, i \mid 0 \le i < n \bullet P) \star P[i := n]$$

$$(\star\, i \mid 0 \le i < n+1 \bullet P) \quad = \quad P[i := 0] \star (\star\, i \mid 0 < i < n+1 \bullet P)$$

- 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:

$m \le n \quad \Rightarrow$
$(\star\, i \mid m \le i < n+1 \bullet P) = (\star\, i \mid m \le i < n \bullet P) \star P[i := n]$

Theorem: Split off term from bottom:

$m \le n \quad \Rightarrow$
$(\star\, i \mid m \le i < n+1 \bullet P) = P[i := m] \star (\star\, i \mid m+1 \le i < n+1 \bullet P)$

Recall: 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>:

$$(\sum i : \mathbb{N} \mid i < x \bullet i + 1) \quad = \quad (\sum j : \mathbb{N} \mid j < x \bullet j + 1)$$

- **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?

$(x + 3 = 5 \cdot i)[i := 9]$
$\equiv \langle$ Substitution, … $\rangle$
$x = 42$

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:

$$(x + 3 = 5 \cdot i)[i := 9] \quad \equiv \quad (x + 3 = 5 \cdot j)[j := 9]$$

- **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 := !**

The *occurs* Meta-Predicate (ctd.)

Definition: $\mathit{occurs}(‘v’, ‘e’)$ means that at least one variable in the list v of variables occurs **free** in at least one expression in expression list e.

$\mathit{occurs}(‘i, n’, ‘(\sum i, n \mid 1 \le i \cdot n \le k \bullet n^i), (\sum n \mid 0 \le n < k \bullet n^i)’)$ ✓

$\mathit{occurs}(‘i’, ‘(i \cdot (5 + i))[i := k + 2]’)$ × **Substitution is a variable binder, too!**

$\mathit{occurs}(‘i’, ‘(i \cdot (5 + i))[i := i + 2]’)$ ✓

The $\neg$*occurs* Proviso for the One-point Rule

(8.14) **Axiom, One-point Rule:** Provided $\neg \mathit{occurs}(‘x’, ‘E’)$,

$$(\forall x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

$$(\exists x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

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:

- $(\forall x \mid x = x + 1 \bullet x = 42)$ **?** $x + 1 = 42$ **— "≡" not valid!**
- $(\exists x \mid x = 2 \cdot x \bullet y + x = 42)$ **?** $y + 2 \cdot x = 42$ **— "≡" not valid!**

Automatic extraction of $\neg$*occurs* Provisos

(8.14) **Axiom, One-point Rule:** Provided $\neg \mathit{occurs}(‘x’, ‘E’)$,

$$(\forall\, x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

$$(\exists\, x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

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 $\neg \mathit{occurs}(‘y’, ‘x, F’)$,

$$(\star\, y \mid R \bullet P)[x := F] \quad = \quad (\star\, y \mid R[x := F] \bullet P[x := F])$$

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 $\rangle$
  $$(\sum z \mid 1 \le z \le 2 \bullet y)[x := y + x]$$
  $=\ \langle$ 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 fresh variables for renaming to avoid variable capture!

In CalcCheck, renaming of bound variables is part of "Reflexivity of =",
but can also be mentioned explicitly.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-07

Part 3: Predicate Logic 1

Generalising De Morgan to Quantification

$$\neg(\exists\, i \mid 0 \le i < 4 \bullet P)$$
$=\ \langle$ Expand quantification $\rangle$
$$\neg(P[i := 0] \lor P[i := 1] \lor P[i := 2] \lor P[i := 3])$$
$=\ \langle$ (3.47) De Morgan $\rangle$
$$\neg P[i := 0] \land \neg P[i := 1] \land \neg P[i := 2] \land \neg P[i := 3]$$
$=\ \langle$ Contract quantification $\rangle$
$$(\forall\, i \mid 0 \le i < 4 \bullet \neg P)$$

(9.18b,c,a) **Generalised De Morgan**:

$$\neg(\exists\, x \mid R \bullet P) \quad \equiv \quad (\forall\, x \mid R \bullet \neg P)$$
$$(\exists\, x \mid R \bullet \neg P) \quad \equiv \quad \neg(\forall\, x \mid R \bullet P)$$
$$\neg(\exists\, x \mid R \bullet \neg P) \quad \equiv \quad (\forall\, x \mid R \bullet P)$$

(9.17) **Axiom, Generalised De Morgan**:

$$(\exists\, x \mid R \bullet P) \quad \equiv \quad \neg(\forall\, x \mid R \bullet \neg P)$$

"Trading" Range Predicates with Body Predicates in $\forall$

(9.2) Axiom, Trading: $\quad (\forall\, x \mid R \bullet P) \quad \equiv \quad (\forall\, x \bullet R \Rightarrow P)$

Trading Theorems for $\forall$:

(9.3a) $\quad (\forall\, x \mid R \bullet P) \quad \equiv \quad (\forall\, x \bullet \neg R \lor P)$

(9.3b) $\quad (\forall\, x \mid R \bullet P) \quad \equiv \quad (\forall\, x \bullet R \land P \equiv R)$

(9.3c) $\quad (\forall\, x \mid R \bullet P) \quad \equiv \quad (\forall\, x \bullet R \lor P \equiv P)$

(9.4a) $\quad (\forall\, x \mid Q \land R \bullet P) \quad \equiv \quad (\forall\, x \mid Q \bullet R \Rightarrow P)$

(9.4b) $\quad (\forall\, x \mid Q \land R \bullet P) \quad \equiv \quad (\forall\, x \mid Q \bullet \neg R \lor P)$

(9.4c) $\quad (\forall\, x \mid Q \land R \bullet P) \quad \equiv \quad (\forall\, x \mid Q \bullet R \land P \equiv R)$

(9.4d) $\quad (\forall\, x \mid Q \land R \bullet P) \quad \equiv \quad (\forall\, x \mid Q \bullet R \lor P \equiv P)$

"Trading" Range Predicates with Body Predicates in $\exists$

(9.2) Axiom, Trading: $\quad (\forall\, x \mid R \bullet P) \quad \equiv \quad (\forall\, x \bullet R \Rightarrow P)$

(9.17) **Axiom, Generalised De Morgan**: $\quad (\exists\, x \mid R \bullet P) \quad \equiv \quad \neg(\forall\, x \mid R \bullet \neg P)$

(9.19) Trading for $\exists$: $\quad (\exists\, x \mid R \bullet P) \quad \equiv \quad (\exists\, x \bullet R \land P)$

(9.20) **Trading for $\exists$:** $\quad (\exists\, x \mid Q \land R \bullet P) \equiv (\exists\, x \mid Q \bullet R \land P)$

Instantiation for $\forall$

$$P[x := E]$$
$\equiv\ \langle$ (8.14) One-point rule $\rangle$
$$(\forall\, x \mid x = E \bullet P)$$
$\Leftarrow\ \langle$ (9.10) Range weakening for $\forall$ $\rangle$
$$(\forall\, x \mid true \lor x = E \bullet P)$$
$\equiv\ \langle$ (3.29) Zero of $\lor$ $\rangle$
$$(\forall\, x \mid true \bullet P)$$
$\equiv\ \langle$ $true$ range in quantification $\rangle$
$$(\forall\, x \bullet P)$$

$$\boxed{\dfrac{\forall\, x \bullet P}{P[x := E]}\ \forall\text{-Elim}}$$

This proves: (9.13) **Instantiation:** $\quad (\forall\, x \bullet P) \quad \Rightarrow \quad P[x := E]$

The one-point rule is **"sharper"** than Instantiation.

Using sharper rules often means fewer dead ends…

A sharp version obtained via (3.60):

$$(\forall\, x \bullet P) \quad \equiv \quad (\forall\, x \bullet P) \land P[x := E]$$

Using Instantiation for $\forall$

(9.13) **Instantiation:** $\quad (\forall\, x \bullet P) \quad \Rightarrow \quad P[x := E]$

A sharp version of Instantiation obtained via (3.60): $\quad (\forall\, x \bullet P) \equiv (\forall\, x \bullet P) \land P[x := E]$

Proving $(\forall\, x \bullet x + 1 > x) \quad \Rightarrow \quad y + 2 > y$:

$$(\forall\, x \bullet x + 1 > x)$$
$=\ \langle$ **Instantiation (9.13) with (3.60)** $\rangle$
$$(\forall\, x \bullet x + 1 > x) \quad \land \quad y + 1 > y$$
$\Rightarrow\ \langle$ **Left-Monotonicity of $\land$ (4.3) with Instantiation (9.13)** $\rangle$
$$(y + 1) + 1 > y + 1 \quad \land \quad y + 1 > y$$
$\Rightarrow\ \langle$ Transitivity of $>$ (15.41) $\rangle$
$$y + 1 + 1 > y$$
$=\ \langle$ $1 + 1 = 2$ $\rangle$
$$y + 2 > y$$

Using Instantiation for $\forall$

(9.13) **Instantiation:** $\quad (\forall\, x \bullet P) \quad \Rightarrow \quad P[x := E]$

A sharp version of Instantiation obtained via (3.60): $\quad (\forall\, x \bullet P) \equiv (\forall\, x \bullet P) \land P[x := E]$

Theorem: $(\forall\, x : \mathbb{Z} \bullet x < x + 1) \ \Rightarrow\ y < y + 2$

Proof:

$\quad (\forall\, x : \mathbb{Z} \bullet x < x + 1)$

$\equiv\langle$ "Instantiation" (9.13) with (3.60) – – – explicit substitution needed! $\rangle$

$\quad (\forall\, x : \mathbb{Z} \bullet x < x + 1) \ \land\ (x < x + 1)[x := y + 1]$

$\equiv\langle$ Substitution, Fact `1 + 1 = 2` $\rangle$

$\quad (\forall\, x : \mathbb{Z} \bullet x < x + 1) \ \land\ y + 1 < y + 2$

$\Rightarrow\langle$ "Monotonicity of $\land$" with "Instantiation" $\rangle$

$\quad (x < x + 1)[x := y] \ \land\ y + 1 < y + 2$

$\equiv\langle$ Substitution $\rangle$

$\quad y < y + 1 \ \land\ y + 1 < y + 2$

$\Rightarrow\langle$ "Transitivity of $<$" $\rangle$

$\quad y < y + 2$

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)** $(\forall x \bullet P) \quad \Rightarrow \quad P[x := E]$:

Theorem: $(\forall x : \mathbb{Z} \bullet x < x + 1) \;\Rightarrow\; y < y + 2$

Proof:

Assuming (1) `$\forall x : \mathbb{Z} \bullet x < x + 1$`:

y

$<$ ⟨ Assumption (1) — implicit instantiation with `E := y` ⟩

$y + 1$

$<$ ⟨ Assumption (1) — implicit instantiation with `E := y + 1` ⟩

$y + 1 + 1$

$=$ ⟨ Fact `$1 + 1 = 2$` ⟩

$y + 2$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-18

Part 1: General Quantification (continued)

Plan for Today

- **General Quantification (LADM chapter 8) — Calculating with Quantifications**
- **Predicate Logic 2**:
  Proving Universal and Existential Quantifications
  (LADM chapter 9)

Leibniz Rules for Quantification

Try to use $\quad x + x = 2 \cdot x \quad$ and Leibniz (1.5) $\quad \dfrac{X \;=\; Y}{E[z := X] \;=\; E[z := Y]} \quad$ 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: $\quad (\sum x \mid 0 \le x < 9 \bullet z)$
- Perform substitution: $\quad (\sum x \mid 0 \le x < 9 \bullet z)[z := x + x]$
  $(\sum y \mid 0 \le y < 9 \bullet x + x)$
- Not possible with (1.5)!
  — $E[z := X] = E[z := Y]$ **renames** x**!**

Special Leibniz rule for quantification:

$$\frac{P \;=\; Q}{(\star x \mid R \bullet E[z := P]) \;=\; (\star x \mid R \bullet E[z := Q])}$$

LADM Leibniz Rules for Quantification

Rewrite equalities in the **range** context of quantifications:

(8.12) **Leibniz** $\quad \dfrac{P \;=\; Q}{(\star x \mid E[z := P] \bullet S) \;=\; (\star x \mid E[z := Q] \bullet S)}$

Rewrite equalities in the **body** context of quantifications:

(8.12) **Leibniz** $\quad \dfrac{R \;\Rightarrow\; (P \;=\; Q)}{(\star x \mid R \bullet E[z := P]) \;=\; (\star x \mid R \bullet E[z := Q])}$

(These inference rules will also be used **implicitly**.)

Important: $P = Q$ needs to be a **theorem**!

These rules are **not** available for local **Assumptions**!

(Because x may occur in P, Q.)

Variable Binding Rearrangements

(8.19) **Axiom, Interchange of dummies:**

$$(\star x \mid R \bullet (\star y \mid S \bullet P)) \;=\; (\star y \mid S \bullet (\star x \mid R \bullet P))$$

provided $\neg occurs('y', 'R')$ and $\neg occurs('x', 'S')$, and each quantification is defined.

(8.20) **Axiom, Nesting:**

$$(\star x, y \mid R \wedge S \bullet P) \;=\; (\star x \mid R \bullet (\star y \mid S \bullet P))$$

provided $\neg occurs('y', 'R')$.

(8.21) **Axiom, Dummy renaming** (α-conversion):

$$(\star x \mid R \bullet P) \;=\; (\star y \mid R[x := y] \bullet P[x := y])$$

provided $\neg occurs('y', 'R, P')$.

***Substitution* (8.11) *prevents capture of* y *by binders in* R *or* P**

Permutation of Bound Variables

Apparently not provable for general quantification from the quantification axioms in the textbook:

Dummy list permutation:

$$(\star x, y \mid R \bullet P) \;=\; (\star y, x \mid R \bullet P)$$

(without side conditions restricting variable occurrences!)

However, the following are easily provable from (8.19) **Interchange of dummies** — **Exercise:**

Dummy list permutation for $\forall$:

$$(\forall x, y \mid R \bullet P) \;=\; (\forall y, x \mid R \bullet P)$$

Dummy list permutation for $\exists$:

$$(\exists x, y \mid R \bullet P) \;=\; (\exists y, x \mid R \bullet P)$$

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.

$(\sum i : \mathbb{N} \mid i < n \bullet f\, i) + (\sum i : \mathbb{N} \mid i < n \bullet g\, i)$

$=$ ⟨ 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:

$$(\forall x \mid R \bullet P) \wedge (\forall x \mid R \bullet Q) = (\forall x \mid R \bullet P \wedge Q)$$

$$(\exists x \mid R \bullet P) \vee (\exists x \mid R \bullet Q) = (\exists x \mid R \bullet P \vee Q)$$

Disjoint Range Split

(8.16) **Axiom, Range split:**

$$(\star x \mid R \vee S \bullet P) \;=\; (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each quantification is defined.

$$(\Sigma x \mid R \vee S \bullet P) \;=\; (\Sigma x \mid R \bullet P) + (\Sigma x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each sum is defined.

$$(\forall x \mid R \vee S \bullet P) \;=\; (\forall x \mid R \bullet P) \wedge (\forall x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$.

$$(\exists x \mid R \vee S \bullet P) \;=\; (\exists x \mid R \bullet P) \vee (\exists x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$.

Range Split "Axioms"

(8.16) **Axiom, Range split:**

$$(\star x \mid R \vee S \bullet P) \;=\; (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each quantification is defined.

(8.17) **Axiom, Range Split:**

$$(\star x \mid R \vee S \bullet P) \star (\star x \mid R \wedge S \bullet P) \;=\; (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided each quantification is defined.

(8.18) **Axiom, Range Split for idempotent** $\star$**:**

$$(\star x \mid R \vee S \bullet P) \;=\; (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided each quantification is defined.

$$(\forall x \mid R \vee S \bullet P) \;=\; (\forall x \mid R \bullet P) \wedge (\forall x \mid S \bullet P)$$

$$(\exists x \mid R \vee S \bullet P) \;=\; (\exists x \mid R \bullet P) \vee (\exists x \mid S \bullet P)$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-18

Part 2: Predicate Logic (continued)

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 $b \le n$".

$(\exists b : \mathbb{Z} \bullet (\forall n : \mathbb{Z} \bullet b \le n))$

- "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 $r < \pi < s$"

$(\forall \varepsilon : \mathbb{R} \mid 0 < \varepsilon$
$\quad \bullet (\exists r, s : \mathbb{Q} \mid r < s < r + \varepsilon \bullet r < \pi < s))$

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.

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \bullet P)$:

For any '$v : \mathbb{N}$': *Proof for P*

Inference rule:

$$\frac{P}{\forall x \bullet P} \;\forall\text{-Intro (prov. } x \text{ not free in assumptions)}$$

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 \`$x : \mathbb{N}$\`:

$$\begin{array}{ll}
 & x < x + 1 \\
\equiv & \langle \text{Identity of } + \rangle \\
 & x + 0 < x + 1 \\
\equiv & \langle \text{Cancellation of } + \rangle \\
 & 0 < 1 \\
\equiv & \langle \text{Fact `} 1 = suc\ 0 \text{`} \rangle \\
 & 0 < suc\ 0 \\
\equiv & \langle \text{Zero is less than successor} \rangle \\
 & true
\end{array}$$

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 \`$x < 2$\`:

$$\begin{array}{ll}
 & x \\
< & \langle \text{Assumption `} x < 2 \text{`} \rangle \\
 & 2 \\
< & \langle \text{Fact `} 2 < 3 \text{`} \rangle \\
 & 3
\end{array}$$

∃-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**".

$$\begin{array}{ll}
 & (\exists x : \mathbb{N} \bullet x \cdot x < x + x) \\
\Leftarrow & \langle \exists\text{-Introduction} \rangle \\
 & (x \cdot x < x + x)[x := 1] \\
\equiv & \langle \text{Substitution} \rangle \\
 & 1 \cdot 1 < 1 + 1 \\
\equiv & \langle \text{Evaluation} \rangle \\
 & true
\end{array}$$

Using ∃-Introduction for "Proof by Counter-Example"

(9.28) **∃-Introduction:** $P[x := E] \Rightarrow (\exists x \bullet P)$

$$\begin{array}{ll}
 & \neg(\forall x : \mathbb{N} \bullet x + x < x \cdot x) \\
\equiv & \langle \text{Generalised De Morgan} \rangle \\
 & (\exists x : \mathbb{N} \bullet \neg(x + x < x \cdot x)) \\
\Leftarrow & \langle \exists\text{-Introduction} \rangle \\
 & (\neg(x + x < x \cdot x))[x := 2] \\
\equiv & \langle \text{Substitution} \rangle \\
 & \neg(2 + 2 < 2 \cdot 2) \\
\equiv & \langle \text{Fact `} 2 + 2 < 2 \cdot 2 \equiv false \text{`} \rangle \\
 & \neg false \\
\equiv & \langle \text{Negation of } false \rangle \\
 & true
\end{array}$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-18

Part 3: Monotonicity of ∀ and ∃

Recall: Monotonicity With Respect To ⇒

Let $\_\le\_$ be an order on T, and let $f : T \to T$ be a function on T. Then f is called

- **monotonic** iff $x \le y \;\Rightarrow\; f\,x \le f\,y$,
- **antitonic** iff $x \le y \;\Rightarrow\; f\,y \le f\,x$.

(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 ⇒:

- **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

- $\_\le_1\_$ be an order on T_1, and $\_\le_2\_$ be an order on T_2,
- $f : T_1 \to T_2$ be a function from T_1 to T_2.

Then f is called

- **monotonic** iff $x \le_1 y \;\Rightarrow\; f\,x \le_2 f\,y$,
- **antitonic** iff $x \le_1 y \;\Rightarrow\; f\,y \le_2 f\,x$.

Transitivity of $\le$ is antitonitcity of $(\_\le r) : \mathbb{Z} \to \mathbb{B}$:

- **Left-Antitonicity of** $\le$: $(p \le q) \Rightarrow (q \le r) \Rightarrow (p \le r)$
- **Right-Monotonicity of** $\le$: $(p \le q) \Rightarrow (r \le p) \Rightarrow (r \le q)$

Weakening/Strengthening for ∀ and ∃ — "Cheap Antitonicity/Monotonicity"

(9.10) **Range weakening/strengthening for** $\forall$: $(\forall\, x \mid Q \vee R \bullet P) \Rightarrow (\forall\, x \mid Q \bullet P)$

(9.11) **Body weakening/strengthening for** $\forall$: $(\forall\, x \mid R \bullet P \wedge Q) \Rightarrow (\forall\, x \mid R \bullet P)$

(9.25) **Range weakening/strengthening for** $\exists$: $(\exists\, x \mid R \bullet P) \Rightarrow (\exists\, x \mid Q \vee R \bullet P)$

(9.26) **Body weakening/strengthening for** $\exists$: $(\exists\, x \mid R \bullet P) \Rightarrow (\exists\, x \mid R \bullet P \vee Q)$

Recall:

(9.2) **Trading for** $\forall$: $(\forall\, x \mid R \bullet P) \equiv (\forall\, x \bullet R \Rightarrow P)$

(9.19) **Trading for** $\exists$: $(\exists\, x \mid R \bullet P) \equiv (\exists\, x \bullet R \wedge P)$

Monotonicity for ∀

(9.12) **Monotonicity of** $\forall$:

$$(\forall\, x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\forall\, x \mid R \bullet P_1) \Rightarrow (\forall\, x \mid R \bullet P_2))$$

Range-Antitonicity of $\forall$:

$$(\forall\, x \bullet R_2 \Rightarrow R_1) \Rightarrow ((\forall\, x \mid R_1 \bullet P) \Rightarrow (\forall\, x \mid R_2 \bullet P))$$

$$
\begin{aligned}
& (\forall\, x \bullet R_2 \Rightarrow R_1) \\
\Rightarrow\ & \langle \text{(9.12) with shunted (3.82a) Transitivity of} \Rightarrow \rangle \\
& (\forall\, x \bullet (R_1 \Rightarrow P) \Rightarrow (R_2 \Rightarrow P)) \\
\Rightarrow\ & \langle \text{(9.12) Monotonicity of } \forall \rangle \\
& (\forall\, x \bullet R_1 \Rightarrow P) \Rightarrow (\forall\, x \bullet R_2 \Rightarrow P) \\
=\ & \langle \text{(9.2) Trading for } \forall \rangle \\
& (\forall\, x \mid R_1 \bullet P) \Rightarrow (\forall\, x \mid R_2 \bullet P)
\end{aligned}
$$

Monotonicity for ∃

(9.27) (Body) **Monotonicity of** $\exists$:

$$(\forall\, x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\exists\, x \mid R \bullet P_1) \Rightarrow (\exists\, x \mid R \bullet P_2))$$

Range-Monotonicity of $\exists$:

$$(\forall\, x \bullet R_1 \Rightarrow R_2) \Rightarrow ((\exists\, x \mid R_1 \bullet P) \Rightarrow (\exists\, x \mid R_2 \bullet P))$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-19

Part 1: Monotonicity of ∀ and ∃

Plan for Today

- Predicate logic: Universal and Existential Quantification
- Introduction to Sequences (Finite Lists)

Monotonicity for ∀

(9.12) **Monotonicity of** $\forall$:

$$(\forall\, x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\forall\, x \mid R \bullet P_1) \Rightarrow (\forall\, x \mid R \bullet P_2))$$

Range-Antitonicity of $\forall$:

$$(\forall\, x \bullet R_2 \Rightarrow R_1) \Rightarrow ((\forall\, x \mid R_1 \bullet P) \Rightarrow (\forall\, x \mid R_2 \bullet P))$$

$$
\begin{aligned}
& (\forall\, x \bullet R_2 \Rightarrow R_1) \\
\Rightarrow\ & \langle \text{(9.12) with shunted (3.82a) Transitivity of} \Rightarrow \rangle \\
& (\forall\, x \bullet (R_1 \Rightarrow P) \Rightarrow (R_2 \Rightarrow P)) \\
\Rightarrow\ & \langle \text{(9.12) Monotonicity of } \forall \rangle \\
& (\forall\, x \bullet R_1 \Rightarrow P) \Rightarrow (\forall\, x \bullet R_2 \Rightarrow P) \\
=\ & \langle \text{(9.2) Trading for } \forall \rangle \\
& (\forall\, x \mid R_1 \bullet P) \Rightarrow (\forall\, x \mid R_2 \bullet P)
\end{aligned}
$$

Monotonicity for ∃

(9.27) (Body) **Monotonicity of** $\exists$:

$$(\forall\, x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\exists\, x \mid R \bullet P_1) \Rightarrow (\exists\, x \mid R \bullet P_2))$$

Range-Monotonicity of $\exists$:

$$(\forall\, x \bullet R_1 \Rightarrow R_2) \Rightarrow ((\exists\, x \mid R_1 \bullet P) \Rightarrow (\exists\, x \mid R_2 \bullet P))$$

Predicate Logic Laws You Really Need To Know

(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 $\neg occurs(\text{'}x\text{'}, \text{'}E\text{'})$, $(\forall x \mid x = E \bullet P) \equiv P[x := E]$
$(\exists x \mid x = E \bullet P) \equiv P[x := E]$

(9.17) **Generalised De Morgan**: $(\exists\, x \mid R \bullet P) \equiv \neg(\forall\, x \mid R \bullet \neg P)$

(9.2) **Trading for** $\forall$: $(\forall\, x \mid R \bullet P) \equiv (\forall\, x \bullet R \Rightarrow P)$

(9.4a) **Trading for** $\forall$: $(\forall\, x \mid Q \wedge R \bullet P) \equiv (\forall\, x \mid Q \bullet R \Rightarrow P)$

(9.19) **Trading for** $\exists$: $(\exists\, x \mid R \bullet P) \equiv (\exists\, x \bullet R \wedge P)$

(9.20) **Trading for** $\exists$: $(\exists\, x \mid Q \wedge R \bullet P) \equiv (\exists\, x \mid Q \bullet R \wedge P)$

(9.13) **Instantiation:** $(\forall\, x \bullet P) \Rightarrow P[x := E]$

(9.28) $\exists$**-Introduction**: $P[x := E] \Rightarrow (\exists\, x \bullet P)$

. . . and correctly handle substitution, Leibniz, renaming of bound variables, and monotonicity/antitonicity . . .

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-19

Part 2: Practice with ∀ and ∃

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.
- The value of an expression (in a state) only depends on its free variables.
- The value of a closed expression does not depend on the state.
- A closed Boolean expression, or sentence,
  - either always evaluates to *true*
  - or always evaluates to *false*
- A closed Boolean expression, or sentence,
  - is either valid
  - or a contradiction
- For a closed Boolean expression, or sentence, ϕ
  - either ϕ is valid
  - or $\neg\phi$ is valid
- For a closed Boolean expression, or sentence, ϕ,
  only one of ϕ and $\neg\phi$ can have a proof!

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 $\neg\phi$ can have a proof!

- Starting "Practice with ∀ and ∃" in H11.1...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-19

Part 3: 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 "[...]" for sequences is not supported by CalcCheck. LADM writes "⟨...⟩".)
- 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 *Seq A* of sequences with elements of type A
  as generated inductively by the following two constructors:

ϵ	:	*Seq A*	`\eps`	empty sequence
$\_\triangleleft\_$	:	$A \to$ *Seq A* $\to$ *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 *Seq A* of sequences with elements of type A
  as generated inductively by the following two constructors:

ϵ	:	*Seq A*	`\eps`	empty sequence
$\_\triangleleft\_$	:	$A \to$ *Seq A* $\to$ *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":

  $\_\triangleright\_$: *Seq A* $\to A \to$ *Seq A* `\snoc` "snoc"

  $\triangleright$ associates to the left.
- (Con-)catenation:

  $\_\frown\_$: *Seq A* $\to$ *Seq A* $\to$ *Seq A* `\catenate`

  $\frown$ associates to the right.

Sequences — Induction Principle

- The set of all **sequences over type A** is written *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 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 $xs :$ *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 $xs :$ *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 $xs :$ *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)
```

$\Longrightarrow$ H11.2

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-21

Part 1: At least five zeroes...

Plan for Today

- **Typing** (see also Textbook Section 8.1)
- **Textbook Chapter 11: Set Theory**

Formalise!

The equation $f\ x = 0$ has at least five solutions.

$\Longrightarrow$ Experiment in the H11.1 notebook...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-21

Part 2: 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: $\mathit{Seq}\ \mathbb{N}$, $\mathbb{N} \to \mathbb{B}$, $\mathit{Seq}\ (\mathit{Seq}\ \mathbb{N}) \to \mathit{Seq}\ \mathbb{B}$, **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:
  - $(x + y) \cdot x \longrightarrow (x : \mathbb{N} + y) \cdot x$
  - $(x + y) \cdot x \longrightarrow ((((x : \mathbb{N}) + (y : \mathbb{N})) : \mathbb{N}) \cdot (x : \mathbb{N})) : \mathbb{N}$

Function Types — Textbook 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 $\mathbb{B}$.

In general: For $f : t_1 \times \cdots \times t_n \to r$,
- if expression a_1 has type t_1, and ..., and a_n has type t_n
- then function application $f(a_1, \ldots, a_n)$ is an expression
- and has type r.

Function Types — 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 $t_1 \to \cdots \to t_n \to r$

$\Rightarrow$ **We write:** $\boxed{f : t_1 \to \cdots \to t_n \to r}$

(The function type constructor $\to$ **associates to the right!**)

Examples: $\neg : \mathbb{B} \to \mathbb{B}$ $\quad$ $\_+\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$ $\quad$ $\_<\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$

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: For $f : A \to B$,
- if expression x has type A,
- then function application $f\ x$ is an expression
- and has type B.

$$\frac{f : A \to B \qquad x : A}{f\ x : B}$$

Well-typed Expressions?

$2 + k$ ✓ $\quad$ $42 - \mathit{true}$ × $\quad$ $\neg(3 \cdot x)$ × $\quad$ $(1/(x : \mathbb{R})) : \mathbb{R}$ ✓

Non-well-typed expressions make no sense!

Function Application — Textbook Version

Consider function g defined by: (1.6) $\quad g(z) = 3 \cdot z + 6$

- Special **function application** syntax for argument that is identifier or constant:

$$g.z = 3 \cdot z + 6$$

Function Application — Mechanised Mathematics Version

Consider function g defined by: (1.6) $\quad g\ z = 3 \cdot z + 6$

- **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 ooperators**.
- As non-associative binary infix operator, function application **associates to the left:**
  If $f : \mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})$, then $f\ 2\ 3 = (f\ 2)\ 3$, and $f\ 2 : \mathbb{Z} \to \mathbb{Z}$
- Typing rule for function application:

$$\frac{f : A \to B \qquad x : A}{f\ x : B}$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-21

Part 3: 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 $\cup$ and $\cap$)
- ...

The Language of Set Theory — Overview

- The type **set** t of sets with elements of type t
- Set membership: For $e : t$ and $S : \mathbf{set}\ t$: $e \in S$
- **Set comprehension:** $\{x : t \mid R \bullet E\}$ — following the pattern of quantification
- Set enumeration: $\{6, 7, 9\}$
- Set size: $\#\{6, 7, 9\} = 3$
- Set inclusion: $\subset, \subseteq, \supset, \supseteq$
- Set union and intersection: $\cup, \cap$
- Set difference: $S - T$
- Set complement: $\sim S$
- Power set (set of subsets): $\mathbb{P}\ S$
- Cartesian product (cross product, direct product) of sets: $S \times T$ (Section 14.1)

Set Membership versus Type Annotation

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.
- $\#\{n : \mathbb{N} \mid true \bullet n\}$ **is undefined!**
- "$\#\mathbb{N}$" **is a type error!** — because $\mathbb{N} : Type$
- Types are not sets — like in Haskell:

```
Integer :: *
Data.Set.Set Integer :: *
```

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 $\neg occurs(`x', `F')$,

$$F \in \{x \mid R \bullet E\} \equiv (\exists x \mid R \bullet E = F)$$

(11.2f) **Empty Set:** $v \in \{\} \equiv false$

(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 $\neg occurs(`x', `S, T')$,

$$S \subseteq T \equiv (\forall x \bullet x \in S \Rightarrow x \in T)$$

(11.14) **Axiom, Proper subset:**	$S \subset T$	$\equiv \quad S \subseteq T \land S \neq T$
(11.20) **Axiom, Union:**	$v \in S \cup T$	$\equiv \quad v \in S \lor v \in T$
(11.21) **Axiom, Intersection:**	$v \in S \cap T$	$\equiv \quad v \in S \land v \in T$
(11.22) **Axiom, Set difference:**	$v \in S - T$	$\equiv \quad v \in S \land v \notin T$
(11.23) **Axiom, Power set:**	$v \in \mathbb{P}\ S$	$\equiv \quad v \subseteq S$

Set Comprehension

Set comprehension examples:

$$\{i : \mathbb{N} \mid i < 4 \bullet 2 \cdot i + 1\} = \{1, 3, 5, 7\}$$
$$\{x : \mathbb{Z} \mid 1 \leq x < 5 \bullet x \cdot x\} = \{1, 4, 9, 16\}$$
$$\{i : \mathbb{Z} \mid 5 \leq i < 8 \bullet i \triangleleft i \triangleleft \epsilon\} = \{(5 \triangleleft 5 \triangleleft \epsilon), (6 \triangleleft 6 \triangleleft \epsilon), (7 \triangleleft 7 \triangleleft \epsilon)\}$$

(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.

Formalise!

$P : Type$ — The type of persons

$\_called\_ : P \to P \to \mathbb{B}$

Jane called more people than Alex.

$$\#\{p : P \mid Jane \text{ called } p\} > \#\{p : P \mid Alex \text{ called } p\}$$

Formalise!

The equation $f\ x = 0$ has at least five solutions.

Without sets: Use $\neq$ to assert "different":

$$(\exists a, b, c, d, e$$
$$\mid a \neq b \neq c \neq d \neq e \neq c \neq a \neq d \neq b \neq e \neq a$$
$$\bullet f\ a = f\ b = f\ c = f\ d = f\ e = 0 \quad)$$

— **does not scale!**

With sets — first attempt:

$\#\{x \mid f\ x = 0\} \geq 5$ — **That does not work for, e.g., $f = \sin$.**

Taking into account possibly infinite sets of solutions:

$$(\exists S : \mathbf{set}\ \mathbb{R} \mid \#S \geq 5 \bullet (\forall x \mid x \in S \bullet f\ x = 0))$$

This "works", because:
Every infinite set contains at least one finite set of size at least 5.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-25

Part 1: Set Theory

Plan for Today

- **Textbook Chapter 11: Set Theory**

Anything Wrong?

Let the set Q be defined by the following:

(R) $Q = \{S \mid S \notin S\}$

Then:

$$\begin{aligned}
& Q \in Q \\
\equiv\ & \langle \text{(R)} \rangle \\
& Q \in \{S \mid S \notin S\} \\
\equiv\ & \langle \text{(11.3) Membership in set comprehension} \rangle \\
& (\exists S \mid S \notin S \bullet Q = S) \\
\equiv\ & \langle \text{(9.19) Trading for } \exists \text{, (8.14) One-point rule} \rangle \\
& Q \notin Q \\
\equiv\ & \langle \text{(11.0) Def. } \notin \rangle \\
& \neg(Q \in Q)
\end{aligned}$$

With (3.15) $p \equiv \neg p \equiv false$, this proves:

(R′) $false$ — **"Russell's paradox"**

$\_\in\_, \_\notin\_ : A \to \mathbf{set}\ A \to \mathbb{B}$

"The mother of all type errors"

$\Longrightarrow$ **birth of type theory...**

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 Membership and Set Enumerations

(11.3) **Axiom, Set membership:** Provided $\neg occurs(`x', `F')$,
$$F \in \{x \mid R \bullet E\} \equiv (\exists x \mid R \bullet E = F)$$

(11.7b) **Simple set compr. membership:**
$$F \in \{x \mid R\} \equiv R[x := F]$$

(11.2) Provided $\neg occurs(`x', `e_0, \ldots, e_{n-1}')$,
$$\{e_0, \ldots, e_{n-1}\} = \{x \mid x = e_0 \lor \cdots \lor x = e_{n-1} \bullet x\}$$

The empty set: $\{x \mid false \bullet x\} = \{\} = \{\}$

Singleton sets: $\{x \mid x = E \bullet x\} = \{E\}$ — provided $\neg occurs(`x', `E')$

One-point set comprehension: $\{x \mid x = E \bullet F\} = \{F[x := E]\}$
— provided $\neg occurs(`x', `E')$

Simplified Set Comprehension Notation

(11.6) Provided $\neg occurs(`y', `R, E')$,
$$\{x \mid R \bullet E\} = \{y \mid (\exists x \mid R \bullet y = E) \bullet y\}$$

This means that each set comprehension of shape $\{x \mid R \bullet E\}$ can be rewritten to shape $\{y \mid R' \bullet y\}$.

Recall: Abbreviated Notation:
$$\{y \mid R\} := \{y \mid R \bullet y\}$$

Set Comprehension versus Predicates

(11.5) $S = \{x \mid x \in S\}$ provided $\neg occurs(`x', `S')$

(11.7) $x \in \{x \mid R\} \equiv R$

(11.8) **Principle of comprehension:** To each predicate R there corresponds a set comprehension $\{x : T \mid R\}$ which contains the objects in T that satisfy R.

R is called a **characteristic predicate** of the set.

$f_R : T \to \mathbb{B}$ with $f_R\ x = R$ is also called the **characteristic function** of the set.

Two alternatives for defining sets:

$S = \{x \mid R\}$	$x \in S \equiv R$
$T = \{x \mid x = 3 \lor x = 5\}$	$x \in T \equiv x = 3 \lor x = 5$

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:**
Let S and T be set expressions and v be a variable. — Using "Set inclusion"
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 $\neg occurs(`x', `S, T')$,
$$S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$$

(11.9) $\{x \mid Q\} = \{x \mid R\} \equiv (\forall x \bullet Q \equiv R)$ — Leibniz for set compr. ranges

(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!
- $\boxed{\text{Using "Set extensionality"}}$ or $\boxed{\text{Using (11.9)}}$ followed by $\boxed{\text{For any} \ldots}$ 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) ⟩

$v \in \{x \mid x = E \lor x = E\}$

$\equiv$ ⟨ Idempotency of $\lor$ (3.26) ⟩

$v \in \{x \mid x = E\}$

$\equiv$ ⟨ Set enumerations (11.2) ⟩

$v \in \{E\}$

Using Set Extensionality — More CalcCheck-Style

Axiom (11.4) "Set extensionality": $S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$
— provided $\neg occurs(`x', `S, T')$

Example (8.2a): $\{E, E\} = \{E\}$

Proof:

  Using "Set extensionality":

    Subproof for `$\forall v \bullet v \in \{E, E\} \equiv v \in \{E\}$`:

      For any `$v$`:

        $v \in \{E, E\}$

        $\equiv$ ⟨ Set enumerations (11.2) ⟩

        $v \in \{x \mid x = E \lor x = E\}$

        $\equiv$ ⟨ Idempotency of $\lor$ (3.26) ⟩

        $v \in \{x \mid x = E\}$

        $\equiv$ ⟨ Set enumerations (11.2) ⟩

        $v \in \{E\}$

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 $\neg occurs(`x', `F')$,
$$F \in \{x \mid R \bullet E\} \equiv (\exists x \mid R \bullet E = F)$$

(11.2f) **Empty Set:** $v \in \{\} \equiv false$

(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 $\neg occurs(`x', `S, T')$,
$$S \subseteq T \equiv (\forall x \bullet x \in S \Rightarrow x \in T)$$

(11.14)	**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:**	$v \in \mathbb{P}\, S \equiv v \subseteq S$
(14.3)	**Axiom, Cross product:**	$S \times T = \{b, c \mid b \in S \land c \in T \bullet \langle b, c \rangle\}$

Calculate!

The size of a finite set S, that is, the number of its elements, is written $\# S$

- $\#\{1,2\}$
- $\#\{1,1\}$
- $\#\{1\}$
- $\#\{\}$
- $\#\{\{\}\}$
- $\#\{\{\{\}\}\}$
- $\#\{\{\},\{\{\}\}\}$
- $\#\{\{\},\{\}\}$
- $\#(\{1,2,3\}\cap\{3,4\})$
- $\#(\{1,2,3\}\cup\{3,4\})$
- $\#(\{1,2,3\}\times\{3,4\})$
- $\#(\{1,2,3\}\cap\{3,2\})$
- $\#(\{1,2,3\}\cup\{3,2\})$
- $\#(\{1,2,3\}\times\{3,2\})$
- $\#(\mathbb{P}\,\{1,2,3\})$
- $\#(\mathbb{P}\,\mathbb{P}\,\{1,2,3\})$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-26

Typed Set Theory

Plan for Today

- **Textbook Chapter 11: Set Theory**

Coming up (interleaved):

- **Explicit Induction Principles**
- **Induction** (LADM Chapter 12)
- **Relations** (LADM Chapter 14)
- Sequences (LADM Chapter 13) may be further developed in Exercises, Assignments, …

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 $\neg occurs(`x', `F')$,
$$F \in \{x \mid R \bullet E\} \quad\equiv\quad (\exists x \mid R \bullet E = F)$$

(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:** $S \subset T \quad\equiv\quad S \subseteq T \wedge S \neq T$
(11.20) **Axiom, Union:** $v \in S \cup T \quad\equiv\quad v \in S \vee v \in T$
(11.21) **Axiom, Intersection:** $v \in S \cap T \quad\equiv\quad v \in S \wedge v \in T$
(11.22) **Axiom, Set difference:** $v \in S - T \quad\equiv\quad v \in S \wedge v \notin T$
(11.23) **Axiom, Power set:** $v \in \mathbb{P}\,S \quad\equiv\quad v \subseteq S$
(14.3) **Axiom, Cross product:** $S \times T \;=\; \{b,c \mid b \in S \wedge c \in T \bullet \langle b,c\rangle\}$

"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: $\{\}, U : \mathbf{set}\ t$

$$(\{\} : \mathbf{set}\ \mathbb{B}) \;=\; \{\} \qquad (U : \mathbf{set}\ \mathbb{B}) \;=\; \{\mathit{false}, \mathit{true}\}$$
$$(\{\} : \mathbf{set}\ \mathbb{N}) \;=\; \{\} \qquad (U : \mathbf{set}\ \mathbb{N}) \;=\; \{k : \mathbb{N} \mid \mathit{true}\}$$

"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:** $v \in \sim S \;\equiv\; v \in \mathbf{U} \wedge v \notin S$

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) $v \in \;\sim S \;\equiv\; v \notin S \quad$ (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: $U : \mathbf{set}\ t$

Axiom: $x \in U$ — remember: $\_\in\_ : t \to \mathbf{set}\ t \to \mathbb{B}$

Theorem: $(U : \mathbf{set}\ t) = \{x : t \bullet x\}$

Types are not sets! — $(U : \mathbf{set}\ t)$ is the set containing all values of type t.

We define a nicer notation: $\llcorner\, t\, \lrcorner = (U : \mathbf{set}\ t)$

"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:** $v \in \sim S \quad\equiv\quad v \in U \wedge v \notin S$

Complement can be expressed via difference: $\sim S \;=\; U - S$

Complement $\sim$ **always implicitly depends** on the universe U!

Example: $\sim\{\mathit{true}\} \;=\; \llcorner\, \mathbb{B}\, \lrcorner - \{\mathit{true}\} \;=\; \{\mathit{false}, \mathit{true}\} - \{\mathit{true}\} \;=\; \{\mathit{false}\}$

LADM: "We can easily prove

(11.18) $v \in \sim S \quad\equiv\quad v \notin S \quad$ (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?

Power Set

(11.23) **Axiom, Power set:** $v \in \mathbb{P}\,S \quad\equiv\quad v \subseteq S$

Declaration: $\mathbb{P}\_ : \mathbf{set}\ t \to \mathbf{set}\ (\mathbf{set}\ t)$

— remember: $\mathbf{set} : \mathit{Type} \to \mathit{Type}$

$\mathbb{P}\,\{0,1\} = \{\{\},\{0\},\{1\},\{0,1\}\}$

- For a type t, the **type of subsets of t** is $\mathbf{set}\ t$
- 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 $\forall$ and $\exists$ rules more complicated.)

  If you find a place where I **accidentally** still follow Z in writing "$\mathbb{P}\ t$" also for "$\mathbf{set}\ t$" or "$\mathbb{P}\,\llcorner\, t\, \lrcorner$", please point it out to me.

What is the Type of Set Complement $\sim\_$?

Consider:

- $\mathbb{Z}_+ : \mathbf{set}\ \mathbb{Z}$
- $S_1 = \{1,3,8\}$
- $S_1 \in \mathbb{P}\,\mathbb{Z}_+$
- $S_1 : \mathbf{set}\ \mathbb{Z}$
- $\sim S_1 : \mathbf{set}\ \mathbb{Z}$
- $\sim S_1 \notin \mathbb{P}\,\mathbb{Z}_+$

Which of the following makes most sense?

- $\sim\_ : \mathbb{P}\,S \to \mathbb{P}\,S$
- $\sim\_ : \mathbb{P}\,S \to \mathbb{P}\,t$ — provided $S : \mathbf{set}\ t$
- $\sim\_ : \mathbb{P}\,S \to \mathbf{set}\ t$ — provided $S : \mathbf{set}\ t$
- $\sim\_ : \mathbf{set}\ t \to \mathbf{set}\ t$

— Sets are not types!

Note: In relation with types, sets are "just some kind of data", like numbers…

- $\mathbf{set} : \mathit{Type} \to \mathit{Type}$
- $\mathbb{P} : \mathbf{set}\ t \to \mathbf{set}\ (\mathbf{set}\ t)$
- $\mathbb{P}\,S : \mathbf{set}\ (\mathbf{set}\ t)$ — provided $S : \mathbf{set}\ t$
- $\_\!\to\!\_ : \mathit{Type} \to \mathit{Type} \to \mathit{Type}$

Calculate!

The size of a finite set S, that is, the number of its elements, is written $\# S$

- $\# \llcorner \mathbb{B} \lrcorner$
- $\# \{S : \mathbf{set}\ \mathbb{B} \mid true \in S \bullet S\}$
- $\# \{T : \mathbf{set\ set}\ \mathbb{B} \mid \{\} \notin T \bullet T\}$
- $\# \{S : \mathbf{set}\ \mathbb{N} \mid (\forall x : \mathbb{N} \mid x \in S \bullet x < n) \wedge \# S = k \bullet S\}$

- $\llcorner \mathbb{B} \lrcorner = \{false, true\}$
- $S \in \llcorner \mathbf{set}\ \mathbb{B} \lrcorner \quad \equiv \quad S \subseteq \llcorner \mathbb{B} \lrcorner$
- $\llcorner \mathbf{set}\ \mathbb{B} \lrcorner = \{\{\}, \{false\}, \{true\}, \{false, true\}\}$
- $T \in \llcorner \mathbf{set\ set}\ \mathbb{B} \lrcorner \quad \equiv \quad T \subseteq \mathbb{P} \llcorner \mathbb{B} \lrcorner$

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 $\cup$, $\cap$, $\sim$, 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	$\vee$	U	with	$true$
$\cap$	with	$\wedge$	$\{\}$	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.
- $E = U$ 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 $\cup$, $\cap$, $\sim$, 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	$\vee$	U	with	$true$
$\cap$	with	$\wedge$	$\{\}$	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.
- $E = U$ is valid iff E_p is valid.

Free theorems!

$$\begin{aligned} P \cap (P \cup Q) &= P \\ P \cap (Q \cup R) &= (P \cap Q) \cup (P \cap R) \\ P \cup (Q \cap R) &\subseteq P \cup Q \\ &\vdots \end{aligned}$$

Tuples and Tuple Types in CalcCheck

Tuples can have arbitrary "arity" at least 2.

Example: A triple with type: $\langle 2, true, \text{"Hello"} \rangle : \langle\!\langle \mathbb{Z}, \mathbb{B}, String \rangle\!\rangle$

Example: A seven-tuple: $\langle 3, true, 5 \triangleleft \epsilon, \langle 5, false \rangle, \text{"Hello"}, \{2, 8\}, \{42 \triangleleft \epsilon\} \rangle$

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.
- Tuple types are enclosed in $\langle\!\langle \ldots \rangle\!\rangle$.
- 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 Cartesian 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:** $S \times T \ = \ \{b, c \mid b \in S \wedge c \in T \bullet \langle b, c \rangle\}$

(14.4) **Membership:** $\langle b, c \rangle \in S \times T \quad \equiv \quad b \in S \ \wedge \ c \in T$

Cartesian product of types: Two-tuple types: $b : t_1 \ ; \ c : t_2 \quad \text{iff} \quad \langle b, c \rangle : \langle\!\langle t_1, t_2 \rangle\!\rangle$

Axiom, Pair projections:

$$\begin{aligned} fst &: \langle\!\langle t_1, t_2 \rangle\!\rangle \to t_1 & \quad fst\ \langle b, c \rangle &= b \\ snd &: \langle\!\langle t_1, t_2 \rangle\!\rangle \to t_2 & \quad snd\ \langle b, c \rangle &= c \end{aligned}$$

Pair equality: For $p, q : \langle\!\langle t_1, t_2 \rangle\!\rangle$,

$$p = q \quad \equiv \quad fst\ p = fst\ q \ \wedge \ snd\ p = snd\ q$$

Some Cross Product Theorems

(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 $\cup$:**

$$\begin{aligned} S \times (T \cup U) &= (S \times T) \cup (S \times U) \\ (S \cup T) \times U &= (S \times U) \cup (T \times U) \end{aligned}$$

(14.9) **Distributivity of $\times$ over $\cap$:**

$$\begin{aligned} S \times (T \cap U) &= (S \times T) \cap (S \times U) \\ (S \cap T) \times U &= (S \times U) \cap (T \times U) \end{aligned}$$

(14.10) **Distributivity of $\times$ over $-$:**

$$\begin{aligned} S \times (T - U) &= (S \times T) - (S \times U) \\ (S - T) \times U &= (S \times U) - (T \times U) \end{aligned}$$

(14.12) **Monotonicity:** $S \subseteq S' \ \wedge \ T \subseteq T' \quad \Rightarrow \quad S \times T \subseteq S' \times T'$

Pairs and Pair Projections

(14.2) **Axiom, Pair equality:** $\langle b, c \rangle = \langle b', c' \rangle \quad \equiv \quad b = b' \wedge c = c'$

(14.4p) **Axiom, Pair projections:**

$$\begin{aligned} fst &: t_1 \times t_2 \to t_1 & \quad fst\ \langle b, c \rangle &= b \\ snd &: t_1 \times t_2 \to t_2 & \quad snd\ \langle b, c \rangle &= c \end{aligned}$$

(14.2p) **Pair equality:** For $p, q : t_1 \times t_2$,

$$p = q \quad \equiv \quad fst\ p = fst\ q \ \wedge \ snd\ p = snd\ q$$

Proving (14.2e) **Pair extensionality:** $p = \langle fst\ p, snd\ p \rangle$:

$$\begin{aligned} & p = \langle fst\ p, snd\ p \rangle \\ =\ & \langle \text{(14.2p) Pair equality} \rangle \\ & fst\ p = fst\ \langle fst\ p, snd\ p \rangle \quad \wedge \quad snd\ p = snd\ \langle fst\ p, snd\ p \rangle \\ =\ & \langle \text{(14.4p) Pair projections} \rangle \\ & fst\ p = fst\ p \quad \wedge \quad snd\ p = snd\ p \\ =\ & \langle \text{(1.2) Reflexivity of equality, (3.38) Idempotency of } \wedge \rangle \\ & true \end{aligned}$$

Some Spice...

Converting between "different ways to take two arguments":

$$\begin{aligned} curry &: (\langle\!\langle A, B \rangle\!\rangle \to C) \to (A \to B \to C) \\ curry\ f\ x\ y &= f\ \langle x, y \rangle \\[1ex] uncurry &: (A \to B \to C) \to (\langle\!\langle A, B \rangle\!\rangle \to C) \\ uncurry\ g\ \langle x, y \rangle &= g\ x\ y \end{aligned}$$

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 2021

Wolfram Kahl

2021-10-28

Part 1: Relative Pseudocomplement

Plan for Today

- A Set Theory Exercise: Relative Pseudocomplement
- **Explicit Induction Principles**
- **Relations** (LADM Chapter 14)

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 $5 \le c$:

$$
\begin{aligned}
& 5 \le c \\
\equiv\ & \langle \text{The given equivalence, with } x := 5 \rangle \\
& 5 \le 5 \quad \text{— This is Reflexivity of } \le
\end{aligned}
$$

Proving $c \le 5$:

$$
\begin{aligned}
& c \le 5 \\
\equiv\ & \langle \text{Given equivalence, with } x := c \rangle \\
& c \le c \quad \text{— This is Reflexivity of } \le
\end{aligned}
$$

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** $A \twoheadrightarrow B$ of A with respect to B is defined by:

$$X \subseteq (A \twoheadrightarrow B) \quad \equiv \quad X \cap A \subseteq B$$

Calculate the **relative pseudocomplement** $A \twoheadrightarrow B$ as a set expression not using $\twoheadrightarrow$! That is:

Calculate $A \twoheadrightarrow B \quad = \quad$ **?**

Using set extensionality, that is:

Calculate $x \in A \twoheadrightarrow B \quad \equiv \quad x \in$ **?**

Characterisation of **relative pseudocomplement of sets:** $X \subseteq (A \twoheadrightarrow B) \quad \equiv \quad X \cap A \subseteq B$

$$
\begin{aligned}
& x \in A \twoheadrightarrow B \\
\equiv\ & \langle e \in S \equiv \{e\} \subseteq S \quad \text{— Exercise!} \rangle \\
& \{x\} \subseteq A \twoheadrightarrow B \\
\equiv\ & \langle \text{Def. } \twoheadrightarrow, \text{ with } X := \{x\} \rangle \\
& \{x\} \cap A \subseteq B \\
\equiv\ & \langle \text{(11.13) Subset} \rangle \\
& (\forall y \mid y \in \{x\} \cap A \bullet y \in B) \\
\equiv\ & \langle \text{(11.21) Intersection} \rangle \\
& (\forall y \mid y \in \{x\} \wedge y \in A \bullet y \in B) \\
\equiv\ & \langle y \in \{x\} \equiv y = x \quad \text{— Exercise!} \rangle \\
& (\forall y \mid y = x \wedge y \in A \bullet y \in B) \\
\equiv\ & \langle \text{(9.4b) Trading for } \forall, \text{ Def. } \notin \rangle \\
& (\forall y \mid y = x \bullet y \notin A \vee y \in B) \\
\equiv\ & \langle \text{(8.14) One-point rule} \rangle \\
& x \notin A \vee x \in B \\
\equiv\ & \langle \text{(11.17) Set complement, (11.20) Union} \rangle \\
& x \in {\sim}A \cup B
\end{aligned}
$$

Theorem: $A \twoheadrightarrow B \quad = \quad {\sim}A \cup B$

Characterisation of **relative pseudocomplement of sets:** $X \subseteq A \twoheadrightarrow B \quad \equiv \quad X \cap A \subseteq B$

Theorem "Pseudocomplement via $\cup$": $A \twoheadrightarrow B \quad = \quad {\sim}A \cup B$

Calculation:

$$
\begin{aligned}
& x \in A \twoheadrightarrow B \\
\equiv\ & \langle \text{Pseudocomplement via } \cup \rangle \\
& x \in {\sim}A \cup B \\
\equiv\ & \langle \text{(11.17) Set complement, (11.20) Union} \rangle \\
& \neg(x \in A) \vee x \in B \\
\equiv\ & \langle \text{(3.59) Definition of } \Rightarrow \rangle \\
& x \in A \Rightarrow x \in B
\end{aligned}
$$

Corollary "Membership in pseudocomplement":
$x \in A \twoheadrightarrow B \quad \equiv \quad x \in A \Rightarrow x \in B$

Easy to see: On sets, relative pseudocomplement wrt. $\{\}$ is complement:
$A \twoheadrightarrow \{\} \quad = \quad {\sim} A$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-28

Part 2: Explicit Induction Principles

Natural Numbers — Induction Principle

The set of all **natural numbers**, written $\mathbb{N}$, is **imductively defined** as generated from the following constructors:

- $0 : \mathbb{N}$
- $suc\ \_ : \mathbb{N} \to \mathbb{N}$

Induction principle for the natural numbers:

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

Natural Numbers — Explicit Induction Principle

Recall: Induction principle for the natural numbers:

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

As **inference rule**:

Informally:

$$\frac{P(0) \qquad \begin{array}{c} \ulcorner P(m) \urcorner \\ \vdots \\ P(suc\ m) \end{array}}{P(m)}$$

Formally:

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

As **axiom / theorem** — corresponding to LADM (12.5):

Axiom "Induction over $\mathbb{N}$":

$$
\begin{aligned}
& P[n := 0] \\
\Rightarrow\ & (\forall n : \mathbb{N} \mid P \bullet P[n := \mathsf{suc}\ n]) \\
\Rightarrow\ & (\forall n : \mathbb{N} \bullet P)
\end{aligned}
$$

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!

Sequences — Induction Principle

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 $xs : Seq\ A$ we have $P(xs)$. — then P holds for all sequences over A.

$$\begin{aligned} P[xs := \epsilon] \quad &\Rightarrow \quad (\forall\ xs : Seq\ A \mid P \bullet (\forall\ x : A \bullet P[xs := x \triangleleft xs]) \\ &\Rightarrow \quad (\forall\ xs : Seq\ A \bullet P) \end{aligned}$$

```
Axiom "Induction over sequences":
    P[xs ≔ ϵ]
    ⇒ (∀ xs : Seq A ❙ P • (∀ x : A • P[xs ≔ x ◃ xs]))
    ⇒ (∀ xs : Seq A • P)
```

$$P[m := 0] \quad \Rightarrow \quad (\forall\ m : \mathbb{N} \mid P \bullet P[m := suc\ m]) \quad \Rightarrow \quad (\forall\ m : \mathbb{N} \bullet P)$$

```
Axiom "Induction over ℕ":
   P[n ≔ 0]
   ⇒ (∀ n : ℕ ❙ P • P[n ≔ suc n])
   ⇒ (∀ n : ℕ • P)
```

Proving "Tail is different" `Using` the Ind. Principle

```
Axiom "Induction over sequences":
    P[xs ≔ ϵ]
    ⇒ (∀ xs : Seq A ❙ P • (∀ x : A • P[xs ≔ x ◃ xs]))
    ⇒ (∀ xs : Seq A • P)

Theorem (13.7) "Tail is different": ∀ xs : Seq A • ∀ x : A • x ◃ xs ≠ xs
Proof:
  Using "Induction over sequences":
    Subproof for `∀ x : A • x ◃ ϵ ≠ ϵ`:
      For any `x : A`:
          x ◃ ϵ ≠ ϵ
        ≡⟨ "Cons is not empty" ⟩
          true
    Subproof for `∀ xs : Seq A ❙             (∀ x : A • x ◃ xs ≠ xs)
                                  • (∀ z : A • (∀ x : A • x ◃ z ◃ xs ≠ z ◃ xs))`:
      For any `xs : Seq A` satisfying "Ind. Hyp." `(∀ x : A • x ◃ xs ≠ xs)`:
        For any `z : A`, `x : A`:
            x ◃ z ◃ xs ≠ z ◃ xs
          ≡⟨ "Definition of ≠", "Injectivity of ◃" ⟩
            ¬ (x = z ∧ z ◃ xs = xs)
          ⇐⟨ "Consequence", "De Morgan", "Weakening", "Definition of ≠" ⟩
            z ◃ xs ≠ xs
          ≡⟨ Assumption "Ind. Hyp." ⟩
            true
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-10-28

Part 3: Relations

Predicates and Tuple Types — Relations are Tuple Sets

$$\_called\_ \quad : \quad P \to P \to \mathbb{B}$$

$(uncurry\ \_called\_) : \langle P, P \rangle \to \mathbb{B}$ is the **characteristic function** of the set

$$R_{called} \quad : \quad \textbf{set}\ \langle P, P \rangle$$

$$R_{called} \quad = \quad \{p, q : P \mid p\ called\ q \bullet \langle p, q \rangle\}$$

R_{called} is a **(binary) relation**.

$$D \quad : \quad P \to City \to City \to \mathbb{B}$$

$$D\ p\ a\ b \quad \equiv \quad \boxed{p \text{ drove from } a \text{ to } b}$$

$$R_D \quad : \quad \textbf{set}\ \langle P, City, City \rangle$$

$$R_D \quad = \quad \{p : P; a, b : City \mid D\ p\ a\ b \bullet \langle p, a, b \rangle\}$$

R_D is a **(ternary) relation**.

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)

Relations

- LADM: A **relation** on $B_1 \times \cdots \times B_n$ is a subset of $B_1 \times \cdots \times B_n$ — where $B_1, \ldots, B_n$ are sets
- CalcCheck: Normally: A **relation** on $\langle t_1, \ldots, t_n \rangle$ is a subset of $\llcorner \langle t_1, \ldots, t_n \rangle \lrcorner$, that is, an item of type $\textbf{set}\ \langle t_1, \ldots, t_n \rangle$ — where $t_1, \ldots, t_n$ are types
- A relation on the tuple (Cartesian product) type $\langle t_1, \ldots, t_n \rangle$ is an **n-ary relation**.

  "Tables" in relational databases are n-ary relations.
- A relation on the pair (Cartesian product) type $\langle t_1, t_2 \rangle$ is a **binary relation**.
- The **type** of binary relations on $\langle t_1, t_2 \rangle$ is written $t_1 \leftrightarrow t_2$, with

$$t_1 \leftrightarrow t_2 \quad = \quad \textbf{set}\ \langle t_1, t_2 \rangle \qquad \text{— } \texttt{\textbackslash rel}$$

- The **set** of binary relations on $B \times C$ is written $B \leftrightarrow\!\!\!\!\circ\ C$, with

$$B \leftrightarrow\!\!\!\!\circ\ C \quad = \quad \mathbb{P}\,(B \times C) \qquad \text{— } \texttt{\textbackslash Rel}$$

What is a Relation?

A **relation**
is a subset
of a Cartesian product.

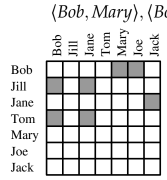
What is a Binary Relation?

A **binary relation**
is a set of pairs.

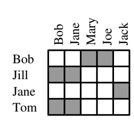
Visualising Binary Relations

$\llcorner Person \lrcorner = \{Bob, Jill, Jane, Tom, Mary, Joe, Jack\}$

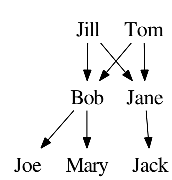
$parentOf = \{\langle Jill, Bob \rangle, \langle Jill, Jane \rangle, \langle Tom, Bob \rangle, \langle Tom, Jane \rangle, \langle Bob, Mary \rangle, \langle Bob, Joe \rangle, \langle Jane, Jack \rangle\}$



$parentOf : Person \leftrightarrow Person$



$parentOf \in (parents \leftrightarrow\!\!\!\!\circ\ children)$



$$\begin{aligned} parents \quad &= \quad Dom\ parentOf \quad = \quad \{Bob, Jill, Jane, Tom\} \\ children \quad &= \quad Ran\ parentOf \quad = \quad \{Bob, Jane, Mary, Joe, Jack\} \end{aligned}$$

Expressing relationship: $Jill \ (\!|\, parentOf \,|\!)\ Bob \quad \equiv \quad \langle Jill, Bob \rangle \in parentOf$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-01

Part 1: Relation Operations

How can you simplify if you know $P_1 \Rightarrow P_2$?

$$\begin{array}{ll} & \vdots \\ \equiv & \langle \ldots \rangle \\ & \ldots \vee P_1 \vee P_2 \vee \ldots \\ \equiv & \langle \quad ? \quad \rangle \\ & ? \end{array} \qquad \begin{array}{ll} & \vdots \\ \equiv & \langle \ldots \rangle \\ & \ldots \wedge P_1 \wedge P_2 \wedge \ldots \\ \equiv & \langle \quad ? \quad \rangle \\ & ? \end{array}$$

$$\begin{array}{ll} & \vdots \\ \equiv & \langle \ldots \rangle \\ & \ldots \vee P_1 \vee P_2 \vee \ldots \\ \equiv & \langle \text{“Reason for } P_1 \Rightarrow P_2\text{”} \\ & \quad \text{with (3.57)} \rangle \\ & \ldots \vee P_2 \vee \ldots \end{array} \qquad \begin{array}{ll} & \vdots \\ \equiv & \langle \ldots \rangle \\ & \ldots \wedge P_1 \wedge P_2 \wedge \ldots \\ \equiv & \langle \text{“Reason for } P_1 \Rightarrow P_2\text{”} \\ & \quad \text{with (3.60)} \rangle \\ & \ldots \wedge P_1 \wedge \ldots \end{array}$$

How can you simplify if you know $S_1 \subseteq S_2$?

$$\begin{array}{ll} & \vdots \\ = & \langle \ldots \rangle \\ & \ldots \cup S_1 \cup S_2 \cup \ldots \\ = & \langle \quad ? \quad \rangle \\ & ? \end{array} \qquad \begin{array}{ll} & \vdots \\ = & \langle \ldots \rangle \\ & \ldots \cap S_1 \cap S_2 \cap \ldots \\ = & \langle \quad ? \quad \rangle \\ & ? \end{array}$$

⟶ Reference Notebook 7.1: Set Theory

- “Set inclusion via ∪”
- “Set inclusion via ∩”

Plan for Today

- Relations
  - Relationship notation and reasoning
  - Set operations as relation operations
  - Set-theoretic definition of relational operations: Converse, composition

Relations

- LADM: A **relation** on $B_1 \times \cdots \times B_n$ is a subset of $B_1 \times \cdots \times B_n$
  — where $B_1, \ldots, B_n$ are sets
- CalcCheck: Normally: A **relation** on $\langle t_1, \ldots, t_n \rangle$ is a subset of $\llcorner \langle t_1, \ldots, t_n \rangle \lrcorner$,
  that is, an item of type **set** $\langle t_1, \ldots, t_n \rangle$
  — where $t_1, \ldots, t_n$ are types
- A relation on the tuple (Cartesian product) type $\langle t_1, \ldots, t_n \rangle$ is an ***n*-ary relation**.
  “Tables” in relational databases are n-ary relations.
- A relation on the pair (Cartesian product) type $\langle t_1, t_2 \rangle$ is a **binary relation**.
- The **type** of binary relations on $\langle t_1, t_2 \rangle$ is written $t_1 \leftrightarrow t_2$, with

  $t_1 \leftrightarrow t_2 \quad = \quad \textbf{set}\ \langle t_1, t_2 \rangle$ — `\rel`
- The **set** of binary relations on $B \times C$ is written $B \leftrightarrow\!\!\!\leftrightarrow C$, with

  $B \leftrightarrow\!\!\!\leftrightarrow C \quad = \quad \mathbb{P}(B \times C)$ — `\Rel`

What is a Relation?

A **relation**
is a subset
of a Cartesian product.

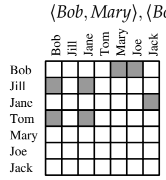
What is a Binary Relation?

A **binary relation**
is a set of pairs.

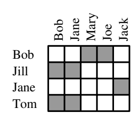
Visualising Binary Relations

$\llcorner Person \lrcorner = \{Bob, Jill, Jane, Tom, Mary, Joe, Jack\}$

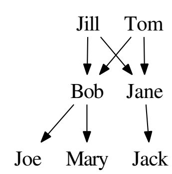
$parentOf = \{\langle Jill, Bob\rangle, \langle Jill, Jane\rangle, \langle Tom, Bob\rangle, \langle Tom, Jane\rangle,$
$\langle Bob, Mary\rangle, \langle Bob, Joe\rangle, \langle Jane, Jack\rangle\}$



$parentOf : Person \leftrightarrow Person$



$parentOf \in (parents \leftrightarrow\!\!\!\leftrightarrow children)$



$parents \quad = \quad Dom\ parentOf \quad = \quad \{Bob, Jill, Jane, Tom\}$
$children \quad = \quad Ran\ parentOf \quad = \quad \{Bob, Jane, Mary, Joe, Jack\}$

Expressing relationship: $Jill \,(\!\, parentOf \,)\!\, Bob \quad \equiv \quad \langle Jill, Bob\rangle \in parentOf$

(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** (N, E) is a pair consisting of

- a set N, the elements of which are called “nodes”, and
- a relation E with $E \in N \leftrightarrow\!\!\!\leftrightarrow N$,
  the element pairs of which are called “edges”.

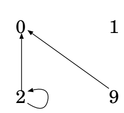
Simple Graphs: Example

Formally: 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 $E \in N \leftrightarrow\!\!\!\leftrightarrow N$, the element pairs of which are called “edges”.

Example: $\quad G_1 = (\{2, 0, 1, 9\}, \{\langle 2, 0\rangle, \langle 9, 0\rangle, \langle 2, 2\rangle\})$

Graphs are normally visualised via **graph drawings**:



Simple graphs are exactly relations!

Reasoning with relations is reasoning about graphs!

Binary Relations, Relationship

Consider $R : t_1 \leftrightarrow t_2$ and $x : t_1$ and $y : t_2$.

$$
\begin{array}{ll}
& R \in \llcorner\, t_1 \leftrightarrow t_2 \,\lrcorner \\
\equiv & \langle \text{Def. } \leftrightarrow \rangle \\
& R \in \llcorner\, \mathbf{set}\, (t_1, t_2) \,\lrcorner \\
\equiv & \langle \text{Membership in } \llcorner\, \mathbf{set}\ \_\ \lrcorner \rangle \\
& R \subseteq \llcorner\, (t_1, t_2) \,\lrcorner \\
\equiv & \langle \text{Def. } \mathbf{set}, \text{Def. } \times, \text{Def. } \llcorner\ \lrcorner \rangle \\
& R \subseteq \llcorner\, t_1 \,\lrcorner \times \llcorner\, t_2 \,\lrcorner
\end{array}
$$

Note that for a type t, the universal set

$U : \mathbf{set}\ t$

is the set of all members of t.

Or, $(U : \mathbf{set}\ t)$ is "type t as a set".

We **abbreviate**: $\llcorner\, t \,\lrcorner := (U : \mathbf{set}\ t)$,
(`\llcorner`...`\lrcorner`) and have:

$$S \in \llcorner\, \mathbf{set}\ t \,\lrcorner \quad \equiv \quad S \subseteq \llcorner\, t \,\lrcorner$$

Notations for "x is in relation R with y":

- explicit membership notation: $\langle x, y \rangle \in R$
- ambiguous traditional infix notation: $x\,R\,y$
- CalcCheck: $x \,(\!|\, R \,|\!)\, y \quad \equiv \quad \langle x, y \rangle \in R$

$\_(\!|\_|\!)\_ : t_1 \to (t_1 \leftrightarrow t_2) \to t_2 \to \mathbb{B}$ — calculational!

Experimental Key Bindings

— US keyboard only! Firefox only?

- `Alt-=` for $\equiv$ in addition to `\==`
- `Alt-<` for $\langle$ in addition to `\<`
- `Alt->` for $\rangle$ in addition to `\>`
- `Alt-(` for $(\!|$ in addition to `\((`
- `Alt-)` for $|\!)$ in addition to `\))`

Set Operations Used as Operations on Binary Relations

Relation union:
$$\langle u, v \rangle \in (R \cup S) \quad \equiv \quad \langle u, v \rangle \in R \lor \langle u, v \rangle \in S$$
$$u \,(\!|\, R \cup S \,|\!)\, v \quad \equiv \quad u \,(\!|\, R \,|\!)\, v \lor u \,(\!|\, S \,|\!)\, v$$

Relation intersection: $u \,(\!|\, R \cap S \,|\!)\, v \quad \equiv \quad u \,(\!|\, R \,|\!)\, v \land u \,(\!|\, S \,|\!)\, v$

Relation difference: $u \,(\!|\, R - S \,|\!)\, v \quad \equiv \quad u \,(\!|\, R \,|\!)\, v \land \neg (u \,(\!|\, S \,|\!)\, v)$

Relation complement: $u \,(\!|\, {\sim} R \,|\!)\, v \quad \equiv \quad \neg (u \,(\!|\, R \,|\!)\, v)$

Relation extensionality:
$$R = S \quad \equiv \quad (\forall x \bullet \forall y \bullet x \,(\!|\, R \,|\!)\, y \equiv x \,(\!|\, S \,|\!)\, y)$$
$$R = S \quad \equiv \quad (\forall x, y \bullet x \,(\!|\, R \,|\!)\, y \equiv x \,(\!|\, S \,|\!)\, y)$$

Relation inclusion:
$$R \subseteq S \quad \equiv \quad (\forall x \bullet \forall y \bullet x \,(\!|\, R \,|\!)\, y \Rightarrow x \,(\!|\, S \,|\!)\, y)$$
$$R \subseteq S \quad \equiv \quad (\forall x \bullet \forall y \mid x \,(\!|\, R \,|\!)\, y \bullet x \,(\!|\, S \,|\!)\, y)$$
$$R \subseteq S \quad \equiv \quad (\forall x, y \bullet x \,(\!|\, R \,|\!)\, y \Rightarrow x \,(\!|\, S \,|\!)\, y)$$
$$R \subseteq S \quad \equiv \quad (\forall x, y \mid x \,(\!|\, R \,|\!)\, y \bullet x \,(\!|\, S \,|\!)\, y)$$

Empty and Universal Binary Relations

- The **empty relation** on (t_1, t_2) is $\{\} : t_1 \leftrightarrow t_2$
$$x \,(\!|\, \{\} \,|\!)\, y \quad \equiv \quad \mathit{false}$$
$$\langle x, y \rangle \in \{\} \quad \equiv \quad \mathit{false}$$

- The **universal relation** on (t_1, t_2) is $\llcorner\, (t_1, t_2) \,\lrcorner : t_1 \leftrightarrow t_2$ or $U : t_1 \leftrightarrow t_2$
$$x \,(\!|\, \llcorner\, (t_1, t_2) \,\lrcorner \,|\!)\, y \quad \equiv \quad \mathit{true} \qquad x \,(\!|\, U \,|\!)\, y \quad \equiv \quad \mathit{true}$$
$$\langle x, y \rangle \in \llcorner\, (t_1, t_2) \,\lrcorner \quad \equiv \quad \mathit{true} \qquad \langle x, y \rangle \in U \quad \equiv \quad \mathit{true}$$

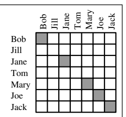
- The **universal relation on** $B \times C$ is $B \times C$
$$x \,(\!|\, B \times C \,|\!)\, y \quad \equiv \quad x \in B \land y \in C$$

(14.4) $\langle x, y \rangle \in B \times C \quad \equiv \quad x \in B \land y \in C$

Sub-identity and Identity Relations

- The **(sub-)identity relation** on $B : \mathbf{set}\ t$ is $\mathrm{id}\ B : t \leftrightarrow t$

id *children* =



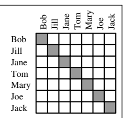
$\mathrm{id}\ B = \{x : t \mid x \in B \bullet \langle x, x \rangle\}$:
$$x \,(\!|\, \mathrm{id}\ B \,|\!)\, y \quad \equiv \quad x = y \in B$$
$$\langle x, y \rangle \in \mathrm{id}\ B \quad \equiv \quad x = y \land y \in B$$

— LADM writes ι_B

— Writing "id B" follows the Z notation

- The **identity relation** on $t : Type$ is $\mathbb{I} : t \leftrightarrow t$ with $\mathbb{I} = \mathrm{id}\ U$
$$x \,(\!|\, \mathbb{I} \,|\!)\, y \quad \equiv \quad x = y$$
$$\langle x, y \rangle \in \mathbb{I} \quad \equiv \quad x = y$$

$(\mathbb{I} : Person \leftrightarrow Person)$ =



- **The "id" and "$\mathbb{I}$" notations are different from previous years!**

Domain and Range of Binary Relations

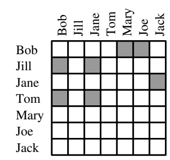
For $R : t_1 \leftrightarrow t_2$, we define $Dom\ R : \mathbf{set}\ t_1$ and $Ran\ R : \mathbf{set}\ t_1$ as follows:

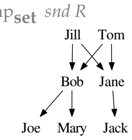
(14.16) $Dom\ R = \{x : t_1 \mid (\exists y : t_2 \bullet x \,(\!|\, R \,|\!)\, y)\} = \{p \mid p \in R \bullet \mathit{fst}\ p\} = \mathrm{map}_{\mathbf{set}}\ \mathit{fst}\ R$

(14.17) $Ran\ R = \{y : t_2 \mid (\exists x : t_1 \bullet x \,(\!|\, R \,|\!)\, y)\} = \{p \mid p \in R \bullet \mathit{snd}\ p\} = \mathrm{map}_{\mathbf{set}}\ \mathit{snd}\ R$

"Membership in \`$Dom$\`":
$$x \in Dom\ R \quad \equiv \quad (\exists y : t_2 \bullet x \,(\!|\, R \,|\!)\, y)$$

"Membership in \`$Ran$\`":
$$y \in Ran\ R \quad \equiv \quad (\exists x : t_1 \bullet x \,(\!|\, R \,|\!)\, y)$$





$$\mathit{parents} = Dom\ \mathit{parentOf} = \{Bob, Jill, Jane, Tom\}$$
$$\mathit{children} = Ran\ \mathit{parentOf} = \{Bob, Jane, Mary, Joe, Jack\}$$

Formalise Without Quantifiers!

$$
\begin{array}{lll}
P & = & \text{type of persons} \\
C & : & P \leftrightarrow P \\
p \,(\!|\, C \,|\!)\, q & \equiv & p \text{ called } q
\end{array}
$$

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.

   $Helen \in Dom\ C$

2. For everybody, there is somebody they haven't called.

   $Dom\ ({\sim} C) = \llcorner\, P \,\lrcorner$

   $Dom\ ({\sim} C) = U$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-01

Part 2: Relational Operations: Converse, Composition

Relation-Algebraic Operations: Operations on Relations

- Set operations ${\sim}, \cup, \cap, -, \multimap$ 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": $\quad c \,(\!|\, R^{\smile} \,|\!)\, b \quad \equiv \quad b \,(\!|\, R \,|\!)\, c$
- 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":
  $$b \,(\!|\, R \,⨾\, S \,|\!)\, d \quad \equiv \quad (\exists c : C \bullet b \,(\!|\, R \,|\!)\, c \,(\!|\, S \,|\!)\, d)$$

The resulting **relation algebra**

- allows concise formalisations **without quantifications**,
- enables simple calculational proofs.

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) $\langle c, b \rangle \in R^{\smile} \quad \equiv \quad \langle b, c \rangle \in R \qquad$ (for $b : B$ and $c : C$)

(14.18) $c \,(\!|\, R^{\smile} \,|\!)\, b \quad \equiv \quad b \,(\!|\, R \,|\!)\, c \qquad$ (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) If $R \in B \leftrightarrow C$, then $R^{\smile} \in C \leftrightarrow B$

(d) $(R^{\smile})^{\smile} = R$

(e) $R \subseteq S \quad \equiv \quad R^{\smile} \subseteq S^{\smile}$

Proving Self-inverse of Converse: $(R^{\smile})^{\smile} = R$

$$\begin{aligned}
& (R^{\smile})^{\smile} = R \\
\equiv\ & \langle \text{ Relation extensionality } \rangle \\
& \forall\, x, y \bullet x \,(\!|\, (R^{\smile})^{\smile} \,|\!)\, y \equiv x \,(\!|\, R \,|\!)\, y \\
\equiv\ & \langle \ldots \rangle \\
& \mathit{true}
\end{aligned}$$

Using "Relation extensionality":
Subproof for `$\forall\, x, y \bullet x \,(\!|\, (R^{\smile})^{\smile} \,|\!)\, y \quad \equiv \quad x \,(\!|\, R \,|\!)\, y$`:
For any x, y:

$$\begin{aligned}
& x \,(\!|\, (R^{\smile})^{\smile} \,|\!)\, y \\
\equiv\ & \langle \text{ Converse } \rangle \\
& y \,(\!|\, R^{\smile} \,|\!)\, x \\
\equiv\ & \langle \text{ Converse } \rangle \\
& x \,(\!|\, R \,|\!)\, y
\end{aligned}$$

Proving Isotonicity of Converse

Proving $R \subseteq S \quad \equiv \quad R^{\smile} \subseteq S^{\smile}$:

$$\begin{aligned}
& R^{\smile} \subseteq S^{\smile} \\
\equiv\ & \langle \text{ Relation inclusion } \rangle \\
& \forall\, y, x \mid y \,(\!|\, R^{\smile} \,|\!)\, x \bullet y \,(\!|\, S^{\smile} \,|\!)\, x \\
\equiv\ & \langle \text{ Converse, dummy permutation } \rangle \\
& \forall\, x, y \mid x \,(\!|\, R \,|\!)\, y \bullet x \,(\!|\, S \,|\!)\, y \\
\equiv\ & \langle \text{ Relation inclusion } \rangle \\
& R \subseteq S
\end{aligned}$$

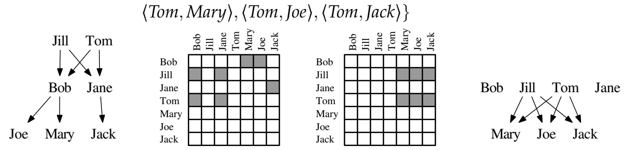
Operations on Relations: Composition $\quad B \xrightarrow{R} C \xrightarrow{S} D$

If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, then their **composition** $R \fatsemi S : B \leftrightarrow D$ is defined by:

(14.20) $\quad b \,(\!|\, R \fatsemi S \,|\!)\, d \ \equiv\ (\exists c : C \bullet b \,(\!|\, R \,|\!)\, c \,(\!|\, S \,|\!)\, d) \qquad$ (for $b : B, d : D$)

(14.20) $\quad b \,(\!|\, R \fatsemi S \,|\!)\, d \ \equiv\ (\exists c : C \bullet b \,(\!|\, R \,|\!)\, c \wedge c \,(\!|\, S \,|\!)\, d) \qquad$ (for $b : B, d : D$)

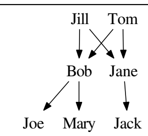
$$\begin{aligned}
\mathit{parentOf} = \ & \{\langle \mathit{Jill}, \mathit{Bob}\rangle, \langle \mathit{Jill}, \mathit{Jane}\rangle, \langle \mathit{Tom}, \mathit{Bob}\rangle, \langle \mathit{Tom}, \mathit{Jane}\rangle, \\
& \langle \mathit{Bob}, \mathit{Mary}\rangle, \langle \mathit{Bob}, \mathit{Joe}\rangle, \langle \mathit{Jane}, \mathit{Jack}\rangle\} \\
\mathit{grandparentOf} \ = \ & \mathit{parentOf} \fatsemi \mathit{parentOf} \\
= \ & \{\langle \mathit{Jill}, \mathit{Mary}\rangle, \langle \mathit{Jill}, \mathit{Joe}\rangle, \langle \mathit{Jill}, \mathit{Jack}\rangle \\
& \langle \mathit{Tom}, \mathit{Mary}\rangle, \langle \mathit{Tom}, \mathit{Joe}\rangle, \langle \mathit{Tom}, \mathit{Jack}\rangle\}
\end{aligned}$$

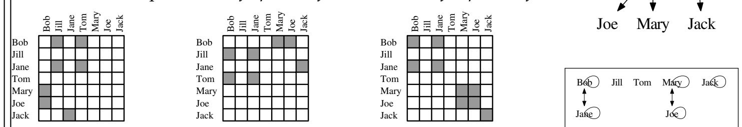


Combining Several Operations

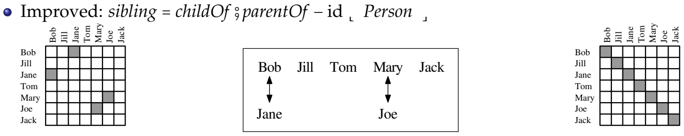
How to define siblings?

- First attempt: $\mathit{childOf} \fatsemi \mathit{parentOf}$, with $\mathit{childOf} = \mathit{parentOf}^{\smile}$





- Improved: $\mathit{sibling} = \mathit{childOf} \fatsemi \mathit{parentOf} - \mathrm{id}\, \llcorner\, \mathit{Person}\, \lrcorner$



Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-02

Part 1: Relation-Algebraic Formalisation Examples

Plan for Today

- Relations
  - Some relation-algebraic formalisation examples
  - Some theorems about relation composition $\fatsemi$
  - Classes of relations
- General Induction

$$\begin{aligned}
P &= \text{type of persons} \\
C &: P \leftrightarrow P \qquad \text{— “called”} \\
B &: P \leftrightarrow P \qquad \text{— “brother of”} \\
\mathit{Aos} &: P \\
\mathit{Jun} &: P
\end{aligned}$$

Convert into English (via predicate logic):

$\mathit{Aos} \,(\!|\, C \,|\!)\, \mathit{Jun}$

$\mathit{Aos} \,(\!|\, C \fatsemi B \,|\!)\, \mathit{Jun}$

$\mathit{Aos} \,(\!|\, {\sim}(C \fatsemi {\sim} B) \,|\!)\, \mathit{Jun}$

$\mathit{Aos} \,(\!|\, {\sim}({\sim} C \fatsemi B) \,|\!)\, \mathit{Jun}$

$\mathit{Aos} \,(\!|\, {\sim}((C \cap {\sim}(B \fatsemi C^{\smile})) \fatsemi {\sim} B) \,|\!)\, \mathit{Jun}$

$(B \fatsemi (\{\mathit{Jun}\} \times \llcorner\, P\, \lrcorner)) \cap (C \fatsemi C^{\smile}) \quad \subseteq \quad \mathrm{id}\, \llcorner\, P\, \lrcorner$

Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall\, x, y \bullet x \,(\!|\, R \,|\!)\, y \equiv x \,(\!|\, S \,|\!)\, y)$
$R \subseteq S$	$\equiv$	$(\forall\, x, y \bullet x \,(\!|\, R \,|\!)\, y \Rightarrow x \,(\!|\, S \,|\!)\, y)$
$u \,(\!|\, \{\} \,|\!)\, v$	$\equiv$	false
$u \,(\!|\, U \,|\!)\, v$	$\equiv$	true
$u \,(\!|\, A \times B \,|\!)\, v$	$\equiv$	$u \in A \wedge v \in B$
$u \,(\!|\, {\sim} S \,|\!)\, v$	$\equiv$	$\neg(u \,(\!|\, S \,|\!)\, v)$
$u \,(\!|\, S \cup T \,|\!)\, v$	$\equiv$	$u \,(\!|\, S \,|\!)\, v \vee u \,(\!|\, T \,|\!)\, v$
$u \,(\!|\, S \cap T \,|\!)\, v$	$\equiv$	$u \,(\!|\, S \,|\!)\, v \wedge u \,(\!|\, T \,|\!)\, v$
$u \,(\!|\, S - T \,|\!)\, v$	$\equiv$	$u \,(\!|\, S \,|\!)\, v \wedge \neg(u \,(\!|\, T \,|\!)\, v)$
$u \,(\!|\, S \rightarrow T \,|\!)\, v$	$\equiv$	$u \,(\!|\, S \,|\!)\, v \Rightarrow (u \,(\!|\, T \,|\!)\, v)$
$u \,(\!|\, \mathbb{I} \,|\!)\, v$	$\equiv$	$u = v$
$u \,(\!|\, \mathrm{id}\ A \,|\!)\, v$	$\equiv$	$u = v \in A$
$u \,(\!|\, R^{\smile} \,|\!)\, v$	$\equiv$	$v \,(\!|\, R \,|\!)\, u$
$u \,(\!|\, R \fatsemi S \,|\!)\, v$	$\equiv$	$(\exists\, x \bullet u \,(\!|\, R \,|\!)\, x \,(\!|\, S \,|\!)\, v)$

$$\begin{aligned}
P &= \text{type of persons} \\
C &: P \leftrightarrow P \qquad \text{— “called”} \\
B &: P \leftrightarrow P \qquad \text{— “brother of”} \\
\mathit{Aos} &: P \\
\mathit{Jun} &: P
\end{aligned}$$

Convert into English (via predicate logic):

$$\begin{aligned}
& \mathit{Aos} \,(\!|\, C \fatsemi B \,|\!)\, \mathit{Jun} \\
=\ & \langle \text{ (14.20) Relation composition } \rangle \\
& (\exists\, b \bullet \mathit{Aos} \,(\!|\, C \,|\!)\, b \,(\!|\, B \,|\!)\, \mathit{Jun})
\end{aligned}$$

"Aos called some brother of Jun."

"Aos called a brother of Jun."

$$\begin{aligned}
& \mathit{Aos} \,(\!|\, {\sim}(C \fatsemi {\sim} B) \,|\!)\, \mathit{Jun} \\
=\ & \langle \text{ (11.17r) Relation complement } \rangle \\
& \neg(\mathit{Aos} \,(\!|\, C \fatsemi {\sim} B \,|\!)\, \mathit{Jun}) \\
=\ & \langle \text{ (14.20) Relation composition } \rangle \\
& \neg(\exists\, p \bullet \mathit{Aos} \,(\!|\, C \,|\!)\, p \,(\!|\, {\sim} B \,|\!)\, \mathit{Jun}) \\
=\ & \langle \text{ (11.17r) Relation complement } \rangle \\
& \neg(\exists\, p \bullet \mathit{Aos} \,(\!|\, C \,|\!)\, p \wedge \neg(p \,(\!|\, B \,|\!)\, \mathit{Jun})) \\
=\ & \langle \text{ (9.18b) Generalised De Morgan } \rangle \\
& (\forall\, p \bullet \neg(\mathit{Aos} \,(\!|\, C \,|\!)\, p \wedge \neg(p \,(\!|\, B \,|\!)\, \mathit{Jun}))) \\
=\ & \langle \text{ (3.47) De Morgan, (3.12) Double negation } \rangle \\
& (\forall\, p \bullet \neg(\mathit{Aos} \,(\!|\, C \,|\!)\, p) \vee p \,(\!|\, B \,|\!)\, \mathit{Jun}) \\
=\ & \langle \text{ (9.3a) Trading for } \forall \rangle \\
& (\forall\, p \mid \mathit{Aos} \,(\!|\, C \,|\!)\, p \bullet p \,(\!|\, B \,|\!)\, \mathit{Jun})
\end{aligned}$$

"Everybody Aos called is a brother of Jun."

"Aos called only brothers of Jun."

Formalise Without Quantifiers! (2)

P := type of persons
C : $P \leftrightarrow P$
$p \,(\!C\!)\, q$:= 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.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-02

Part 2: Some Properties of Relation Composition

First Simple Properties of Composition

If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, then their **composition** $R \,⨾\, S : B \leftrightarrow D$ is defined by:

(14.20) $b \,(\!R ⨾ S\!)\, d \equiv (\exists c : C \bullet b \,(\!R\!)\, c \wedge c \,(\!S\!)\, d)$ (for $b : B, d : D$)

(14.22) **Associativity of ⨾:** $Q ⨾ (R ⨾ S) = (Q ⨾ R) ⨾ S$

Left- and Right-identities of ⨾: If $R : B \leftrightarrow C$, then:

$$\mathrm{id}\llcorner B \lrcorner ⨾ R = R = R ⨾ \mathrm{id}\llcorner C \lrcorner$$

We defined: $\mathbb{I} = \mathrm{id}\ U$

Relationship via $\mathbb{I}$: $x \,(\!\mathbb{I}\!)\, y \equiv x = y$

$\mathbb{I}$ is "the" identity of composition:

Identity of ⨾: $\mathbb{I} ⨾ R = R = R ⨾ \mathbb{I}$

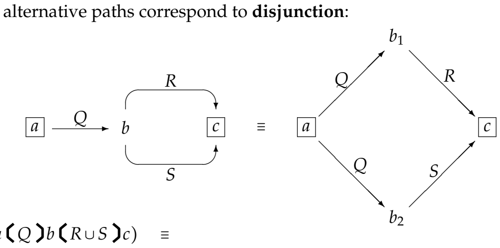
Contravariance: $(R ⨾ S)^{\smile} = S^{\smile} ⨾ R^{\smile}$

Distributivity of Relation Composition over Union

Composition distributes over **union** from both sides:

(14.23) $Q ⨾ (R \cup S) = Q ⨾ R \cup Q ⨾ S$
$(P \cup Q) ⨾ R = P ⨾ R \cup Q ⨾ R$

In **control flow** diagrams (NFA) — boxed variables are free; others existentially quantified; alternative paths correspond to **disjunction**:



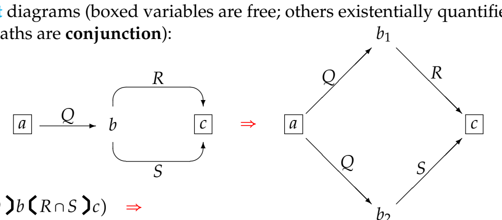
$(\exists b \bullet a \,(\!Q\!)\, b \,(\!R \cup S\!)\, c) \equiv$
$(\exists b_1, b_2 \bullet a \,(\!Q\!)\, b_1 \,(\!R\!)\, c \vee a \,(\!Q\!)\, b_2 \,(\!S\!)\, c)$

Sub-Distributivity of Composition over Intersection

Composition **sub**-distributes over **intersection** from both sides:

(14.24) $Q ⨾ (R \cap S) \subseteq Q ⨾ R \cap Q ⨾ S$
$(P \cap Q) ⨾ R \subseteq P ⨾ R \cap Q ⨾ R$

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a \,(\!Q\!)\, b \,(\!R \cap S\!)\, c) \Rightarrow$
$(\exists b_1, b_2 \bullet a \,(\!Q\!)\, b_1 \,(\!R\!)\, c \wedge a \,(\!Q\!)\, b_2 \,(\!S\!)\, c)$

Counterexample for $\Leftarrow$:
Q := neighbour of R := brother of S := parent of

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 $Q ⨾ S \subseteq R ⨾ S$:

$R ⨾ S$
$= \langle$ Assumption $Q \cup R = R$ $\rangle$
$(Q \cup R) ⨾ S$
$= \langle$ (14.23) Distributivity of ⨾ over $\cup$ $\rangle$
$Q ⨾ S \cup R ⨾ S$
$\supseteq \langle$ (11.31) Strengthening $S \subseteq S \cup T$ $\rangle$
$Q ⨾ S$

Logical Reasoning for Computer Science

COMPSCI 2LC3

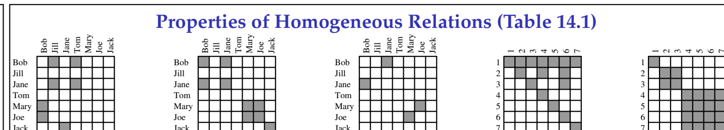
McMaster University, Fall 2021

Wolfram Kahl

2021-11-02

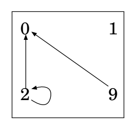
Part 3: Classes of Relations

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.)

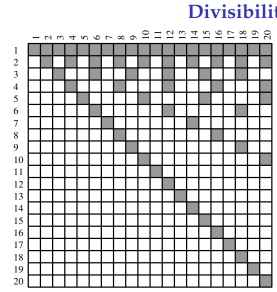
reflexive	$\mathbb{I} \subseteq R$	$(\forall b : B \bullet b \,(\!R\!)\, b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall b : B \bullet \neg(b \,(\!R\!)\, b))$
symmetric	$R^{\smile} = R$	$(\forall b, c : B \bullet b \,(\!R\!)\, c \equiv c \,(\!R\!)\, b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall b, c \bullet b \,(\!R\!)\, c \wedge c \,(\!R\!)\, b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall b, c : B \bullet b \,(\!R\!)\, c \Rightarrow \neg(c \,(\!R\!)\, b))$
transitive	$R ⨾ R \subseteq R$	$(\forall b, c, d \bullet b \,(\!R\!)\, c \wedge c \,(\!R\!)\, d \Rightarrow b \,(\!R\!)\, d)$

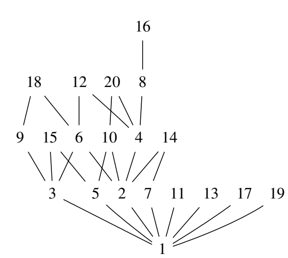
R is an **equivalence (relation) on** B iff it is reflexive, transitive, and symmetric.

R is a **(partial) order on** B iff it is reflexive, transitive, and antisymmetric.
(E.g., $\leq, \geq, \subseteq, \supseteq, \mid$)

R is a **strict-order on** B iff it is irreflexive, transitive, and asymmetric.
(E.g., $<, >, \subset, \supset$)

Divisibility Order with Hasse Diagram

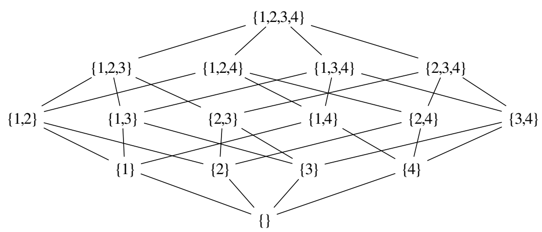




Hasse diagram for an **order**:

- Edge direction is **upwards** — **antisymmetric**
- Loops not drawn — **reflexive**
- Transitive edges not drawn — **transitive**

Inclusion Order on Powerset 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:

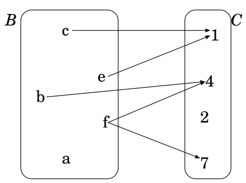
univalent determinate	$R^{\smile} \fatsemi R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b ⦅R⦆ c_1 \wedge b ⦅R⦆ c_2 \Rightarrow c_1 = c_2$
total	$\mathrm{Dom}\,R = \llcorner B \lrcorner$ $\mathbb{I} \subseteq R \fatsemi R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b ⦅R⦆ c)$
injective	$R \fatsemi R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 ⦅R⦆ c \wedge b_2 ⦅R⦆ c \Rightarrow b_1 = b_2$
surjective	$\mathrm{Ran}\,R = \llcorner C \lrcorner$ $\mathbb{I} \subseteq R^{\smile} \fatsemi R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b ⦅R⦆ c)$
a **mapping**	iff it is univalent and total	
bijective	iff it is injective and surjective	

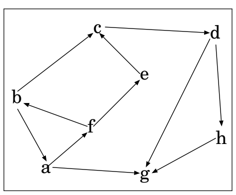
Univalent relations are also called **(partial) functions**.

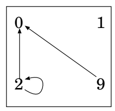
Mappings are also called **total functions**.

Properties of Heterogeneous Relations — Examples 1

univalent	$R^{\smile} \fatsemi R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b ⦅R⦆ c_1 \wedge b ⦅R⦆ c_2 \Rightarrow c_1 = c_2$
total	$\mathrm{Dom}\,R = \llcorner B \lrcorner$ $\mathbb{I} \subseteq R \fatsemi R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b ⦅R⦆ c)$
a **mapping**	iff it is univalent and total	

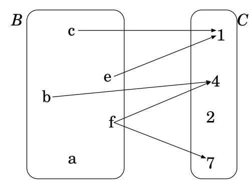


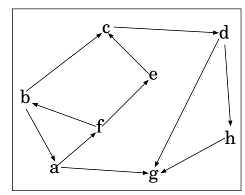


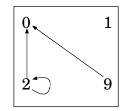


Properties of Heterogeneous Relations — Examples 2

injective	$R \fatsemi R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 ⦅R⦆ c \wedge b_2 ⦅R⦆ c \Rightarrow b_1 = b_2$
surjective	$\mathrm{Ran}\,R = \llcorner C \lrcorner$ $\mathbb{I} \subseteq R^{\smile} \fatsemi R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b ⦅R⦆ c)$
bijective	iff it is injective and surjective	







Properties of Heterogeneous Relations — Notes

univalent	$R^{\smile} \fatsemi R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b ⦅R⦆ c_1 \wedge b ⦅R⦆ c_2 \Rightarrow c_1 = c_2$
surjective	$\mathbb{I} \subseteq R^{\smile} \fatsemi R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b ⦅R⦆ c)$
total	$\mathbb{I} \subseteq R \fatsemi R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b ⦅R⦆ c)$
injective	$R \fatsemi R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 ⦅R⦆ c \wedge b_2 ⦅R⦆ c \Rightarrow b_1 = b_2$

All these properties are defined for arbitrary relations! (Not only for functions!)

- R is univalent and surjective
  iff $R^{\smile} \fatsemi R = \mathbb{I}$
  iff $R^{\smile}$ is a left-inverse of R
- R is total and injective
  iff $R \fatsemi R^{\smile} = \mathbb{I}$
  iff $R^{\smile}$ is a right-inverse of R

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-04

Part 1: 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 $\prec$ with no infinite (descending) $\succ$-chains are **well-founded**.

Loops **terminate** iff they are "going down" some well-founded 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 $(\forall\, x : U \bullet P\,x)$ for some property $P : U \to \mathbb{B}$ (with $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$)
- Situation: Elements of U are related via $\_\succ\_ : U \to U \to \mathbb{B}$ 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 $x : U$ there is a proof that

  if $P\,y$ for all predecessors y of x, then $P\,x$,

  then for every $z : U$ 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 (12.19) **Mathematical induction over $(U, \prec)$:**
If there are no infinite $\succ$-chains in U, that is, **if $\prec$ is well-founded**, then:

$$(\forall\, x \bullet P\,x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \prec x \bullet P\,y) \Rightarrow P\,x)$$

Mathematical Induction in $\mathbb{N}$

Consider $\_\prec\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ with $(x \prec y) = (y \succ x) = (y = suc\ x)$. $\quad \_\prec\_ = \ulcorner suc \urcorner$

Mathematical induction over $(\mathbb{N}, \prec)$**:**

$$
\begin{array}{ll}
& (\forall\, x : \mathbb{N} \bullet P\, x) \\
= & \langle \text{(12.19) Math. induction; Def. } \prec \rangle \\
& (\forall\, x : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid suc\ y = x \bullet P\, y) \Rightarrow P\, x) \\
= & \langle \text{(8.18) Range split, with } true \equiv x = 0 \lor x > 0 \rangle \\
& (\forall\, x : \mathbb{N} \mid x = 0 \bullet (\forall\, y : \mathbb{N} \mid suc\ y = x \bullet P\, y) \Rightarrow P\, x) \land \\
& (\forall\, x : \mathbb{N} \mid x > 0 \bullet (\forall\, y : \mathbb{N} \mid suc\ y = x \bullet P\, y) \Rightarrow P\, x) \\
= & \langle \text{(8.14) One-point rule; (8.22) Change of dummy} \rangle \\
& ((\forall\, y : \mathbb{N} \mid suc\ y = 0 \bullet P\, y) \Rightarrow P\, 0) \land \\
& (\forall\, z : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid suc\ y = suc\ z \bullet P\, y) \Rightarrow P\, (suc\ z)) \\
= & \left\langle \begin{array}{l} \text{(8.13) Empty range, with } suc\ y = 0 \equiv false; \\ \text{Cancellation of } suc\text{, (8.14) One-point rule for } \forall \end{array} \right\rangle \\
& P\, 0 \land (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\, (suc\ z))
\end{array}
$$

Mathematical Induction in $\mathbb{N}$ (ctd.)

Mathematical induction over $(\mathbb{N}, \ulcorner suc \urcorner)$**:**

$$(\forall\, x : \mathbb{N} \bullet P\, x) \quad \equiv \quad P\, 0 \land (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\, (suc\ z))$$

$$(\forall\, x : \mathbb{N} \bullet P\, x) \quad \equiv \quad P\, 0 \land (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\, (z + 1))$$

Absence of infinite **descending** $\ulcorner suc \urcorner$ 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.

Example for Complete Induction in $\mathbb{N}$

Mathematical induction over $(\mathbb{N}, <)$ **"Complete induction over** $\mathbb{N}$**":**

$$(\forall\, x : \mathbb{N} \bullet P\, x) \equiv (\forall\, x : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid y < x \bullet P\, y) \Rightarrow P\, x)$$

Theorem: Every natural number greater than 1 is a product of (one or more) prime numbers.
Formalisation: $\forall\, n : \mathbb{N} \bullet 1 < n \Rightarrow (\exists B : Bag\ \mathbb{N} \mid (\forall p \mid p \in B \bullet isPrime\ p) \bullet bagProd\ B = n)$
Proof:
Using "Complete induction":
- **For any** `$n$`:
  - **Assuming** `$\forall\, m \mid m < n \bullet 1 < m \Rightarrow (\exists B : Bag\ \mathbb{N} \mid (\forall p \mid p \in B \bullet isPrime\ p) \bullet bagProd\ B = m)$`:
    - **Assuming** `$1 < n$`:
      - **By cases:** `$isPrime\ n$`, `$\neg(isPrime\ n)$`
        - **Completeness:** By "Excluded middle"
        - **Case** `$isPrime\ n$`:
          …"∃-Introduction": $B := ⦃n⦄$ …
        - **Case** `$\neg(isPrime\ n)$`:
          …then $n = n_1 \cdot n_2$ with $n_1 < n > n_2$
          …with witness: $bagProd\ B_1 = n_1$ and $bagProd\ B_2 = n_2$
          …then $bagProd\ (B_1 \cup B_2) = n$

Mathematical Induction on Sequences

Cons induction: Mathematical induction over $(Seq\ A, \prec)$ where

$$\prec\ := \{x : A; xs, ys : Seq\ A \mid x \triangleleft xs = ys \bullet \langle xs, ys \rangle\}$$

$$(\forall\, xs : Seq\ A \bullet P\, xs) \quad \equiv \quad P\, \epsilon \land (\forall\, xs : Seq\ A \mid P\, xs \bullet (\forall\, x : A \bullet P(x \triangleleft xs)))$$

Snoc induction: Mathematical induction over $(Seq\ A, \prec)$ where

$$\prec\ := \{x : A; xs, ys : Seq\ A \mid xs \triangleright x = ys \bullet \langle xs, ys \rangle\}$$

$$(\forall\, xs : Seq\ A \bullet P\, xs) \quad \equiv \quad P\, \epsilon \land (\forall\, xs : Seq\ A \mid P\, xs \bullet (\forall\, x : A \bullet P(xs \triangleright x)))$$

Strict prefix induction: Mathematical induction over $(Seq\ A, \prec)$ where

$$\prec\ := \{us, xs, ys : Seq\ A \mid us \neq \epsilon \land xs \frown us = ys \bullet \langle xs, ys \rangle\}$$

$$(\forall\, xs : Seq\ A \bullet P\, xs) \quad \equiv \quad (\forall\, xs : Seq\ A \bullet (\forall\, ys : Seq\ A \mid ys \prec xs \bullet P\, ys) \Rightarrow P\, xs)$$

Different induction hypotheses make certain proofs easier.

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 2021

Wolfram Kahl

2021-11-04

Part 2: The While Rule

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.

In general, you have to identify an appropriate invariant yourself!

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
```

⟶ H15

Invariant: $\quad s = \sum j : \mathbb{N} \mid j < i \bullet f\, j$

— Generalised postcondition using the negated loop condition
(This is a frequent pattern.)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-08

Part 1: Correctness of Reversing Singly-linked Lists

Correctness of Reversing Singly-linked Lists

Theorem "Reversing of singly-linked lists":

$\mathsf{xs} = xs_0$

$\Rightarrow[\ \mathsf{ys} := \epsilon\ ⨾$

$\quad \mathsf{while\ xs} \neq \epsilon$

$\quad\quad \mathsf{do}$

$\quad\quad\quad \mathsf{ys} := \mathsf{head\ xs} \triangleleft \mathsf{ys}\ ⨾$

$\quad\quad\quad \mathsf{xs} := \mathsf{tail\ xs}$

$\quad\quad \mathsf{od}$

$\ ]$

$\mathsf{ys} = \mathsf{reverse}\ xs_0$

Proof:

?

Correctness of Reversing Singly-linked Lists

Theorem "Reversing of singly-linked lists":

$\mathsf{xs} = xs_0$

$\Rightarrow[\ \mathsf{ys} := \epsilon\ ⨾ \mathsf{while\ xs} \neq \epsilon\ \mathsf{do\ ys} := \mathsf{head\ xs} \triangleleft \mathsf{ys}\ ⨾ \mathsf{xs} := \mathsf{tail\ xs\ od}\ ]$

$\mathsf{ys} = \mathsf{reverse}\ xs_0$

Proof:

$\mathsf{xs} = xs_0$ ▪▪▪▪▪▪ Precondition

$\Rightarrow[\ \mathsf{ys} := \epsilon\ ]$ ⟨ ? ⟩

$\mathsf{reverse\ xs} \frown \mathsf{ys} = \mathsf{reverse}\ xs_0$ ▪▪▪▪▪▪ Invariant

$\Rightarrow[\ \mathsf{while\ xs} \neq \epsilon\ \mathsf{do}$

$\quad \mathsf{ys} := \mathsf{head\ xs} \triangleleft \mathsf{ys}\ ⨾$

$\quad \mathsf{xs} := \mathsf{tail\ xs}$

$\mathsf{od}\ ]$ ⟨ "While" with ? ⟩

$\neg(\mathsf{xs} \neq \epsilon) \wedge \mathsf{reverse\ xs} \frown \mathsf{ys} = \mathsf{reverse}\ xs_0$ ▪▪▪▪▪▪ Negated loop condition, and invariant

$\Rightarrow$ ⟨ ? ⟩

$\mathsf{ys} = \mathsf{reverse}\ xs_0$ ▪▪▪▪▪▪ Postcondition

Correctness of Reversing Singly-linked Lists

Theorem "Reversing of singly-linked lists":

$\mathsf{xs} = xs_0$

$\Rightarrow[\ \mathsf{ys} := \epsilon\ ⨾ \mathsf{while\ xs} \neq \epsilon\ \mathsf{do\ ys} := \mathsf{head\ xs} \triangleleft \mathsf{ys}\ ⨾ \mathsf{xs} := \mathsf{tail\ xs\ od}\ ]$

$\mathsf{ys} = \mathsf{reverse}\ xs_0$

Proof:

$\mathsf{xs} = xs_0$ ▪▪▪▪▪▪ Precondition

$\Rightarrow[\ \mathsf{ys} := \epsilon\ ]$ ⟨ "Proper initialisation for `rev`" ⟩

$\mathsf{reverse\ xs} \frown \mathsf{ys} = \mathsf{reverse}\ xs_0$ ▪▪▪▪▪▪ Invariant

$\Rightarrow[\ \mathsf{while\ xs} \neq \epsilon\ \mathsf{do}$

$\quad \mathsf{ys} := \mathsf{head\ xs} \triangleleft \mathsf{ys}\ ⨾$

$\quad \mathsf{xs} := \mathsf{tail\ xs}$

$\mathsf{od}\ ]$ ⟨ "While" with "Invariant for `rev`" ⟩ ▪▪▪▪▪▪ A4.3

$\neg(\mathsf{xs} \neq \epsilon) \wedge \mathsf{reverse\ xs} \frown \mathsf{ys} = \mathsf{reverse}\ xs_0$ ▪▪▪▪▪▪ Negated loop condition, and invariant

$\Rightarrow$ ⟨ ? ⟩

$\mathsf{ys} = \mathsf{reverse}\ xs_0$ ▪▪▪▪▪▪ Postcondition

Correctness of Initialisation for Reversing Singly-linked Lists

Theorem "Proper initialisation for `rev`":

$\mathsf{xs} = xs_0$

$\Rightarrow[\ \mathsf{ys} := \epsilon\ ]$

$\mathsf{reverse\ xs} \frown \mathsf{ys} = \mathsf{reverse}\ xs_0$

Proof:

$\mathsf{reverse\ xs} \frown \mathsf{ys} = \mathsf{reverse}\ xs_0$

$[\ \mathsf{ys} := \epsilon\ ] \Leftarrow$ ⟨ "Assignment" with substitution ⟩

$\mathsf{reverse\ xs} \frown \epsilon = \mathsf{reverse}\ xs_0$

$\equiv$ ⟨ "Right-identity of ⌢" ⟩

$\mathsf{reverse\ xs} = \mathsf{reverse}\ xs_0$

$\equiv$ ⟨ Substitution ⟩

$(\mathsf{reverse}\ z)[z := xs] = (\mathsf{reverse}\ z)[z := xs_0]$

$\Leftarrow$ ⟨ "Leibniz" ⟩

$\mathsf{xs} = xs_0$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-08

Part 2: Midterm 1

M1.1A

Theorem (M1.1): $y = 2 \Rightarrow x \cdot (y \cdot y - 4) = 0$

Proof:

$y = 2 \Rightarrow x \cdot (y \cdot y - 4) = 0$

$\equiv$ ⟨ Substitution ⟩

$y = 2 \Rightarrow (x \cdot (u \cdot u - 4) = 0)[u := y]$

$\equiv$ ⟨ "Replacement" (3.84b) ⟩

$y = 2 \Rightarrow (x \cdot (u \cdot u - 4) = 0)[u := 2]$

$\equiv$ ⟨ Substitution ⟩

$y = 2 \Rightarrow (x \cdot (2 \cdot 2 - 4) = 0)$

$\equiv$ ⟨ Evaluation ⟩

$y = 2 \Rightarrow (x \cdot 0 = 0)$

$\equiv$ ⟨ "Zero of ·" ⟩

$y = 2 \Rightarrow \mathsf{true}$

$\equiv$ ⟨ "Right-zero of ⇒" ⟩

true

> **Theorem** (3.84a) "Replacement":
>
> $(e = f) \wedge E[z := e]$
>
> $\equiv\ (e = f) \wedge E[z := f]$
>
> **Theorem** (3.84b) "Replacement":
>
> $(e = f) \Rightarrow E[z := e]$
>
> $\equiv\ (e = f) \Rightarrow E[z := f]$

M1.1B

Theorem (M1.1): $x = 3 \Rightarrow (9 - x \cdot x) \cdot y = 0$

Proof:

$x = 3 \Rightarrow (9 - x \cdot x) \cdot y = 0$

$\equiv$ ⟨ Substitution ⟩

$x = 3 \Rightarrow ((9 - u \cdot u) \cdot y = 0)[u := x]$

$\equiv$ ⟨ "Replacement" (3.84b) ⟩

$x = 3 \Rightarrow ((9 - u \cdot u) \cdot y = 0)[u := 3]$

$\equiv$ ⟨ Substitution ⟩

$x = 3 \Rightarrow ((9 - 3 \cdot 3) \cdot y = 0)$

$\equiv$ ⟨ Evaluation ⟩

$x = 3 \Rightarrow (0 \cdot y = 0)$

$\equiv$ ⟨ "Zero of ·" ⟩

$x = 3 \Rightarrow \mathsf{true}$

$\equiv$ ⟨ "Right-zero of ⇒" ⟩

true

> **Theorem** (3.84a) "Replacement":
>
> $(e = f) \wedge E[z := e]$
>
> $\equiv\ (e = f) \wedge E[z := f]$
>
> **Theorem** (3.84b) "Replacement":
>
> $(e = f) \Rightarrow E[z := e]$
>
> $\equiv\ (e = f) \Rightarrow E[z := f]$

M1.2A — Even Product

Theorem "Even product": $\mathsf{even}\ a \Rightarrow \mathsf{even}\ (a \cdot b)$

Proof:

By induction on `$b : \mathbb{N}$`:

Base case:

$\mathsf{even}\ a \Rightarrow \mathsf{even}\ (a \cdot 0)$

$\equiv$ ⟨ "Zero of ·" ⟩

$\mathsf{even}\ a \Rightarrow \mathsf{even}\ 0$

$\equiv$ ⟨ "Zero is even" ⟩

$\mathsf{even}\ a \Rightarrow \mathsf{true}$

— **This is** "Right-zero of ⇒"

Induction step:

$\mathsf{even}\ a \Rightarrow \mathsf{even}\ (a \cdot \mathsf{suc}\ b)$

$\equiv$ ⟨ "Multiplying the successor" ⟩

$\mathsf{even}\ a \Rightarrow \mathsf{even}\ (a + a \cdot b)$

$\equiv$ ⟨ "Even addition" ⟩

$\mathsf{even}\ a \Rightarrow (\mathsf{even}\ a \equiv \mathsf{even}\ (a \cdot b))$

$\equiv$ ⟨ "Distributivity of ⇒ over ≡" ⟩

$(\mathsf{even}\ a \Rightarrow \mathsf{even}\ a) \equiv (\mathsf{even}\ a \Rightarrow \mathsf{even}\ (a \cdot b))$

$\equiv$ ⟨ Induction hypothesis ⟩

$(\mathsf{even}\ a \Rightarrow \mathsf{even}\ a) \equiv \mathsf{true}$

— **This is** "Reflexivity of ⇒"

M1.2A — Odd Product

Theorem "Odd product": $\mathsf{odd}\ (a \cdot b) \equiv \mathsf{odd}\ a \wedge \mathsf{odd}\ b$

Proof:

By induction on `$a : \mathbb{N}$`:

Base case:

$\mathsf{odd}\ (0 \cdot b) \equiv \mathsf{odd}\ 0 \wedge \mathsf{odd}\ b$

$\equiv$ ⟨ "Zero of ·" ⟩

$\mathsf{odd}\ 0 \equiv \mathsf{odd}\ 0 \wedge \mathsf{odd}\ b$

$\equiv$ ⟨ "Definition of ⇒ via ∧" ⟩

$\mathsf{odd}\ 0 \Rightarrow \mathsf{odd}\ b$

$\equiv$ ⟨ "Double negation" ⟩

$\neg\neg\,\mathsf{odd}\ 0 \Rightarrow \mathsf{odd}\ b$

$\equiv$ ⟨ "Zero is not odd" ⟩

$\neg\,\mathsf{true} \Rightarrow \mathsf{odd}\ b$

$\equiv$ ⟨ "Definition of `false`" ⟩

$\mathsf{false} \Rightarrow \mathsf{odd}\ b$

— **This is** "ex falso quodlibet"

Induction step:

$\mathsf{odd}\ (\mathsf{suc}\ a \cdot b)$

$\equiv$ ⟨ "Definition of · for `suc`" ⟩

$\mathsf{odd}\ (b + a \cdot b)$

$\equiv$ ⟨ "Odd addition" ⟩

$\mathsf{even}\ b \equiv \mathsf{odd}\ (a \cdot b)$

$\equiv$ ⟨ Induction hypothesis ⟩

$\mathsf{even}\ b \equiv \mathsf{odd}\ a \wedge \mathsf{odd}\ b$

$\equiv$ ⟨ "Even is not odd" ⟩

$\neg\,\mathsf{odd}\ b \equiv \mathsf{odd}\ a \wedge \mathsf{odd}\ b$

$\equiv$ ⟨ (3.48) ⟩

$\neg\,\mathsf{odd}\ a \wedge \mathsf{odd}\ b$

$\equiv$ ⟨ "Odd successor" ⟩

$\mathsf{odd}\ (\mathsf{suc}\ a) \wedge \mathsf{odd}\ b$

M1.2B — Odd Product

Theorem "Odd product": $\mathsf{odd}\ (a \cdot b) \Rightarrow \mathsf{odd}\ a$
Proof:
By induction on `$b : \mathbb{N}$`:
Base case:
$\mathsf{odd}\ (a \cdot 0) \Rightarrow \mathsf{odd}\ a$
$\equiv\langle$ "Zero of $\cdot$" $\rangle$
$\mathsf{odd}\ 0 \Rightarrow \mathsf{odd}\ a$
$\equiv\langle$ "Material implication" $\rangle$
$\neg\, \mathsf{odd}\ 0 \vee \mathsf{odd}\ a$
$\equiv\langle$ "Zero is not odd" $\rangle$
$\mathsf{true} \vee \mathsf{odd}\ a$
$\equiv\langle$ "Zero of $\vee$" $\rangle$
true
Induction step:
$\mathsf{odd}\ (a \cdot \mathsf{suc}\ b) \Rightarrow \mathsf{odd}\ a$
$\equiv\langle$ "Multiplying the successor" $\rangle$
$\mathsf{odd}\ (a + a \cdot b) \Rightarrow \mathsf{odd}\ a$
$\equiv\langle$ "Odd addition" $\rangle$
$(\mathsf{even}\ a \equiv \mathsf{odd}\ (a \cdot b)) \Rightarrow \mathsf{odd}\ a$
$\equiv\langle$ "Material implication" $\rangle$
$\neg\, (\mathsf{even}\ a \equiv \mathsf{odd}\ (a \cdot b)) \vee \mathsf{odd}\ a$
$\equiv\langle$ "Commutativity of $\neg$ with $\equiv$" $\rangle$
$(\mathsf{even}\ a \equiv \neg\, \mathsf{odd}\ (a \cdot b)) \vee \mathsf{odd}\ a$
$\equiv\langle$ "Distributivity of $\vee$ over $\equiv$" $\rangle$
$(\mathsf{even}\ a \vee \mathsf{odd}\ a) \equiv (\neg\, \mathsf{odd}\ (a \cdot b) \vee \mathsf{odd}\ a)$
$\equiv\langle$ "Material implication" $\rangle$
$(\mathsf{even}\ a \vee \mathsf{odd}\ a) \equiv (\mathsf{odd}\ (a \cdot b) \Rightarrow \mathsf{odd}\ a)$
$\equiv\langle$ Induction hypothesis $\rangle$
$(\mathsf{even}\ a \vee \mathsf{odd}\ a) \equiv \mathsf{true}$
— **This is** "Odd or even"

M1.2B — Even Product

Theorem "Even product": $\mathsf{even}\ (a \cdot b) \equiv \mathsf{even}\ a \vee \mathsf{even}\ b$
Proof:
By induction on `$a : \mathbb{N}$`:
Base case:
$\mathsf{even}\ (0 \cdot b) \equiv \mathsf{even}\ 0 \vee \mathsf{even}\ b$
$\equiv\langle$ "Zero of $\cdot$" $\rangle$
$\mathsf{even}\ 0 \equiv \mathsf{even}\ 0 \vee \mathsf{even}\ b$
$\equiv\langle$ "Zero is even" $\rangle$
$\mathsf{true} \equiv \mathsf{true} \vee \mathsf{even}\ b$
— **This is** "Zero of $\vee$"
Induction step:
$\mathsf{even}\ (\mathsf{suc}\ a \cdot b)$
$\equiv\langle$ "Definition of $\cdot$ for `suc`" $\rangle$
$\mathsf{even}\ (b + a \cdot b)$
$\equiv\langle$ "Even addition" $\rangle$
$\mathsf{even}\ b \equiv \mathsf{even}\ (a \cdot b)$
$\equiv\langle$ Induction hypothesis $\rangle$
$\mathsf{even}\ b \equiv \mathsf{even}\ a \vee \mathsf{even}\ b$
$\equiv\langle$ (3.32) $\rangle$
$\neg\, \mathsf{even}\ a \vee \mathsf{even}\ b$
$\equiv\langle$ "Even successor" $\rangle$
$\mathsf{even}\ (\mathsf{suc}\ a) \vee \mathsf{even}\ b$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-08

Part 3: Quantifier Reasoning Examples: H14

H14 — 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 `$x$`:
$x \in \mathsf{Dom}\ (R \cup S)$
$\equiv\langle\ ?\ \rangle$
$x \in \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

H14 — 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 `$x$`:
$x \in \mathsf{Dom}\ (R \cup S)$
$\equiv\langle$ "Membership in `Dom`" $\rangle$
$\exists y \bullet x \,(\!| R \cup S |\!)\, y$
$\equiv\langle$ "Relation union" $\rangle$
$\exists y \bullet x \,(\!| R |\!)\, y \vee x \,(\!| S |\!)\, y$
$\equiv\langle\ ?\ \rangle$
$(\exists y \bullet x \,(\!| R |\!)\, y) \vee (\exists y \bullet x \,(\!| S |\!)\, y)$
$\equiv\langle$ "Membership in `Dom`" $\rangle$
$x \in \mathsf{Dom}\ R \vee x \in \mathsf{Dom}\ S$
$\equiv\langle$ "Union" $\rangle$
$x \in \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

H14 — 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 `$x$`:
$x \in \mathsf{Dom}\ (R \cup S)$
$\equiv\langle$ "Membership in `Dom`" $\rangle$
$\exists y \bullet x \,(\!| R \cup S |\!)\, y$
$\equiv\langle$ "Relation union" $\rangle$
$\exists y \bullet x \,(\!| R |\!)\, y \vee x \,(\!| S |\!)\, y$
$\equiv\langle$ "Distributivity of $\exists$ over $\vee$" $\rangle$
$(\exists y \bullet x \,(\!| R |\!)\, y) \vee (\exists y \bullet x \,(\!| S |\!)\, y)$
$\equiv\langle$ "Membership in `Dom`" $\rangle$
$x \in \mathsf{Dom}\ R \vee x \in \mathsf{Dom}\ S$
$\equiv\langle$ "Union" $\rangle$
$x \in \mathsf{Dom}\ R \cup \mathsf{Dom}\ S$

H14 — Domain of ∩ — 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" $\rangle$
$x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

H14 — 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" $\rangle$
$x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

H14 — 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" $\rangle$
$x \in \mathsf{Dom}\ R \cap \mathsf{Dom}\ S$

H14 — 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 $\neg occurs(‘x’, ‘P’)$

H14 — 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 \`$x$\`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in \`Dom\`" $\rangle$
      $\exists y \bullet x \,(\!| R \cap S |\!)\, y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; x \,(\!| S |\!)\, y$

    $\Rightarrow\langle\ ?\ \rangle$

      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; (\exists y \bullet x \,(\!| S |\!)\, y)$
    $\equiv\langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
      $(\exists y \bullet x \,(\!| R |\!)\, y) \;\wedge\; (\exists y \bullet x \,(\!| S |\!)\, y)$
    $\equiv\langle$ "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 $\neg occurs(\text{'}x\text{'}, \text{'}P\text{'})$

H14 — 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 \`$x$\`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in \`Dom\`" $\rangle$
      $\exists y \bullet x \,(\!| R \cap S |\!)\, y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; x \,(\!| S |\!)\, y$
    $\equiv\langle$ Substitution $\rangle$
      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; (x \,(\!| S |\!)\, y)[y := y]$
    $\Rightarrow\langle\ ?$ with "$\exists$-Introduction" $\rangle$
      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; (\exists y \bullet x \,(\!| S |\!)\, y)$
    $\equiv\langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
      $(\exists y \bullet x \,(\!| R |\!)\, y) \;\wedge\; (\exists y \bullet x \,(\!| S |\!)\, y)$
    $\equiv\langle$ "Membership in \`Dom\`" $\rangle$
      $x \in \mathsf{Dom}\,R \;\wedge\; x \in \mathsf{Dom}\,S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$

H14 — 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 \`$x$\`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in \`Dom\`" $\rangle$
      $\exists y \bullet x \,(\!| R \cap S |\!)\, y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; x \,(\!| S |\!)\, y$
    $\equiv\langle$ Substitution $\rangle$
      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; (x \,(\!| S |\!)\, y)[y := y]$
    $\Rightarrow\langle$ "Body monotonicity of $\exists$" with "Monotonicity of $\wedge$" with "$\exists$-Introduction" $\rangle$
      $\exists y \bullet x \,(\!| R |\!)\, y \;\wedge\; (\exists y \bullet x \,(\!| S |\!)\, y)$
    $\equiv\langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
      $(\exists y \bullet x \,(\!| R |\!)\, y) \;\wedge\; (\exists y \bullet x \,(\!| S |\!)\, y)$
    $\equiv\langle$ "Membership in \`Dom\`" $\rangle$
      $x \in \mathsf{Dom}\,R \;\wedge\; x \in \mathsf{Dom}\,S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-08

Part 4: Witnesses

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:
  $(\exists x \mid R \bullet P) \Rightarrow Q$
$=\ \langle$ (9.19) Trading for $\exists$ $\rangle$
  $(\exists x \bullet R \wedge P) \Rightarrow Q$
$=\ \langle$ (3.59) $p \Rightarrow q \equiv \neg p \vee q$, (9.18b) Gen. De Morgan $\rangle$
  $(\forall x \bullet \neg(R \wedge P)) \vee Q$
$=\ \langle$ (9.5) Distributivity of $\vee$ over $\forall$ — $\neg occurs(\text{'}x\text{'}, \text{'}Q\text{'})$ $\rangle$
  $(\forall x \bullet \neg(R \wedge P) \vee Q)$
$=\ \langle$ (3.59) $p \Rightarrow q \equiv \neg p \vee q$ $\rangle$
  $(\forall x \bullet R \wedge P \Rightarrow Q)$

The last line is, by Metatheorem (9.16), a theorem iff $(R \wedge P) \Rightarrow Q$ is.

LADM Theory of Integers — Axioms

(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 $+$:** $a + b = a + c \quad\equiv\quad b = c$
(15.10b) **Unique mult. identity:** $a \neq 0 \Rightarrow (a \cdot z = a \;\equiv\; z = 1)$
(15.12) **Unique additive inverse:** $x + a = 0 \wedge y + a = 0 \Rightarrow x = y$

```
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” ⟩
          ∀ x : ℤ • x + a = 0  ⇒  a + b = a + c  ⇒  b = c
      Proof for this:
        For any `x : ℤ`:
          Assuming `x + a = 0`, `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:**
$(\exists x \bullet x + a = 0)$

(15.8) **Cancellation of $+$:**
$a + b = a + c \quad\equiv\quad b = c$

```
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:**
$(\exists x \bullet x + a = 0)$

(15.8) **Cancellation of $+$:**
$a + b = a + c \quad\equiv\quad b = c$

Witnesses (ctd.)

(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

(9.30) **Metatheorem Witness**: If $\neg occurs(\text{'}\hat{x}\text{'}, \text{'}P, Q, R\text{'})$, then:

$(\exists x \mid R \bullet P) \Rightarrow Q$ is a theorem iff
$(R \wedge P)[x := \hat{x}] \Rightarrow Q$ is a theorem.

Corresponding to inference rule $\exists$-elimination:

$$\frac{(\exists x \bullet P) \qquad \begin{array}{c} \ulcorner P \urcorner \\ \vdots \\ Q \end{array}}{Q}\ \exists\text{-Elim}$$

(prov. x not free in Q, assumptions)

Witnesses: Using Existential Assumptions/Theorems following LADM

(9.30) **Metatheorem Witness**: If $\neg occurs(\text{'}\hat{x}\text{'}, \text{'}P, Q, R\text{'})$, then:
$(\exists x \mid R \bullet P) \Rightarrow Q$ is a theorem iff
$(R \wedge P)[x := \hat{x}] \Rightarrow Q$ is a theorem.

Prove: $a + b = a + c \Rightarrow b = c$, using:
(9.31) $(\exists x : \mathbb{Z} \bullet x + a = 0)$
(9.30) turns this into $(x + a = 0)[x := \alpha]$, so we use $\alpha + a = 0$.

  $a + b = a + c$
$\Rightarrow\ \langle$ Leibniz, with Deduction Theorem (4.4) $\rangle$
  $\alpha + a + b = \alpha + a + c$
$=\ \langle$ Assumption $\alpha + a = 0$ $\rangle$
  $0 + b = 0 + c$
$=\ \langle$ Additive identity (15.3) $\rangle$
  $b = c$

```
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” ⟩
          (∃ 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
```

(15.6) **Additive Inverse:**
$(\exists x \bullet x + a = 0)$

(15.8) **Cancellation of +:**
$a + b = a + c \quad \equiv \quad b = c$

(15.6) Additive Inverse
$(\exists x \bullet x + a = 0)$

$$\dfrac{(\exists x \bullet P) \qquad \begin{array}{c}\ulcorner P \urcorner \\ \vdots \\ R\end{array}}{R}\ \exists\text{-Elim}$$
(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`:
      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
```

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,

$$\dfrac{(\exists x \bullet P) \qquad \begin{array}{c}\ulcorner P \urcorner \\ \vdots \\ R\end{array}}{R}\ \exists\text{-Elim}$$
(prov. x not free in R, assumptions)

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.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-09

Part 1: Residuals

Given: $x \le z \quad \equiv \quad x \le 5$

What do you know about z? Why? (Prove it!)

Given: $X \subseteq A \twoheadrightarrow B \quad \equiv \quad X \cap A \subseteq B$

Calculate the **relative pseudocomplement** $A \twoheadrightarrow B$!

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\quad X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S$

$R \backslash S$ is the largest solution $X : B \leftrightarrow C$ for $R \fatsemi X \subseteq S$.

Calculate the **right residual** (“left division”) $R \backslash S$!

$$\begin{tikzcd} A \arrow[r, "S"] \arrow[dr, "R"'] & C \\ & B \arrow[u, "R \backslash S"'] \end{tikzcd}$$

Same idea as for “$\twoheadrightarrow$”:
Using extensionality, calculate $b \llparenthesis R \backslash S \rrparenthesis c \quad \equiv \quad b \llparenthesis ? \rrparenthesis c$

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\quad X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S$

Calculate the **right residual** (“left division”) $R \backslash S$!

$$\begin{tikzcd} A \arrow[r, "S"] \arrow[dr, "R"'] & C \\ & B \arrow[u, "R \backslash S"'] \end{tikzcd}$$

$$\begin{array}{ll}
 & b \llparenthesis R \backslash S \rrparenthesis c \\
= & \langle \text{Similar to the calculation for relative pseudocomplement} \rangle \\
 & (\forall a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c) \\
= & \langle \text{Generalised De Morgan, Relation conversions} \rangle \\
 & b \llparenthesis \sim (R^{\smile} \fatsemi \sim S) \rrparenthesis c
\end{array}$$

Therefore: $R \backslash S = \sim (R^{\smile} \fatsemi \sim S)$

— monotonic in second argument; antitonic in first argument

Proving $b \llparenthesis R \backslash S \rrparenthesis c \equiv (\forall a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c)$:

$$\begin{array}{ll}
 & b(R \backslash S)c \\
= & \langle e \in S \equiv \{e\} \subseteq S \text{ — Exercise! } \rangle \\
 & \{\langle b,c \rangle\} \subseteq (R \backslash S) \\
= & \langle \text{Def. } \backslash\text{: } X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S \rangle \\
 & R \fatsemi \{\langle b,c \rangle\} \subseteq S \\
= & \langle \text{(11.13r) Relation inclusion} \rangle \\
 & (\forall a, c' \mid a \llparenthesis R \fatsemi \{\langle b,c \rangle\} \rrparenthesis c' \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(14.20) Relation composition} \rangle \\
 & (\forall a, c' \mid (\exists b' \bullet a \llparenthesis R \rrparenthesis b' \wedge b' \llparenthesis \{\langle b,c \rangle\} \rrparenthesis c') \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle y \in \{x\} \equiv y = x \text{ — Exercise! } \rangle \\
 & (\forall a, c' \mid (\exists b' \bullet a \llparenthesis R \rrparenthesis b' \wedge b' = b \wedge c = c') \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(9.19) Trading for } \exists \rangle \\
 & (\forall a, c' \mid (\exists b' \mid b' = b \bullet a \llparenthesis R \rrparenthesis b' \wedge c = c') \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(8.14) One-point rule} \rangle \\
 & (\forall a, c' \mid a \llparenthesis R \rrparenthesis b \wedge c = c' \bullet a \llparenthesis S \rrparenthesis c') \\
= & \langle \text{(8.20) Quantifier nesting} \rangle \\
 & (\forall a \mid a \llparenthesis R \rrparenthesis b \bullet (\forall c' \mid c = c' \bullet a \llparenthesis S \rrparenthesis c')) \\
= & \langle \text{(1.3) Symmetry of =, (8.14) One-point rule} \rangle \\
 & (\forall a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c)
\end{array}$$

Right Residual: $\quad X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S$

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 $S : A \leftrightarrow C$: $\quad X \subseteq R \backslash S \quad \equiv \quad R \fatsemi X \subseteq S$

Calculate the **right residual** (“left division”) $R \backslash S$! (“R under S”)

$$\begin{tikzcd} A \arrow[r, "S"] \arrow[dr, "R"'] & C \\ & B \arrow[u, "R \backslash S"'] \end{tikzcd}$$

$$\begin{array}{ll}
 & b \llparenthesis R \backslash S \rrparenthesis c \\
= & \langle \text{Similar to the calculation for relative pseudocomplement} \rangle \\
 & (\forall a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c) \\
= & \langle \text{Generalised De Morgan, Relation conversions} \rangle \\
 & b \llparenthesis \sim (R^{\smile} \fatsemi \sim S) \rrparenthesis c
\end{array}$$

Therefore: $R \backslash S = \sim (R^{\smile} \fatsemi \sim S)$

— monotonic in second argument; antitonic in first argument

Formalisations Using Residuals

Relationship via $\backslash$:

$$\begin{array}{ll}
 & b \llparenthesis R \backslash S \rrparenthesis c \\
\equiv & (\forall a \mid a \llparenthesis R \rrparenthesis b \bullet a \llparenthesis S \rrparenthesis c)
\end{array}$$

“Aos called only brothers of Jun.”
“Everybody called by Aos is a brother of Jun.”

$$\begin{array}{ll}
 & (\forall p \mid Aos \llparenthesis C \rrparenthesis p \bullet p \llparenthesis B \rrparenthesis Jun) \\
\equiv & \langle \text{(14.18) Relation converse} \rangle \\
 & (\forall p \mid p \llparenthesis C^{\smile} \rrparenthesis Aos \bullet p \llparenthesis B \rrparenthesis Jun) \\
\equiv & \langle \text{Right residual} \rangle \\
 & Aos \llparenthesis C^{\smile} \backslash B \rrparenthesis Jun
\end{array}$$

“Aos called every brother of Jun.”
“Every brother of Jun has been called by Aos.”

$$\begin{array}{ll}
 & (\forall p \mid p \llparenthesis B \rrparenthesis Jun \bullet Aos \llparenthesis C \rrparenthesis p) \\
\equiv & \langle \text{(14.18) Relation converse} \rangle \\
 & (\forall p \mid p \llparenthesis B \rrparenthesis Jun \bullet p \llparenthesis C^{\smile} \rrparenthesis Aos) \\
\equiv & \langle \text{Right residual} \rangle \\
 & Jun \llparenthesis B \backslash C^{\smile} \rrparenthesis Aos
\end{array}$$

Some Properties of Right Residuals

Characterisation of right residual: $\forall\, R : A \leftrightarrow B;\ S : A \leftrightarrow C \bullet \quad X \subseteq R \setminus S \quad \equiv \quad R \mathbin{;} X \subseteq S$

Two sub-cancellation properties follow easily:

$$R \mathbin{;} (R \setminus S) \quad \subseteq \quad S$$
$$(Q \setminus R) \mathbin{;} (R \setminus S) \quad \subseteq \quad (Q \setminus S)$$

Theorem "$\mathbb{I} \setminus$": $\mathbb{I} \setminus R = R$
Proof:
  Using "Mutual inclusion":
    Subproof:
      $\mathbb{I} \setminus R$
      $= \langle$ "Identity of $;$" $\rangle$
      $\mathbb{I} \mathbin{;} (\mathbb{I} \setminus R)$
      $\subseteq \langle$ "Cancellation of $\setminus$" $\rangle$
      R
    Subproof:
      $R \subseteq \mathbb{I} \setminus R$
      $\equiv \langle$ "Characterisation of $\setminus$" $\rangle$
      $\mathbb{I} \mathbin{;} R \subseteq R$
      $\equiv \langle$ "Identity of $;$", "Reflexivity of $\subseteq$" $\rangle$
      true

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
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis {\sim} S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \twoheadrightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\ A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^{\smile} \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \mathbin{;} S \rrparenthesis v$	$\equiv$	$(\exists\, x \bullet u \llparenthesis R \rrparenthesis x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \setminus S \rrparenthesis v$	$\equiv$	$(\forall\, x \mid x \llparenthesis R \rrparenthesis u \bullet x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall\, x \mid v \llparenthesis R \rrparenthesis x \bullet u \llparenthesis S \rrparenthesis x)$

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
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis {\sim} S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \twoheadrightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\ A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^{\smile} \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \mathbin{;} S \rrparenthesis v$	$\equiv$	$(\exists\, x \mid u \llparenthesis R \rrparenthesis x \bullet x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \setminus S \rrparenthesis v$	$\equiv$	$(\forall\, x \mid x \llparenthesis R \rrparenthesis u \bullet x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall\, x \mid v \llparenthesis R \rrparenthesis x \bullet u \llparenthesis S \rrparenthesis x)$

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
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis {\sim} S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \twoheadrightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\ A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^{\smile} \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \mathbin{;} S \rrparenthesis v$	$\equiv$	$(\exists\, x \bullet u \llparenthesis R \rrparenthesis x \wedge x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \setminus S \rrparenthesis v$	$\equiv$	$(\forall\, x \bullet x \llparenthesis R \rrparenthesis u \Rightarrow x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall\, x \bullet v \llparenthesis R \rrparenthesis x \Rightarrow u \llparenthesis S \rrparenthesis x)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-09

Part 2: More on Sets and Relations

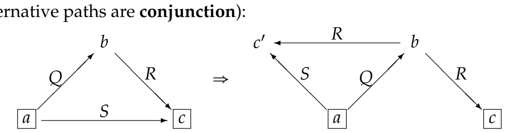
Modal Rules— Converse as Over-Approximation of Inverse

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \subseteq Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \subseteq (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Useful to "**make information available locally**" (Q is replaced with $Q \cap S \mathbin{;} R^{\smile}$) for use in further proof steps.

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \Rightarrow$
$\quad (\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c')$

Proving a Modal Rule — Straight-forward Calculation

Theorem "Modal rule": $(Q \mathbin{;} R) \cap S \subseteq (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$
Proof:
  Using "Relation inclusion":
    Subproof for `$\forall\, a \bullet \forall\, c \bullet a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c \Rightarrow a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$`:
      For any `$a$`, `$c$`:
        $a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$
        $\equiv \langle$ "Relation composition" $\rangle$
        $\exists\, b \bullet a \llparenthesis Q \cap S \mathbin{;} R^{\smile} \rrparenthesis b \wedge b \llparenthesis R \rrparenthesis c$
        $\equiv \langle$ "Relation intersection", "Relation composition", "Relation converse" $\rangle$
        $\exists\, b \bullet a \llparenthesis Q \rrparenthesis b \wedge (\exists\, c_2 \bullet a \llparenthesis S \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c_2) \wedge b \llparenthesis R \rrparenthesis c$
        $\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
        $\exists\, b \bullet \exists\, c_2 \bullet a \llparenthesis Q \rrparenthesis b \wedge a \llparenthesis S \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c$

        $\Leftarrow \langle\ ?\ \rangle$ ······ This is the implication from the previous slide

        $\exists\, b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c$
        $\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
        $(\exists\, b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c) \wedge a \llparenthesis S \rrparenthesis c$
        $\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
        $a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c$

Proving a Modal Rule — Straight-forward Calculation (filled)

Theorem "Modal rule": $(Q \mathbin{;} R) \cap S \subseteq (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$
Proof:
  Using "Relation inclusion":
    Subproof for `$\forall\, a \bullet \forall\, c \bullet a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c \Rightarrow a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$`:
      For any `$a$`, `$c$`:
        $a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$
        $\equiv \langle$ "Relation composition" $\rangle$
        $\exists\, b \bullet a \llparenthesis Q \cap S \mathbin{;} R^{\smile} \rrparenthesis b \wedge b \llparenthesis R \rrparenthesis c$
        $\equiv \langle$ "Relation intersection", "Relation composition", "Relation converse" $\rangle$
        $\exists\, b \bullet a \llparenthesis Q \rrparenthesis b \wedge (\exists\, c_2 \bullet a \llparenthesis S \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c_2) \wedge b \llparenthesis R \rrparenthesis c$
        $\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
        $\exists\, b \bullet \exists\, c_2 \bullet a \llparenthesis Q \rrparenthesis b \wedge a \llparenthesis S \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c$
        $\Leftarrow \langle$ "Body monotonicity of $\exists$" with "$\exists$-Introduction" $\rangle$
        $\exists\, b \bullet (a \llparenthesis Q \rrparenthesis b \wedge a \llparenthesis S \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c_2 \wedge b \llparenthesis R \rrparenthesis c)[c_2 := c]$
        $\equiv \langle$ Substitution, "Idempotency of $\wedge$" $\rangle$
        $\exists\, b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c$
        $\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
        $(\exists\, b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c) \wedge a \llparenthesis S \rrparenthesis c$
        $\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
        $a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c$

Proving a Modal Rule — Artificial `Assuming witness` Variant

Theorem "Modal rule": $(Q \mathbin{;} R) \cap S \subseteq (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$
Proof:
  Using "Relation inclusion":
    Subproof for `$\forall\, a \bullet \forall\, c \bullet a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c \Rightarrow a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$`:
      For any `$a$`, `$c$`:
        Assuming (1) `$a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c$`:
          Side proof for (2) `$\exists\, b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c$`:
            $a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c$ — **This is** assumption (1)
            $\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
            $(\exists\, b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c) \wedge a \llparenthesis S \rrparenthesis c$
            $\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$
            $\exists\, b_2 \bullet a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c$
          Continuing:
            Assuming witness `$b_2$` **satisfying**
                (3) `$a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c$` **by** local property (2):
              $a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$
              $\equiv \langle$ "Relation composition" $\rangle$
              $\exists\, b \bullet a \llparenthesis Q \cap S \mathbin{;} R^{\smile} \rrparenthesis b \wedge b \llparenthesis R \rrparenthesis c$
              $\Leftarrow \langle$ "$\exists$-Introduction" $\rangle$
              $(a \llparenthesis Q \cap S \mathbin{;} R^{\smile} \rrparenthesis b \wedge b \llparenthesis R \rrparenthesis c)[b := b_2]$
              $= \langle$ Substitution, assumption (3), "Identity of $\wedge$" $\rangle$

Proof:
  Using "Relation inclusion":
    Subproof for `$\forall\, a \bullet \forall\, c \bullet a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c \Rightarrow a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$`:
      For any `$a$`, `$c$`:
        Assuming (1) `$a \llparenthesis (Q \mathbin{;} R) \cap S \rrparenthesis c$`:
          Assuming witness `$b_2$` **satisfying** (3) `$a \llparenthesis Q \rrparenthesis b_2 \wedge b_2 \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c$`
              by "Distributivity of $\wedge$ over $\exists$" and "Relation intersection"
                and "Relation composition" and assumption (1):
            $a \llparenthesis (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \rrparenthesis c$
            $\equiv \langle$ "Relation composition" $\rangle$
            $\exists\, b \bullet a \llparenthesis Q \cap S \mathbin{;} R^{\smile} \rrparenthesis b \wedge b \llparenthesis R \rrparenthesis c$
            $\Leftarrow \langle$ "$\exists$-Introduction" $\rangle$
            $(a \llparenthesis Q \cap S \mathbin{;} R^{\smile} \rrparenthesis b \wedge b \llparenthesis R \rrparenthesis c)[b := b_2]$
            $\equiv \langle$ Substitution, assumption (3), "Identity of $\wedge$" $\rangle$
            $a \llparenthesis Q \cap S \mathbin{;} R^{\smile} \rrparenthesis b_2$
            $\equiv \langle$ "Relation intersection", "Relation composition", "Relation converse" $\rangle$
            $a \llparenthesis Q \rrparenthesis b_2 \wedge \exists\, c_2 \bullet a \llparenthesis S \rrparenthesis c_2 \wedge b_2 \llparenthesis R \rrparenthesis c_2$
            $\equiv \langle$ Assumption (3), "Identity of $\wedge$" $\rangle$
            $\exists\, c_2 \bullet a \llparenthesis S \rrparenthesis c_2 \wedge b_2 \llparenthesis R \rrparenthesis c_2$
            $\Leftarrow \langle$ "$\exists$-Introduction" $\rangle$
            $(a \llparenthesis S \rrparenthesis c_2 \wedge b_2 \llparenthesis R \rrparenthesis c_2)[c_2 := c]$
            $\equiv \langle$ Substitution, assumption (3), "Identity of $\wedge$" $\rangle$
            true

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**: $A \lhd R \;=\; R \cap (A \times U)$
- **Domain antirestriction**: $A ⩤ R \;=\; R - (A \times U) \;=\; R \cap (\sim A \times U)$
- **Range restriction**: $R \rhd B \;=\; R \cap (U \times B)$
- **Range antirestriction**: $R ⩥ B \;=\; R - (U \times B) \;=\; R \cap (U \times \sim B)$

$$
\begin{array}{ll}
 & B \fatsemi (\{Jun\} \times \llcorner P \lrcorner) \;\cap\; (C \fatsemi C^{\smile}) \;\subseteq\; \mathbb{I} \\
\equiv & \langle \text{ Domain- and range restriction properties } \rangle \\
 & Dom(B \rhd \{Jun\}) \lhd (C \fatsemi C^{\smile}) \;\subseteq\; \mathbb{I}
\end{array}
$$

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 Chales Sanders Peirce.)

Relational Image and Relation Overriding

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**: $R\,\llparenthesis A \rrparenthesis \;=\; Ran(A \lhd R)$

  "Relational image of set A under relation R

  Notation as "generalised function application"...

$$
\begin{array}{ll}
 & B \fatsemi (\{Jun\} \times \llcorner P \lrcorner) \;\cap\; (C \fatsemi C^{\smile}) \;\subseteq\; \mathbb{I} \\
\equiv & \langle \text{ Domain- and range restriction properties } \rangle \\
 & Dom(B \rhd \{Jun\}) \lhd (C \fatsemi C^{\smile}) \;\subseteq\; \mathbb{I} \\
\equiv & \langle \text{ Relational image } \rangle \\
 & (B^{\smile}\,\llparenthesis \{Jun\} \rrparenthesis) \lhd (C \fatsemi C^{\smile}) \;\subseteq\; \mathbb{I}
\end{array}
$$

- **Relation overriding**: $R \oplus S \;=\; (Dom\ S ⩤ R) \cup S$

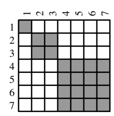
  "Updating R exactly where S relates with anything"

  In $\quad C \oplus \{\langle Aos, Jun \rangle\} \quad$, Aos called only Jun.

Recall: Equivalence Relations

Recall: A (homogeneous) relation $R : B \leftrightarrow B$ is called:

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \mathrel{⟮R⟯} b)$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : B \bullet b \mathrel{⟮R⟯} c \equiv c \mathrel{⟮R⟯} b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \mathrel{⟮R⟯} c \mathrel{⟮R⟯} d \Rightarrow b \mathrel{⟮R⟯} d)$
idempotent	$R \fatsemi R = R$	
equivalence	$\mathbb{I} \subseteq R = R \fatsemi R = R^{\smile}$	reflexive, transitive, symmetric



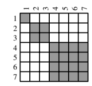
Equivalence Classes, Partitions

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:

$$x \in [b]_\Xi \;\equiv\; x \mathrel{⟮\Xi⟯} b \qquad \text{Equivalently:} \qquad [b]_\Xi = \Xi\,\llparenthesis \{b\} \rrparenthesis$$

Theorem: For an equivalence relation Ξ on B, the set $\{\, b : B \bullet \Xi\,\llparenthesis \{b\} \rrparenthesis \,\}$ of equivalence classes of Ξ is a partition of $\llcorner B \lrcorner$.

$$\{\ \{1\},\ \{2,3\},\ \{4,5,6,7\}\ \}$$



Definition (11.76): If $\;T$: **set** $t\;$ and $\;S$: **set** (**set** t), then:

S is a **partition of** T

$$
\begin{array}{l}
\equiv\; (\forall u, v \mid u \in S \wedge v \in S \wedge u \neq v \bullet u \cap v = \{\}) \\
\wedge\; (\bigcup u \mid u \in S \bullet u) = T
\end{array}
$$

Theorem: There is a bijective mapping
between equivalence relations on B and partitions of B.

The partition view can be useful for **implementing** equivalence relations.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-11

Part 1: Relational Formalisation of Graph Properties 1

Plan for Today

- Relational Formalisation of Simple Graph Properties
- Starting relation-algebraic calculational proofs

Relation-algebraic proof

- Will be an important topic of Exercises 10.*
- Will not be on Midterm 2

Midterm 2: Up to H14, H15, A5, Ex9.*

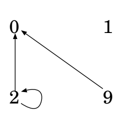
Recall: Simple Graphs

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: $\qquad G_1 = (\{2, 0, 1, 9\}, \{\langle 2, 0 \rangle, \langle 9, 0 \rangle, \langle 2, 2 \rangle\})$

Graphs are normally visualised via **graph drawings**:



Simple graphs are exactly relations!

Reasoning with relations is reasoning about graphs!

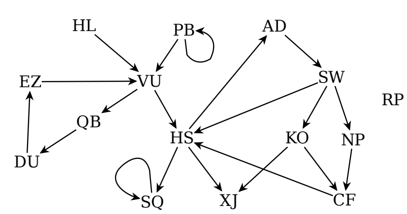
Simple Reachability Statements in Graph $G = (V, E)$

- No edge ends at node s

  $s \notin Ran\ E \qquad$ or $\qquad s \in \sim(Ran\ E) \qquad$ — s is called a **source** of G
- No edge starts at node s

  $s \notin Dom\ E \qquad$ or $\qquad s \in \sim(Dom\ E) \qquad$ — s is called a **sink** of G
- Node n_2 is reachable from node n_1 via a three-edge path

  $n_1 \mathrel{⟮E \fatsemi E \fatsemi E⟯} n_2$



Simple Reachability Statements in Graph $G_{\mathbb{N}} = (\llcorner \mathbb{N} \lrcorner, \ulcorner suc \urcorner)$

- No edge ends at node 0

  $0 \notin Ran\ \ulcorner suc \urcorner \qquad$ or $\qquad 0 \in \sim(Ran\ \ulcorner suc \urcorner) \qquad$ — 0 is a **source** of $G_{\mathbb{N}}$

  0 is the only source of $G_{\mathbb{N}}$: $\qquad \sim(Ran\ \ulcorner suc \urcorner) = \{0\}$
- s is a sink iff no edge starts at node s

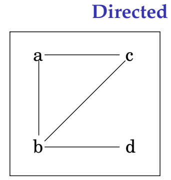
  $s \notin Dom\ \ulcorner suc \urcorner \qquad$ or $\qquad s \in \sim(Dom\ \ulcorner suc \urcorner)$

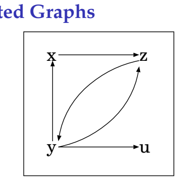
  $G_{\mathbb{N}}$ has no sinks: $\qquad Dom\ \ulcorner suc \urcorner = \llcorner \mathbb{N} \lrcorner \qquad$ or $\qquad \sim(Dom\ \ulcorner suc \urcorner) = \{\}$
- Node 5 is reachable from node 2 via a three-edge path:

  $2 \mathrel{⟮\ulcorner suc \urcorner \fatsemi \ulcorner suc \urcorner \fatsemi \ulcorner suc \urcorner⟯} 5$

$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow 6 \longrightarrow 7 \longrightarrow \ldots$$

Directed versus Undirected Graphs





- Edges in 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 if 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 graphs.

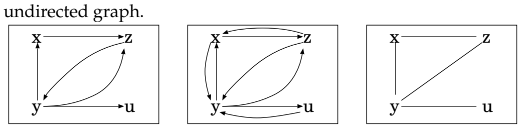
Symmetric Closure

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.



Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-11

Part 2: Starting Relation-Algebraic Calculational Proofs

Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall x, y \bullet x (\!\| R \|\!) y \equiv x (\!\| S \|\!) y)$
$R \subseteq S$	$\equiv$	$(\forall x, y \bullet x (\!\| R \|\!) y \Rightarrow x (\!\| S \|\!) y)$
$u (\!\| \{\} \|\!) v$	$\equiv$	$false$
$u (\!\| A \times B \|\!) v$	$\equiv$	$u \in A \land v \in B$
$u (\!\| \sim S \|\!) v$	$\equiv$	$\neg(u (\!\| S \|\!) v)$
$u (\!\| S \cup T \|\!) v$	$\equiv$	$u (\!\| S \|\!) v \lor u (\!\| T \|\!) v$
$u (\!\| S \cap T \|\!) v$	$\equiv$	$u (\!\| S \|\!) v \land u (\!\| T \|\!) v$
$u (\!\| S - T \|\!) v$	$\equiv$	$u (\!\| S \|\!) v \land \neg(u (\!\| T \|\!) v)$
$u (\!\| S \twoheadrightarrow T \|\!) v$	$\equiv$	$u (\!\| S \|\!) v \Rightarrow u (\!\| T \|\!) v$
$u (\!\| \mathrm{id}\ A \|\!) v$	$\equiv$	$u = v \in A$
$u (\!\| \mathbb{I} \|\!) v$	$\equiv$	$u = v$
$u (\!\| R^{\smile} \|\!) v$	$\equiv$	$v (\!\| R \|\!) u$
$u (\!\| R \fatsemi S \|\!) v$	$\equiv$	$(\exists x \bullet u (\!\| R \|\!) x \land x (\!\| S \|\!) v)$
$u (\!\| R \backslash S \|\!) v$	$\equiv$	$(\forall x \bullet x (\!\| R \|\!) u \Rightarrow x (\!\| S \|\!) v)$
$u (\!\| S / R \|\!) v$	$\equiv$	$(\forall x \bullet v (\!\| R \|\!) x \Rightarrow u (\!\| S \|\!) x)$

Using Extensionality/Inclusion and the Translation Table, you Proved:

Theorem "Self-inverse of $^{\smile}$": $R^{\smile\smile} = R$
Theorem "Converse of $\cap$": $(R \cap S)^{\smile} = R^{\smile} \cap S^{\smile}$
Theorem "Converse of $\fatsemi$": $(R \fatsemi S)^{\smile} = S^{\smile} \fatsemi R^{\smile}$
Theorem "Converse of $\mathbb{I}$": $\mathbb{I}^{\smile} = \mathbb{I}$
Theorem "Isotonicity of $^{\smile}$": $R \subseteq S \equiv R^{\smile} \subseteq S^{\smile}$
Theorem "Converse of $\cup$": $(R \cup S)^{\smile} = R^{\smile} \cup S^{\smile}$
Theorem "Distributivity of $\fatsemi$ over $\cup$": $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
Theorem "Sub-distributivity of $\fatsemi$ over $\cap$": $Q \fatsemi (R \cap S) \subseteq Q \fatsemi R \cap Q \fatsemi S$
Theorem "Left-identity of $\fatsemi$" "Identity of $\fatsemi$": $\mathbb{I} \fatsemi R = R$
Theorem "Right-identity of $\fatsemi$" "Identity of $\fatsemi$": $R \fatsemi \mathbb{I} = R$
Theorem "Composition of reflexive relations": reflexive $R \Rightarrow$ reflexive $S \Rightarrow$ reflexive $(R \fatsemi S)$
Theorem "Converse of reflexive relations": reflexive $R \Rightarrow$ reflexive $(R^{\smile})$
Theorem "Converse reflects reflectivity": reflexive $(R^{\smile}) \Rightarrow$ reflexive R
Theorem "Converse of transitive relations": transitive $R \Rightarrow$ transitive $(R^{\smile})$
Theorem "Associativity of $\fatsemi$": $(Q \fatsemi R) \fatsemi S = Q \fatsemi (R \fatsemi S)$
Theorem "Distributivity of $\fatsemi$ over $\cup$": $(Q \cup R) \fatsemi S = Q \fatsemi S \cup R \fatsemi S$
Theorem "Sub-distributivity of $\fatsemi$ over $\cap$": $(Q \cap R) \fatsemi S \subseteq Q \fatsemi S \cap R \fatsemi S$
Theorem "Monotonicity of $\fatsemi$": $Q \subseteq R \Rightarrow Q \fatsemi S \subseteq R \fatsemi S$
Theorem "Converse of $\{\}$": $\{\}^{\smile} = \{\}$
Theorem "Co-difunctionality" "Hesitation": $R \subseteq R \fatsemi R^{\smile} \fatsemi R$
Theorem "Modal rule": $(Q \fatsemi R) \cap S \subseteq Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$
Theorem "Dedekind rule": $(Q \fatsemi R) \cap S \subseteq (Q \cap S \fatsemi R^{\smile}) \fatsemi (R \cap Q^{\smile} \fatsemi S)$
Theorem "Schröder": $Q \fatsemi R \subseteq S \equiv {\sim} S \fatsemi R^{\smile} \subseteq {\sim} Q$

All subexpressions have $\mathbb{B}$ or $\_\leftrightarrow\_$ types!
Equations of relational expressions:
Relation algebra

Relation Algebra

- 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 $\_\cap\_$ and maxima $\_\cup\_$ (which are monotonic)
  - least relation $\{\}$ and largest ("universal") relation U $(= \llcorner B \lrcorner \times \llcorner C \lrcorner)$
  - complement operation $\sim\_$ such that $R \cap {\sim} R = \{\}$ and $R \cup {\sim} R = U$
  - relative pseudo-complement $R \twoheadrightarrow S = {\sim} R \cup S$
- The composition operation $\_\fatsemi\_$
  - 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: $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
- The converse operation $\_^{\smile}$
  - maps relation $R : B \leftrightarrow C$ to $R^{\smile} : C \leftrightarrow B$
  - is self-inverse ($R^{\smile\smile} = R$) and monotonic
  - is contravariant wrt. composition: $(R \fatsemi S)^{\smile} = S^{\smile} \fatsemi R^{\smile}$
- The Dedekind rule holds: $Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^{\smile}) \fatsemi (R \cap Q^{\smile} \fatsemi S)$
- The Schröder equivalences hold:
  $Q \fatsemi R \subseteq S \equiv Q^{\smile} \fatsemi {\sim} S \subseteq {\sim} R$ and $Q \fatsemi R \subseteq S \equiv {\sim} S \fatsemi R^{\smile} \subseteq {\sim} Q$
- $\fatsemi$ has left-residuals $S / R = {\sim}({\sim} S \fatsemi R^{\smile})$ and right-residuals $Q \backslash S = {\sim}(Q^{\smile} \fatsemi {\sim} S)$

Monotonicity of Relation Composition

Relation composition is monotonic in both arguments:

$$Q \subseteq R \quad \Rightarrow \quad Q \fatsemi S \subseteq R \fatsemi S$$
$$Q \subseteq R \quad \Rightarrow \quad P \fatsemi Q \subseteq P \fatsemi R$$

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 $Q \fatsemi S \subseteq R \fatsemi S$:

$R \fatsemi S$
$=$ ⟨ Assumption $Q \cup R = R$ ⟩
$(Q \cup R) \fatsemi S$
$=$ ⟨ (14.23) Distributivity of $\fatsemi$ over $\cup$ ⟩
$Q \fatsemi S \cup R \fatsemi S$
$\supseteq$ ⟨ (11.31) Strengthening $S \subseteq S \cup T$ ⟩
$Q \fatsemi S$

Relation-Algebraic Proof of Sub-Distributivity

Use set-algebraic properties and **Monotonicity of $\fatsemi$**: $Q \subseteq R \Rightarrow P \fatsemi Q \subseteq P \fatsemi R$

to prove: **Subdistributivity of $\fatsemi$ over $\cap$**: $Q \fatsemi (R \cap S) \subseteq (Q \fatsemi R) \cap (Q \fatsemi S)$

$Q \fatsemi (R \cap S)$
$=$ ⟨ Idempotence of $\cap$ (11.35) ⟩
$(Q \fatsemi (R \cap S)) \cap (Q \fatsemi (R \cap S))$
$\subseteq$ ⟨ **Mon. of $\cap$ with Mon. of $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ ⟩
$(Q \fatsemi (R \cap S)) \cap (Q \fatsemi S)$
$\subseteq$ ⟨ **Mon. of $\cap$ with Mon. of $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ — separate $\subseteq$-steps normally needed in CalcCheck! ⟩
$(Q \fatsemi R) \cap (Q \fatsemi S)$

(Previously we proved monotonicity from subdistributivity.)

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	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$	$(\forall b, c \bullet b (\!\| R \|\!) c \land c (\!\| R \|\!) b \Rightarrow b = c)$
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:

$\mathbb{I}$
$=$ ⟨ Symmetry of $\mathbb{I}$ ⟩
$\mathbb{I}^{\smile}$
$\subseteq$ ⟨ **Mon. $^{\smile}$ with** Reflexivity of R ⟩
$R^{\smile}$

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	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

Theorem "Idempotency from reflexive and transitive":
reflexive $R \Rightarrow$ transitive $R \Rightarrow$ idempotent R
Proof:
Assuming \`reflexive $R$\`, \`transitive $R$\`:
idempotent R
$\equiv$ ⟨ "Definition of idempotency" ⟩
$R \fatsemi R = R$
$\equiv$ ⟨ "Mutual inclusion" ⟩
$R \fatsemi R \subseteq R \land R \subseteq R \fatsemi R$
$\equiv$ ⟨ "Definition of transitivity", assumption \`transitive $R$\`, "Identity of $\land$" ⟩
$R \subseteq R \fatsemi R$
$\equiv$ ⟨ "Identity of $\fatsemi$" ⟩
$R \fatsemi \mathbb{I} \subseteq R \fatsemi R$
$\Leftarrow$ ⟨ "Monotonicity of $\fatsemi$" ⟩
$\mathbb{I} \subseteq R$
$\equiv$ ⟨ Assumption \`reflexive $R$\` with "Definition of reflexivity" ⟩
true

Reflexive and Transitive Implies Idempotent — “and using with”

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” ⟩
$R \fatsemi R = R$
≡ ⟨ “Mutual inclusion” ⟩
$R \fatsemi R \subseteq R \wedge R \subseteq R \fatsemi R$
≡ ⟨ Assumption \`transitive $R$\`, “Identity of ∧” ⟩
$R \subseteq R \fatsemi R$
≡ ⟨ “Identity of $\fatsemi$” ⟩
$R \fatsemi \mathbb{I} \subseteq R \fatsemi R$
⇐ ⟨ “Monotonicity of $\fatsemi$” ⟩
$\mathbb{I} \subseteq R$
≡ ⟨ Assumption \`reflexive $R$\` ⟩
true

reflexive	$\mathbb{I}$	$\subseteq$	R
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

Reflexive and Transitive Implies Idempotent — Semi-formal

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.
Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R \fatsemi R$:

R
$=$ ⟨ Identity of $\fatsemi$ ⟩
$R \fatsemi \mathbb{I}$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Reflexivity of R ⟩
$R \fatsemi R$

Reflexive and Transitive Implies Idempotent — Cyclic ⊆-chain Proving \` = \`

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”:
Using “Definition of idempotency”:
Subproof for \`$R \fatsemi R = R$\`:
$R \fatsemi R$
⊆ ⟨ Assumption \`transitive $R$\` ⟩
R
= ⟨ “Identity of $\fatsemi$” ⟩
$R \fatsemi \mathbb{I}$
⊆ ⟨ “Monotonicity of $\fatsemi$” with assumption \`reflexive $R$\` ⟩
$R \fatsemi R$

reflexive	$\mathbb{I}$	$\subseteq$	R
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

Symmetric and Transitive Implies Idempotent

symmetric	$R^{\smile}$	$=$	R	$(\forall\, b, c : B \bullet b\,(R)\,c \equiv c\,(R)\,b)$
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	

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 ∩, Identity of $\fatsemi$ ⟩
$R \fatsemi \mathbb{I} \cap R$
$\subseteq$ ⟨ **Modal rule** $Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$ ⟩
$R \fatsemi (\mathbb{I} \cap R^{\smile} \fatsemi R)$
$\subseteq$ ⟨ **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 ⟩
$R \fatsemi R$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-15

Part 1: Relational Formalisation of Graph Properties 2

Plan for Today

- Relational Formalisation of Simple Graph Properties 2
  - Reachability: (Reflexive) transitive closures
- Relation-algebraic calculational proofs 2

Relation-algebraic proof

- Will be an important topic of Exercises 10.*
- Will not be on Midterm 2

Midterm 2: Up to H14, H15, A5, Ex9.*

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., $<$, $>$, $\subset$, $\supset$)

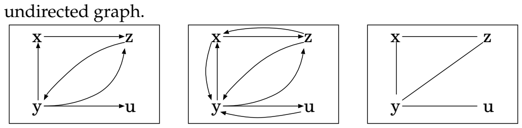
Recall: Symmetric Closure

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.



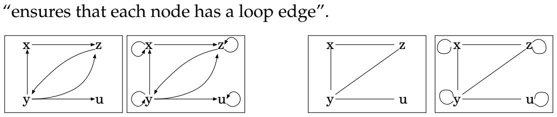
Reflexive Closure

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
- $R \subseteq Q$
- $Q \fatsemi Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge P \fatsemi P \subseteq P \bullet Q \subseteq P)$

Definition: The transitive closure of $R : B \leftrightarrow B$ is written R^+.

Theorem: $R^+ = (\bigcap P \mid R \subseteq P \wedge P \fatsemi P \subseteq P \bullet P)$.

Theorem: $R^+ = (\bigcup i : \mathbb{N} \mid i > 0 \bullet R^i)$

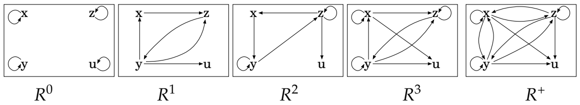
Powers of a homogeneous relation $R : B \leftrightarrow B$:

- $R^0 = \mathbb{I}$
- $R^1 = R$
- $R^{n+1} = R^n \fatsemi R$

Transitive Closure via Powers

Powers of a homogeneous relation $R : B \leftrightarrow B$:

- $R^0 = \mathbb{I}$
- $R^1 = R$
- $R^{n+1} = R^n \fatsemi R$
- $R^2 = R \fatsemi R$
- $R^3 = R \fatsemi R \fatsemi R$
- $R^4 = R \fatsemi R \fatsemi R \fatsemi R$
- R^i is reachability via exactly i many R-steps



- $R^+ = (\bigcup\, i : \mathbb{N} \mid i > 0 \bullet R^i)$
- $R^+ = R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Transitive closure R^+ is reachability via at least one R-step

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

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q \wedge Q \fatsemi Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P \fatsemi P \subseteq P \bullet Q \subseteq P)$

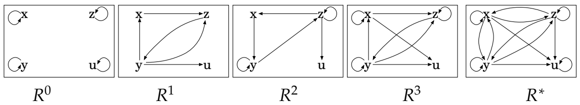
Definition: The reflexive transitive closure of R is written R^*.

Theorem: $R^* = (\bigcap P \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P \fatsemi P \subseteq P \bullet P)$.

Theorem: $R^* = (\bigcup\, i : \mathbb{N} \bullet R^i)$

Transitive and Reflexive Transitive Closure via Powers

- R^i is reachability via exactly i many R-steps



- $R^+ = (\bigcup\, i : \mathbb{N} \mid i > 0 \bullet R^i)$
- $R^+ = R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Transitive closure R^+ is reachability via at least one R-step

---

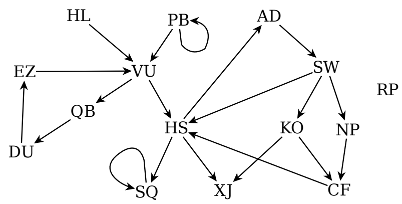
- $R^* = (\bigcup\, i : \mathbb{N} \bullet R^i)$
- $R^* = \mathbb{I} \cup R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Reflexive transitive closure R^* is reachability via any number of R-steps

---

- Variants of the **Warshall algorithm** calculate these closures in cubic time.

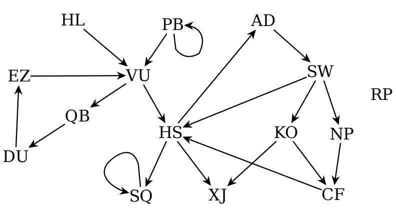
Reachability in graph $G = (V, E)$ — 1 (ctd.)

- No edge ends at node s
  $s \notin Ran\ E$ or $s \in {\sim}(Ran\ E)$ — s is called a **source** of G
- No edge starts at node s
  $s \notin Dom\ E$ or $s \in {\sim}(Dom\ E)$ — s is called a **sink** of G
- Node n_2 is reachable from node n_1 via a three-edge path
  $n_1 \,(E^3)\, n_2$ or $n_1 \,(E \fatsemi E \fatsemi E)\, n_2$
- Node y is **reachable** from node x
  $x \,(E^*)\, y$ — **reachability**



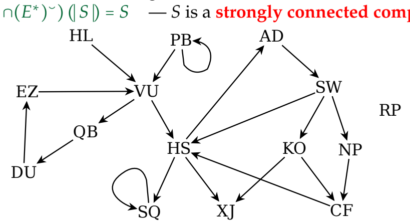
Reachability in graph $G = (V, E)$ — 2

- Node y is **reachable** from node x
  $x \,(E^*)\, y$ — **reachability**
- Every node is reachable from node r
  $\{r\} \times V \subseteq E^*$ or $E^* (\!| \{r\} |\!) = V$ — r is called a **root** of G
- Node y is **reachable via a non-empty path** from node x: $x \,(E^+)\, y$
- Nodes x lies on a cycle: $x \,(E^+)\, x$ or $x \,(E^+ \cap \mathbb{I})\, x$ 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
  $n_1 \,(E^* \cap (E^*)^{\smile})\, n_2$ — n_1 and n_2 are **strongly connected**
- S is an equivalence class of strong connectedness between nodes
  $S \times S \subseteq E^* \wedge (E^* \cap (E^*)^{\smile}) (\!| S |\!) = S$ — S is a **strongly connected component (SCC)** of G

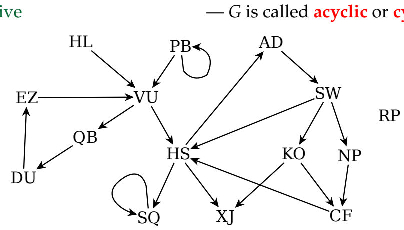


Reachability in graph $G = (V, E)$ — 4

- A node n is said to "lie on a cycle" if there is a non-empty path from n to n

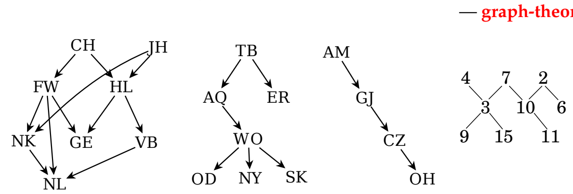
  $cycleNodes \;:=\; Dom(E^+ \cap \mathbb{I})$

- 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: $E \fatsemi E^{\smile} \subseteq \mathbb{I}$ 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 2021

Wolfram Kahl

2021-11-15

Part 2: Continuing Relation-Algebraic Calculational Proofs

Recall: Relation Algebra

- 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 $\_\cap\_$ and maxima $\_\cup\_$ (which are monotonic)
  - least relation $\{\}$ and largest ("universal") relation U $(= \llcorner B \lrcorner \times \llcorner C \lrcorner)$
  - complement operation ${\sim}\_$ such that $R \cap {\sim}R = \{\}$ and $R \cup {\sim}R = U$
  - relative pseudo-complement $R \Rightarrow S \;=\; {\sim}R \cup S$
- The composition operation $\_\fatsemi\_$
  - 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: $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
- The converse operation $\_^{\smile}$
  - maps relation $R : B \leftrightarrow C$ to $R^{\smile} : C \leftrightarrow B$
  - is self-inverse $(R^{\smile\smile} = R)$ and monotonic
  - is contravariant wrt. composition: $(R \fatsemi S)^{\smile} = S^{\smile} \fatsemi R^{\smile}$
- The Dedekind rule holds: $Q \fatsemi R \cap S \;\subseteq\; (Q \cap S \fatsemi R^{\smile}) \fatsemi (R \cap Q^{\smile} \fatsemi S)$
- The Schröder equivalences hold:
  $Q \fatsemi R \subseteq S \;\equiv\; Q^{\smile} \fatsemi {\sim}S \subseteq {\sim}R$ and $Q \fatsemi R \subseteq S \;\equiv\; {\sim}S \fatsemi R^{\smile} \subseteq {\sim}Q$
- $\fatsemi$ has left-residuals $S \,/\, R = {\sim}({\sim}S \fatsemi R^{\smile})$ and right-residuals $Q \backslash S = {\sim}(Q^{\smile} \fatsemi {\sim}S)$

Recall: Properties of Homogeneous Relations

reflexive	$\mathbb{I} \subseteq R$	$(\forall b : B \bullet b \llparenthesis R \rrparenthesis b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall b : B \bullet \neg(b \llparenthesis R \rrparenthesis b))$
symmetric	$R^{\smile} = R$	$(\forall b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall b, c \bullet b \llparenthesis R \rrparenthesis c \wedge c \llparenthesis R \rrparenthesis b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall b, c : B \bullet b \llparenthesis R \rrparenthesis c \Rightarrow \neg(c \llparenthesis R \rrparenthesis b))$
transitive	$R \mathbin{;} R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \wedge c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$

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., $<, >, \subset, \supset$)

Most Homogeneous Relation Properties are Preserved by Intersection

reflexive	$\mathbb{I} \subseteq R$
irreflexive	$\mathbb{I} \cap R = \{\}$
transitive	$R \mathbin{;} R \subseteq R$
idempotent	$R \mathbin{;} R = R$

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:

$\mathbb{I}$
$= \langle$ Idempotence of $\cap$ $\rangle$
$\mathbb{I} \cap \mathbb{I}$
$\subseteq \langle$ Mon. of $\cap$ with Refl. R $\rangle$
$R \cap \mathbb{I}$
$\subseteq \langle$ Mon. of $\cap$ with Refl. S $\rangle$
$R \cap S$

Transitivity:

$(R \cap S) \mathbin{;} (R \cap S)$
$\subseteq \langle$ Sub-distributivity of $\mathbin{;}$ over $\cap$ $\rangle$
$(R \mathbin{;} R) \cap (R \mathbin{;} S) \cap (S \mathbin{;} R) \cap (S \mathbin{;} S)$
$\subseteq \langle$ Weakening $X \cap Y \subseteq X$ $\rangle$
$(R \mathbin{;} R) \cap (S \mathbin{;} S)$
$\subseteq \langle$ Mon. $\cap$ with transitivity of R and S $\rangle$
$R \cap S$

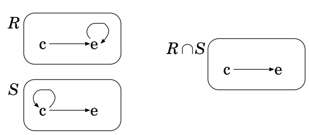
Most Homogeneous Relaton Properties are Preserved by Intersection

reflexive	$\mathbb{I} \subseteq R$
irreflexive	$\mathbb{I} \cap R = \{\}$
transitive	$R \mathbin{;} R \subseteq R$
idempotent	$R \mathbin{;} R = R$

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	$R \mathbin{;} R \subseteq R$
idempotent	$R \mathbin{;} R = R$

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:

$\mathbb{I}$
$\subseteq \langle$ Reflexivity of R $\rangle$
R
$\subseteq \langle$ Weakening $X \subseteq X \cup Y$ $\rangle$
$R \cup S$

Irreflexivity:

$\mathbb{I} \cap (R \cup S)$
$= \langle$ Distributivity of $\cap$ over $\cup$ $\rangle$
$(\mathbb{I} \cap R) \cup (\mathbb{I} \cap S)$
$= \langle$ Irreflexivity of R and S $\rangle$
$\{\} \cup \{\}$
$= \langle$ Idempotence of $\cup$ $\rangle$
$\{\}$

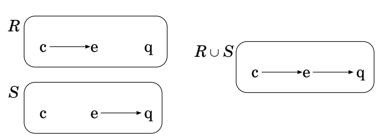
Some Homogeneous Relation Properties are Preserved by Union

reflexive	$\mathbb{I} \subseteq R$
irreflexive	$\mathbb{I} \cap R = \{\}$
transitive	$R \mathbin{;} R \subseteq R$
idempotent	$R \mathbin{;} R = R$

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	$R \mathbin{;} R \subseteq R$
idempotent	$R \mathbin{;} R = R$

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 $R \subseteq R^{\smile}$:

R
$= \langle$ Self-inverse of converse $\rangle$
$(R^{\smile})^{\smile}$
$\subseteq \langle$ Mon. of $^{\smile}$ with Assumption $R^{\smile} \subseteq R$ $\rangle$
$R^{\smile}$

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	$R \mathbin{;} R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \mathbin{;} R = R$	

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 $R \subseteq R \mathbin{;} R$:

R
$= \langle$ Idempotence of $\cap$, Identity of $\mathbin{;}$ $\rangle$
$R \mathbin{;} \mathbb{I} \cap R$
$\subseteq \langle$ **Modal rule** $Q \mathbin{;} R \cap S \subseteq Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$ $\rangle$
$R \mathbin{;} (\mathbb{I} \cap R^{\smile} \mathbin{;} R)$
$\subseteq \langle$ **Mon. $\mathbin{;}$ with** Weakening $X \cap Y \subseteq X$ $\rangle$
$R \mathbin{;} R^{\smile} \mathbin{;} R$
$= \langle$ Symmetry of R $\rangle$
$R \mathbin{;} R \mathbin{;} R$
$\subseteq \langle$ **Mon. $\mathbin{;}$ with** Transitivity of R $\rangle$
$R \mathbin{;} R$

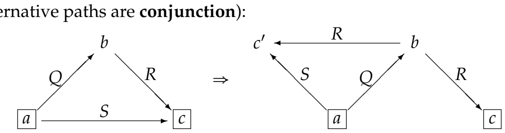
Modal Rules— Converse as Over-Approximation of Inverse

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \subseteq Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \subseteq (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Useful to "**make information available locally**" (Q is replaced with $Q \cap S \mathbin{;} R^{\smile}$) 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

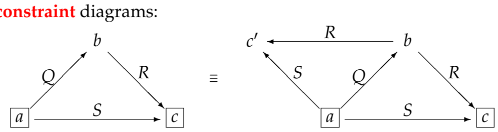
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \subseteq Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \subseteq (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Equivalently, using $M \subseteq N \equiv M = M \cap N$ etc.:

$$Q \mathbin{;} R \cap S = Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S) \cap S$$
$$Q \mathbin{;} R \cap S = (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \cap S$$

In **constraint** diagrams:



$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \equiv$
$\equiv \quad (\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c' \wedge b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-18

Part 1: Inductive Datastructures: Trees

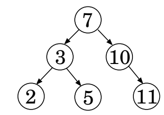
Plan for Today

- **Tree Datastructures; Structural Induction**
- Relation-Algebraic Proof: Modal Rules, Dedekind Rule

Inductively-defined Tree Data Structures

Binary (search) trees

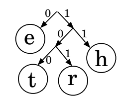
```
data BTree = EmptyB
  | Branch BTree Int BTree
```



```
bt1left = Branch
  (Branch EmptyB 2 EmptyB)
  3
  (Branch EmptyB 5 EmptyB)
bt1right = Branch
  EmptyB
  10
  (Branch EmptyB 11 EmptyB)
```

Huffman trees

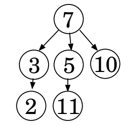
```
data HTree = Leaf Char
  | HBranch HTree HTree
```



```
hTree1 = HBranch (Leaf 'e')
  (HBranch
    (HBranch (Leaf 't') (Leaf 'r'))
    (Leaf 'h'))
decode hTree1 "100110" = "the"
```

Arbitrarily branching

```
data Tree
  = Branch Int [Tree]
```

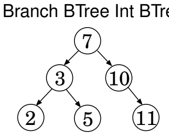


```
t1left = Branch 7
  [Branch 3 [Branch 2 []]
  ,Branch 5 [Branch 11 []]
  ,Branch 10 []
  ]
```

Binary Trees (Exercise 10.4)

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)
```

```
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 ▷ ◬))

Fact “Alternative definition of `t1`”:
  t1 = (⌈ 2 ⌋ ◁ 3 ▷ ⌈ 5 ⌋)
       ◁ 7 ▷
       (◬ ◁ 10 ▷ ⌈ 11 ⌋)
```

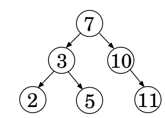
Binary Trees (Exercise 10.4)

```
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 ▷ ◬))

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)
```

Recall: Induction — Reduction via Well-founded Relations

- Goal: prove $(\forall x : U \bullet P\,x)$ for some property $P : U \to \mathbb{B}$ (with $\neg occurs(\text{‘}x\text{’}, \text{‘}P\text{’})$)
- Situation: Elements of U are related via $\_\succsim\_ : U \to U \to \mathbb{B}$ with “simpler” elements (constituents, predecessors, parts, …)
  “$y \precsim x$” 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 $x : U$ there is a proof that

  if $P\,y$ for all predecessors y of x, then $P\,x$,

  then for every $z : U$ with $\neg(P\,z)$:
  - there is a predecessor u of z with $\neg(P\,u)$
  - and so there is an infinite $\succsim$-chain (of elements c with $\neg(P\,c)$) starting at z.

Theorem (12.19) **Mathematical induction over** $(U, \precsim)$**:**
If there are no infinite $\succsim$-chains in U, that is, **if $\precsim$ is well-founded**, then:

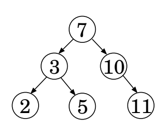
$$(\forall x \bullet P\,x) \quad \equiv \quad (\forall x \bullet (\forall y \mid y \precsim x \bullet P\,y) \Rightarrow P\,x)$$

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** $(U, \precsim)$, **if $\precsim$ is well-founded**

$$(\forall x \bullet P\,x) \quad \equiv \quad (\forall x \bullet (\forall y \mid y \precsim x \bullet P\,y) \Rightarrow P\,x)$$

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)
```

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
- …
- Every “data” in Haskell defines a (possibly degenerated) tree datastructure

In programming:

- **Trees are easy to deal with.**
- Graphs, even DAGs, can be tricky
  - — even with good APIs.
  - Choosing “the right” API is already hard!
  - The same holds for relations!

    — Because relations *are* graphs…

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-18

Part 2: Continuing Relation-Algebraic Calculational Proofs

Recall: Relation Algebra

- 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 $\_\cap\_$ and maxima $\_\cup\_$ (which are monotonic)
  - least relation $\{\}$ and largest (“universal”) relation U ($= \llcorner B \lrcorner \times \llcorner C \lrcorner$)
  - complement operation $\sim\_$ such that $R \cap \sim R = \{\}$ and $R \cup \sim R = U$
  - relative pseudo-complement $R \Rightarrow S = \sim R \cup S$
- The composition operation $\_\fatsemi\_$
  - 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: $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
- 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: $(R \fatsemi S)\breve{} = S\breve{} \fatsemi R\breve{}$
- The Dedekind rule holds: $Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R\breve{}) \fatsemi (R \cap Q\breve{} \fatsemi S)$
- The Schröder equivalences hold:
  $Q \fatsemi R \subseteq S \equiv Q\breve{} \fatsemi \sim S \subseteq \sim R$ and $Q \fatsemi R \subseteq S \equiv \sim S \fatsemi R\breve{} \subseteq \sim Q$
- $\fatsemi$ has left-residuals $S \mathbin{/} R = \sim(\sim S \fatsemi R\breve{})$ and right-residuals $Q \backslash S = \sim(Q\breve{} \fatsemi \sim S)$

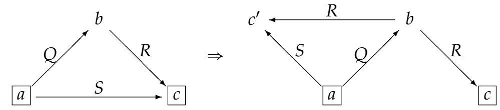
Modal Rules— Converse as Over-Approximation of Inverse

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Useful to "**make information available locally**" (Q is replaced with $Q \cap S \mathbin{;} R^{\smile}$) for use in further proof steps.

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \Rightarrow$
$(\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c')$

Modal Rules modulo Inclusion via Intersection

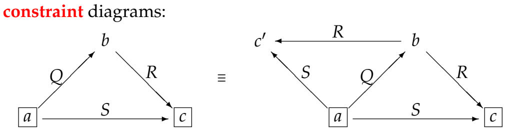
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Equivalently, using $M \subseteq N \;\equiv\; M = M \cap N$ etc.:

$$Q \mathbin{;} R \cap S \;=\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S) \cap S$$
$$Q \mathbin{;} R \cap S \;=\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R \cap S$$

In **constraint** diagrams:



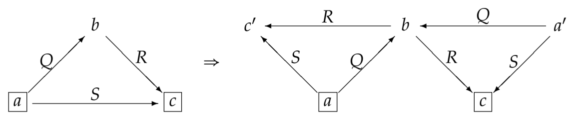
$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \equiv$
$\equiv \quad (\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c' \wedge b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c)$

Modal Rules and Dedekind Rule

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Equivalent: **Dedekind Rule:** $\quad Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$



Dedekind Rule modulo Inclusion via Intersection

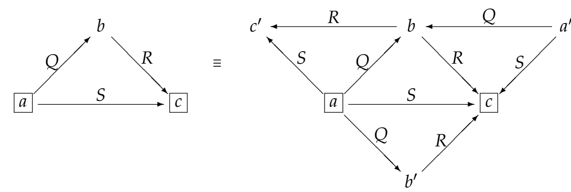
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$$
$$Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} R$$

Equivalent: **Dedekind Rule:** $\quad Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$

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: $\quad Q \mathbin{;} R \cap S \;\subseteq\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$

Dedekind (sharp version): $\quad Q \mathbin{;} R \cap S \;=\; (Q \cap S \mathbin{;} R^{\smile}) \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S) \cap S$

Proofs: Exercise!

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	$R \mathbin{;} R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \mathbin{;} R = R$	

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 $R \subseteq R \mathbin{;} R$:

$\quad R$
$= \langle$ Idempotence of $\cap$, Identity of $;$ $\rangle$
$\quad R \mathbin{;} \mathbb{I} \cap R$
$\subseteq \langle$ **Modal rule** $\quad Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$ $\rangle$
$\quad R \mathbin{;} (\mathbb{I} \cap R^{\smile} \mathbin{;} R)$
$\subseteq \langle$ **Mon. $;$ with** Weakening $X \cap Y \subseteq X$ $\rangle$
$\quad R \mathbin{;} R^{\smile} \mathbin{;} R$
$= \langle$ Symmetry of R $\rangle$
$\quad R \mathbin{;} R \mathbin{;} R$
$\subseteq \langle$ **Mon. $;$ with** Transitivity of R $\rangle$
$\quad R \mathbin{;} R$

Recall: Properties of Heterogeneous Relations

A relation $R : B \leftrightarrow C$ is called:

univalent determinate	$R^{\smile} \mathbin{;} R \subseteq \mathbb{I}$	$\forall b, c_1, c_2 \bullet b \llparenthesis R \rrparenthesis c_1 \wedge b \llparenthesis R \rrparenthesis c_2 \Rightarrow c_1 = c_2$
total	$\mathit{Dom}\ R = B$ $\mathbb{I} \subseteq R \mathbin{;} R^{\smile}$	$\forall b : B \bullet (\exists c : C \bullet b \llparenthesis R \rrparenthesis c)$
injective	$R \mathbin{;} R^{\smile} \subseteq \mathbb{I}$	$\forall b_1, b_2, c \bullet b_1 \llparenthesis R \rrparenthesis c \wedge b_2 \llparenthesis R \rrparenthesis c \Rightarrow b_1 = b_2$
surjective	$\mathit{Ran}\ R = C$ $\mathbb{I} \subseteq R^{\smile} \mathbin{;} R$	$\forall c : C \bullet (\exists b : B \bullet b \llparenthesis R \rrparenthesis c)$
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 $F \mathbin{;} (R \cap S) = (F \mathbin{;} R) \cap (F \mathbin{;} S)$

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, $F^{\smile} \mathbin{;} F \subseteq \mathbb{I}$

$\quad (F \mathbin{;} R) \cap (F \mathbin{;} S)$
$\subseteq \langle$ **"Modal rule"** $\quad Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$ $\rangle$
$\quad F \mathbin{;} (R \cap (F^{\smile} \mathbin{;} F \mathbin{;} S))$
$\subseteq \langle$ **"Mon. of $;$" with "Mon. of $\cap$" with "Mon. of $;$" with** assumption \`$F^{\smile} \mathbin{;} F \subseteq \mathbb{I}$\` $\rangle$
$\quad F \mathbin{;} (R \cap (\mathbb{I} \mathbin{;} S))$
$= \langle$ "Identity of $;$" $\rangle$
$\quad F \mathbin{;} (R \cap S)$

Ex10.* will practice such relation-algebraic proofs.

New Keywords: **Monotonicity** and **Antitonicity**

If $F : A \leftrightarrow B$ is univalent, then $F \mathbin{;} (R \cap S) = (F \mathbin{;} R) \cap (F \mathbin{;} S)$

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, $F^{\smile} \mathbin{;} F \subseteq \mathbb{I}$

$\quad (F \mathbin{;} R) \cap (F \mathbin{;} S)$
$\subseteq \langle$ **"Modal rule"** $\quad Q \mathbin{;} R \cap S \;\subseteq\; Q \mathbin{;} (R \cap Q^{\smile} \mathbin{;} S)$ $\rangle$
$\quad F \mathbin{;} (R \cap (F^{\smile} \mathbin{;} F \mathbin{;} S))$
$\subseteq \langle$ **Monotonicity with** assumption \`$F^{\smile} \mathbin{;} F \subseteq \mathbb{I}$\` $\rangle$
$\quad F \mathbin{;} (R \cap (\mathbb{I} \mathbin{;} S))$
$= \langle$ "Identity of $;$" $\rangle$
$\quad F \mathbin{;} (R \cap S)$

Ex10.* will practice such relation-algebraic proofs.

For Univalent Relations . . . — LADM Hint, for M2-like Context

Theorem: If $F : A \leftrightarrow B$ is univalent, then $F \mathbin{;} (R \cap S) = (F \mathbin{;} R) \cap (F \mathbin{;} S)$

Hint: Assume determinacy; then show the equation using **relation extensionality**, and start from the RHS $\langle b, d \rangle \in (F \mathbin{;} R) \cap (F \mathbin{;} S)$. In the expansions of the two relation compositions here, introduce different bound variables.

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x \llparenthesis F \fatsemi R \cap F \fatsemi S \rrparenthesis z$
$\equiv \langle ? \rangle$
$x \llparenthesis F \fatsemi (R \cap S) \rrparenthesis z$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x \llparenthesis F \fatsemi R \cap F \fatsemi S \rrparenthesis z$
$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
$(\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z) \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle ? \rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \rrparenthesis z \wedge y \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Relation intersection" $\rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \cap S \rrparenthesis z$
$\equiv \langle$ "Relation composition" $\rangle$
$x \llparenthesis F \fatsemi (R \cap S) \rrparenthesis z$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x \llparenthesis F \fatsemi R \cap F \fatsemi S \rrparenthesis z$
$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
$(\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z) \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle ? \rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \rrparenthesis z \wedge y \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Relation intersection" $\rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \cap S \rrparenthesis z$
$\equiv \langle$ "Relation composition" $\rangle$
$x \llparenthesis F \fatsemi (R \cap S) \rrparenthesis z$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet$
$a \llparenthesis R \rrparenthesis b_1 \wedge a \llparenthesis R \rrparenthesis b_2$
$\Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x \llparenthesis F \fatsemi R \cap F \fatsemi S \rrparenthesis z$
$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
$(\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z) \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle ? \rangle$
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \wedge x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle ? \rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \rrparenthesis z \wedge y \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Relation intersection" $\rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \cap S \rrparenthesis z$
$\equiv \langle$ "Relation composition" $\rangle$
$x \llparenthesis F \fatsemi (R \cap S) \rrparenthesis z$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet$
$a \llparenthesis R \rrparenthesis b_1 \wedge a \llparenthesis R \rrparenthesis b_2$
$\Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x \llparenthesis F \fatsemi R \cap F \fatsemi S \rrparenthesis z$
$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
$(\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z) \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle ? \rangle$
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \wedge x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Trading for ∃", "One-point rule for ∃",
substitution, "Idempotency of ∧" $\rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \rrparenthesis z \wedge y \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Relation intersection" $\rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \cap S \rrparenthesis z$
$\equiv \langle$ "Relation composition" $\rangle$
$x \llparenthesis F \fatsemi (R \cap S) \rrparenthesis z$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet$
$a \llparenthesis R \rrparenthesis b_1 \wedge a \llparenthesis R \rrparenthesis b_2$
$\Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x \llparenthesis F \fatsemi R \cap F \fatsemi S \rrparenthesis z$
$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
$(\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z) \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle ? \rangle$
$\exists y_1 \bullet \exists y_2 \bullet (x \llparenthesis F \rrparenthesis y_1 \wedge x \llparenthesis F \rrparenthesis y_2 \Rightarrow y_2 = y_1)$
$\wedge\, x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Strong modus ponens" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \wedge x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Trading for ∃", "One-point rule for ∃",
substitution, "Idempotency of ∧" $\rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \rrparenthesis z \wedge y \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Relation intersection" $\rangle$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet$
$a \llparenthesis R \rrparenthesis b_1 \wedge a \llparenthesis R \rrparenthesis b_2$
$\Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x \llparenthesis F \fatsemi R \cap F \fatsemi S \rrparenthesis z$
$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$
$(\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z) \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge (\exists y_2 \bullet x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z)$
$\equiv \langle$ "Distributivity of ∧ over ∃" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle$ Assumption \`univalent $F$\`, "Identity of ∧" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet (x \llparenthesis F \rrparenthesis y_1 \wedge x \llparenthesis F \rrparenthesis y_2 \Rightarrow y_2 = y_1)$
$\wedge\, x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Strong modus ponens" $\rangle$
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \wedge x \llparenthesis F \rrparenthesis y_1 \llparenthesis R \rrparenthesis z \wedge x \llparenthesis F \rrparenthesis y_2 \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Trading for ∃", "One-point rule for ∃",
substitution, "Idempotency of ∧" $\rangle$
$\exists y \bullet x \llparenthesis F \rrparenthesis y \llparenthesis R \rrparenthesis z \wedge y \llparenthesis S \rrparenthesis z$
$\equiv \langle$ "Relation intersection" $\rangle$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet$
$a \llparenthesis R \rrparenthesis b_1 \wedge a \llparenthesis R \rrparenthesis b_2$
$\Rightarrow b_1 = b_2$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-22

Part 1: M2

Plan for Today

- Midterm 2
- Relation-Algebraic Reasoning
  - Limitations of *with*$_2$
  - General relation closures as introduced in Ref11.2
  - Inverses
- **Topological Sort:** Introduction (see LADM section 14.4)

M2: “Domain/Range of `id` ”

Theorem “Domain of `id` ”: Dom (id A) = A
Proof:
  Using “Set extensionality ”:
    For any `$x$`:
      x ∈ Dom (id A)
    ≡ ⟨ “Membership in `Dom` ” ⟩
      ∃ y • x ⦇ id A ⦈ y

      x ∈ A

Theorem “Range of `id` ”: Ran (id A) = A
Proof:
  Using “Set extensionality ”:
    For any `$y$`:
      y ∈ Ran (id A)
    ≡ ⟨ “Membership in `Ran` ” ⟩
      ∃ x • x ⦇ id A ⦈ y

      y ∈ A

Provided:

Declaration: Dom : (A ↔ B) → set A
Declaration: Ran : (A ↔ B) → set B
Axiom “Membership in `Dom` ”: x ∈ Dom R ≡ ∃ y • x ⦇ R ⦈ y
Axiom “Membership in `Ran` ”: y ∈ Ran R ≡ ∃ x • x ⦇ R ⦈ y

M2: “Domain/Range of `id` ”

Theorem “Domain of `id` ”: Dom (id A) = A
Proof:
  Using “Set extensionality ”:
    For any `$x$`:
      x ∈ Dom (id A)
    ≡ ⟨ “Membership in `Dom` ” ⟩
      ∃ y • x ⦇ id A ⦈ y
    ≡ ⟨ “Relationship via `id` ” ⟩
      ∃ y • x = y ∈ A
    ≡ ⟨ “Trading for ∃ ” ⟩
      ∃ y ❙ y = x • y ∈ A
    ≡ ⟨ “One-point rule for ∃ ”, substitution ⟩
      x ∈ A

Theorem “Range of `id` ”: Ran (id A) = A
Proof:
  Using “Set extensionality ”:
    For any `$y$`:
      y ∈ Ran (id A)
    ≡ ⟨ “Membership in `Ran` ” ⟩
      ∃ x • x ⦇ id A ⦈ y
    ≡ ⟨ “Relationship via `id` ” ⟩
      ∃ x • x = y ∈ A
    ≡ ⟨ “Trading for ∃ ” ⟩
      ∃ x ❙ x = y • y ∈ A
    ≡ ⟨ “One-point rule for ∃ ”, substitution ⟩
      y ∈ A

Provided:

Declaration: Dom : (A ↔ B) → set A
Declaration: Ran : (A ↔ B) → set B
Axiom “Membership in `Dom` ”: x ∈ Dom R ≡ ∃ y • x ⦇ R ⦈ y
Axiom “Membership in `Ran` ”: y ∈ Ran R ≡ ∃ x • x ⦇ R ⦈ y

M2: Antitonicity / Monotonicity

Theorem “Antitonicity of ⩤ ”:
  A ⊆ B ⇒ B ⩤ R ⊆ A ⩤ R
Proof:

Theorem “Monotonicity of ▷ ”:
  A ⊆ B ⇒ R ▷ A ⊆ R ▷ B
Proof:

Declaration: _ ◁ _ , _ ⩤ _ : set t_1 → (t_1 ↔ t_2) → (t_1 ↔ t_2)
Declaration: _ ▷ _ , _ ⩥ _ : (t_1 ↔ t_2) → set t_2 → (t_1 ↔ t_2)
Axiom “Relationship via ◁ ” “Domain restriction ”:
  x ⦇ A ◁ R ⦈ y ≡ x ∈ A ∧ x ⦇ R ⦈ y
Axiom “Relationship via ▷ ” “Range restriction ”:
  x ⦇ R ▷ B ⦈ y ≡ x ⦇ R ⦈ y ∈ B
Axiom “Relationship via ⩤ ” “Domain antirestriction ”:
  x ⦇ A ⩤ R ⦈ y ≡ ¬ (x ∈ A) ∧ x ⦇ R ⦈ y
Axiom “Relationship via ⩥ ” “Range antirestriction ”:
  x ⦇ R ⩥ B ⦈ y ≡ x ⦇ R ⦈ y ∧ ¬ (y ∈ B)
Declaration: _ ⦇ _ ⦈ : (t_1 ↔ t_2) → set t_1 → set t_2
Axiom “Definition of ⦇ _ ⦈ ”: R ⦇ A ⦈ = Ran (A ◁ R)

M2: Antitonicity / Monotonicity

Theorem “Antitonicity of ⩤ ”:
  A ⊆ B ⇒ B ⩤ R ⊆ A ⩤ R
Proof:
  Assuming `$A$ ⊆ $B$`:
    Using “Relation inclusion ”:
      For any `$x$`, `$y$`:
        x ⦇ B ⩤ R ⦈ y
      ≡ ⟨ “Domain antirestriction ” ⟩
        ¬ (x ∈ B) ∧ x ⦇ R ⦈ y

      ⇒ ⟨ ? ⟩

        ¬ (x ∈ A) ∧ x ⦇ R ⦈ y
      ≡ ⟨ “Domain antirestriction ” ⟩
        x ⦇ A ⩤ R ⦈ y

Theorem “Monotonicity of ▷ ”:
  A ⊆ B ⇒ R ▷ A ⊆ R ▷ B
Proof:
  Assuming `$A$ ⊆ $B$` **and using with** “Set inclusion ”:
    Using “Relation inclusion ”:
      For any `$x$`, `$y$`:
        x ⦇ R ▷ A ⦈ y
      ≡ ⟨ “Range restriction ” ⟩
        y ∈ A ∧ x ⦇ R ⦈ y

      ⇒ ⟨ ? ⟩

        y ∈ B ∧ x ⦇ R ⦈ y
      ≡ ⟨ “Range restriction ” ⟩
        x ⦇ R ▷ B ⦈ y

M2: Antitonicity / Monotonicity

Theorem “Antitonicity of ⩤ ”:
  A ⊆ B ⇒ B ⩤ R ⊆ A ⩤ R
Proof:
  Assuming `$A$ ⊆ $B$`:
    Using “Relation inclusion ”:
      For any `$x$`, `$y$`:
        x ⦇ B ⩤ R ⦈ y
      ≡ ⟨ “Domain antirestriction ” ⟩
        ¬ (x ∈ B) ∧ x ⦇ R ⦈ y
      ⇒ ⟨ “Monotonicity of ∧ ” with “Contrapositive ” with
          “Casting ” with assumption `$A$ ⊆ $B$` ⟩
        ¬ (x ∈ A) ∧ x ⦇ R ⦈ y
      ≡ ⟨ “Domain antirestriction ” ⟩
        x ⦇ A ⩤ R ⦈ y

Theorem “Monotonicity of ▷ ”:
  A ⊆ B ⇒ R ▷ A ⊆ R ▷ B
Proof:
  Assuming `$A$ ⊆ $B$` **and using with** “Set inclusion ”:
    Using “Relation inclusion ”:
      For any `$x$`, `$y$`:
        x ⦇ R ▷ A ⦈ y
      ≡ ⟨ “Range restriction ” ⟩
        y ∈ A ∧ x ⦇ R ⦈ y
      ⇒ ⟨ “Monotonicity of ∧ ” with assumption `$A$ ⊆ $B$` ⟩
        y ∈ B ∧ x ⦇ R ⦈ y
      ≡ ⟨ “Range restriction ” ⟩
        x ⦇ R ▷ B ⦈ y

M2 Notes

- The first proof “Domain/Range of `id` ” was intended as **free points for all**
- The second proof “Antitonicity / Monotonicity” was intended as **free points for all who paid some attention**
- The third proof works like the one shown at the end of last Thursday’s lecture (Nov. 18)
- “Closed book” means that looking things up is wasting your time.
- Copying somewhat-related proofs from all kinds of sources generally did not work out very well. (Last year’s “𝕀” is different from this year’s “𝕀”. . .)
- You have to be pretty strong to be able to adapt a somewhat-related proof that you didn’t write. . .
- ***The way to succeed:***
  — Read the current notebook — only! — in detail!
  — Have the skills to construct your proofs yourself!
  — Do construct your proofs yourself when you need them!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-22

Part 2: Abstract Relational-Algebraic Reasoning

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 $A_1\sigma$, $A_2\sigma$, . . . using *ThmB* and *ThmB*$_2$ . . .
  - Rewrite goal applying $L\sigma \mapsto R\sigma$ rigidly.

ThmA `with` *ThmB* and *ThmB*$_2$ . . .

- E.g.: “Transitivity of ⊆” with Assumptions `$Q \cap S \subseteq Q$` and `$Q \subseteq R$` when trying to prove `$Q \cap S \subseteq R$`
  - “Transitivity of ⊆” 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: $q \subseteq s \mapsto true$, 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 ⊆” with `$R := Q$` and assumption `$Q \cap S \subseteq Q$` and assumption `$Q \subseteq R$`

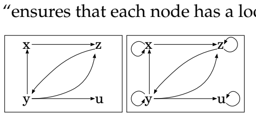
Recall: Reflexive Closure

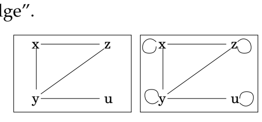
Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$
iff Q is the smallest reflexive relation containing R,

or, equivalently, iff
- $R \subseteq Q$
- $\mathbb{I} \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

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”.





Reflexive Closure Operator `reflClos` (in Ref11.2)

Axiom “Definition of `reflClos` ”: reflClos R = R ∪ 𝕀

Theorem “Closure properties of `reflClos`: Expanding ”:
  R ⊆ reflClos R
Proof:

  ?

Theorem “Closure properties of `reflClos`: Reflexivity ”:
  reflexive (reflClos R)
Proof:

  ?

Theorem “Closure properties of `reflClos`: Minimality ”:
  R ⊆ S ∧ reflexive S ⇒ reflClos R ⊆ S
Proof:

  ?

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

- $R \subseteq Q$
- $pred\ Q$
- $(\forall P : B \leftrightarrow C \mid R \subseteq P \land pred\ P \bullet Q \subseteq P)$

(For some properties, closures are not defined, or not always defined.)

Closures

Let *pred* (for "predicate") be a property on relations, i.e.: $\quad pred \;:\; (B \leftrightarrow C) \to \mathbb{B}$

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

- $R \subseteq Q \quad$ and $\quad pred\ Q \quad$ and $\quad (\forall P : B \leftrightarrow C \mid R \subseteq P \land pred\ P \bullet Q \subseteq P)$

General Relation Closures in Ref11.2:

Precedence 50 **for:** $\_is\_closure-of\_$
Conjunctional: $\quad \_is\_closure-of\_$
Declaration: $\quad \_is\_closure-of\_$:
$\quad (A \leftrightarrow B) \to ((A \leftrightarrow B) \to \mathbb{B}) \to (A \leftrightarrow B) \to \mathbb{B}$

Axiom "Relation closure":
$\quad Q$ **is** *pred* closure-of R
$\equiv \; R \subseteq Q \;\land\; pred\ Q \;\land\; (\forall P \bullet R \subseteq P \land pred\ P \Rightarrow Q \subseteq P)$

Theorem "Well-definedness of `reflClos`":

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`":

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 `$\forall P \bullet R \subseteq P \land$ reflexive $P \Rightarrow$ reflClos $R \subseteq P$`:
For any `$P$`:
Assuming `$R \subseteq P$`, `reflexive $P$`:
?

Recall: Properties of Heterogeneous Relations

A relation $R : B \leftrightarrow C$ is called:

univalent determinate	$R^\smile \fatsemi R \;\subseteq\; \mathbb{I}$	$\forall b, c_1, c_2 \bullet b \,(R)\, c_1 \land b \,(R)\, c_2 \Rightarrow c_1 = c_2$
total	$Dom\ R \;=\; \llcorner B \lrcorner$ $\mathbb{I} \;\subseteq\; R \fatsemi R^\smile$	$\forall b : B \bullet (\exists c : C \bullet b \,(R)\, c)$
injective	$R \fatsemi R^\smile \;\subseteq\; \mathbb{I}$	$\forall b_1, b_2, c \bullet b_1 \,(R)\, c \land b_2 \,(R)\, c \Rightarrow b_1 = b_2$
surjective	$Ran\ R \;=\; \llcorner C \lrcorner$ $\mathbb{I} \;\subseteq\; R^\smile \fatsemi R$	$\forall c : C \bullet (\exists b : B \bullet b \,(R)\, c)$
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 "between Sets"

Let $R : B \leftrightarrow C$ be a relation and $X :$ **set** B and $Y :$ **set** C be sets. Then R is called:

univalent determinate	$R^\smile \fatsemi R \;\subseteq\; \mathbb{I}$	$\forall b, c_1, c_2 \bullet b \,(R)\, c_1 \land b \,(R)\, c_2 \Rightarrow c_1 = c_2$
total on X	$Dom\ R \;\supseteq\; X$ $\mathrm{id}\ X \;\subseteq\; R \fatsemi R^\smile$	$\forall b : B \mid b \in X \bullet (\exists c : C \bullet b \,(R)\, c)$
injective	$R \fatsemi R^\smile \;\subseteq\; \mathbb{I}$	$\forall b_1, b_2, c \bullet b_1 \,(R)\, c \land b_2 \,(R)\, c \Rightarrow b_1 = b_2$
surjective onto Y	$Ran\ R \;\supseteq\; Y$ $\mathrm{id}\ Y \;\subseteq\; R^\smile \fatsemi R$	$\forall c : C \mid c \in Y \bullet (\exists b : B \bullet b \,(R)\, c)$
a **mapping from** X **to** Y	$R^\smile \fatsemi R \subseteq \mathrm{id}\ Y \quad \land \quad Dom\ R = X$	

We define $\quad X \twoheadrightarrow Y \quad$ to be the set of all mappings from X to Y.

We therefore write "$f \in X \twoheadrightarrow Y$" for "$f$ is a mapping from X to Y".

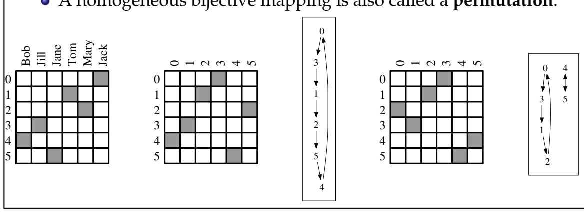
(We continue to write $T_1 \to T_2$ for the function type of functions ("operators") from type T_1 to type T_2.
Such functions do not have any relation type.)

Inverses of Total Functions — Between Sets

We write "$f \in S_1 \twoheadrightarrow S_2$" for "$f$ is a mapping fron S_1 to S_2".

(14.43) **Definition:** Let f with $f \in S_1 \twoheadrightarrow S_2$ 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 $f \fatsemi g = \mathrm{id}\ S_1$ and $g \fatsemi f = \mathrm{id}\ S_2$.

- 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 $f : B \leftrightarrow C$ be a **mapping** between types B and C.
An **inverse of** f is a mapping $g : C \leftrightarrow B$ such that $f \fatsemi g = \mathbb{I} = \mathrm{id}\ \llcorner B \lrcorner$ and $g \fatsemi f = \mathbb{I} = \mathrm{id}\ \llcorner C \lrcorner$.

Theorem: If g is an inverse of a mapping $f : B \to C$, then $g = f^\smile$.
Proof: (Using antisymmetry of $\subseteq$)

$$
\begin{array}{ll}
 & f^\smile \\
= & \langle \text{Identity of } \fatsemi \rangle \\
 & f^\smile \fatsemi \mathbb{I} \\
= & \langle g \text{ is an inverse of } f \rangle \\
 & f^\smile \fatsemi f \fatsemi g \\
\subseteq & \langle \textbf{Mon. of } \fatsemi \textbf{ with } f \text{ is univalent, that is, } f^\smile \fatsemi f \subseteq \mathbb{I} \rangle \\
 & \mathbb{I} \fatsemi g \\
= & \langle \text{Identity of } \fatsemi \rangle \\
 & g \\
\subseteq & \langle \text{Identity of } \fatsemi, \;\; \textbf{Mon. of } \fatsemi \textbf{ with } f \text{ is total, that is, } \mathbb{I} \subseteq f \fatsemi f^\smile \rangle \\
 & g \fatsemi f \fatsemi f^\smile \\
= & \langle g \text{ is an inverse of } f\text{; Identity of } \fatsemi \rangle \\
 & f^\smile
\end{array}
$$

More complicated in the set-based view!

Logical Reasoning for Computer Science

COMPSCI 2LC3

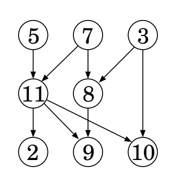
McMaster University, Fall 2021

Wolfram Kahl

2021-11-22

Part 3: Topological Sort: Intro

Topological Sort — Introduction



A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $\quad E \cap E^\smile \subseteq \mathbb{I} \subseteq E \supseteq E \fatsemi E \quad$ and $E \cup E^\smile = V \times V \quad$ and $\quad B \subseteq E$.

Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* \fatsemi B^*$

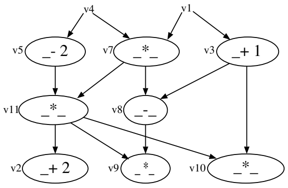
E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.

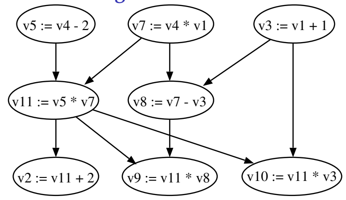
Example: The DAG above has, among others, the following topological sorts:

- $[5, 7, 3, 11, 8, 2, 9, 10]$ — *visual left-to-right, top-to-bottom*
- $[3, 5, 7, 8, 11, 2, 9, 10]$ — *smallest-numbered available vertex first*
- $[5, 7, 3, 8, 11, 10, 9, 2]$ — *fewest edges first*
- $[7, 5, 11, 3, 10, 8, 9, 2]$ — *largest-numbered available vertex first*
- $[5, 7, 11, 2, 3, 8, 9, 10]$ — *attempting top-to-bottom, left-to-right*
- $[3, 7, 8, 5, 11, 10, 2, 9]$ — *(arbitrary)*

$B = \{\langle 3, 8\rangle, \langle 3, 10\rangle, \langle 5, 11\rangle, \langle 7, 8\rangle, \langle 7, 11\rangle, \langle 8, 11\rangle, \langle 11, 2\rangle, \langle 11, 9\rangle, \langle 11, 10\rangle\}$

Topological Sort — Code Scheduling — SSA





Static single assignment form: Each variable is assigned **once**, and assigned before use.

```
v5  := v4 - 2
v7  := v4 * v1
v3  := v1 + 1
v11  := v5 * v7
v8  := v7 - v3
v2  := v11 + 2
v9  := v11 * v8
v10  := v11 * v3
```

We can consider SSA as **encoding data-flow graphs**.

Each admissible re-ordering of an SSA sequence is a different topological sort of that graph.

It is frequently easier to think in terms of that graph than in terms of re-orderings!

Logical Reasoning for Computer Science

COMPSCI 2LC3

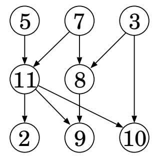
McMaster University, Fall 2021

Wolfram Kahl

2021-11-23

Topological Sort

Topological Sort — Introduction



A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E \fatsemi E$ and $E \cup E^{\smile} = V \times V$ and $B \subseteq E$.

Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* \fatsemi B^*$

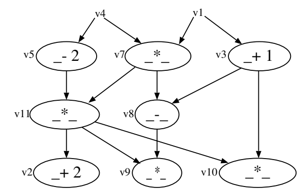
E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.

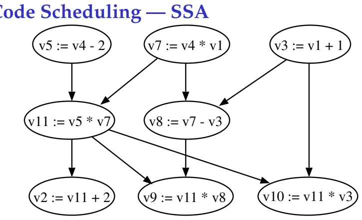
Example: The DAG above has, among others, the following topological sorts:

- [5, 7, 3, 11, 8, 2, 9, 10] — *visual left-to-right, top-to-bottom*
- [3, 5, 7, 8, 11, 2, 9, 10] — *smallest-numbered available vertex first*
- [5, 7, 3, 8, 11, 10, 9, 2] — *fewest edges first*
- [7, 5, 11, 3, 10, 8, 9, 2] — *largest-numbered available vertex first*
- [5, 7, 11, 2, 3, 8, 9, 10] — *attempting top-to-bottom, left-to-right*
- [3, 7, 8, 5, 11, 10, 2, 9] — *(arbitrary)*

$B = \{\langle 3,8\rangle, \langle 3,10\rangle, \langle 5,11\rangle, \langle 7,8\rangle, \langle 7,11\rangle, \langle 8,11\rangle, \langle 11,2\rangle, \langle 11,9\rangle, \langle 11,10\rangle\}$

Topological Sort — Code Scheduling — SSA





Static single assignment form: Each variable is assigned **once**, and assigned before use.

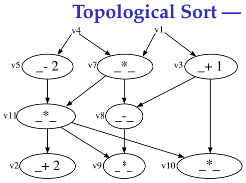
```
v5  := v4 - 2
v7  := v4 * v1
v3  := v1 + 1
v11  := v5 * v7
v8  := v7 - v3
v2  := v11 + 2
v9  := v11 * v8
v10  := v11 * v3
```

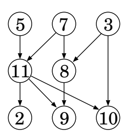
We can consider SSA as **encoding data-flow graphs**.

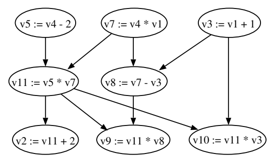
Each admissible re-ordering of an SSA sequence is a different topological sort of that graph.

It is frequently easier to think in terms of that graph than in terms of re-orderings!

Topological Sort — Code Scheduling — SSA — Pipeline Stalls







Static single assignment form: Each variable is assigned **once**, and assigned before use.

[7, 5, 11, 3, 10, 8, 9, 2]

```
v7  := v4 * v1
v5  := v4 - 2
v11  := v5 * v7
v3  := v1 + 1
v10  := v11 * v3
v8  := v7 - v3
v9  := v11 * v8
v2  := v11 + 2
```

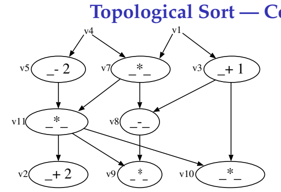
Let E be the topological sort of (V, B);
let $C = E - \mathbb{I}$ be the associated strict-order.

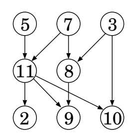
Depth-2 pipelining requires $B \subseteq C \fatsemi C$.
Depth-3 pipelining requires $B \subseteq C \fatsemi C \fatsemi C$.

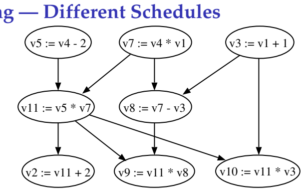
The "next-step" relation: $S = C - C \fatsemi C^+$

Depth-2 pipelining requires $B \cap S = \{\}$.
Depth-3 pipelining requires $B \cap (S \cup S \fatsemi S) = \{\}$.

Topological Sort — Code Scheduling — Different Schedules





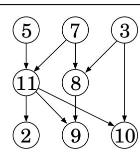


Example: Most of the original example topological sorts induce pipeline stalls:

- [5, 7, 3, 11, 8, 2, 9, 10] — *visual left-to-right, top-to-bottom*
- [3, 5, 7, 8, 11, 2, 9, 10] — *smallest-numbered available vertex first*
- [5, 7, 3, 8, 11, 10, 9, 2] — *fewest edges first*
- [7, 5, 11, 3, 10, 8, 9, 2] — *largest-numbered available vertex first*
- [5, 7, 11, 2, 3, 8, 9, 10] — *attempting top-to-bottom, left-to-right*
- [3, 7, 8, 5, 11, 10, 2, 9] — *(arbitrary)*

$B = \{\langle 3,8\rangle, \langle 3,10\rangle, \langle 5,11\rangle, \langle 7,8\rangle, \langle 7,11\rangle, \langle 8,11\rangle, \langle 11,2\rangle, \langle 11,9\rangle, \langle 11,10\rangle\}$

Topological Sort — Simple Algorithm



Given a DAG (V, B) (with V : **set** T),
calculate sequence s encoding a topological sort E.

var vs : **set** T
var s : *Seq* T
vs **:=** V ⨾ — **not-yet-used vertices**
$\{ vs = V \}$ — **Precondition**
s **:=** ϵ ⨾ — **accumulator for result sequence**
$\{ (vs \text{ and } \{v \mid v \in s\} \text{ partition } V) \wedge$
$\quad (\forall v \mid v \in s \bullet \forall u \mid u \,(B)\, v \bullet u \text{ precedes } v \text{ in } s) \}$ — **Invariant**
while $vs \neq \{\}$ **do**
$\quad$ Choose a source u of the subgraph $(vs, B \cap (vs \times vs))$ induced by vs ⨾
$\quad vs, s$ **:=** $vs - \{u\}, s \triangleright u$
od
$\{ (\forall u, v : V \mid u \,(B)\, v \bullet u \text{ precedes } v \text{ in } s) \}$ — **Postcondition**

How to "Choose a source u of the subgraph induced by vs" **efficiently?**

Data Refinement

Initialisation — Operations — Finalisation

Abstract states:

$$\begin{tikzcd}
 & X \arrow[r, "f_1"] \arrow[dd, dotted, "R"] & X \arrow[r, "f_2"] \arrow[dd, dotted, "R"] & X \arrow[r, "f_3"] \arrow[dd, dotted, "R"] & X \arrow[dr, "p"] \arrow[dd, dotted, "R"] & \\
I \arrow[ur, "i"] \arrow[dr, "j"'] & & & & & T \\
 & Y \arrow[r, "g_1"'] & Y \arrow[r, "g_2"'] & Y \arrow[r, "g_3"'] & Y \arrow[ur, "q"'] &
\end{tikzcd}$$

Implementation states:

Representation relation: $R : X \leftrightarrow Y$
relates abstract states X with concrete implementation states Y:

- Compatible initialisation: $j \subseteq i \fatsemi R$
- Operation simulation: $R \fatsemi g_k \subseteq f_k \fatsemi R$
- Compatible results: $R \fatsemi q \subseteq p$

Topological Sort — Making Choosing Minimal Elements Easier

To store mappings $V \twoheadrightarrow X$ in "**array ... of** X", "assume" $V = 0..k = \{i : \mathbb{N} \mid 0 \leq i \leq k\}$.

var *sources* : *Seq* $(0..k)$ — three new variables make vs superfluous
var *preCount* : **array** $0..k$ **of** $\llcorner \mathbb{N} \lrcorner$
var *postSet* : **array** $0..k$ **of** $\mathbb{P}\,(0..k)$ — read-only version of $B : V \leftrightarrow V$ as $V \twoheadrightarrow \mathbb{P}V$

Coupling invariant:

$\quad \{u \mid u \in sources\} = vs - (\mathrm{Ran}\, B') \wedge$ — *sources* contains sources of $B' = B \cap (vs \times vs)$
$\quad (\forall v \mid v \in vs \bullet preCount[v] = \#(B'^{\smile} (\!|\{v\}|\!))) \wedge$
$\quad (\forall u \mid u \in vs \bullet postSet[u] = B' (\!|\{u\}|\!))$

Initialisation:

for $v \in 0..k$ **do** $preCount[v]$ **:=** $\#(B^{\smile} (\!|\{v\}|\!))$ **od** ⨾
for $u \in 0..k$ **do** $postSet[u]$ **:=** $B (\!|\{u\}|\!)$ **od** ⨾
sources **:=** ϵ ⨾
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** *sources* **:=** *sources* $\triangleright v$ **fi od**

Topological Sort — Complete "Translated" LADM Algorithm

for $v \in 0..k$ **do** $preCount[v]$ **:=** $\#(B^{\smile} (\!|\{v\}|\!))$ **od** ⨾
for $u \in 0..k$ **do** $postSet[u]$ **:=** $B (\!|\{u\}|\!)$ **od** ⨾
sources **:=** ϵ ⨾
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** *sources* **:=** *sources* $\triangleright v$ **fi od**
ghost $vs := 0..k$ ⨾
s **:=** ϵ
while *sources* $\neq \epsilon$ **do**
$\quad u$ **:=** *head sources* ⨾
$\quad s$ **:=** $s \triangleright u$ ⨾
$\quad$ *sources* **:=** *tail sources* ⨾ — remove u from *sources*
$\quad$ **ghost** $vs := vs - \{u\}$ ⨾
$\quad$ **for** $v \in postSet[u]$ **do**
$\qquad preCount[v]$ **:=** $preCount[v] - 1$ ⨾
$\qquad$ **if** $preCount[v] = 0$ **then** *sources* **:=** *sources* $\triangleright v$ **fi**
$\quad$ **od**
od

Logical Reasoning for Computer Science

COMPSCI 2LC3

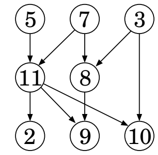
McMaster University, Fall 2021

Wolfram Kahl

2021-11-25

Topological Sort

Topological Sort — Specification



A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E ; E$ and $E \cup E^{\smile} = V \times V$ and $B \subseteq E$.

Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* ; B^*$

E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.

Interface types: **var** vs : **set** T ⁃⁃⁃ input V
var s : *Seq* T ⁃⁃⁃ output representing E

Next week: Procedure declaration, e.g.: *Seq* T topSort(**set** T vs)

Precondition: $vs = V$

Postcondition: $(\forall u, v \mid u \,(B)\, v \bullet u$ precedes v in $s)$

One Formalisation of _precedes_in_

Precedence 50 **for:** _precedes_in_
Conjunctional: _precedes_in_
Declaration: _precedes_in_ : $A \to A \to$ Seq $A \to \mathbb{B}$

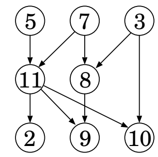
Axiom "Def. `_precedes_in_`": x precedes y in ϵ ≡ false
Axiom "Def. `_precedes_in_`": x precedes y in $(x \triangleleft zs)$ ≡ $y \in zs$
Axiom "Def. `_precedes_in_`": $x \neq z \Rightarrow (x$ precedes y in $(z \triangleleft zs) \equiv x$ precedes y in $zs)$

1 *precedes* 3 *in* $[1, 2]$ ≡ ?

1 *precedes* 3 *in* $[3]$ ≡ ?

1 *precedes* 3 *in* $[3, 1, 3]$ ≡ ?

Topological Sort — Specification (ctd.)



A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E ; E$ and $E \cup E^{\smile} = V \times V$ and $B \subseteq E$.

Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* ; B^*$

E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.

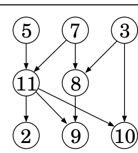
Interface types: **var** vs : **set** T ⁃⁃⁃ input V
var s : *Seq* T ⁃⁃⁃ output representing E

Next week: Procedure declaration, e.g.: *Seq* T topSort(**set** T vs)

Precondition: $vs = V$

Postcondition: $(\forall u, v \mid u \,(B)\, v \bullet u$ precedes v in $s)$
$\wedge \{v \mid v \in s\} = V$
$\wedge$ *length* $s = \# V$

Topological Sort — Simple Algorithm



Given a DAG (V, B) (with V : **set** T),
calculate sequence s encoding a topological sort E.

var vs : **set** T; s : *Seq* T
$vs := V$; — **not-yet-used vertices**
$\{ vs = V \}$ — **Precondition**
$s := \epsilon$; — **Initialising accumulator for result sequence**
$\{ (vs$ and $\{v \mid v \in s\}$ partition $V) \wedge$ *length* $s + \# vs = \# V \wedge$
$(\forall u, v \mid v \in s \wedge u \,(B)\, v \bullet u$ precedes v in $s) \}$ — **Invariant**
while $vs \neq \{\}$ **do**
Choose a source u of the subgraph $(vs, B \cap (vs \times vs))$ induced by vs ;
$vs, s := vs - \{u\}, s \triangleright u$
od
$\{ (\forall u, v \mid u \,(B)\, v \bullet u$ precedes v in $s)$
$\wedge \{v \mid v \in s\} = V \wedge$ *length* $s = \# V \}$ — **Postcondition**

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.

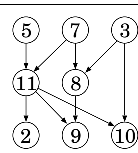
Frequent pattern: Generalised postcondition using the negated loop condition

Using the "While" Rule

Theorem "While-example":
Pre
$\Rightarrow$[INIT ;
while B
do
C
od ;
FINAL
]
Post

Proof:
Pre ⁃⁃⁃ Precondition
$\Rightarrow$[INIT] ⟨ ? ⟩
Q ⁃⁃⁃ Invariant
$\Rightarrow$[while B do
C
od] ⟨ "While" with subproof:
$B \wedge Q$ ⁃⁃⁃ Loop condition and invariant
$\Rightarrow$[C] ⟨ ? ⟩
Q ⁃⁃⁃ Invariant
⟩
$\neg B \wedge Q$ ⁃⁃⁃ Negated loop condition, and invariant
$\Rightarrow$[FINAL] ⟨ ? ⟩
Post ⁃⁃⁃ Postcondition

Topological Sort — Simple Algorithm

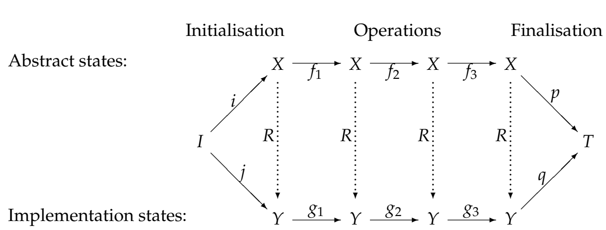


Given a DAG (V, B) (with V : **set** T),
calculate sequence s encoding a topological sort E.

var vs : **set** T; s : *Seq* T
$vs := V$; — **not-yet-used vertices**
$\{ vs = V \}$ — **Precondition**
$s := \epsilon$; — **Initialising accumulator for result sequence**
$\{ (vs$ and $\{v \mid v \in s\}$ partition $V) \wedge$ *length* $s + \# vs = \# V \wedge$
$(\forall u, v \mid v \in s \wedge u \,(B)\, v \bullet u$ precedes v in $s) \}$ — **Invariant**
while $vs \neq \{\}$ **do**
Choose a source u of the subgraph $(vs, B \cap (vs \times vs))$ induced by vs ;
$vs, s := vs - \{u\}, s \triangleright u$
od
$\{ (\forall u, v \mid u \,(B)\, v \bullet u$ precedes v in $s)$
$\wedge \{v \mid v \in s\} = V \wedge$ *length* $s = \# V \}$ — **Postcondition**

How to "Choose a source u of the subgraph induced by vs" **efficiently?**

Data Refinement



Representation relation: $R : X \leftrightarrow Y$
relates abstract states X with concrete implementation states Y:

- Compatible initialisation: $j \subseteq i ; R$
- Operation simulation: $R ; g_k \subseteq f_k ; R$
- Compatible results: $R ; q \subseteq p$

Topological Sort — Making Choosing Minimal Elements Easier

To store mappings $V \mathrel{-\!\!\circ\!\!\!\rightarrow} X$ in "**array** ... **of** X", "assume" $V = 0..k = \{i : \mathbb{N} \mid 0 \le i \le k\}$.

var *sources* : *Seq* $(0..k)$ — three new variables make vs superfluous
var *preCount* : **array** $0..k$ **of** $\llcorner \mathbb{N} \lrcorner$
var *postSet* : **array** $0..k$ **of** $\mathbb{P}(0..k)$ — read-only version of $B : V \leftrightarrow V$ as $V \mathrel{-\!\!\circ\!\!\!\rightarrow} \mathbb{P}V$

Coupling invariant:

$\{u \mid u \in sources\} = vs - (\mathrm{Ran}\, B') \wedge$ — *sources* contains sources of $B' = B \cap (vs \times vs)$
$(\forall v \mid v \in vs \bullet preCount[v] = \#(B'^{\smile} (\![\{v\}]\!))) \wedge$
$(\forall u \mid u \in vs \bullet postSet[u] = B' (\![\{u\}]\!))$

Initialisation:

for $v \in 0..k$ **do** $preCount[v] := \#(B^{\smile} (\![\{v\}]\!))$ **od** ;
for $u \in 0..k$ **do** $postSet[u] := B (\![\{u\}]\!)$ **od** ;
$sources := \epsilon$;
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**

Topological Sort — Complete "Translated" LADM Algorithm

for $v \in 0..k$ **do** $preCount[v] := \#(B^{\smile} (\![\{v\}]\!))$ **od** ;
for $u \in 0..k$ **do** $postSet[u] := B (\![\{u\}]\!)$ **od** ;
$sources := \epsilon$;
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
ghost $vs := 0..k$;
$s := \epsilon$
while $sources \neq \epsilon$ **do**
 $u := head\ sources$;
 $s := s \triangleright u$;
 $sources := tail\ sources$; — remove u from *sources*
 ghost $vs := vs - \{u\}$;
 for $v \in postSet[u]$ **do**
  $preCount[v] := preCount[v] - 1$;
  if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
 od
od

Topological Sort — Complete $O(\# B + \# V)$ Algorithm

for $p \in B$ **do**
 $preCount[snd\ p] := preCount[snd\ p] + 1$
 $postSet[fst\ p] := postSet[fst\ p] \cup \{v\}$
od ;
$sources := \epsilon$; **for** $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
ghost $vs := 0..k$;
$s := \epsilon$
while $sources \neq \epsilon$ **do**
 $u := head\ sources$;
 $s := s \triangleright u$;
 $sources := tail\ sources$; — remove u from *sources*
 ghost $vs := vs - \{u\}$;
 for $v \in postSet[u]$ **do**
  $preCount[v] := preCount[v] - 1$;
  if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
 od
od

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

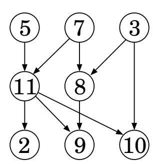
Wolfram Kahl

2021-11-29

Part 1: Topological Sort

Recall: Topological Sort — Specification

A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E ; E$ and $E \cup E^{\smile} = V \times V$ and $B \subseteq E$.



Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* ; B^*$

E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.

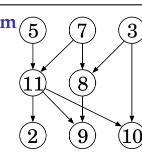
Interface types: **var** vs : **set** T ▪▪▪▪▪▪ input V
var s : *Seq* T ▪▪▪▪▪▪ output representing E

Next week: Procedure declaration, e.g.: *Seq* T topSort(**set** T vs)

Precondition: $vs = V$

Postcondition: $(\forall u, v \mid u \,(\!\!\!\mid B \mid\!\!\!)\, v \bullet u$ precedes v in $s)$
$\wedge \{v \mid v \in s\} = V$
$\wedge\ length\ s = \# V$

Recall: Topological Sort — Simple Algorithm



Given a DAG (V, B) (with V : **set** T),
calculate sequence s encoding a topological sort E.

var vs : **set** T; s : *Seq* T
$vs := V$; — **not-yet-used vertices**
$\{ vs = V \}$ — **Precondition**
$s := \epsilon$; — **Initialising accumulator for result sequence**
$\{ (vs$ and $\{v \mid v \in s\}$ partition $V) \wedge length\ s + \# vs = \# V \wedge$
$(\forall u, v \mid v \in s \wedge u \,(\!\!\!\mid B \mid\!\!\!)\, v \bullet u$ precedes v in $s) \}$ — **Invariant**
while $vs \neq \{\}$ **do**
 Choose a source u of the subgraph $(vs, B \cap (vs \times vs))$ induced by vs ;
 $vs, s := vs - \{u\}, s \triangleright u$
od
$\{ (\forall u, v \mid u \,(\!\!\!\mid B \mid\!\!\!)\, v \bullet u$ precedes v in $s)$
$\wedge \{v \mid v \in s\} = V \wedge length\ s = \# V \}$ — **Postcondition**

How to "Choose a source u of the subgraph induced by vs" **efficiently?**

Topological Sort — Making Choosing Minimal Elements Easier

To store mappings $V \mathrel{-\!\!\circ\!\!\!\rightarrow} X$ in "**array** ... **of** X", "assume" $V = 0..k = \{i : \mathbb{N} \mid 0 \le i \le k\}$.

var *sources* : *Seq* $(0..k)$ — three new variables make vs superfluous
var *preCount* : **array** $0..k$ **of** $\llcorner \mathbb{N} \lrcorner$
var *postSet* : **array** $0..k$ **of** $\mathbb{P}(0..k)$ — read-only version of $B : V \leftrightarrow V$ as $V \mathrel{-\!\!\circ\!\!\!\rightarrow} \mathbb{P}V$

Coupling invariant:

$\{u \mid u \in sources\} = vs - (\mathrm{Ran}\, B') \wedge$ — *sources* contains sources of $B' = B \cap (vs \times vs)$
$(\forall v \mid v \in vs \bullet preCount[v] = \#(B'^{\smile} (\![\{v\}]\!))) \wedge$
$(\forall u \mid u \in vs \bullet postSet[u] = B' (\![\{u\}]\!))$

Initialisation:

for $v \in 0..k$ **do** $preCount[v] := \#(B^{\smile} (\![\{v\}]\!))$ **od** ;
for $u \in 0..k$ **do** $postSet[u] := B (\![\{u\}]\!)$ **od** ;
$sources := \epsilon$;
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**

Topological Sort — Complete "Translated" LADM Algorithm

for $v \in 0..k$ **do** $preCount[v] := \#(B^{\smile} (\![\{v\}]\!))$ **od** ;
for $u \in 0..k$ **do** $postSet[u] := B (\![\{u\}]\!)$ **od** ;
$sources := \epsilon$;
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
ghost $vs := 0..k$;
$s := \epsilon$
while $sources \neq \epsilon$ **do**
 $u := head\ sources$;
 $s := s \triangleright u$;
 $sources := tail\ sources$; — remove u from *sources*
 ghost $vs := vs - \{u\}$;
 for $v \in postSet[u]$ **do**
  $preCount[v] := preCount[v] - 1$;
  if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
 od
od

Topological Sort — Complete $O(\# B + \# V)$ Algorithm

for $p \in B$ **do**
 $preCount[snd\ p] := preCount[snd\ p] + 1$
 $postSet[fst\ p] := postSet[fst\ p] \cup \{snd\ p\}$
od ;
$sources := \epsilon$; **for** $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
ghost $vs := 0..k$;
$s := \epsilon$
while $sources \neq \epsilon$ **do**
 $u := head\ sources$;
 $s := s \triangleright u$;
 $sources := tail\ sources$; — remove u from *sources*
 ghost $vs := vs - \{u\}$;
 for $v \in postSet[u]$ **do**
  $preCount[v] := preCount[v] - 1$;
  if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
 od
od

Modelling Arrays as Partial Functions

Precedence 97 **for:** $\_ \mathrel{-\!\!\circ\!\!\!\rightarrow} \_$

Associating to the right: $\_ \mathrel{-\!\!\circ\!\!\!\rightarrow} \_$

Declaration: $\_ \mathrel{-\!\!\circ\!\!\!\rightarrow} \_ : \mathsf{set}\ A \to \mathsf{set}\ B \to \mathsf{set}\ (A \leftrightarrow B)$

Axiom "Definition of $\mathrel{-\!\!\circ\!\!\!\rightarrow}$":

$X \mathrel{-\!\!\circ\!\!\!\rightarrow} Y = \{ f \mid f^{\smile} ; f \subseteq \mathsf{id}\ Y \wedge \mathsf{Dom}\ f = X \}$

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

```
z := xs[i] ;
xs[i] := xs[j] ;
xs[j] := z
```

Theorem "Array swap":

$$i \le k \ge j \ \wedge\ \mathsf{xs} = xs_0 \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner$$
$$\Rightarrow [\ z := \mathsf{xs} @ i\ ;$$
$$\quad \mathsf{xs} := \mathsf{xs} \oplus \{ \langle i, \mathsf{xs} @ j \rangle \}\ ;$$
$$\quad \mathsf{xs} := \mathsf{xs} \oplus \{ \langle j, z \rangle \}$$
$$]$$
$$\mathsf{xs} = xs_0 \oplus \{ \langle i, xs_0 @ j \rangle, \langle j, xs_0 @ i \rangle \}$$

Sortedness

Declaration: sorted : $(\mathbb{N} \leftrightarrow \mathbb{N}) \to \mathbb{B}$

Axiom "Definition of `sorted`":

$$\mathsf{sorted}\ R \ \equiv\ R\,\breve{}\ ;\ \ulcorner\_<\_\urcorner\ ;\ R \ \subseteq\ \ulcorner\_\le\_\urcorner$$

Theorem "Sortedness":

$$\mathsf{sorted}\ R \equiv \forall\, i \bullet \forall\, j \mid i < j \bullet \forall\, m \bullet \forall\, n \mid i\ (R)\ m \wedge j\ (R)\ n \bullet m \le n$$

Proof:

Theorem "Sorting 0":

$$\mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner$$
$$\Rightarrow [\ p := 0\ ;$$
$$\quad \mathsf{while}\ p \ne k\ \mathsf{do}$$
$$\qquad \mathsf{xs} := \mathsf{xs} \oplus \{ \langle p, 42 \rangle \}\ ;$$
$$\qquad p := p + 1$$
$$\quad \mathsf{od}$$
$$]$$
$$\mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted\ xs}$$

```
p := 0 ;
while p ≠ k do
   xs[p] := 42 ;
   p := p + 1
```

Proof:

$$\mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner$$
$$\Rightarrow \langle\ ?\ \rangle$$
$$\mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted}\,((0\,..\,0) \lhd \mathsf{xs})$$
$$\Rightarrow [\ p := 0\ ]\langle \text{"Assignment" with substitution} \rangle$$
$$\mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted}\,((0\,..\,p) \lhd \mathsf{xs})$$
$$\Rightarrow [\ \mathsf{while}\ p \ne k\ \mathsf{do}\ \ \mathsf{xs} := \mathsf{xs} \oplus \{ \langle p, 42 \rangle \}\ ;\ p := p + 1\ \mathsf{od}$$
$$]\langle \text{"While" with subproof:}$$
$$\qquad ?$$
$$\rangle$$
$$\neg(p \ne k) \wedge \mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted}\,((0\,..\,p) \lhd \mathsf{xs})$$
$$\Rightarrow \langle\ ?\ \rangle$$
$$\mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted\ xs}$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-29

Part 2: 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 $\Lbag$ and $\Rbag$.

Sets versus bags example:

$$\{ x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x \} = \{4,1,0\} = \{0,1,4\} = \{0,0,0,1,1,4\}$$
$$\Lbag x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x \Rbag = \Lbag 4,1,0,1,4 \Rbag = \Lbag 0,1,1,4,4 \Rbag \ne \Lbag 0,1,4 \Rbag$$

The operator $\_\#\_ : t \to Bag\ t \to \mathbb{N}$ counts the number of occurrences of an element in a bag:

$$1 \# \Lbag 0,0,0,1,1,4 \Rbag = 2$$

Bag extensionality and **bag inclusion** are defined via all occurrence counts:

$$B = C \ \equiv\ (\forall\, x \bullet x \# B = x \# C) \qquad B \subseteq C \ \equiv\ (\forall\, x \bullet x \# B \le x \# C)$$

Bag operations:

$$x \# (B \cup C) = (x \# B) + (x \# C)$$
$$x \# (B \cap C) = (x \# B) \downarrow (x \# C)$$
$$x \# (B - C) = (x \# B) \mathbin{\dot-} (x \# C)$$

Bag Product and Bag Reconstitution

Recall: A **bag** is "like a set, but each element can occur any (finite) number of times".

$$\Lbag x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x \Rbag = \Lbag 4,1,0,1,4 \Rbag = \Lbag 0,1,1,4,4 \Rbag \ne \Lbag 0,1,4 \Rbag$$

$\_\#\_ : t \to Bag\ t \to \mathbb{N}$ counts the number of occurrences: $1 \# \Lbag 0,0,0,1,1,4 \Rbag = 2$

$\_\sqsubseteq\_ : t \to Bag\ t \to \mathbb{B}$ is membership, with $x \sqsubseteq B \equiv x \# B \ne 0$: $1 \sqsubseteq \Lbag 0,0,0,1,1,4 \Rbag \equiv true$

Calculate: $\Lbag x \mid x \sqsubseteq \Lbag 0,0,0,1,1,4 \Rbag \Rbag = ?$

Define $bagProd : Bag\ \mathbb{N} \to \mathbb{N}$ such that: $bagProd\ \Lbag e_1, e_2, \ldots, e_n \Rbag = e_1 \cdot e_2 \cdot \ldots \cdot e_n$

e.g., $bagProd \Lbag 2,2,3,3,5 \Rbag = 180$

- 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: $B = \Lbag\ ? \mid \ ? \ \bullet\ ? \Rbag$

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
- $av\ S$ is the average of the elements of S
- $max\ S$ is the maximum of the elements of S

Reformulating the pigeonhole principle: (16.44) $av\ S > 1 \ \Rightarrow\ max\ S > 1$

Generalising:

(16.45) Pigeonhole principle:

If $S : Bag\ \mathbb{R}$ is non-empty, then: $av\ S \le max\ S$

Stronger on integers:

(16.46) Pigeonhole principle:

If $S : Bag\ \mathbb{Z}$ is non-empty, then: $\lceil av\ S \rceil \le max\ S$

Generalised Pigeonhole Principle — Application

(16.45) Pigeonhole principle: If $S : Bag\ \mathbb{R}$ is non-empty, then: $av\ S \le max\ S$

(16.46) Pigeonhole principle: If $S : Bag\ \mathbb{Z}$ is non-empty, then $\lceil av\ S \rceil \le max\ S$

(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. Therefore:

$$max\ S$$
$$\ge \langle \text{Pigeonhole principle (16.46) — } S \text{ contains integers} \rangle$$
$$\lceil av\ S \rceil$$
$$= \langle S \text{ has 7 values that sum to 8} \rangle$$
$$\lceil 8/7 \rceil$$
$$= \langle \text{Definition of ceiling} \rangle$$
$$2$$

Bag-based Specification of Sorting

Theorem "Sorting 1":

$$xs_0 = \mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner$$
$$\Rightarrow [\ \mathsf{SORT}$$
$$]$$
$$\mathsf{xs} \in (0\,..\,k) \twoheadrightarrow \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted\ xs}$$
$$\wedge \Lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \Rbag = \Lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \Rbag$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-30

Part 1: Total Correctness

Bag-based Specification of Sorting

Theorem "Sorting 1":

$$
\begin{array}{l}
\quad xs_0 = \mathsf{xs} \in (0 \mathrel{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \\
\Rightarrow [\ \mathsf{SORT} \\
\quad] \\
\quad \mathsf{xs} \in (0 \mathrel{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge\ \mathsf{sorted\ xs} \\
\qquad \wedge \Lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \Rbag = \Lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \Rbag
\end{array}
$$

A Verified Sorting Algorithm

Theorem "Sorting 0′":

$$
\begin{array}{l}
\quad \mathsf{xs} \in (0 \mathrel{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \\
\Rightarrow [\ \mathsf{while\ true\ do} \\
\qquad\quad \mathsf{xs} := \mathsf{xs} \oplus \{ \langle 0, 42 \rangle \} \\
\qquad \mathsf{od} \\
\quad] \\
\quad \mathsf{xs} \in (0 \mathrel{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge\ \mathsf{sorted\ xs} \\
\qquad \wedge \Lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \Rbag = \Lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \Rbag
\end{array}
$$

```
while true do
    xs[0] := 42
```

Proof:

$$
\begin{array}{l}
\quad \mathsf{xs} \in (0 \mathrel{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \\
\Rightarrow \langle \text{“Right-zero of } \Rightarrow \text{”} \rangle \\
\quad \mathsf{true} \\
\Rightarrow [\ \mathsf{while\ true\ do} \quad \mathsf{xs} := \mathsf{xs} \oplus \{ \langle 0, 42 \rangle \} \quad \mathsf{od} \\
\quad] \langle \text{“While” with subproof:} \\
\qquad\qquad \mathsf{true} \wedge \mathsf{true} \\
\qquad\quad \Rightarrow [\ \mathsf{xs} := \mathsf{xs} \oplus \{ \langle 0, 42 \rangle \} \ ] \\
\qquad\qquad \langle \text{“Idempotency of } \wedge \text{”, “Assignment” with substitution} \rangle \\
\qquad\qquad \mathsf{true} \\
\quad \rangle \\
\quad \neg\, \mathsf{true} \wedge \mathsf{true} \\
\Rightarrow \langle \text{“Contradiction”, “ex falso quodlibet”} \rangle \\
\quad \mathsf{xs} \in (0 \mathrel{..} k) \twoheadrightarrow \llcorner \mathbb{N} \lrcorner \ \wedge\ \mathsf{sorted\ xs} \\
\qquad \wedge \Lbag p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p \Rbag = \Lbag p \mid p \in xs_0 \bullet \mathsf{snd}\ p \Rbag
\end{array}
$$

Precondition-Postcondition Specifications in Dynamic Logic Notation

- 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:
Commands that do not terminate properly:

- Commands that crash — evaluating undefined expressions
- Infinite loops

Rules That Work for Both

Sequential composition:

```
Primitive inference rule “Sequence”:
    `P  ⇒[ C₁ ]  Q`,  `Q  ⇒[ C₂ ]  R`
  ├──────────────────────────────────
      `P  ⇒[ C₁ ; C₂ ]  R`
```

Strengthening the precondition:

```
     `P₁ ⇒ P₂`,    `P₂ ⇒[ C ] Q`
  ├───────────────────────────────
           `P₁ ⇒[ C ] Q`
```

Weakening the postcondition:

```
     `P ⇒[ C ] Q₁`,    `Q₁ ⇒ Q₂`
  ├───────────────────────────────
           `P ⇒[ C ] Q₂`
```

Total Correctness Rule for Assignment

Used so far: **Dynamic Logic Partial Correctness Assignment Axiom:**

$$Q[x := E] \quad \Rightarrow [\ x \mathrel{\mathbf{:=}} E\ ] \quad Q$$

LADM Total Correctness Assignment Axiom (10.1):

$$\{ \mathit{dom}\ \text{‘}E\text{’} \wedge Q[x := E] \} \quad x \mathrel{\mathbf{:=}} E \quad \{ Q \}$$

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.)

Assignment ":=":
Two characters;
type ":="

Substitution ":=":
One Unicode character;
type "\:="

Examples:

- $\mathit{dom}\ \text{‘}\mathit{sqrt}\ (x / y)\text{’} \quad \equiv \quad y \neq 0 \wedge x / y \geq 0$
- $\mathit{dom}\ \text{‘}a\ @\ i\text{’} \quad \equiv \quad i \in \mathit{Dom}\ a$
- For *int*-variables i and j:
  $\mathit{dom}\ \text{‘}i + j\text{’} \quad \equiv \quad \mathit{minint} \leq x + y \leq \mathit{maxint}$

Conditional Rule

Each evaluation of an expression E needs to be guarded by a precondition *dom* ‘E’:

$$\frac{\{ B \wedge P \}\quad C_1 \quad \{ Q \} \qquad\qquad \{ \neg B \wedge P \} \quad C_2 \quad \{ Q \}}{\{ \mathit{dom}\ \text{‘}B\text{’} \wedge P \} \quad \mathit{if}\ B\ \mathit{then}\ C_1\ \mathit{else}\ C_2\ \mathit{fi} \quad \{ Q \}}$$

"While" Rule

So far:

```
        `B ∧ Q  ⇒[ C ]  Q`
  ├──────────────────────────────────
   `Q  ⇒[ while B do C od ]  ¬ B ∧ Q`
```

Now **two** additional ingredients:

- **Invariant:** $Q : \mathbb{B}$ — as before, ensuring functional correctness
- **Variant** (or "bound function"): $T : \mathbb{Z}$ — ensuring termination

$$\frac{\{ B \wedge Q \} \quad C \quad \{ Q \} \qquad \{ B \wedge Q \wedge T = t_0 \} \quad C \quad \{ T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ \mathit{dom}\ \text{‘}B\text{’} \wedge Q \} \quad \mathit{while}\ B\ \mathit{do}\ C\ \mathit{od} \quad \{ \neg B \wedge Q \}}$$

In each iteration:

- The invariant Q is preserved.
- 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

$$\frac{\{ B \wedge Q \wedge T = t_0 \} \quad C \quad \{ Q \wedge T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ \mathit{dom}\ \text{‘}B\text{’} \wedge Q \} \quad \mathit{while}\ B\ \mathit{do}\ C\ \mathit{od} \quad \{ \neg B \wedge Q \}}\ \text{prov. } \neg \mathit{occurs}(\text{‘}t_0\text{’}, \text{‘}B, C, Q, T\text{’})$$

In each iteration:

- The invariant Q is preserved.
- The variant T decreases.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-11-30

Part 2: Frama-C

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!**

Start reading:
`https://allan-blanchard.fr/publis/frama-c-wp-tutorial-en.pdf`

The *findMax* Frame

findMax0.c:

```
/*@ requires ???;
    ensures ???;
 */
int findMax(int n, int a[]) {
  ???
}
```

Overall program correctness is based on **function contracts**:

- "**requires**": Procedure call precondition
- "**ensures**": Procedure call postcondition
  May refer to **result** for the return value.

Loops are "Opaque" — Need Annotations to Help Automatic Provers

Total correctness **While** rule:

$$\frac{\{ B \wedge Q \wedge T = t_0 \} \quad C \quad \{ Q \wedge T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ \mathit{dom}\ \text{‘}B\text{’} \wedge Q \} \quad \mathit{while}\ B\ \mathit{do}\ C\ \mathit{od} \quad \{ \neg B \wedge Q \}} \text{prov. } \neg \mathit{occurs}(\text{‘}t_0\text{’}, \text{‘}B, C, Q, T\text{’})$$

Loop invariant Q: Property always true in a 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 variant: To prove termination

- Show a metric that is **strictly decreasing** at each iteration and **bounded** by 0

Loop assigns: What is assigned within the loop

- More modular than integrating this into the pre-postcondition spec.

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
```

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;
}
```

findMax Attempt 2

findMax2.c:

```
/*@ requires 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[]) {
  int i;
  /*@
      loop invariant 0 ≤ i ≤ n;
      loop assigns i;
   */
  for( i = 0; i < n; i++) ;
  return 0;
}
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-12-02

Frama-C

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!**

Start reading:
`https://allan-blanchard.fr/publis/frama-c-wp-tutorial-en.pdf`

ACSL Function Contracts

Overall program correctness is based on **function contracts**, mainly:

- "**requires**": Procedure call precondition
- "**assigns**": Global variables that may be updated
- "**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

Total correctness **While** rule:

$$\frac{\{ B \wedge Q \wedge T = t_0 \} \quad C \quad \{ Q \wedge T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ \mathit{dom}\ \text{‘}B\text{’} \wedge Q \} \quad \mathit{while}\ B\ \mathit{do}\ C\ \mathit{od} \quad \{ \neg B \wedge Q \}} \text{prov. } \neg \mathit{occurs}(\text{‘}t_0\text{’}, \text{‘}B, C, Q, T\text{’})$$

"**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;
}
```

`findMax1.c`:

findMax Attempt 1

```
/*@ 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-gui -wp -wp-rte findMax1.c
```

`findMax1a.c`:

The *findMax* Attempt 1a

```
/*@ 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;
}
```

`findMax2.c`:

findMax Attempt 2

```
/*@ requires 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[]) {
  int i;
  /*@
      loop invariant 0 ≤ i ≤ n;
      loop assigns i;
  */
  for( i = 0; i < n; i++) ;
  return 0;
}
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-12-06

Part 1: Total/Partial Correctness, Relational Semantics

Recall: Total Correctness versus 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.
- 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.

Differences between partial and total correctness:

Commands that do not terminate properly:

- Commands that crash — evaluating undefined expressions
- Infinite loops

The Programming Language: Expressions and Commands

The types Cmd, ExprV, and Expr$\mathbb{B}$ are abstract syntax tree (AST) types

Declaration: $ExprV$, Expr$\mathbb{B}$: Type
Declaration: Cmd : Type

Declaration: _;_ : Cmd → Cmd → Cmd
Declaration: _:=_ : Var → ExprV → Cmd
Declaration: if_then_else_fi : Expr$\mathbb{B}$ → Cmd → Cmd → Cmd
Declaration: while_do_od : Expr$\mathbb{B}$ → Cmd → Cmd

Types for Semantics of Expressions and Commands

Imperative programs, such as Cmd, transform a State that assigns values to variables.

Declaration: Value : Type
Declaration: State : Type
Declaration: Var : Type

Axiom "Definition of `State`": State = Var → Value

Declaration: eval : State → ExprV → Value
Declaration: sat : Expr$\mathbb{B}$ → set State

Declaration: $\_ \oplus' \_ : (A \to B) \to \langle\!\langle A , B \rangle\!\rangle \to (A \to B)$
Axiom "Definition of function override":
$(x = z \Rightarrow (f \oplus' \langle x, y \rangle)\ z = y)$
$\wedge\ (x \neq z \Rightarrow (f \oplus' \langle x, y \rangle)\ z = f\ z)$

Semantics of Commands

Program execution induces a **state transformation relation**.

Declaration: $[\![\_]\!]$: Cmd → (State ↔ State)

Axiom "Semantics of := ":
$[\![\ x := e\ ]\!] = \{ s : \mathsf{State} \bullet \langle s , s \oplus' \langle x , \mathsf{eval}\ s\ e \rangle \rangle \}$
Axiom "Semantics of ; ": $[\![\ C_1 ; C_2\ ]\!] = [\![\ C_1\ ]\!] \mathbin{⨾} [\![\ C_2\ ]\!]$
Axiom "Semantics of `if`":
$[\![\ \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{fi}\ ]\!] = (\mathsf{sat}\ B \lhd [\![\ C_1\ ]\!]) \cup (\mathsf{sat}\ B \mathrel{⩤} [\![\ C_2\ ]\!])$

Axiom "Semantics of `while`":
$[\![\ \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}\ ]\!] = (\mathsf{sat}\ B \lhd [\![\ C\ ]\!])^{*} \mathrel{⩥} \mathsf{sat}\ B$

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":
$(P \Rightarrow [\langle\ C\ \rangle] Q) \equiv \mathsf{sat}\ P \subseteq \mathsf{Dom}\ [\![\ C\ ]\!] \wedge [\![\ C\ ]\!]\ (\!|\ \mathsf{sat}\ P\ |\!) \subseteq \mathsf{sat}\ Q$

- 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.

Axiom "Partial Correctness":
$(P \Rightarrow [\ C\ ] Q) \equiv [\![\ C\ ]\!]\ (\!|\ \mathsf{sat}\ P\ |\!) \subseteq \mathsf{sat}\ Q$

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":
$(P \Rightarrow [\langle\ C\ \rangle] Q)$
$\equiv (\forall s_1 \mid s_1 \in \mathsf{sat}\ P \bullet \exists s_2 \mid s_1\ (\ [\![\ C\ ]\!]\ )\ s_2 \bullet s_2 \in \mathsf{sat}\ Q)$
$\wedge (\forall s_1, s_2 \bullet s_1 \in \mathsf{sat}\ P \wedge s_1\ (\ [\![\ C\ ]\!]\ )\ s_2 \Rightarrow s_2 \in \mathsf{sat}\ Q)$

- 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":
$(P \Rightarrow [\ C\ ] Q)$
$\equiv \forall s_1, s_2 \bullet s_1 \in \mathsf{sat}\ P \wedge s_1\ (\ [\![\ C\ ]\!]\ )\ s_2 \Rightarrow s_2 \in \mathsf{sat}\ Q$

H16: Blanchard: Hoare Triples

2.1.3. Hoare triples

Hoare logic is a program formalization method proposed by Tony Hoare in 1969 in a paper entitled *An Axiomatic Basis for Computer Programming*. This method defines:

- axioms, that are properties we admit, such as "the skip action does not change the program state",
- rules to reason about the different allowed combinations of actions, for example "the skip action followed by the action A" is equivalent to "the action A".

The behavior of the program is defined by what we call "Hoare triples":

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

Where P and Q are predicates, logic formulas that express properties about the memory at particular program points. C is a list of instructions that defines the program. This syntax expresses the following idea: "if we are in a state where P is verified, after executing C and if C terminates, then Q is verified for the new state of the execution". Put in another way, P is a sufficient precondition to ensure that C will bring us to the postcondition Q. For example,

---

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-12-06

Part 2: Frama-C: Behaviours, . . .

"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));
*/
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-12-07

Part 1: Z Function Set Arrows

Professional Behaviour for Students

Learn a lot!

Behave with Academic Integrity!

Fill in the evaluations for **all your courses!** ⟶ https://evals.mcmaster.ca/

- Response rates are noted at the Faculty level
- The better the Faculty sees CompSci, the more interesting electives **you** will have available in Level IV
- Do all you can to get the response rates up for all COMPSCI courses!

Plan for Today

- The Z Specification Notation
- λ-abstraction, . . .
- "Natural Deduction" — A different presentation of logics (LADM ch. 7)
- **Conclusion**

Review Sessions — **Details to be announced — likely dates:**

- Mon., Dec. 13th
- Tue., Dec. 14th
- Wed., Dec. 15th

- 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 evaluations for **all your courses!** ⟶ https://evals.mcmaster.ca/

The Z Specification Notation

- Mathematical notation intended for software specification
- ISO-standardised
- Two parts:
  - Typed set theory in first-order predicate logic
    — essentially the logic and set theory you are using in CalcCheck
    — except that in Z, types **are** maximal sets
  - "Schemas" modelling of states and state transitions
- Avenue ⟶ Resources ⟶ Links ⟶ Z

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

$$X ⇸ Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$$

$$X \longrightarrow Y == \{ f : X ⇸ Y \mid \operatorname{dom} f = X \}$$

$$X ⤔ Y == \{ f : X ⇸ Y \mid (\forall x_1, x_2 : \operatorname{dom} f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \}$$

$$X \rightarrowtail Y == (X ⤔ Y) \cap (X \longrightarrow Y)$$

$$X ⤀ Y == \{ f : X ⇸ Y \mid \operatorname{ran} f = Y \}$$

$$X \twoheadrightarrow Y == (X ⤀ Y) \cap (X \longrightarrow Y)$$

$$X ⤖ Y == (X \twoheadrightarrow Y) \cap (X \rightarrowtail Y)$$

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 $X \to Y$ 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 $S \circ R = R \fatsemi S$.

$X \nrightarrow Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \wedge (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$

$X \longrightarrow Y == \{ f : X \nrightarrow Y \mid \operatorname{dom} f = X \}$

$X \mathrel{⤔} Y == \{ f : X \nrightarrow Y \mid (\forall x_1, x_2 : \operatorname{dom} f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \}$

$X \rightarrowtail Y == (X \mathrel{⤔} Y) \cap (X \longrightarrow Y)$

Laws:

$f \in X \nrightarrow Y \Leftrightarrow f \circ f^\sim = \mathrm{id}(\operatorname{ran} f)$

$f \in X \mathrel{⤔} Y \Leftrightarrow f \in X \nrightarrow Y \wedge f^\sim \in Y \nrightarrow X$

$f \in X \rightarrowtail Y \Leftrightarrow f \in X \longrightarrow Y \wedge f^\sim \in Y \nrightarrow X$

$f \in X \mathrel{⤔} Y \Rightarrow f(\!| S \cap T |\!) = f(\!| S |\!) \cap f(\!| T |\!)$

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 $S \circ R = R \fatsemi S$.

$X \nrightarrow Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \wedge (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$

$X \longrightarrow Y == \{ f : X \nrightarrow Y \mid \operatorname{dom} f = X \}$

$X \mathrel{⤀} Y == \{ f : X \nrightarrow Y \mid \operatorname{ran} f = Y \}$

$X \twoheadrightarrow Y == (X \mathrel{⤀} Y) \cap (X \longrightarrow Y)$

$X \mathrel{⤖} Y == (X \twoheadrightarrow Y) \cap (X \rightarrowtail Y)$

Laws:

$f \in X \mathrel{⤖} Y \Leftrightarrow f \in X \longrightarrow Y \wedge f^\sim \in Y \longrightarrow X$

$f \in X \mathrel{⤀} Y \Rightarrow f \circ f^\sim = \mathrm{id}\ Y$

Z Function Sets in CalcCheck

For two sets $A : \mathbf{set}\ t_1$ and $B : \mathbf{set}\ t_2$, we define the following **function sets**:

CalcCheck				Z
$f \in A \longrightarrow B$	`\tfun`	total function	$Dom\ f = A \wedge f^\smile \fatsemi f \subseteq \mathrm{id}\ B$	$f \in A \to B$
$f \in A \nrightarrow B$	`\pfun`	partial function	$Dom\ f \subseteq A \wedge f^\smile \fatsemi f \subseteq \mathrm{id}\ B$	$f \in A \nrightarrow B$
$f \in A \rightarrowtail B$	`\tinj`	total injection	$f \fatsemi f^\smile = \mathrm{id}\ A \wedge f^\smile \fatsemi f \subseteq \mathrm{id}\ B$	$f \in A \rightarrowtail B$
$f \in A \mathrel{⤔} B$	`\pinj`	partial injection	$f \fatsemi f^\smile \subseteq \mathrm{id}\ A \wedge f^\smile \fatsemi f \subseteq \mathrm{id}\ B$	$f \in A \mathrel{⤔} B$
$f \in A \twoheadrightarrow B$	`\tsurj`	total surjection	$Dom\ f = A \wedge f^\smile \fatsemi f = \mathrm{id}\ B$	$f \in A \twoheadrightarrow B$
$f \in A \mathrel{⤀} B$	`\psurj`	partial surjection	$Dom\ f \subseteq A \wedge f^\smile \fatsemi f = \mathrm{id}\ B$	$f \in A \mathrel{⤀} B$
$f \in A \mathrel{⤖} B$	`\tbij`	total bijection	$f \fatsemi f^\smile = \mathrm{id}\ A \wedge f^\smile \fatsemi f = \mathrm{id}\ B$	$f \in A \mathrel{⤖} B$
$f \in A \mathrel{⤗} B$	`\pbij`	partial bijection	$f \fatsemi f^\smile \subseteq \mathrm{id}\ A \wedge f^\smile \fatsemi f = \mathrm{id}\ B$	

Counting …

Let A and B be finite sets with $\# A = a$ and $\# B = b$:

- $\# (A \times B) = ?$ — pairs
- $\# (A \leftrightarrow B) = \# (\mathbb{P} (A \times B)) = ?$ — relations
- $\# (A \longrightarrow B) = ?$ — total functions
- $\# (A \nrightarrow B) = ?$ — partial functions
- $\# (A \mathrel{⤖} A) = ?$ — homogeneous total bijections
- $\# (A \mathrel{⤖} B) = ?$ — total bijections
- $\# (A \rightarrowtail B) = ?$ — total injections
- $\# (A \mathrel{⤗} B) = ?$ — partial bijections
- $\# (A \mathrel{⤔} B) = ?$ — partial injections
- $\# (A \twoheadrightarrow B) = ?$ — total surjections
- $\# \{ S \mid S \subseteq B \wedge \# S = a \} = ?$ — a-combinations of B

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-12-07

Part 2: λ, O

λ-Abstraction

λ-abstraction creates nameless functions: If $E : B$, then $(\lambda x : A \bullet E) : A \to B$.
The following are usually introduced as left-to-right reduction rules:

Theorem "β-reduction": $(\lambda x \bullet E)\ a = E[x := a]$

Theorem "η-reduction": $(\lambda x : A \bullet F\ x) = F$ — provided $\neg occurs(\text{‘}x\text{’}, \text{‘}F\text{’})$

In addition, "α-conversion" is capture-avoiding renaming of bound variables.

Theorem "Function extensionality": $f = g \equiv \forall x \bullet f\ x = g\ x$

Theorem "Refl.-trans. closure": $R\,{*}$ **is** $(\lambda S \bullet \text{reflexive } S \wedge \text{transitive } S)$ closure-of R

Proof:
Using "Relation closure":
Subproof for \`$R \subseteq R\,{*}$\`:
By "Characterisation of \`_*\`: Expanding"
Subproof for \`$(\lambda S \bullet \text{reflexive } S \wedge \text{transitive } S)\ (R\,{*})$\`:
$(\lambda S \bullet \text{reflexive } S \wedge \text{transitive } S)\ (R\,{*})$
$\equiv \langle$ "β-reduction", substitution $\rangle$
$\text{reflexive } (R\,{*}) \wedge \text{transitive } (R\,{*})$
Proof for this:
By "Characterisation of \`_*\`: Reflexivity"
and "Characterisation of \`_*\`: Transitivity"
and "Idempotency of $\wedge$"
Subproof for \`$\forall S \bullet (\lambda S \bullet \text{reflexive } S \wedge \text{transitive } S)\ S \wedge R \subseteq S \Rightarrow R\,{*} \subseteq S$\`:
For any \`$S$\`:
Assuming (rt) \`$(\lambda S \bullet \text{reflexive } S \wedge \text{transitive } S)\ S$\`

Big-O

Does $O(n \cdot log\ n)$ talk about n? — Abuse of notation!

$O(n \cdot log\ n)$ talks about the function "$\lambda n \bullet n \cdot log\ n$"!

Declaration: $O : (\mathbb{R} \to \mathbb{R}) \to \text{set } (\mathbb{R} \to \mathbb{R})$

Axiom "Definition of big O":
$f \in O\,g \equiv \exists b \bullet \exists c \mid c > 0 \bullet \forall x \mid x > b \bullet \text{abs } (f\ x) < c \cdot g\ x$

Theorem: $(\lambda x \bullet 4 \cdot x + 7) \in O\ (\lambda x \bullet x)$

Proof:
$(\lambda x \bullet 4 \cdot x + 7) \in O\ (\lambda x \bullet x)$
$\equiv \langle$ "Definition of big O" $\rangle$
$\exists b \bullet \exists c \mid c > 0 \bullet \forall x \mid x > b \bullet \text{abs } ((\lambda x \bullet 4 \cdot x + 7)\ x) < c \cdot (\lambda x \bullet x)\ x$
$\equiv \langle$ "β-reduction", substitution $\rangle$
$\exists b \bullet \exists c \mid c > 0 \bullet \forall x \mid x > b \bullet \text{abs } (4 \cdot x + 7) < c \cdot x$
$\Leftarrow \langle$ "$\exists$-Introduction" $\rangle$
$(\exists c \mid c > 0 \bullet \forall x \mid x > b \bullet \text{abs } (4 \cdot x + 7) < c \cdot x)[b := 2]$
$\equiv \langle$ Substitution, "Trading for $\exists$" $\rangle$
$(\exists c \bullet c > 0 \wedge \forall x \mid x > 2 \bullet \text{abs } (4 \cdot x + 7) < c \cdot x)$
$\Leftarrow \langle$ "$\exists$-Introduction" $\rangle$
$(c > 0 \wedge \forall x \mid x > 2 \bullet \text{abs } (4 \cdot x + 7) < c \cdot x)[c := 8]$
$\equiv \langle$ Substitution, Fact \`$8 > 0$\`, "Identity of $\wedge$" $\rangle$
$(\forall x \mid x > 2 \bullet \text{abs } (4 \cdot x + 7) < 8 \cdot x)$
Proof for this:
For any \`$x$\` **satisfying** \`$2 < x$\`:
Side proof for (1) \`$4 \cdot x + 7 > 0$\`:

Recall: Topological Sort — Complete $O(\# B + \# V)$ Algorithm

for $p \in B$ **do**
  $preCount[snd\ p] := preCount[snd\ p] + 1$
  $postSet[fst\ p] := postSet[fst\ p] \cup \{snd\ p\}$
od ;
$sources := \epsilon$; **for** $v \in 0\,..\,k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
ghost $vs := 0\,..\,k$;
$s := \epsilon$
while $sources \neq \epsilon$ **do**
  $u := head\ sources$;
  $s := s \triangleright u$;
  $sources := tail\ sources$; — remove u from $sources$
  ghost $vs := vs - \{u\}$;
  for $v \in postSet[u]$ **do**
    $preCount[v] := preCount[v] - 1$;
    if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
  od
od

Topological Sort — Complete $O(\# B + \# V)$-ghosted Algorithm

ghost int stepCount = 0 ;
for $p \in B$ **do**
  $preCount[snd\ p] := preCount[snd\ p] + 1$; **ghost** stepCount++ ;
  $postSet[fst\ p] := postSet[fst\ p] \cup \{snd\ p\}$; **ghost** stepCount++
od ;
$sources := \epsilon$;
for $v \in 0\,..\,k$ **do** **ghost** stepCount++ ; **if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
$s := \epsilon$
while $sources \neq \epsilon$ **do**
  $u := head\ sources$; $s := s \triangleright u$; **ghost** stepCount++ ;
  $sources := tail\ sources$; — remove u from $sources$
  for $v \in postSet[u]$ **do**
    $preCount[v] := preCount[v] - 1$; **ghost** stepCount++ ;
    if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
  od
od ;
ghost assert $\text{stepCount} \leq C_1 \cdot \# B + C_2 \cdot \# V$;

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2021

Wolfram Kahl

2021-12-07

Part 3: Natural Deduction, Conclusion

(Simplified) Inference Rules —- See LADM p. 133, “Using Z” ch. 2&3

“Natural Deduction” — A Presentation of Logic for Mathematical Study of Logic

$\dfrac{P \wedge Q}{P}$ $\wedge$-Elim$_1$ $\quad$ $\dfrac{P \wedge Q}{Q}$ $\wedge$-Elim$_2$ $\quad$ $\dfrac{\forall\, x \bullet P}{P[x := E]}$ Instantiation ($\forall$-Elim)

$\dfrac{P}{P \vee Q}$ $\vee$-Intro$_1$ $\quad$ $\dfrac{Q}{P \vee Q}$ $\vee$-Intro$_2$ $\quad$ $\dfrac{P[x := E]}{\exists\, x \bullet P}$ $\exists$-Intro

$\dfrac{P \Rightarrow Q \qquad P}{Q}$ $\Rightarrow$-Elim $\quad$ $\dfrac{P \qquad Q}{P \wedge Q}$ $\wedge$-Intro $\quad$ $\dfrac{P}{\forall\, x \bullet P}$ $\forall$-Intro (prov. x not free in assumptions)

$\dfrac{\begin{matrix}\ulcorner P \urcorner \\ \vdots \\ Q\end{matrix}}{P \Rightarrow Q}$ $\Rightarrow$-Intro $\quad$ $\dfrac{P \vee Q \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix} \qquad \begin{matrix}\ulcorner Q \urcorner \\ \vdots \\ R\end{matrix}}{R}$ $\vee$-Elim $\quad$ $\dfrac{(\exists x \bullet P) \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix}}{R}$ $\exists$-Elim (prov. x not free in R, assumptions)

About Natural Deduction

Example proof (using the inference rules as shown in Using Z):

$$\dfrac{\dfrac{\ulcorner \exists x : a \bullet p \Rightarrow q \urcorner^{[1]} \qquad \dfrac{\ulcorner x \in a \urcorner^{[3]} \qquad \dfrac{\ulcorner p \Rightarrow q \urcorner^{[3]} \qquad \dfrac{\ulcorner \forall x : a \bullet p \urcorner^{[2]} \qquad \ulcorner x \in a \urcorner^{[3]}}{p}\ \forall\text{-elim}}{q}\ \Rightarrow\text{-elim}}{\exists x : a \bullet q}\ \exists\text{-intro}}{\dfrac{\exists x : a \bullet q}{(\forall x : a \bullet p) \Rightarrow (\exists x : a \bullet q)}\ \Rightarrow\text{-intro}^{[2]}}\ \exists\text{-elim}^{[3]}}{(\exists x : a \bullet p \Rightarrow q) \quad \Rightarrow \quad ((\forall x : a \bullet p) \Rightarrow (\exists x : a \bullet q))}\ \Rightarrow\text{-intro}^{[1]}$$

- 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 **run-time errors** imposes additional preconditions on commands
- Termination is typically dealt with separately 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

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 natural way:
  - 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 od 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

- **2AC3 Automata and Computability**
  — formal languages, grammars, finite automata, transition relations, Kleene algebra! acceptance predicates, ...
- **CS 2SD3 / SE 3BB4 Concurrent Systems Design**
  —correctness of concurrent programs, 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
- **3RA3 Software Requirements**
  — Capturing **precisely** what the customer wants, formalisation
- COMPSCI **3EA3 Software and System Correctness**
  — Formal specifications, validation, verification
- **3FP3 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!**