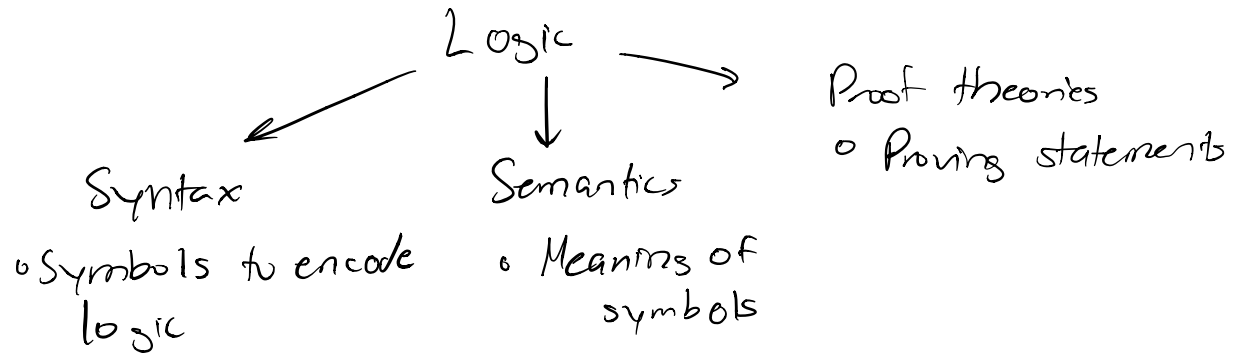
SE 212

Introduction: Logic



- Argument ⟶ premise ⇒ conclusion

Syntax

- Creates a well-formed formula
- Linking w/ semantics:
  1. Entailment: '$\models$'

     $P \models Q$: Q is true wherever P is true.

     $\models P$: P is always true

     $P_1, P_2, \ldots, P_n \models Q$: $P_1 \ldots P_n$ tell us that we can conclude Q.

  2. Prove: '$\vdash$'

     $P_1, P_2, \ldots, P_n \vdash Q$: Can derive Q from premises.

- Soundness & Completeness: Proof theories.
  - Soundness: $P \vdash Q \Rightarrow P \models Q$ (proving the arguments)
    - If I prove something, then whenever premise is true, conclusion is true.

- Completeness: $P \models Q \Rightarrow P \vdash Q$
  - I can prove valid arguments in this proof theory

PROPOSITIONAL LOGIC: SYNTAX

- Syntax:
  - Formulas: T/F, prop. symbols, connectives, brackets
    - Symbols: P, Q, A . . . (either T or F)
    - Connectives: operations between propositions

Symbol	Operation	George
$\neg$	not	!
$\wedge$	and	&
$\vee$	or	|
$\Rightarrow$	implication	=>
$\Leftrightarrow$	equivalence	<=>

(Order of precedence ↓)

    - Right-associative: $a \wedge (b \wedge c)$
    - Ex:// $\neg p \wedge \neg q \vee r \equiv (\neg p) \wedge (\neg q) \vee r$
      $\equiv ((\neg p) \wedge (\neg q)) \vee r$
  - Terminology:
    - $P \vee Q$: P, Q disjuncts
    - $P \wedge Q$: P, Q conjuncts

$$P \Rightarrow Q$$

P: premise, antecedent; Q: conclusion, consequent

- Formalizing natural language:

- Proposition identification:
  1. Highlight the statements only true or false
  2. No connective words
- Prime proposition: statements that cannot be broken down
  - Ex:// rain is falling, I bought a car...
- Identify connectives: use Nissanke's rules
  - ____ unless ____ : $P \vee Q$
- Put in order of sentence.
- Ex:// It is snowing only if it cold
  1. Prop. identify:
     S = 'it is snowing'
     P = 'it is cold'
  2. Connect:
     $S \Rightarrow P$ (Nissanke's rule)
- Ex:// If a request occurs, then it is acknowledged or process does not make progress.
  r = 'request occurs', a = 'acknowledged',
  p = 'make progress'

  1) $(r \Rightarrow a) \vee (\neg r \Rightarrow \neg p)$

  or

  $r \Rightarrow (a \vee \neg p)$
- Ambiguous: perform truth table analysis + add

column to represent intuition ⇒ see which version makes most sense.

PROPOSITIONAL LOGIC: SEMANTICS

- Semantics: wff → truth values / truth functions.

- Boolean valuation: $\int$: syntax → T/F
  - Models / interpretations.

$$[p \wedge q \Rightarrow z]$$

$$[p] = T, \quad [q] = F \quad [z] = T$$

  - Substituting truth values into syntax!
  - Use square brackets to represent BV.
  - Steps:
    1. Identify the syntax for BV

$$[p \wedge q \Rightarrow z]$$

    2. Map syntax to functions:

$\wedge$: AND $\Rightarrow$: IMP $\neg$: NOT
$\vee$: OR $\Leftrightarrow$: IFF

$$[p \wedge q \Rightarrow z] = ([p] \text{ AND } [q]) \text{ IMP } [z]$$

    3. Substitute truth values:

$$([p] \text{ AND } [q]) \text{ IMP } [z] = (T \text{ AND } F) \text{ IMP } T$$

    4. Evaluate: F IMP T = T

- Truth tables: show all possible BV

	p (p=4)	q (p=2)	z (p=1)	syntax
2^n	T	T	T	
	T	T	F	
	T	F	T	
	T	F	F	

-

p	q	z	S_1	S_2	S_3
			T	T	F
			F	T	F
			T	T	F
			T	T	F

↙ Contradiction: $\forall [S] = F$

↑ $\exists [S] = T \Rightarrow$ satisfiable

↳ tautology: $\forall [S] = T$

- How do you show satisfiability / tautology / contradiction?
  1) Truth table
  2) Think of BV s.t. condition is true.

Ex:// Show that $\neg(p \Rightarrow q) \wedge \neg(\neg r \Rightarrow \neg p)$ is satisfiable?

$$[\neg(p \Rightarrow q) \wedge \neg(\neg r \Rightarrow \neg p)] = T$$

false · false · true · true

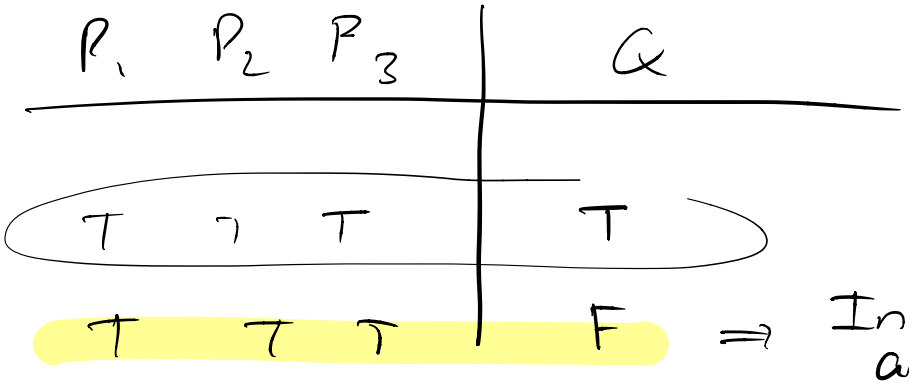
i) $[p \Rightarrow q] = \text{false}$? $\Rightarrow$ $p = [T]$, $[q] = \text{false}$

ii) $[\neg r \Rightarrow \neg p] = \text{false}$? $\Rightarrow$ $r = [F]$, $[p] = \text{true}$.

- Tautology: p is a tautology: $\underline{\underline{\models p}}$

- Logical implication: $P \models Q \Leftrightarrow \forall B.V\ [P]=T, [Q]=T$

  1) Truth table:



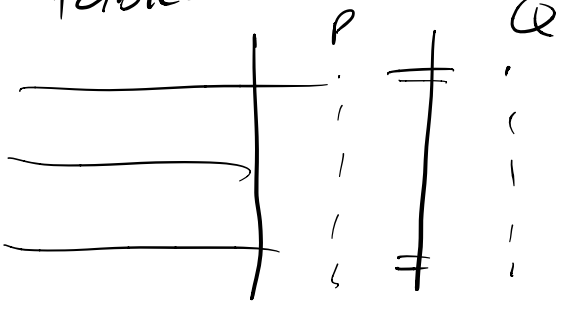
$\Rightarrow$ Invalid argument

  - Invalidity: show situation where $[P_{1\ldots n}]=T$ but $[Q]=F$
    1) Truth table
    2) Boolean valuation

  2) Show $\models P \Rightarrow Q$
  - Use some sort of proof theory to show $P \Rightarrow Q \leftrightarrow T$

- Contingent: argument is a collection of T/F
- Logical equivalence: $P \Leftrightarrow Q \Leftrightarrow [P] = [Q]$

  1) Truth tables:

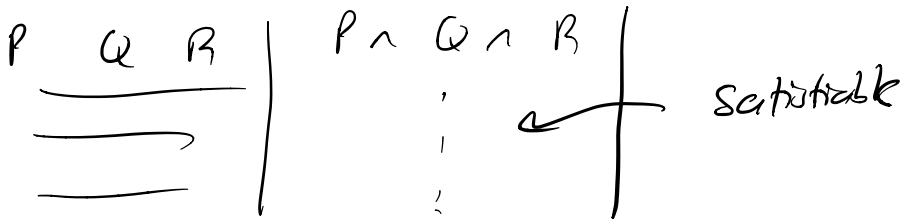


{ columns are identical }

  2) Proof theory

- Consistency: $\{P, Q, R \ldots\}$ is consistent if $\exists$ a B.V. s.t. all formulas are true.
  - Concerns a set of formulas

- Consistency can be shown:
  1) Conjunction is satisfiable / not a contradiction



  2) Truth table

- Ex: // 1. Sales of house falls as interest ↑
  2. Ave not happy if sales of houses ↓
  3. Interest rates are rising
  4. Alchemeers.

Step #1: Formalization.

s = sales ↓
r = rates ↑
h = happy

1. $r \Rightarrow s$
2. $s \Rightarrow \neg h$
3. r
4. h

Step #2: TT

s	r	h	$r \Rightarrow s$	$s \Rightarrow \neg h$

PROOF THEORY: TRANSFORMATIONAL PROOFS

- Determine equivalence of P and Q
- Use Boolean logic laws to turn $P \rightarrow Q$
  - List of Boolean logic laws
- General structure:
  1. State what you are trying to prove.

$$((c \Rightarrow d) \wedge (b \vee a)) \wedge \neg(b \vee a) \Leftrightarrow F$$

2. Start with 1 side:

1) $((c \Rightarrow d) \wedge (b \vee c)) \wedge \neg(b \vee a)$

3. Transform:

1) select a law: make sure you can actually use it

2) Apply it and put result on new line

3) Next to the result, write: by $\frac{\text{law}}{\quad}$

1) $((c \Rightarrow d) \wedge (b \vee c)) \wedge \neg(b \vee a)$

$\Leftrightarrow (c \Rightarrow d) \wedge ((b \vee c) \wedge \neg(b \vee a))$ by assoc

4. Use those steps to get your proof.

1) $((c \Rightarrow d) \wedge (b \vee c)) \wedge \neg(b \vee a)$

$\Leftrightarrow (c \Rightarrow d) \wedge ((b \vee c) \wedge \neg(b \vee a))$ by assoc

$\Leftrightarrow (c \Rightarrow d) \wedge$ false by contr

$\Leftrightarrow$ false by simpl

- Able to use 1 law multiple times on 1 line
- 2 implicit laws that we are using in TP:
  1. Rule of substitution: $P \Leftrightarrow Q$ and $P \wedge \ldots R$, then $P \wedge \ldots R \Leftrightarrow P \wedge \ldots Q$.
  2. Transitivity: $P \Leftrightarrow Q$, $Q \Leftrightarrow R \Rightarrow P \Leftrightarrow R$
- Sound and complete!
- Ex:// $p \wedge (\neg(\neg q \wedge \neg p) \vee p) \Leftrightarrow p$

$p \wedge (\neg(\neg q \wedge \neg p) \vee p) \Leftrightarrow p \wedge (\neg\neg(q \vee p) \vee p)$ by dm

$\Leftrightarrow p \wedge (q \vee p \vee p)$ by neg

$\leftrightarrow p \wedge (q \vee p)$ by simp1

$\leftrightarrow p$ by simp2

- Ex:// $\neg true \leftrightarrow false$

$\neg true \leftrightarrow \neg(p \vee \neg p)$ by lem

$\leftrightarrow \neg p \wedge \neg\neg p$ by dn

$\leftrightarrow \neg p \wedge p$ by neg

$\leftrightarrow false$ by contr

- Tips:
  1. Simplify impl. + equiv. as fast as possible
  2. Simplify ASAP
  3. Don't sleep on LEM and simp2
  4. Use distribution law backwards for simplification (factoring)

- Applications:
  1. Proving code reachibility: if else that's nested
  2. Normal forms: POS/SOP from ECE 124
     - Literal: either a single prop. symbol or its negation $(p, \neg p)$
     - Conjunctive NF (CNF): conjuction of clauses

       clause $\wedge$ clause $\wedge$ clause

       ↳ Disjunction of literals or a literal itself

       ▷ $p \wedge \neg p \wedge (p \vee q)$ ✓
     - Disjunctive NF (DNF): disjunction of clause

       clause $\vee$ clause $\vee$ clause

↳ Conjunction of literals or a literal itself

- What about: p, $\neg p$, true, false (both)
- You can use TP to turn anything into CNF or DNF
- Mistake: $p \vee (q \wedge (p \wedge \neg q))$ ✗ (compound clause)
- Tips: De Morgan's laws are useful and use distribution.
- Ex:// $\neg(p \vee q) \wedge \neg r)$ into CNF

$\neg(p \vee q) \wedge \neg r) \leftrightarrow \neg(p \vee \neg q) \vee \neg\neg r$ by dm

$\leftrightarrow \neg(p \vee \neg q) \vee r$ by neg

$\leftrightarrow (\neg p \wedge \neg\neg q) \vee r$ by dm

$\leftrightarrow (\neg p \wedge q) \vee r$ by neg

✓ $\leftrightarrow (\neg p \vee r) \wedge (q \vee r)$ by dist

PROOF THEORY: NATURAL DEDUCTION

$P_1, P_2, P_3, \ldots, P_n \vdash C$

premises — conclusion

premises —(inference)→ conclusion

- To prove: take premises + use inference rules to get conclusion
- Inference rules: eliminating / introducing connective
  - How to use: match up premises w/ needed premises in rule

- Ex:// $a \wedge (a \Rightarrow c), c \vdash (a \Rightarrow c) \wedge c$

① List premises

1) $a \wedge (a \Rightarrow c)$ premise

2) c premise

② Think backwards + determine what you NEED

1) $a \wedge (a \Rightarrow c)$ premise

2) c premise

3) $a \Rightarrow c$ by and-e on 1

4) $(a \Rightarrow c) \wedge c$ by and-i on 2, 3

- Ex:// $\neg a \Leftrightarrow \neg b, \neg b \wedge a \vdash c$

1) $\neg a \Leftrightarrow \neg b$ premise

2) $\neg b \wedge a$ premise

3) $\neg b$ by and-e on 2

4) a by and-e on 2

5) $\neg a$ by iff_mp on 3, 1

6) c by not-e on 4, 5

- Subproofs: break down proof into manageable parts
  - Generally:

x) P

x+1) sub-proof opening R {

⋮ $A \wedge B$

⋮

x+y) a

}

x+y+1) conclusion by subproof-rule on x+1 — x+y

Use R to get to conclusion

Things in subproof are not accessible outside of proof

- imp-i: you want to introduce $a \Rightarrow b$

  assume x {

  ⋮

  y

  }

  $x \Rightarrow y$

  - Ex:// $a \Rightarrow b \Rightarrow c, b \vdash a \Rightarrow c$

    1) $a \Rightarrow b \Rightarrow c$ premise

    2) b premise

    3) assume a {

    4) $b \Rightarrow c$ by imp-e on 1, 3

    5) c by imp-e on 2, 4

    }

    6) $a \Rightarrow c$ by imp-i on 3 - 5

- Indirect proof (raa): disprove opposite of conclusion

  disprove R {

  ⋮

  false ⇒ Often done not-e

  ⇓

  Show contradiction w/ another line in proof.

  }

  $\neg R$

  - Hint: negations in premises, unreachable conc. w/out contr. or no premises
  - Ex:// $p \wedge \neg q \Rightarrow r, \neg r, p \vdash q$

    1) $p \wedge \neg q \Rightarrow r$ premise

    2) $\neg r$ premise

    3) p premise

    4) $\neg(p \wedge \neg q)$ by imp-e on 1, 2

5) assume $\neg q$ {

6) $p \wedge \neg q$ by and-i on 3, 5

7) false by not-e on 4, 6

}

8) q by raa 5-7

- Case analysis: used a lot w/ ors

→ $p \vee q$ ⇒ Law of Excluded middle can help introduce or

{ case p {

⋮

r

}

{ case q {

⋮

r

}

r by cases

- Ex:// $p \Rightarrow s, j \Rightarrow s, p \vee j \vdash s$

1) $p \Rightarrow s$ premise

2) $j \Rightarrow s$ premise

3) $p \vee j$ premise

4) case p {

5) s by imp-e on 1, 4

}

6) case j {

7) s by imp-e on 2, 6

}

8) s by cases 3, 4-5, 6-7

- Tip: find subgoals in order to prove statement

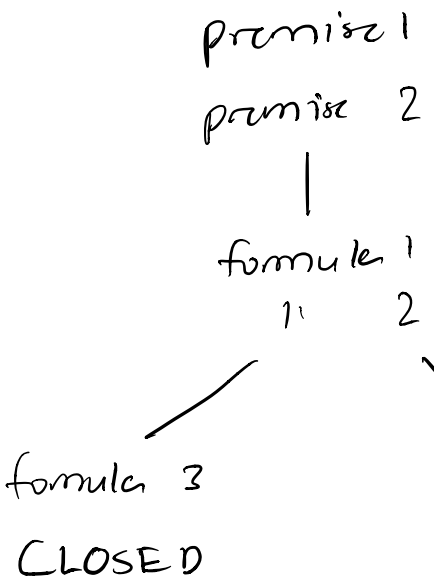
- Ex: // $b \Rightarrow c \vdash \neg b \vee c$

1) $b \Rightarrow c$ premise
2) $b \vee \neg b$ by lem
3) case b {
   4) c by imp-e on 1, 3
   5) $\neg b \vee c$ by or-i on 4
   }
6) case $\neg b$ {
   7) $\neg b \vee c$ by or-i on 6
   }
8) $\neg b \vee c$ by cases on 2, 3-5, 6-7

Use if or needed!

PROOF THEORY: SEMANTIC TABLEAUX

- ST: way to show all possible ways of making a conjunction true:



↳ Contradiction along line of argument

- Really good way of showing invalid argument:
  - ∀ branches ⇒ contradiction!
- Steps:

1. Pivit out formulas using branching rules
2. Close branches through contradictory arguments

- Ex:// Show that $r \Rightarrow s$, $s \Rightarrow \neg h$, r, h are inconsistant

1. $r \Rightarrow s$
2. $s \Rightarrow \neg h$
3. r
4. h

imp-br on 1

5. $\neg r$ — CLOSED on 3, 5

6. s

7. $\neg s$ — CLOSED 6, 7

8. $\neg h$ — Closed 8, 4

- Tip: use non-branching rules to get to shorter proofs
- George: # check ST

1) . .
2) . . .
3) rule-name on line {

4) ——

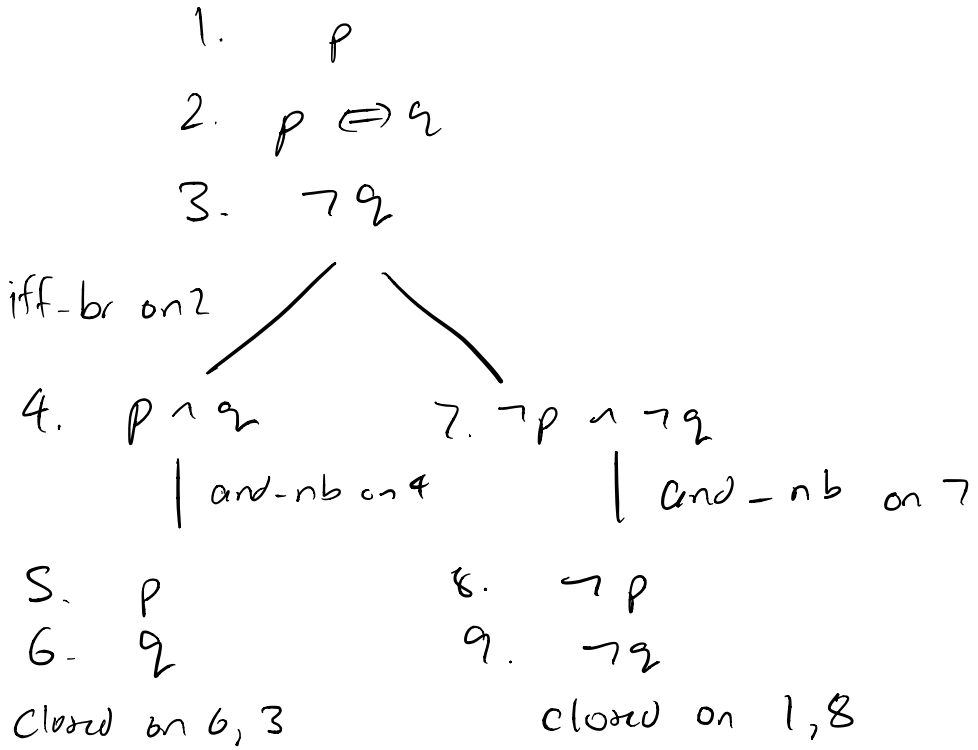
↓ → Close on —, —

5) ——

↓

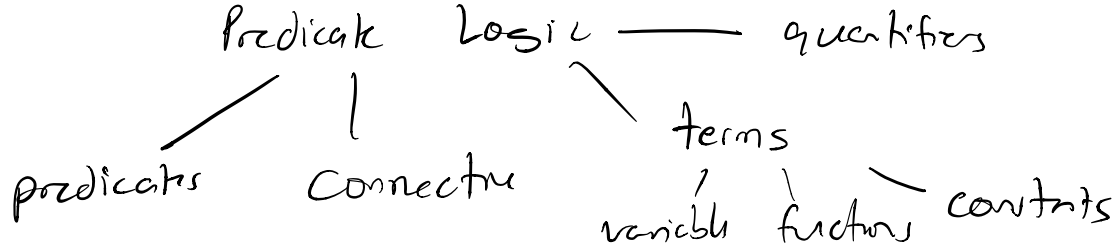
}

- Prove valid arguments: show conjunction of premise & negation of conclusion is inconsistant
  - Ex:// $p, p \Leftrightarrow q \vdash q$



PREDICATE LOGIC: BUILDING BLOCKS

- Cons of prop. logic: unable to quantify & show relations between prop.



- Predicates: T/F valuation of relationship between 2 objects
  - Ex: // child ("Billy"), likes-ice-cream ("Anna")
  - This is a n-ary predicate (2, 3, 4, ... n objects)
- Constants: values in a domain
- Variables: placeholder for a constant
  - Ex: // between (x, y, z) ⇒ x = "Waterloo" z = "Ottawa" y = "Toronto"
- Quantifiers:
  - Universal quantifier: $\forall x$ (forall) in a domain
  - Existential quantifier: $\exists x$ (there exists) in a domain

- Ex:// 1. Everything likes Fridays.
  ① Define our predicates
  likes_fridays(x) = x likes Fridays
  ② Formalize using quantifiers.
  $\forall x \cdot$ likes_fridays(x)
- Two predicates that need to be true in a quantifier:
  $\forall x \cdot a(x) \Rightarrow b(x)$, $\exists x \cdot a(x) \wedge b(x)$
  ↳ Implicitly has conjunction ($x_1, x_2 \ldots \in$ domain)
  ↳ Need to introduce conjunction

- Functions: return attribute of object ∉ Bool
  - Ex:// age(y) ⇒ 20, location("Taj Mahal") ⇒ "India"
  - Combine functions + predicates!
    - Ex:// James is born north of Montreal
      north_of(birthplace(James), Montreal)

For all x, if input has value x then output is x^2
$\forall x \cdot (\text{value(input)} == x) \Rightarrow (\text{value(output)} == x^2)$

PREDICATE LOGIC: SYNTAX

Predicate logic
- Untyped
- Typed } Python vs. C++

- Untyped:
  - Alphabet: all possible symbols
  - Terms: constants, variables, function of terms
  - Formulas: have truth values

- T/F (atomic wff), predicates, connectives, quantifiers
  - ↳ accept only terms.

- Ex:// Predicates: P, q. Variables: x, y, z. Functions: k, g, h. constants c, d

  ① $\forall x \cdot \exists y \cdot \underbrace{p(c, d)}_{\text{formula}} \Rightarrow$ Formula ✓

  ② $\exists x \cdot \underbrace{k(x, y, g(x))}_{\text{Term}} \Rightarrow$ Not wff

  ③ $\forall x \cdot q(\underbrace{p(x)}_{\text{Formula}}) \Rightarrow$ Not wff (predicates cannot take predicates)

- Formalization:
  - Ex:// ① Everything doesn't like something

    $\forall x \cdot \exists y \cdot \neg like(x, y)$

    ② Nothing likes everything

    $\neg(\exists x \cdot \forall y \cdot likes(x, y))$

  - Important "De-Morgan" formulas:

    $\neg(\forall x \cdot P(x)) \iff \exists x \cdot \neg P(x)$

    $\neg(\exists x \cdot P(x)) \iff \forall x \cdot \neg P(x)$

  - Conjunctions:
    - $\exists$: $\exists x \cdot P(x) \wedge G(x)$
    - $\forall$: $\forall x \cdot P(x) \Rightarrow G(x)$

  - Ex:// ① All bicycles in garage

    * $\forall x \cdot b(x) \Rightarrow g(x)$

    ② Some bicycles in garage

    * $\exists x \cdot b(x) \wedge g(x)$

    ③ Everything is either a bicycle or a garage?

    $\forall x \cdot b(x) \vee g(x)$

④ Something

$$\exists x \cdot b(x) \wedge s(x)$$

- Binding: variables are tied to quantifiers ⇒ order matter.

$$\forall x \cdot \exists y \cdot \text{likes}(x, y) \neq \forall x, \exists y \cdot \text{likes}(y, x)$$

- Scoping: quantifiers sometimes only applies to part of formula

$$(\forall x \cdot G(x)) \wedge P(x) \neq \forall x \cdot G(x) \wedge P(x)$$

- Bound variable: falls in quantifier scope (closest left quantifier)
  - If 2 variables bound to same quantifier ⇒ same value.
  - Free variable: not bound (closed WFF has no free vars)
  - Ex:// ① $\forall x \cdot (\exists y \cdot p(x, y)) \wedge (\exists y \cdot q(y))$
    1. Is this well formed? ✓
    2. Are there any free variables? ✓

    ② $\forall x \cdot \exists y \cdot p(x, y) \wedge (\exists x \cdot p(x, y))$
    1. Well-formed? ✓
    2. Free variables? ✓
- Formalizing English: no rules!

  Connectives → quantifiers → constants → functions → predicates
  - Ex:// All rich actors collect valuables
    a) $\forall x \cdot \text{rich\_actor}(x) \Rightarrow \text{collect\_valuables}(x)$
    b) $\forall x \cdot \text{rich}(x) \wedge \text{actor}(x) \Rightarrow \exists y \cdot \text{collects}(x, y) \wedge \text{valuable}(y)$

No unique formalization!

- Typed: putting limit on domain or return values

quantifier var : type . . . .

- Special type: Bool (T/F)
- Well-typed formulas: correct typed arguments for functions
  - Well-formed formula requires well-typed formula
  - Ex// $c: T_1$, $f: T_1 \to T_2$, $g: T_2 \to T_3$, $p: T_1 \to Bool$, $q: T_3 \times T_1 \to Bool$ (2 diff. types)
    $x: T_1$, $y: T_3$

    ① $\forall x \cdot \exists y \cdot q(f(x), y)$ ×
    ↳ Not returning T_1 / T_3

    ② $\exists x \cdot p(f(x)) \Rightarrow \forall y \cdot q(y, g(f(c)))$ ×
    ↳ Not returning T_1
- Type inference: find general maximum typing s.t. well-typed
  - Ex// ① $\exists x, y \cdot p(x, y) \wedge p(f(x), y)$

    $x: T_1$ $\quad p: T_1 \times T_2 \to Bool$

    $y: T_2$ $\quad f: T_1 \to T_1$

    ② $\exists x, y \cdot q(f(y)) \wedge q(g(x))$

    $x: T_1$ $\quad q: T_3 \to Bool$ $\quad g: T_1 \to T_3$

    $y: T_2$ $\quad f: T_2 \to T_3$
  - Strategy:
    1. Note down everything you need to type
    2. Give different types to variables
    3. Type the terms/predicates that IMMEDIATELY use variables.
    4. If no restriction on types ⇒ give it a new type
    5. Work until typed everything
- Typing ⇒ predicates: $\forall x : T_1 \cdot P(x) \equiv \forall x \cdot T_1(x) \Rightarrow P(x)$

- Can combine quantifiers:

  $\forall x : Q, \forall y : Q \Rightarrow \forall x, y : Q \cdot \ldots$

  $\forall x : Q, \forall y : P \Rightarrow \forall x : Q, y : P \cdot \ldots$

PREDICATE LOGIC: SEMANTICS

Meaning of predicate logic

Domain Interpretation Quantifiers

- Domain: set of all values of interest ($\{...\}$, $\mathbb{N}$, $\mathbb{R}$)
- Interpretation: non-empty domain + mappings
  - Mapping: for all domain values, how does each func./predicate work?

Syntax:	Meaning
$P(\cdot)$	$P(D_1) := T$
	$P(D_2) := F$
$g(\cdot,\cdot)$	

(# of dots = # of args!) (Arbitrary!)

- Quantifier meaning:
  - Universal: formula is T for all domain values (conjunction)

$$[\forall x \cdot P(x)] \Rightarrow [P(d_1)] \text{ AND } [P(d_2)] \text{ AND } \ldots$$

  - Existential: formula is T for at least 1 domain value (disjunction)

$$[\exists x \cdot P(x)] \Rightarrow [P(d_1)] \text{ OR } [P(d_2)] \text{ OR } \ldots$$

- Determine meaning of predicate formula:
  1. Establish domain
  2. Establish mapping for all domain values
  3. Substitute domain values into formula

- Ex: // $\exists x \cdot b(x) \Rightarrow g(x)$

① Domain: $\{Trek, RM, AO\}$

② Mapping:

Syntax	Meaning
$b(\cdot)$	bike(Trek) := T
	bike(RM) := T
	bike(AO) := F
$g(\cdot)$	garage(Trek) := T
	" (RM) := F
	" (AO) := F

② Substitute:

$$[\exists x \cdot b(x) \Rightarrow g(x)] = [b(\hat{}Trek) \Rightarrow g(\hat{}Trek)] \text{ OR } [b(\hat{}RM) \Rightarrow g(\hat{}RM)] \text{ OR}$$

(Allows us to use domain values even if still in syntax!)

Removed quantifier { [b[^A0] ⇒ g[^A0]]

Substitute meaning ⇒ = (bike[Tak) IMP garage[Tak)) OR ...

Evaluate ⇒ = T

- Showing counter examples: construct interpretation where premises = T, conc. = F
  - Tip: think about HOW premise/conc. evaluate before thinking of interpretation
- Typed predicate logic: domain for each type
  - Ex:// $c: T_1$, $d: T_2$. Show following is satisfiable: $q(c,d) \wedge \exists z: T_1 . \neg(z=c) \wedge q(z,d)$

1. Domain:

   T_1: $\{v_1, v_2\}$

   T_2: $\{w_1\}$

2. Mapping:

Syntax	Meaning
c	v_1
d	w_1
$q(\cdot,\cdot)$	$Q(v_1, w_1) := T$
	$Q(v_2, w_1) := T$

3. Substitution:

[formula] = [q(c,d)] AND (([¬^v1 = c] AND [q(^v1, d)]) OR ([¬^v2 = c] AND [q(^v2, d)]))

= ...

= T

PREDICATE LOGIC: NATURAL DEDUCTION

New Rules

∃-intro, ∀-intro, ∃-elim., ∀-elim.

- Genuine vs. unknown variables
  - Genuine variables (x_g): universal quant of variables makes formula true
  - Unknown variable (x_u): existential

- ∀-elimination:

$$\frac{\forall x . P(x)}{}$$

$P[t/x]$ ⇒ t is substituted for x

- No variable capture: substitutions have to be variables not already bound
- Ex:// $p(c), \forall x \cdot p(x) \Rightarrow \neg(q(x)) \vdash \neg q(c)$

  1) $p(c)$ premise
  2) $\forall x \cdot p(x) \Rightarrow \neg(q(x))$ premise
  3) $p(c) \Rightarrow \neg(q(c))$ forall-e on 2
  4) $\neg(q(c))$ by imp-e on 1, 3

- Ex:// $(\forall y \cdot \exists x \cdot q(x,y,z)) \wedge r(y)$ — free!

  ① $P[x/y]$: $(\forall y \cdot \exists x \cdot q(x,y,z)) \wedge r(x)$ ✓

  ② $P[x/z]$: $(\forall y \cdot \exists x \cdot q(x,y,\boxed{x})) \wedge r(y)$ ✗
  ↳ Already bound!

- Tip: use this last to get to conclusion

- ∃- introduction:

$$\frac{P(t)}{\exists x \cdot P(x)}$$

t ⇐ witness, t is not bound

⇒ x should not be bound

- Ex:// $\forall x \cdot a(x) \wedge b(x) \vdash \exists x \cdot a(x)$

  1) $\forall x \cdot a(x) \wedge b(x)$ premise
  2) $a(c) \wedge b(c)$ by forall-e on 1 ⇒ ∀ is outermost, cannot do and-e yet!
  3) $a(c)$ by and-e on 2
  4) $\exists x \cdot a(x)$ by exists-i on 3

- ∀- introduction:

for every x_0 {
  ⋮ ↳ x_0 must be a new variable
  $P[x_0/x]$ ⇒ Unconstrained + not bound to x
}

$$\frac{}{\forall x \cdot P(x)}$$

- ∃- elimination:

$\exists x \cdot P(x)$
for some x_U $P[x_U/x]$ { ← x_U is an unused variable
  ⋮
  Q ⇒ Q must be indep. of x_U
}

$$\frac{}{}$$

- Trick: deal w/ exists into/elimination before ∀ into/elimination ⇒ introduce the unknown variable in ∀ statements!!

- Ex:// $\forall x \cdot p(x) \Rightarrow q(x), \forall x \cdot p(x) \vdash \forall x \cdot q(x)$

1) $\forall x \cdot p(x) \Rightarrow q(x)$ premise
2) $\forall x \cdot p(x)$ premise
3) for every x_0 {
   4) $p(x_0) \Rightarrow q(x_0)$ by forall-e on 1
   5) $p(x_0)$ by forall-e on 2
   6) $q(x_0)$ by imp-e on 4,5
   }
7) $\forall x \cdot q(x)$ by forall-i on 3-6

- Ex:// $\forall x \cdot p(x,x) \vdash \forall x \cdot \exists y \cdot p(x,y)$

1) $\forall x \cdot p(x,x)$ premise
2) for every x_0 {
   3) $p(x_0,x_0)$ by forall-e on 1
   4) $\exists y \cdot p(x_0,y)$ by exist-i on 3
   }
5) $\forall x \cdot \exists y \cdot p(x,y)$ by forall-i on 2-4

- Ex:// $\exists x \cdot p(x) \vdash \neg(\forall x \cdot \neg p(x))$

1) $\exists x \cdot p(x)$ premise
2) disprove $\forall x \cdot \neg p(x)$ {
   3) for some x_0 $p(x_0)$ {
      4) $\neg p(x_0)$ by forall-e on 2
      5) false by not-e on 3,4
      }
   6) false by exists-e on 1, 3-5
   }
7) $\neg(\forall x \cdot \neg p(x))$ by raa on 2-6

- Ex:// $\neg(\forall x \cdot p(x)) \vdash \exists x \cdot \neg p(x)$

1) $\neg(\forall x \cdot p(x))$ premise ⇒ If premises negated, good indication to use raa + prove contradiction using negated conclusion
2) disprove $\neg(\exists x \cdot \neg p(x))$ {
   3) for every x_0 {
      4) disprove $\neg(p(x_0))$ {
         5) $\exists x \cdot \neg p(x)$
         6) false by not-e on 2,5
         }
      7) $p(x_0)$
      }

} Needed to contradict conclusion!

8) $\forall x \cdot p(x)$ by forall-i on 3-7

9) false by not-e 1, 8

}

10) $\exists x \cdot \neg p(x)$ by raa on 2-9

- Type has no impact on proofs

PREDICATE LOGIC: SEMANTIC TABLEAUX

New Rules

forall-nb exists-nb

- Universal quantification (forall-nb):

$$\forall x \cdot P(x)$$
$$\mid \text{ forall-nb}$$
$$P(t) \Rightarrow t \text{ is free in } x$$

- Existential quantification (exists-nb):

$$\exists x \cdot P(x)$$
$$\mid \text{ exists-nb}$$
$$P(x_0) \Rightarrow \text{new variable (free)}$$

- Neg. universal quantification:

$$\neg(\forall x \cdot P(x))$$
$$\mid \text{ not-forall-nb}$$
$$\exists x \cdot \neg P(x)$$

- Neg. existential quantification:

$$\neg(\exists x \cdot P(x))$$
$$\mid \text{ not-exists-nb}$$
$$\forall x \cdot \neg P(x)$$

- Ex:// $\forall x \cdot p(x) \Rightarrow q(x), \forall x \cdot p(x) \vdash \forall x \cdot q(x)$

1) $\forall x \cdot p(x) \Rightarrow q(x)$

2) $\forall x \cdot p(x)$

3) $\neg(\forall x \cdot q(x))$

| not-forall-nb on 3

4) $\exists x \cdot \neg q(x)$

| exists-nb on 4

5) $\neg q(x_0)$

| forall-nb on 2

6) $p(x_0)$

| forall-nb on 1

7) $p(x_0) \Rightarrow q(x_0)$

imp-br on 7

8) $\neg p(x_0)$

Closed on 6, 8

9) $q(x_0)$

Closed on 5, 9

- Ex:// $\forall x \cdot p(x, c) \vdash \exists x \cdot p(g(x), x)$

1) $\forall x \cdot p(x, c)$

2) $\neg(\exists x \cdot p(g(x), x))$

| not-exists-nb on 2

3) $\forall x \cdot \neg p(g(x), x)$

| forall-nb on 3

4) $\neg(p(g(c), c)$

| forall-nb on 1

5) $p(g(c), c)$

closed on 4, 5

← Introduce more complex stuff first!

⇓

Use forall to show contradiction

- Tip: deal with existential before universal

PREDICATE LOGIC: FORMALIZATION

- Q: how much detail should we formalize?
  - Depends on what you are trying to prove
  - Look at common phrases to turn into predicates
- Q: purpose of types?
  1. Limit domain of universal quantification for counterexamples
  2. Limit use of predicates/functions to catch errors
  - Introduce: ← Happen frequently ⇒ convert to type

$$\forall x \cdot \boxed{T(x)} \Rightarrow P(x) \equiv \forall x : T \cdot P(x)$$

$$\exists x \cdot \boxed{T(x)} \wedge P(x) \equiv \exists x : T \cdot P(x)$$

- Ex:// Premises:
  1. 2 parallel lines that are not identical do not intersect
  2. If 2 parallel lines, then no common points or all common pts
  3. Two lines w/ all points in common intersect

Conclusion:
Two parallel lines that are not identical have no points in common

Formalize this argument.

① Formalize conclusion

① Common predicates:

twoLines $(\cdot,\cdot)$, notIdentical $(\cdot,\cdot)$, intersect $(\cdot,\cdot)$

allPtsCommon $(\cdot,\cdot)$, noPtsCommon $(\cdot,\cdot)$

① Unique predicates: parallel (a,b)

$\therefore \forall x, y \cdot twoLines(x,y) \wedge parallel^{(x,y)} \wedge notIdentical(x,y) \Rightarrow noPtsCommon(x,y)$

② Formalize predicates

1. $\forall x,y \cdot twoParallelLines(x,y) \wedge notIdentical(x,y) \Rightarrow \neg intersect(x,y)$
2. $\forall x,y \cdot twoParallelLines(x,y) \Rightarrow noPtsCommon(x,y) \vee allPtsCommon(x,y)$
3. $\forall x,y \cdot \ldots$

HIDDEN PREMISES

- Hidden premise: unwritten information needed to complete proof
- Enthymeme: argument w/ hidden premise
- To identify hidden premises
  - (A) Formalize premises → prove → try to find hidden premise, based on where you get stuck in proof
  - (B) Look @ problem description + determine what should be true if conclusion is true.

- Ex:// Formalize + prove:
  - ① Crime committed by someone @ 6 pm
  - ② Billy was in jail at 6pm
  - ③ Billy did not commit the crime (conclusion)

1. Formalize
   - Common predicate: CC(suspect, location, time), locn(suspect, time)
   - ① $\exists x \cdot CC(x, BS, 6)$
   - ② locn(Billy, J, 6)

③ $\neg cc(Billy, GS, 6)$

2. Hidden premises

① If commit crime, then at location

$\forall x,y,z \cdot cc(x,y,z) \Rightarrow locn(x,y,z)$

② Cannot be at 2 locations at same time

$\forall x,y,z,w : locn(x,y,w) \wedge locn(x,y,z) \Rightarrow w = z$

Forall common in hidden premise

③ General store + jail are not same

$\neg (J = GS)$

3. Prove via natural deduction / semantic tableaux

- Tips on choosing hidden premises:
  1. Don't choose conclusion
  2. Valid w/ all interpretations
  3. Should not be sufficient to prove conclusion
  4. Often universally quantified

Intro To Theories

- Defn: collection of constants, predicates, functions all relate to same info
  - Axioms: fundamental building blocks of theories, given
- Reason: restrict usage of predicates for certain interpretation
  - Model of theory: interpretation where theory is true
  - Standard/normal interpretation: model we're aiming at
- Multiple theories can be used together
- To make a theory:
  1. Define consistent axioms
  2. Define standard interpretation

Theory Of Equality

- Syntax: '=' predicate
  - Only terms can be compared w/ equality
  - $a = b$ is atomic wff
  - Precedence & associativity don't matter (cannot do $a = b = c$)
  - $\neg (a = b)$ is $a \; != b$
- Equality is about replacing something w/ equivalent
- "Only" examples:
  - Ex:// ① Alice likes only bubblegum ice cream

$$likes(A, bg) \wedge \forall x . (A, x) \Rightarrow x = Bg$$

OR

$$likes(A, bg) \wedge \neg (\exists x . likes(A, x) \wedge \neg (x = Bg))$$

OR

$$\forall x . likes(A, x) \Leftrightarrow x = Bg$$

② Only Alice likes bubblegum ice cream

$$likes(A, bg) \wedge \forall x . (x, Bg) \Rightarrow x = A$$

③ The only kind of ice cream Alice likes is bubblegum

$$likes(A, Bg) \wedge \forall x . ic(x) \wedge likes(A, x) \Rightarrow x = Bg$$

OR

$$\forall x . ic(x) \wedge likes(A, x) \Leftrightarrow Bg = x$$

- Natural deduction:

$$\frac{}{t = t} \; eq\text{-}i \qquad \frac{t_1 = t_2 \quad P[t_2/x]}{P[t_1/x]} \; eq\text{-}e$$

- Properties / axioms:

① Symmetry: $\forall x, y.\ (x = y) \Rightarrow (y = x)$

② Transitivity: $\forall x, y, z.\ (x = y) \wedge (y = z) \Rightarrow x = z$

③ Leibniz's Law: $\forall x, y.\ x = y \Rightarrow P(x) = P(y)$

↳ Introduction of functions / predicates!

- Ex:// $\forall x.\ P(x) \Leftrightarrow (x = b) \vdash P(b) \wedge \forall x, y.\ P(x) \wedge P(y) \Rightarrow (x = y)$

1) $\forall x.\ P(x) \Leftrightarrow (x = b)$ premise

2) $P(b) \Leftrightarrow (b = b)$ by forall-e on 1

3) $b = b$ by eq-i

4) $P(b)$ by iff-mp on 2, 3

5) for every xs {

6) for every ys {

Need to quantify implication so used quantified variables

7) assume $P(xs) \wedge P(ys)$ {

8) $P(xs) \Leftrightarrow (xs = b)$ by forall-e on 1

9) $P(ys) \Leftrightarrow (ys = b)$ by forall-e on 1

10) $P(xs)$ by and-e on 7

11) $P(ys)$ by and-e on 7

12) $xs = b$ by iff-mp on 8

13) $ys = b$ by iff-mp on 9

14) $xs = ys$ by eq-e on 12, 13

} Good application of eq-e

}

15) $P(xs) \wedge P(ys) \Rightarrow xs = ys$ by imp-i on 7-14

}

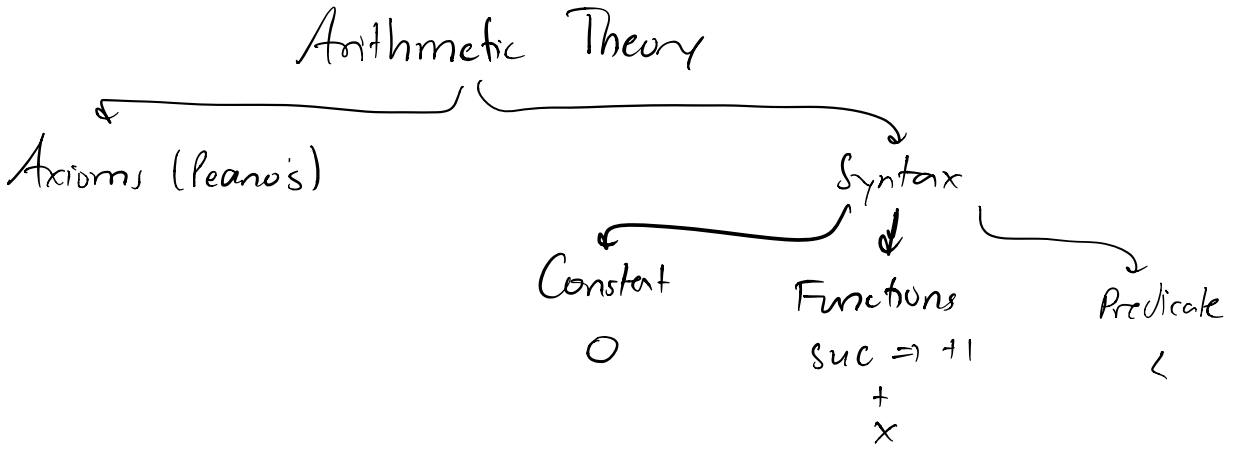
16) $\forall y.\ P(xs) \wedge P(y) \Rightarrow xs = y$ by forall-i on 6-15

}

17) $\forall x, y.\ P(x) \wedge P(y) \Rightarrow x = y$ by forall-i on 5-16

16) $P(b) \wedge \forall x,y \cdot P(x) \wedge P(y) \Rightarrow x = y$ by and-i on 4, 17

THEORY OF ARITHMETIC



- Peano's axioms:
  1. $\forall n \cdot \neg (suc(n) = 0)$ zero is not a successor
  2. $\forall x,y \cdot (suc(x) = suc(y)) \Rightarrow (x = y)$ distinct successors
  3. $\forall m \cdot m + 0 = m$, $\forall m \cdot 0 + m = m$ 0 is addition identity
  4. $\forall m,n \cdot m + suc(n) = suc(m + n)$ $m + (n+1) = (m+n)+1$
  5. $\forall m \cdot m \times 0 = 0$ multiplication by zero
  6. $\forall m,n \cdot m \times suc(n) = m \times n + m$ ⇒ recursive multiplication
  7. $(P(0) \wedge \forall k \cdot P(k) \Rightarrow P(k+1)) \Rightarrow \forall n \cdot P(n)$ induction

- Standard model of arithmetic

  Domain: $\mathbb{N}$

  Mappings:

Syntax	Meaning
0	0
$suc(\cdot)$	$Suc(x) := x + 1$
$\cdot + \cdot$	$Plus(x,y) := x + y$
$\cdot \times \cdot$	$Times(x,y) := x \times y$
$\cdot < \cdot$	$LessThan(x,y) := x < y$

- Induction:
  - Condition: finite + ordered set (ℕ)
  - Framework:

```
bc) P(0)  ⇒ base case
    :
ih) inductionstep P(k_S) {  ⇒ inductive hypothesis
        :
    x) P(k_S + 1)           } inductive step
    }
zt) ∀n. P(n) by induction on bc, ih - x
```

  - Tips:
    1. If you need arithmetic on a line: "by arith ∀ on ..."
       ↳ If $n_1 \neq n_2$ ⇒ by arith on ...
    2. If base case ≠ 0
       a) Still prove P(0)

```
disprove 0 ≥ n {
    ¬(0 ≥ n)
        P(0) by not.e
```

       b) Prove base cases up to n

```
bc) P(0)
    :
ih) induction step  k_S ≥ n ⇒ P(k_S) {
        assume k_S + 1 ≥ n {
            k_S = n - 1  ∨  k_S ≥ n  by arith
            case k_S = n - 1 {
                :
```

P(ks+1) }

Can ks ≥ n $

⋮

P(ks+1) }

P(ks+1) by cases

- Recursive functions:
  - Defined on ordered domain (ℕ)
  - Can use induction w/ recursive functions but recursive cond. proven
- Ex:// $f(0)=0,\ f(1)=1,\ f(n)=f(n-1)+f(n-2)$ for $n \geq 2$

⊢ $\forall n : \mathbb{N} \cdot \exists y : \mathbb{N} \cdot f(3 \times n) = 2 \times y$

P(0)

1) $f(3\times 0) = f(0)$ by arith
2) $f(3\times 0) = 0$ by arith w/ defn of fib
3) $f(3\times 0) = 2\times 0$ by arith on 2

b0) $\exists y : \mathbb{N} \cdot f(3\times 0) = 2\times y$ by exists_i on 3

ih) induction step $\exists y \cdot f(3\times n) = 2\times y$ {

4) for some $y_0 \cdot f(3\times n) = 2\times y_0$ {

will use later in proof, but showing that cond. upholds!

5) $3\times n + 3 \geq 2$ by arith
6) $3\times n + 2 \geq 2$ by arith

induction work

7) $f(3\times(n+1)) = f(3\times n + 3)$ by arith
8) $= f(3\times n + 2) + f(3\times n + 1)$ by arith w/ fib defn
9) $= f(3\times n + 1) + f(3\times n) + f(3\times n + 1)$ by arith w/ fib defn
10) $= f(3\times n) + 2 \cdot f(3\times n + 1)$ by arith
11) $= 2\times y_0 + 2 \cdot f(3\times n + 1)$ by arith on 4
12) $f(3\times n + 1) = 2\times(y_0 + f(3\times n + 1))$ by arith on 11
13) $\exists y \cdot f(3\times(n+1)) = 2\times y$

induction step ⇒ }

14) $\exists y \cdot f(3 \times (n+1)) = 2xy$ by exists-e on 4-13

}

15) $\forall n \cdot \exists y \cdot f(3 \times n) = 2xy$ by induction on bc, ih - 14

- Counterexamples: just use standard model for counterexamples
  - Tip: If trying to disprove $\forall$, show one example of failure

$= \ldots \text{ AND } ((1 < 3) \text{ IMP } (3^2 < 1^2 + 5)) \text{ AND } \ldots$

$= F$

- Gödel's incompleteness theorem: axioms not sufficient to prove/disprove

THEORY OF SETS

$\{ obj1, obj2, \ldots, \{\}, \emptyset \}$

- To define a set:
  1. Set enumeration: $\{a_1, a_2, \ldots, a_n\}$
  2. Set comprehension: $\{x \mid x \in \mathbb{N}, x \leq 6\}$

     element | conditions
     - Condition (wff) is called characteristic predicate w/ variable being free
  3. Z notation:

$\{ \langle term \rangle . \langle signature \rangle \mid \langle wff \rangle \}$

term: funct, const., var. ↳ If variable: omit

signature: types; var1 : type1, var2 : type 2

  - Ex: // $\{x : \mathbb{N} \mid x \leq 6\}$

- Types: types = sets

$\forall x : B \cdot P(x) \Leftrightarrow \forall x \cdot x \in B \Rightarrow P(x)$

$\exists x : B \cdot P(x) \Leftrightarrow \exists x \cdot x \in B \wedge P(x)$

- Ex:// Formalization : just identify collections of objects

① Set of natural divisors of 10

Comprehension: $\{x \mid x \in \mathbb{N} \wedge \exists y \cdot y \in \mathbb{N} \wedge x \times y = 10\}$

Enumeration: $\{1, 2, 5, 10\}$

② Set of students attending lectures from BC

Comprehension: $\{x \mid Student(x) \wedge from(x, Bc) \wedge atLecture(x)\}$

③ Set of number double of some $\mathbb{N}$ between 5, 10

Z-notation: $\{2x \cdot x : \mathbb{N} \mid (5 < x) \wedge (x < 10)\}$

④ Age of students who attend Waterloo

Z-notation: $\{age(x) \cdot x : Student \mid attend(x, UW)\}$

- Set comprehension axioms:

① $x \in \{y : S \mid P(y)\} \Leftrightarrow x \in S \wedge P(x)$

② $x \in \{t(a, b, \ldots) \cdot a : S, b : R \mid P(a, b, \ldots)\}$ (x is part of function; predicate is true)

$\Leftrightarrow \exists a, b, \ldots \cdot a \in S \wedge b \in R \wedge \ldots \wedge x = t(a, b, \ldots) \wedge P(a, b \ldots)$

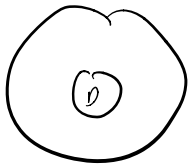
- Set equality: exact same elements

$$D = B \Leftrightarrow (\forall x \cdot x \in D \Leftrightarrow x \in B)$$

- Subsets: all elements of S1 in S2

$$D \subseteq B \Leftrightarrow (\forall x \cdot x \in D \Rightarrow x \in B)$$

- Proper subset: $D \subseteq B \wedge D \neq B$



$\Rightarrow D \subset B$

- Axioms:

③ Empty set is subset of every set

$$\vdash \emptyset \subseteq B$$

④ Every set is subset of itself

$$\vdash B \subseteq B$$

⑤ Subset is transitive

$$\vdash A \subseteq B \wedge B \subseteq C \Rightarrow A \subseteq C$$

⑥ Subset $\leftrightarrow$ equality

$$\vdash D = B \Leftrightarrow D \subseteq B \wedge B \subseteq D$$

- Size of set:

$$\#D \quad \text{or} \quad |D|$$

  - $|D| = 1 \Rightarrow$ singleton set

- Power set: set of all subsets

$$\mathbb{P}(D) = \{B \mid B \subseteq D\}$$

  - $\#(\mathbb{P}(D)) = 2^{\#D}$

- Proofs: use axioms in natural deduction via "by set/..."

- Ex:// Proofs

① $B \subseteq C, C \subseteq D \vdash B \subseteq D$

1) $B \subseteq C$ premise
2) $C \subseteq D$ premise
3) for every x_0 {
   4) assume $x_0 \in B$ {
      5) $\forall x. x \in B \Rightarrow x \in C$ by set
      6) $x_0 \in B \Rightarrow x_0 \in C$ by forall-e
      7) $\forall x. x \in C \Rightarrow x \in D$ by set
      8) $x_0 \in C \Rightarrow x_0 \in D$ by forall-e
      9) $x_0 \in C$ by imp-e on 4, 6

10) $x_9 \in D$ by imp-e on 7, 9

}

11) $x_9 \in B \Rightarrow x_9 \in D$ by imp-i on 9-10

}

12) $\forall x . x \in B \Rightarrow x \in D$ by forall-i on 3-11

13) $B \subseteq D$ by set

② $S \in \mathbb{P}(Q) \vdash \forall x . x \in S \Rightarrow x \in Q$

1) $S \in \mathbb{P}(Q)$ premise

2) $S \in \{B \mid B \in Q\}$ by set

3) $S \in Q$ by set

4) $\forall x . x \in S \Rightarrow x \in Q$ by set

- That vs. which:

① Cars that drive fast are dangerous

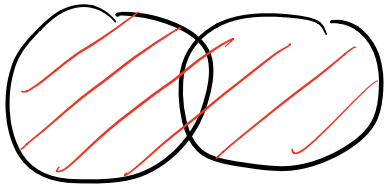
$$Cars \cap Fast \subseteq Dangerous$$

② Cars which drive fast are dangerous

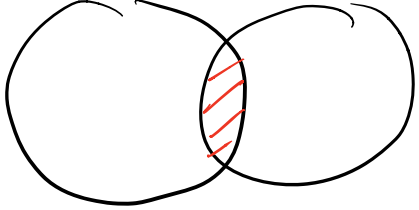
$$Cars \subseteq Dangerous \wedge Fast \subseteq Dangerous$$

SET FUNCTIONS, PREDICATES + PROOFS

- Functions:
  - Set union: $A \cup B = \{x \mid x \in A \vee x \in B\}$

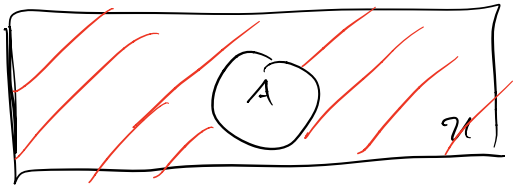


  - Set intersectn: $A \cap B : \{x \mid x \in A \wedge x \in B\}$

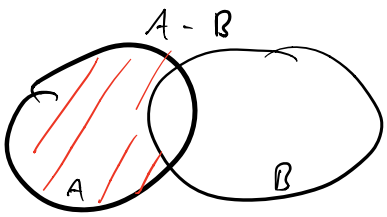


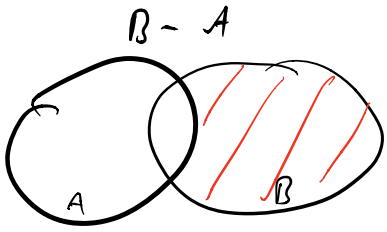
- Disjoint set: $A \cap B = \emptyset$
- Absolute complement: $A' = \{x \mid x \notin A \wedge x \in \mathcal{U}\}$

$= \{x \mid x \notin A\}$



- Set difference: $A - B = \{x \mid x \in A \wedge x \notin B\}$
  - $A - B \neq B - A$





- Multiple unions or intersections

$\bigcup \mathcal{J} = J_1 \cup J_2 \cup \ldots \cup J_n$ $\qquad$ $\bigcap \mathcal{J} = J_1 \cap J_2 \cap J_3 \ldots \cap J_n$

set of sets

- Formalization:
  1. Identify parts of sentences that are collections
  2. Use set functions to relate to another

- Ex:// (1) Canadian students who like hockey or volleyball

Canadian $\wedge$ Students $\wedge$ (like Hockey $\cup$ like Volleyball)

(2) No one from SK has a fav colour of blue

From SK $\wedge$ FavBlue $= \emptyset$

or FavBlue $\subseteq$ FromSk'

- Derived laws:
  - Commutative: $D \cap B = B \cap D$, $D \cup B = B \cup D$
  - Associative: $(D \cap B) \cap C = D \cap (B \cap C)$
  - Distributive: $(D \cap B) \cup C = (D \cup C) \cap (B \cup C)$
  - De Morgans: $(D \cup B)' = D' \cap B'$, $(D \cap B)' = D' \cup B'$
  - Empty:

$$D \cap \emptyset = \emptyset \qquad D - \emptyset = D \qquad \emptyset = \mathcal{U}'$$
$$D \cup \emptyset = D \qquad \emptyset - D = \emptyset$$

  - Universal:

$$D \cap \mathcal{U} = D \qquad D - \mathcal{U} = \emptyset \qquad \mathcal{U}' = \emptyset$$
$$D \cup \mathcal{U} = \mathcal{U} \qquad \mathcal{U} - D = D'$$

  - Intersection is a subset: $D \cap B \subseteq D/B$
  - Subset of a union: $D \subseteq D \cup B$
- Transformational proofs
  - 1st style: prove $A \Leftrightarrow B$ by showing $A \leftrightarrow B$
    - Use axioms + derived law
    - Ex: // $\vdash B = C \Leftrightarrow B \subseteq C \wedge C \subseteq B$

$$B = C \leftrightarrow \forall x. \in B \Leftrightarrow x \in C \quad \text{by set}$$
$$\leftrightarrow \forall x. (x \in B \Rightarrow x \in C) \wedge (x \in C \Rightarrow x \in B) \quad \text{by equiv}$$
$$\leftrightarrow (\forall x. x \in B \Rightarrow x \in C) \wedge (\forall x. x \in C \Rightarrow x \in B) \quad \text{by forall\_over\_and}$$
$$\leftrightarrow B \subseteq C \wedge C \subseteq B \quad \text{by set}$$

  - 2nd style: $A = B \Rightarrow A = B \leftrightarrow \text{true}$
    - Can either use set comp. or just use sets (faith)

- Ex:// $\vdash U - (B \cap U) = B'$

$U - (B \cap U) = B' \leftrightarrow U - (B \cap U) = U - B$ by set

$\leftrightarrow U - (B \cap U) = U - (B \cap U)$ by set

$\leftrightarrow$ true by set

- 3rd style: $A = B \Rightarrow x \in A \leftrightarrow x \in B$ (faster than 2nd)

- Ex:// $\vdash (B \cap C)' = B' \cup C'$

$x \in (B \cap C)' \leftrightarrow \neg (x \in B \cap C)$ by set

$\leftrightarrow \neg (x \in B \cap x \in C)$ by set

$\leftrightarrow \neg (x \in B) \cup \neg (x \in C)$ by DM

$\leftrightarrow x \in B' \cup x \in C'$ by set

$\leftrightarrow x \in B' \cup C'$ by set

- Counter examples: use predicate logic disproof

- Ex:// Disprove $\vdash G \cup (G \cup B) = G$

① Domain:

$U = \{c, d\}$ ⇒ universal domain

② Mappings:

G	$\{c\}$
B	$\{d\}$

⇒ Define individual set elements

③ Operations

$[G \cup (G \cup B)] = \{c\} \cup (\{c\} \cup \{d\}) = \{c\}$

$= \{c\} \cup \{c, d\} = \{c\}$

$= \{c, d\} = \{c\}$

$= F$

RELATIONS

- Tuple: 2 ≤ component sets

$(\ldots, \ldots, \ldots, \ldots)$

- Pairs: 2 comp., Triple: 3 comp.,
- Order matters:

$$(x, y) \neq (y, x)$$

- Cartesian product:

$$C \times B = \{(c, b) \mid c \in C \wedge b \in B\}$$

- Ex:// $C = \{1, 2, 3\}$

$B = \{4, 5\}$

$C \times B = \{(1,4), (1,5), (2,4), (2,5), (3,4), (3,5)\}$

- Relations: subset of Cartesian product
  - Type: $R : C \leftrightarrow B$, or $R : \mathbb{P}(C \times B)$
  - Binary relation: set of pairs
  - Ex:// own = {(Rima, Honda), (Joe, Honda), (Sarah, BMW)}

$\therefore$ own : People $\leftrightarrow$ Cars

- Relations $\leftrightarrow$ multi-arity predicates

$$P(a, b, c) \leftrightarrow (a, b, c) \in P$$

- Same functions + predicates for sets apply for relations
- Domain: set of all first elements of pairs

$$R : A \leftrightarrow B \Rightarrow dom(R) = \{x : A \mid \exists y : B \cdot (x, y) \in R\}$$

- Range: set of all second elements of pairs

$R : A \leftrightarrow B \Rightarrow \text{ran}(R) = \{y : B \mid \exists x : A \cdot (x, y) \in R\}$

- Formulas:
  - Ex: // $fix \subseteq owns \equiv F \Rightarrow$ Everyone fixes car owns the car is false

    $dom(fix) \subseteq dom(fix - owns) \cup dom(fix) \equiv dom(fix - own)$

    ↳ Everyone who fixes a car fixes at least 1 car that they don't own

- Domain + range properties: $S : A \leftrightarrow B$, $T : A \leftrightarrow B$
  - Domain:
    1. $dom(S) \subseteq A$
    2. $dom(S \cup T) = dom(S) \cup dom(T)$
    3. $dom(S \cap T) = dom(S) \cap dom(T)$
  - Range:
    1. $ran(S) \subseteq B$
    2. $ran(S \cup T) = ran(S) \cup ran(T)$
    3. $ran(S \cap T) = ran(S) \cap ran(T)$

- Inverse relation: $R^\sim$ reverse order of elements of R

  $R^\sim = \{(b, a) \cdot a : A, b : B \mid (a, b) \in R\}$

  - Laws:
    1. $(R^\sim)^\sim = R$
    2. $dom(R^\sim) = ran(R)$
    3. $ran(R^\sim) = dom(R)$

- Identity relation: id (B) is pairing of every element of B w/ itself
  - $id(B) = \{(a, a) \mid a \in B\}$
  - Ex:// $S = \{a, b, c\}$

    $id(S) = \{(a,a), (b,b), (c,c)\}$
- Relational composition: same thing as functional composition

  $R: A \leftrightarrow B, \quad S: B \leftrightarrow C$

  $R ; S = \{(a,c) \cdot a \in A, c : C \mid \exists b \cdot (a,b) \in R \wedge (b,c) \in S\}$
  - Ex:// $R: \{(a,1), (b,2), (c,3)\}$

    $S: \{(1,z), (2,w)\}$

    $R ; S: \{(a,z), (b,w)\}$
  - Binding the sep between 2 relations!
  - Laws:

    1. $R ; (S ; T) = (R ; S) ; T$
    2. $(R ; S)^{\sim} = S^{\sim} ; R^{\sim}$
    3. $id(dom(R)) \subseteq R ; R^{\sim}$
    4. $id(ran(R)) \subseteq R^{\sim} ; R$
    5. $id(A) ; R = R$
    6. $R ; id(B) = R$

    $R: A \leftrightarrow B$
    $S: B \leftrightarrow C$
    $T: C \leftrightarrow D$
- Iteration: recursive relation

  Set $A \sim R: A \leftrightarrow A$.

  $R^{\circ} = id(A)$

$$R^n = R ; R^{n-1}$$

- Ex:// $D = \{S, L, W\}$

$R = \{(S, L), (L, W)\}$ $(R : A \leftrightarrow A)$

$R^0 = \{(S, S), (L, L), (W, W)\}$

$R^2 = R ; R^1$

$= R ; (R ; R^0)$

$= R ; (R ; id(D))$

$= R ; R$ ← $R ; id(A) = R$

$= \{(S, W)\}$

- Steps of solving iteration problem:
  1. Find R^0
  2. Apply recursive relation
  3. Simplify using derived laws / defn of composition
- Derived laws:
  1. $R^{m+n} = R^m ; R^n$
  2. $R^{mn} = (R^m)^n$

  } $R : X \leftrightarrow X$, $m, n : \mathbb{N}$, Use induction to prove

- Relational image:

A, B, C where $C \subseteq A$. $R : A \leftrightarrow B$.

$$R(|C|) = \{y : B \mid \exists x : A \cdot (x, y) \in R \wedge x \in C\}$$

- 2nd element whose 1st element is in C!
- Ex:// own = {(Rima, Honda), (Joe, Honda), (Sarah, BMW)}

$own(\{Rima\}) = \{Honda\}$

$own(\{Rima, Sarah\}) = \{Honda, BMW\}$

- Derived laws:
  1. $R(|A \cup B|) = R(|A|) \cup R(|B|)$
  2. $R(|A \cap B|) = R(|A|) \cap R(|B|)$
  3. $R(|dom(R)|) = ran(R)$

- Restrictions: create new binary relations from old

Restrictions

Restriction
- Retaining pairs whose elements satisfy set rule

Subtraction / Co-restriction
- Retaining pairs whose elements NOT satisfy set rule

- $R : A \leftrightarrow B$, $A1 \subseteq A$, $B1 \subseteq B$
  1. Domain restriction:

$$A1 \lhd R = \{(a,b) \cdot a : A, b : B \mid (a,b) \in R \wedge a \in A1\}$$

($A1$ → restriction)

  2. Domain subtraction:

$$A1 \lhd R : \{(a,b) \cdot a : A, b : B \mid (a,b) \in R \wedge a \notin A1\}$$

  3. Range restriction:

$$R \rhd B1 = \{(a,b) \cdot a : A, b : B \mid (a,b) \in R \wedge b \in B1\}$$

($B1$ → restriction)

  4. Range subtraction

$$R \rhd B1 = \{(a,b) \cdot a : A, b : B \mid (a,b) \in R \wedge b \notin B1\}$$

- Ex: // $R: \{(a,1),(b,2),(c,3)\}$

$\{a,b\} \lhd R = \{(a,1),(b,2)\}$

$\{a,b\} \lhd\!\!\!\!- R = \{(c,3)\}$

$R \rhd \{1,2\} = \{(a,1),(b,2)\}$

$R \rhd\!\!\!\!- \{1,2\} = \{(c,3)\}$

- Derived laws: $R: X \leftrightarrow Y$, $A, B: \mathbb{P}(X)$

1. $A \lhd (B \lhd R) = (A \cap B) \lhd R$
2. $A \lhd\!\!\!\!- (B \lhd\!\!\!\!- R) = (A \cup B) \lhd\!\!\!\!- R$
3. $(A \lhd R) \cup (A \lhd\!\!\!\!- R) = R$

- Relational overriding: updating relations

$R: A \leftrightarrow B$, $S: A \leftrightarrow B$

$R \oplus S = (\mathrm{dom}(S) \lhd\!\!\!\!- R) \cup S$

$= \{(a,b) \mid (a,b) \in R \wedge a \notin \mathrm{dom}(S) \vee (a,b) \in S\}$

- Ex: // $own = \{(Rima, Honda), (Joe, Honda), (Sarah, BMW)\}$

$S = \{(Rima, BMW)\}$

$own \oplus S = \{(Joe, Honda), (Sarah, BMW), (Rima, BMW)\}$

- Remove all pairs in original w/ domain = overriding relation → add overriding relation

- Formalization: no algorithm, just understand operations

- Ex: // owns, rents: People $\leftrightarrow$ Dwellings, Students: $\mathbb{P}$(People), Houses: $\mathbb{P}$(Dwellings)

① All houses that are rented are owned

1. What relations are we talking about?

rents, owns

2. What parts of relations are we discussing

Houses → talking about range

3. Restrictions

Dwellings must be Houses.

4. Relate:

$$ran(rents \rhd Houses) \subseteq ran(owned \rhd Houses)$$

OR

$$ran(rents \rhd Houses) \subseteq ran(owned)$$

OR

$$ran(rents) \cap Houses \subseteq ran(owned)$$

② Student that rents a house does not own dwellings

1. Relations: rents, owns
2. Parts of relations: domain
3. Restrictions: People → Students
4. Relate:

$$dom(students \lhd rents) \cap dom(own) = \emptyset$$

OR

$$dom(students \lhd rents) \lhd dom(own) = \emptyset$$

- Proof: use set theory proofs

Get definition → manipulate → rework definition

- Ex: // $R ; (S ; T) = (R ; S) ; T$

1. Determine style:

$$(a, d) \in R ; (S ; T) \leftrightarrow (a, d) \in (R ; S) ; T$$

2. Initial + end:

$$\exists b \cdot a, b \in R \wedge (b, d) \in S ; T$$

↳ $\exists c \cdot a,c \in R;S \wedge (c,d) \in T$

3. Link proof:

$(a,d) \in R;(S;T)$

$\hookleftarrow \exists b \cdot a,b \in R \wedge (b,d) \in S;T$ by set & rel. comp.

$\hookleftarrow \exists b \cdot (a,b) \in R \wedge \exists c \cdot (b,c) \in S \wedge (c,d) \in T$ by set

$\hookleftarrow \exists b \cdot \exists c \cdot (a,b) \in R \wedge (b,c) \in S \wedge (c,d) \in T$ by move_exists

$\hookleftarrow \exists c \cdot \exists b \cdot$ ||

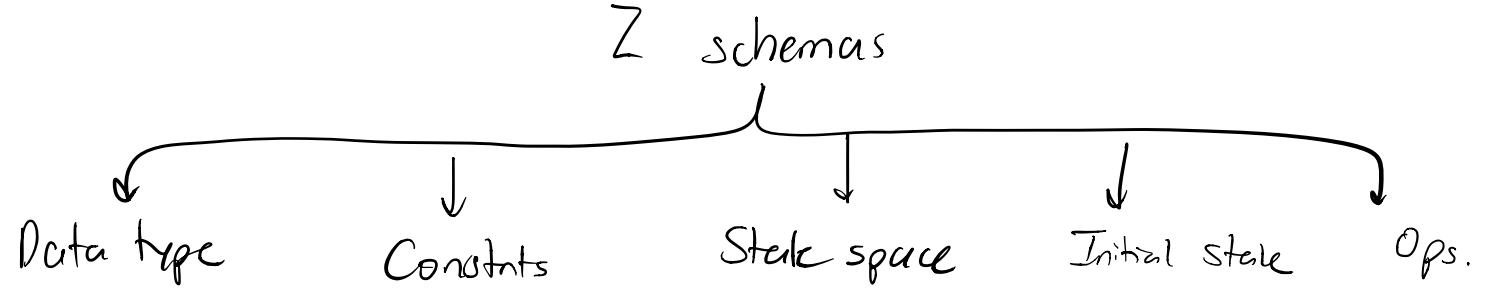
$\hookleftarrow \exists c \cdot (\exists b \cdot (a,b) \in R \wedge (b,c) \in S) \wedge (c,d) \in T$ by move_exists

$\hookleftarrow \exists c \cdot (a,c) \in R;S \wedge (c,d) \in T$ by set

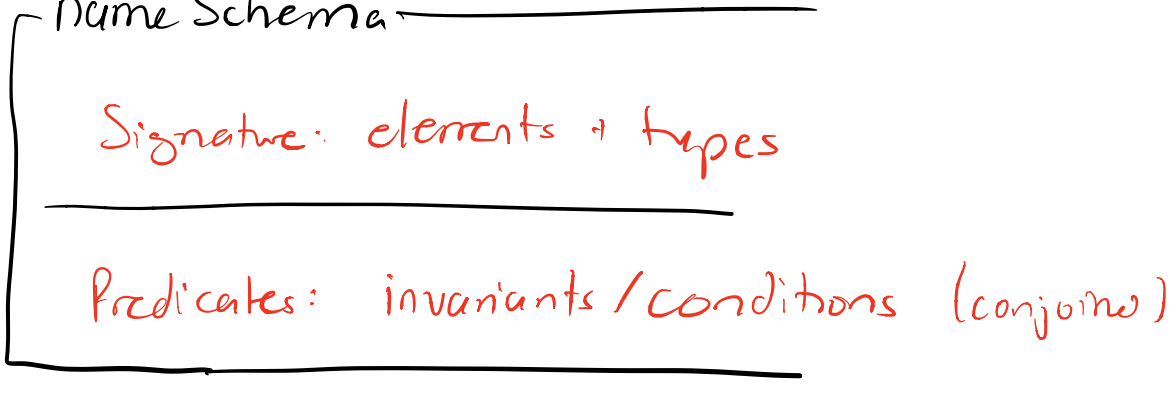
$\hookleftarrow (a,d) \in (R;S);T$ by set

Z SCHEMAS

- Formal specification: description of system in unambiguous forms



- Generally:



- State space: What are variables that we are working w/
  - Ex:

$x < y$

Schema Inclusion

Normal

Delta

Use this to break down schema

- Normal: including all states of schema in new one
  - Ex: //

My System

Initial System

⋮

- Delta: used to show state change (operations)

$var \longrightarrow var'$

old new

  - Δ schema: all states in "schema" are included + primed ver.
  - Ex: //

Operation

Δ MySystem

$x' = x + 1$
$y' = y + 1$ } Updating state

  - Operations: precondition + post condition predicates needed
    - Preconditions: must satisfy before conducting op
    - Post condition: what is the operation

- Make sure you are not violating prev. invariants

- I/O:

  a? : input a! : output

  - Local variable: not an I/O

- Constants:

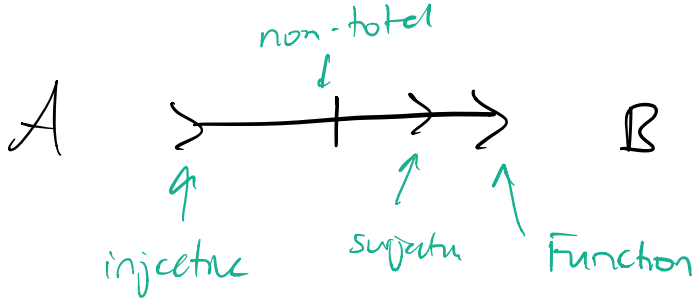
  - Create own constat schema + include
  - Include definiton + constraints
  - Ex:

Constants
MaxBooks : ℕ
100 ≤ MaxBooks

- Types:

  1. Built-in : ℕ, ℤ
  2. Generic: elements not defined. Flight, Person, ...
  3. Free : generic + values defined (Colors ::= Red | Blue | Green)
  4. Compound: relation

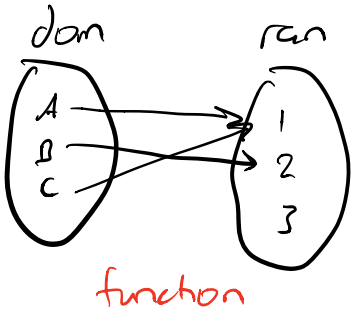
- Compound type:



  - Checklist for determining arrow:

1. Is this a function:

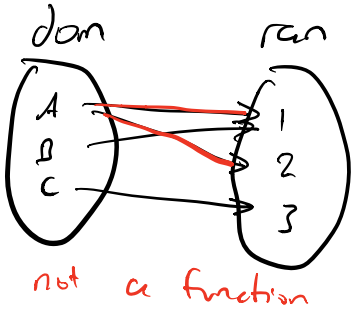
   1 element of domain maps to <u>at most</u> 1 element of range



function

⇓

→



not a function

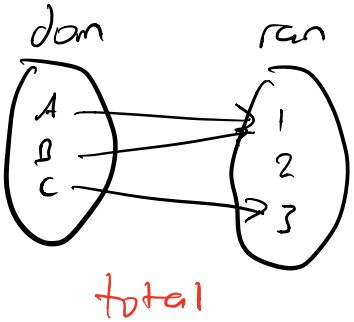
⇓

←→

Done

2. Is this total:

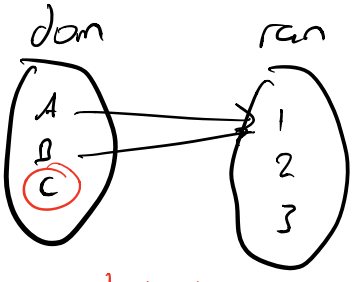
   Every element of domain maps to element in range.



total

⇓

Nothing



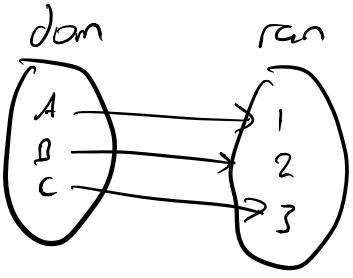
non-total

⇓

→

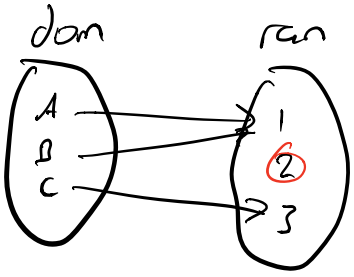
3. Is it surjective?

   Every element of range mapping to some element in domain



surjective

⇓



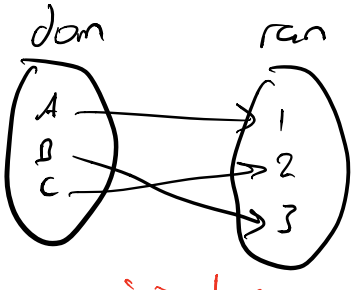
not surjective

⇓

Nothing

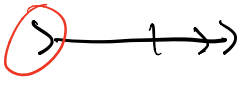
4. Is it injective?

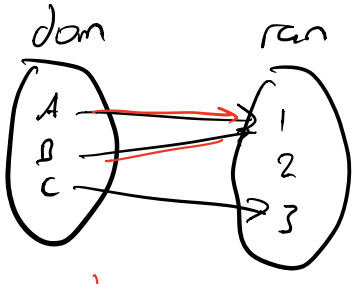
Is mapping between elements of dom + range unique?



injective

⇓





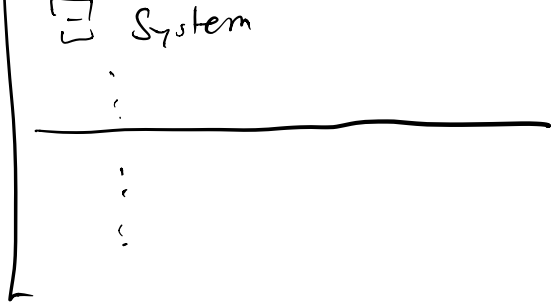
not injective

⇓

Nothing

- Partial function: both total/non-total (commonly assoc. w/ N.T.)
- If f is injective, then $f^{\sim}$ is injective
- Bijective: injective + surjective

- All other relation operations apply
- Use set comprehension for functions w/ infinite domain
- Schema: use states but no change to system (exception handling)
  - Ex:// Error A

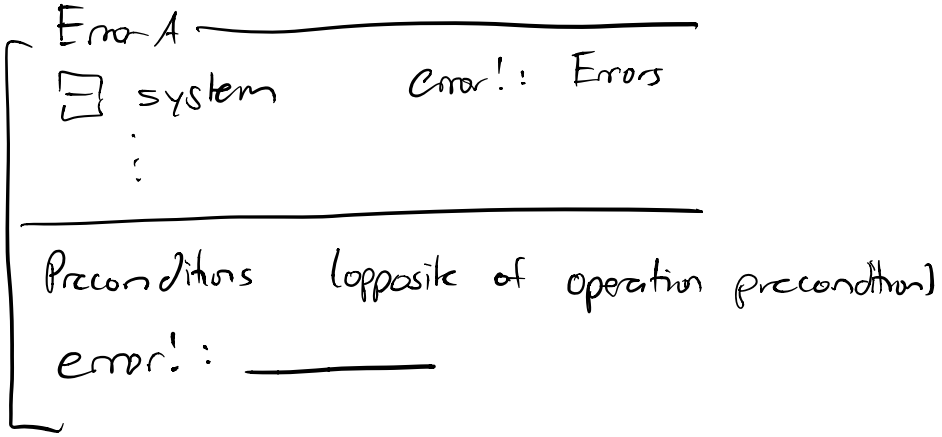


- Exception handling
  1. Define error types:

     Errors ::= ErrorA | ErrorB | . . .

  2. Define exception handling schema if preconditions in operations don't work

  3. Define exception

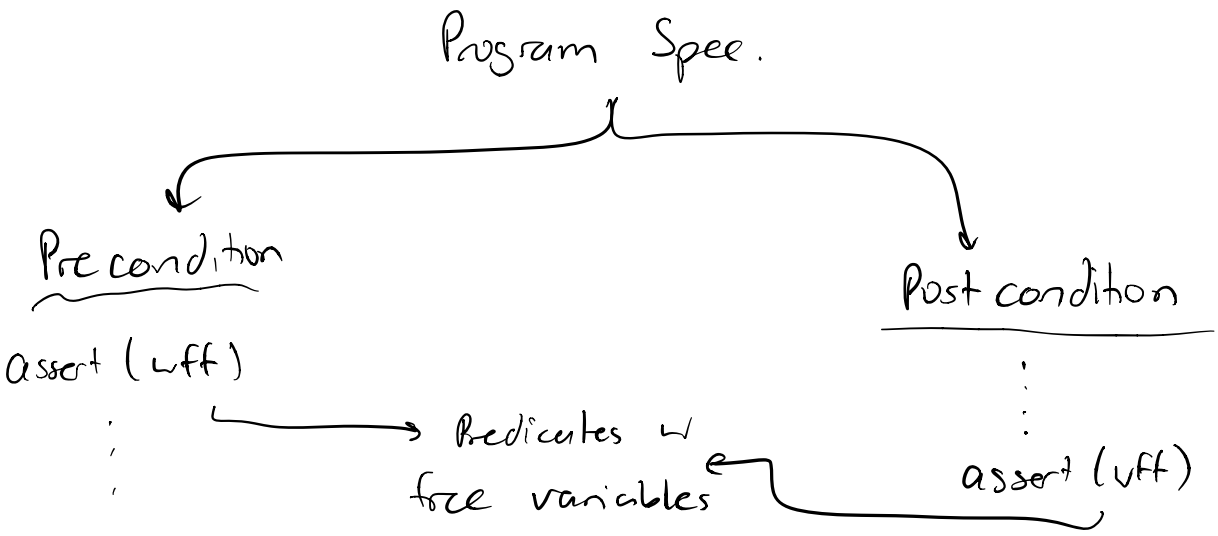


- Accessing parts of tuple:
  1. Tuple is a total function: $f(x) \Rightarrow$ second element of (x, y)
  2. " bijective: $(f^{\sim})(y) \Rightarrow$ first element of (x, y)
  3. " relation: set of second elemb for first element x
     ↳ $y \in R(\{x\})$

     set of first elements for second element y
     ↳ $R \rhd \{y\}$ or $R^{\sim}(\{y\})$

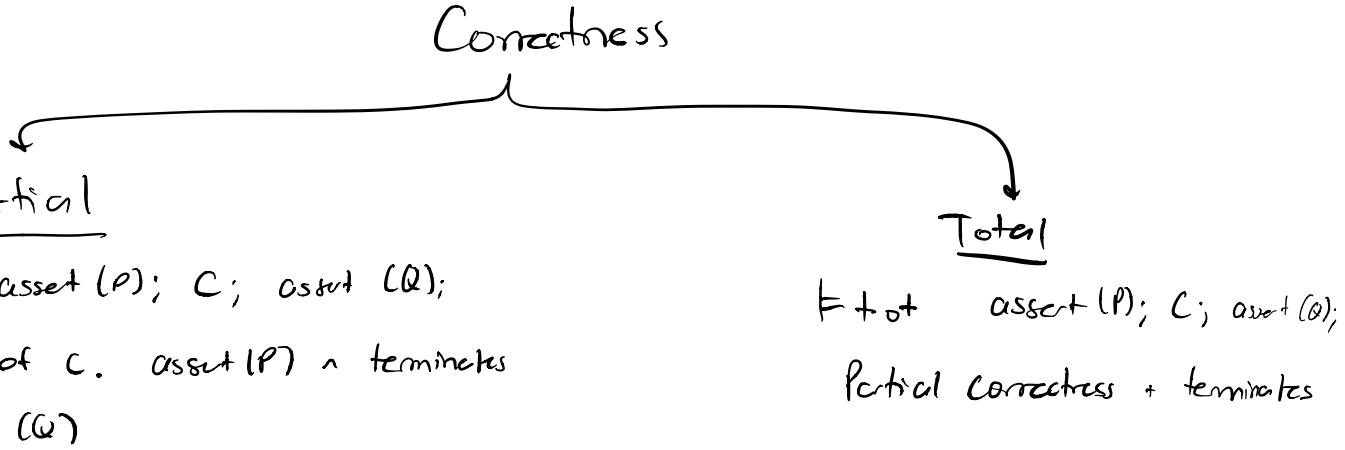
Introduction To Program Correctness

- Goal: find bugs + prove program will work
- Correct behaviour is defined by program specification



- Prove the following format:

assert (P);
C;
assert (Q);



- Ex: // assert (true);
  while true do {
    x := 0
  }
  assert (x > 0)

⇒ Does not meet spec because:
1. Does not terminates
2. End conditon will not pass even if terminates

∴ Not correct

- Logical variables: keeps track of initial states
  - Ex:// assert $(x = x_0 \wedge x > 0)$;
    C; $\Rightarrow$ x_0 does not appear in prog.
    assert $(y^2 < x_0)$;

- Ex:// What is precondition + post-condition s.t. partial correct.

1. y := 1
   While !(x == 0) do {
   y := y * x
   x := x - 1
   }

① Understand the program: fix value of variables.

x = 5,

Iter	X	Y	Z
0	5	1	
1	4	5	
2	3	20	
3	2	60	
4	1	120	
5	0	120	

$\Rightarrow$ $Y = x_0!$

② Write precondition + post condition s.t. always work.

Pre: assert $(x = x_0 \wedge x \geq 0)$

Post: assert $(y = x_0!)$

2. z := 0
   While (x > 0) {
   z = z + y;
   z := x - 1;
   }

① Understand program

Summing $1 \ldots x_0$

② Conditions.

Preconditions: assert $(x = x_0 \wedge x \geq 0)$

Postconditions: assert $\left(z = \frac{x_0 (x_0 + 1)}{2}\right)$

FLOYD-HOARE LOGIC

Assignment Rule

$$\frac{}{\text{assert}(P[E/Var]);\ Var := E;\ \text{assert}(P);}\ \text{Asn}$$

- If assertion has variable E, then E assigned to variable Var THEN assertion afterwards replaces E with Var
- Make sure that variables are free

- Proofs are done backwards ⇒ postcondition derives precondition

- Ex:// ①

```
assert (Y > 0);
X := Y;
assert (X > 0); %. Asn
```

↑ Replace LHS var w/ RHS expression

②

```
assert (2 > 0);
X := 2
assert (X > 0); %. Asn
```

③

```
assert (Z = X + Y + 1 - 4);
X := X + Y + 1;
assert (Z = X - 4); %. Asn
```

Composition Rule

$$\frac{\begin{array}{ll}\{P\}; & \{P\};\\ C_1; & C_2;\\ \{Q\}; & \{Q\};\end{array}}{\begin{array}{l}\{P\};\\ C_1;\\ C_2;\\ \{Q\};\end{array}}\ \text{Comp.}$$

- Ex:// Create a spec to swap X and Y + prove it.

① Create a spec.

$\{Y = Y_0 \wedge X = X_0\}$

$R := X$

$X := Y$

$Y := R$

$\{Y == X_0 \wedge X == Y_0\}$

② Proof:

$\{Y = Y_0 \wedge X = X_0\}$

$\{X == X_0 \wedge Y = Y_0\}$

Annotated Program: → $R := X$

$\{R == X_0 \wedge Y == Y_0\}$;. Asn

$X := Y$

$\{R == X_0 \wedge X == Y_0\}$;. Asn

$\{R == X_0 \wedge X == Y_0\}$

$Y := R$

$\{Y == X_0 \wedge X == Y_0\}$;. Asn

Every command has assertion

Chain

Combine through composition ⇒ Correct

- General proof guideline:

① Derive spec.

② From postcondition, crack precondition for immediately prev. command

③ Use composition implicitly to find precondition for next com.

- Ex://

```
assert (X ≥ 5 ∧ Y ≥ 0);
Z := X;
Z := Z + Y;
V := Z;
assert (V ≥ 5);
```

⇒

```
assert (X ≥ 5 ∧ Y ≥ 0);
assert (X + Y ≥ 5);
Z := X;
assert (Z + Y ≥ 5); Asn
Z := Z + Y
assert (Z ≥ 5); Asn
V := Z;
assert [V ≥ 5]; Asn
```

These are same

Implied Proof Rule

$$\frac{\begin{array}{l} P' \vdash P \\ \text{assert } (P); \\ C; \\ \text{assert } (Q); \\ Q \vdash Q' \end{array}}{\begin{array}{l} \text{assert } (P') \\ C; \\ \text{assert } (Q') \end{array}}$$

Verification proofs: must be proven seperately (natural deduction, trans. proofs)

- Use whenever we need to connect one assertion to another w/out program

- Ex://

```
assert (X ≥ 5 ∧ Y ≥ 0);
assert (X + Y ≥ 5); Implied (VC 1)
Z := X;
assert (Z + Y ≥ 5); Asn
Z := Z + Y
assert (Z ≥ 5); Asn
V := Z;
assert [V ≥ 5]; Asn
```

Verification ↓ 1

VC1:

$X \geq 5 \wedge Y \geq 0 \vdash X + Y \geq 5$

1) $X \geq 5 \wedge Y \geq 0$ premise
2) $X + Y \geq 5$ by arith

- Changing proof strength if using implied rule

- $A \vdash B \Rightarrow$ A is stronger than B
- $P' \vdash P \wedge$ assert $(P) \wedge \ldots \Rightarrow$ assert (P'): precondition strengthening
- $Q \vdash Q' \wedge \ldots$ assert $(Q) \Rightarrow$ assert (Q'): postcondition weakening

Conditional

- 1-armed conditional:

$$\frac{\text{assert}(P \wedge B);\ C;\ \text{assert}(Q);\quad P \wedge \neg B \vdash Q}{\text{assert}(P);\ \text{if } B \text{ then } C;\ \text{assert } Q}$$

```
assert (P);
if B then {
    assert (P ∧ B)
    C;
    assert (Q)
}
assert (Q)   If-then (VC 1)
```

- Ex: // assert (true)

```
if (max < B) then {
    assert (true ∧ max > B)   If-else
    assert (B >= B)   Implied (arith)
    max := B
    assert (max >= B)   Asn
}
assert (max >= B)   If-else rule (VC 1)
```

#1: Prove inner by bringing precondition + post-condition

#2: Prove inner via Asn / Impl.

#3: Verification condition.

VC1: $\text{true} \wedge \neg(\text{max} < B) \vdash \text{max} >= B$

- Two armed conditional:

```
assert (P);
if B then {
    assert (P ∧ B);   If-then-else
    C1;
    assert (Q)        Rule
} else {
    assert (P ∧ ¬B);  If-then-else
    C2;
    assert (Q);       Rule
}
assert (Q);           If-then-else
```

Use same steps as defined above.

- Ex://

```
assert (true)
assert (x+1 == x+1)   Implied (eq_i)
a := x+1
assert (a == x+1)     Asn
if (a-1 == 0) then {
    assert (a == x+1 ∧ a-1 == 0);   If-then-else
    assert (1 == x+1);              Implied (VC 1)
    y := 1
    assert (y == x+1);
} else {
    assert (a == x+1 ∧ ¬(a-1 == 0))   If-then-else
    assert (a == x+1);                Implied (land-e)
    y := a
    assert (y == x+1);                Asn
}
assert (y == x+1);                    If-then-else
```

Trick: be lazy + use following asserts as basis for condition

- Another method:

```
assert ((B ⇒ P1) ∧ (¬B ⇒ P2))
if B then {
    assert (P1);   If-then-else
```

Use same steps as defined above.

```
    C1;
    assert (Q);   Rule
} else {
    assert (P2);   If-then-else
    C2;
    assert (Q)   Rule
}
assert (Q)   If-then-else
```

Use this if P_1 is v. diff. than P_2

While Loop

- Loop invariant: true at beginning + end of loop
  - No unique invariant ⇒ make it simple
  - Look at behavior of loop & postcondition
  - Can't prove smth. w/ invariant ⇒ so stronger

- Rule:

```
assert (Inv);
while B do {
    assert (Inv ∧ B);   Partial-while
    C;
    assert (Inv);   Proof rule
}
assert (Inv ∧ ¬B);   Partial-while
```

- Ex://

```
assert (0 ≤ n ∧ a ≥ 0);
assert (1 == a^0 ∧ 0 ≤ n);   Implied (VC 1)
s := 1;
assert (s = a^0 ∧ 0 ≤ n);   Asn
i := 0;

assert (s = a^i ∧ i ≤ n);   Asn
```

① Identify invariant
   - Trace through loop
   - Look at post-condition

② Fill in partial while invariant

③ Perform inner proof (while)

④ Return outer proof

```
while (i < n) do {
    assert (s = a^i ∧ i ≤ n ∧ i < n);   Partial while
    assert (s * a = a^(i+1) ∧ i+1 ≤ n);   Implied (VC 2)
    s := s x a
    assert (s = a^(i+1) ∧ i+1 ≤ n);
    i := i + 1
    assert (s = a^i ∧ i ≤ n);   Asn
}
```

Inv — Neg. of cond.

assert $(s = a^i \wedge i \leq n \wedge \neg(i < n))$ Partial-while

assert $(s = a^n)$ Implied (VC 3)

Counter examples

- To show smth. does not work:
  1. Create initial states of all variables
  2. Construct final states by tracing through pros.
  3. Show that initial states satisfy precondition but final state does not satisfy postconditions.

- Ex: //

```
assert (true);
a := x + 1;
if (a - 1 == 0) {
    y := 1;
} else {
    y := a + 1;
}
assert (y = x + 1);
```

Initial state:

$a = ?$ $x = 1$ $y = ?$

Final state:

$a = 2$ $x = 1$ $y = 3$

Precondition: $[true]$

Postcondition: $[y = x + 1]$

$= 3 = 1 + 1$

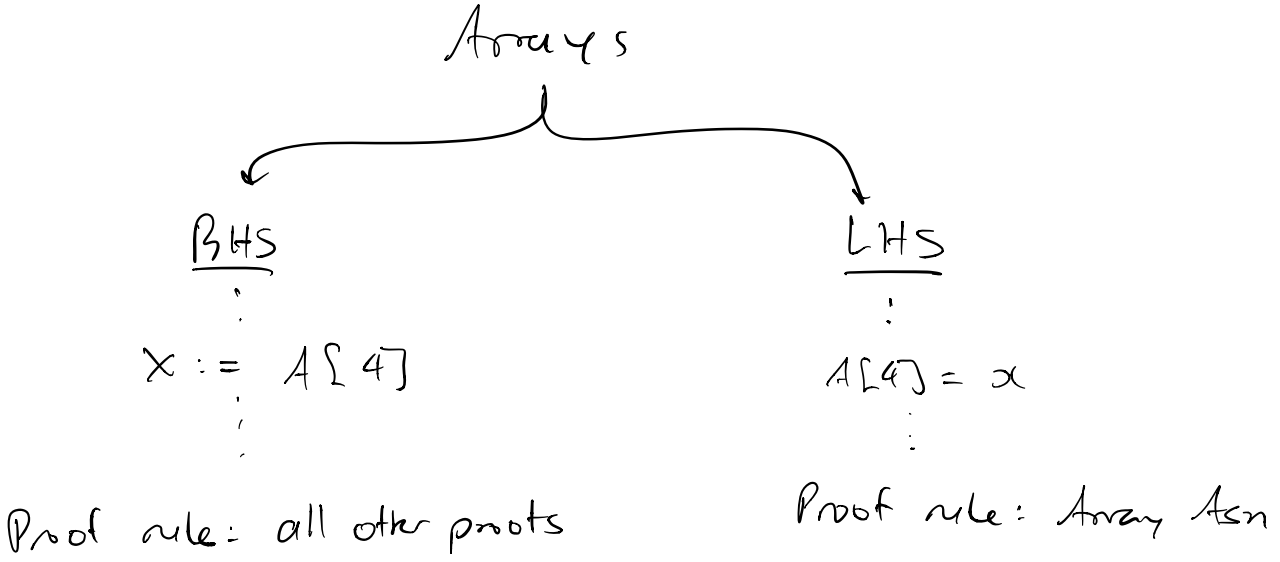
$= F$

Arrays

- Representation: $B[i]$

  ↳ Indices: $\{i : \mathbb{N} \mid 0 \leq i \leq size(B)\}$

↳ B: Indices → Value (mapping)



- Array assignment rule:

assert (P[B ⊕ {(i, e)}] / B])
B[i] := e
assert (P); Array Asn

↳ Relational override of index i with value e for all mentions of B in P

- Ex: // ①

assert ((B ⊕ {(k, 4)})[k] == c);
B[k] := 4;
assert (B[k] == c); Array Asn

②

assert ((A ⊕ {(k+1), c})[k] == d);
A[k+1] := c;
assert (A[k] = d); Array Asn

- Ex: //

assert (∀ i. 0 ≤ i ∧ i < n ⇒ B[i] = B0[i] , 0 < n);
assert (Inv[k := 0]);
k := 0;
assert (Inv); Asn
while (k < n-1) {
  assert (Inv ∧ k < n-1);

assert (Inv [B ⊕ {(k-1, B[k])}] / B', [k+1 / k]);
k := k+1;
assert (Inv [B ⊕ {(k-1, B[k])}] / B) Assn
B[k-1] := B[k]
assert (Inv); Array Asn

Inp.

}

assert (Inv ∧ ¬(k < n-1)) Partial while
assert (∀ i. 0 ≤ i ∧ i < n-1 ⇒ B[i] = B0[i+1]);

Inv: ∀ i. 0 ≤ i ∧ i ≤ k ⇒ B[i] = B0[i+1]
∧ ∀ i. k ≤ i ∧ i ≤ n ⇒ B[i] = B0[i]

- Special natural deduction: override

1. B ⊕ {(i, e)}[i] = e Override

2. x) i = j
x+1) B ⊕ {(j, e)}[j] = e Override on 3

3. x) ¬(i = j)
x+1) B ⊕ {(i, e)}[j] = B[j] override on 3

Functions + Procedures

Function

fun foo(...) {
assert (P);
⋮
assert (Q);
return val;
}

No propagations of changes ⇒

Procedures

proc bar(...) {
assert (R);
⋮
assert (Q);
}

Changes propagate

- Functions:

```
fun foo(...) {
    assert (R);
    :
    assert (S);
    return val;
}
```

```
assert (P);
:
assert (Q[f(a1, a2)/y])   Proof rule
y := f(a1, a2);
assert (Q);               Proof rule
:
```

- This will generate a function lemma:

  $(R \Rightarrow S)$ [param in func. / call params].

- Function proof steps:

  1. Prove body of function
  2. Conclude lemma
  3. Perform rest of proof w/ lemma

- Ex:

```
fun f(x, y) {
    assert (x ≥ 0);
    :
    assert (ret = x+y);
    return ret;
}
```

```
assert (b ≥ 0);
assert (f(b,c) ≥ b+c);   Implied (VC 1)
d := f(b, c);
assert (d ≥ b + c);      Asn
```

Replace return w/ function call

Function lemma: $b \geq 0 \Rightarrow f(b,c) = b+c$

- Procedures:

```
proc bar(...) {
    assert (R);
    :
    assert (Q);
}
```

```
:
assert (R [local params req. w/ called] ∧ H)
p(a1, a2);
assert (Q [local params req. w/ called params]
        ∧ H)   Procedure
:
```

- H: part of assertion w/ no variables changed by procedure.
- Replace logical variable in code like R, Q
- Steps:
  1. Prove procedure
  2. Use procedure rule ⇒ assertions
  3. Prove rest of program
- Ex:

```
proc inc (x) {
  assert (x = x0);
  :
  assert (x = x0 + 1);
}
```

```
assert (b = b0);
inc(b);
assert (b = b0 + 1);        Procedure
assert (b = b1 ∧ b1 = b0 + 1);   Impl.
inc (b);
assert (b = b1 + 1 ∧ b1 = b0 + 1);
assert (b = b0 + 2);        Impl.   Proc.
```

IMPORTANT: use multiple logical variables if chained procedures like above. Propagate post conditions as well.

Recursive functions + procedures

- To prove:
  1. Assume correct call correct ⇒ lemma if func.
  2. Prove the body of recursive

- Ex:

Pre: $\exists k.\ 0 \le k \wedge k \le i \wedge B[k] = c$

```
fun find (i) {
  assert (Pre);
  if (B[i] == c) {
```

```
                assert (B[i] == c ∧ Pre); If-then-else
Base case       assert (B[i] == c); Implies (And_e);
                ret := i
                assert (B[ret] == c); Asn
            } else {
                assert (!(B[i] == c) ∧ Pre); If-then-else
                assert (B[find(i-1)] == c); Impl.
⇒               ret := find(i-1);
                assert (B[ret] == c); Asn
            }
            assert (B[ret] == c); If-then-else
            return ret;
        }
```

Lemma: (by function rule)

$\exists k.\ 0 \le k \wedge k \le i-1 \wedge B[k] = c \Rightarrow B[find(i-1)] == c;$

→ Replaced local param i with called param i-1

- Ex:

```
proc add (x, y) {
    assert ((x = x0) ∧ (y ≥ 0));
    if (y != 0) {
        assert (y != 0 ∧ x = x0 ∧ y ≥ 0); If-then-else
        assert (x = x0 ∧ y-1 ≥ 0)
        add (x, y-1);
        assert (x = x0 + y - 1)          Proc.
        assert (x+1 = x0 + y); Implied (VC ___)
        x := x+1;
```

→ Don't NEED H. You could prove without.

assert ($x = x_0 + 5$); Asn

}

Base case ⇒ assert ($x = x_0 + y$); If-then (VC —)

Termination

Termination matter → Loops (variant / bounding exp.)

Termination matter → Recursive (bounding function)

- Showing loop termination:
  - Bounding expr:
    - ☑ Integer statement
    - ☑ Loop variables involved
    - ☑ Non-neg.
    - ☑ Decreases in iteration
    - ☑ Statement → 0 ⇒ guard is false
  - Ex: //

```
assert (n > 0);
sum := 0;
j := 0;
while (j != n) {
    sum := sum + a
    j := j + 1;
};
```

Bounding expression: $n - j$

Satisfies checklist

- Tip: look carefully @ loop behaviour → guard

- Showing recursive termination:
  - Show bounding function + base case

☑ Int. expr.
☑ Use arguments
☑ Non-neg.
☑ Decrease over time
} Bounding func.

☑ Section of code w/ no recursion + only enter if bounding func → 0 } Base case.

- Ex: //

```
proc add (x, y) {
    assert (x = x0 ∧ y ≥ 0);
    if (y != 0) then {
        add (x, y - 1);
        x := x + 1;
    }
    assert (x = x0 + y);
}
```

Bounding func. y

- Tip: look at conditions of recursion.

- Decidable problem: problem w/ yes/no answer.
  - Unsolvable prob.: no algo to solve it

- Prop. logic is decidable but not predicate logic (as a whole)

- Program correctness is undecidable
- Halting problem: no also. to determine if program will halt.