System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

A Logic Primer
(DRAFT)

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

Logic

The α and ω in science.

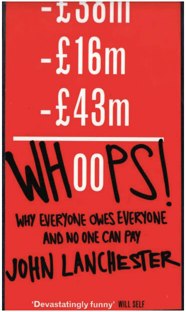
- Basis of mathematics.
- Also of engineering.
  - Particularly useful for verification (model-checking = checking a model against a logical formula).
  - But also used in other domains, e.g.: Prolog, Datalog, UML OCL (Object Constraint Language), ...

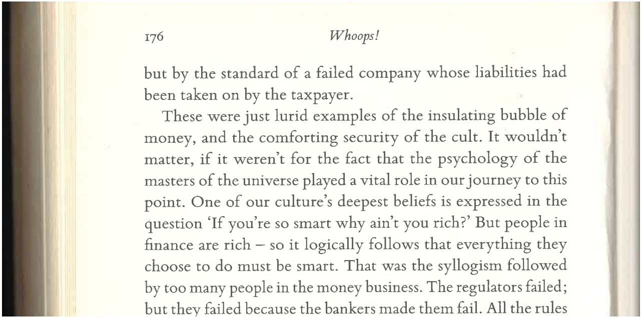
A myriad of logics:

- Propositional logic
- First-order logic
- Constructive logic
- Temporal logic
- ...

A fascinating history: read Logicomix [Doxiadis et al., 2009]!
The story is still evolving in our days!

Language and logic

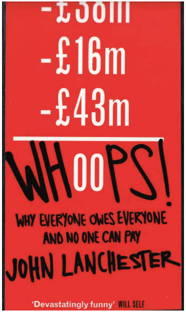


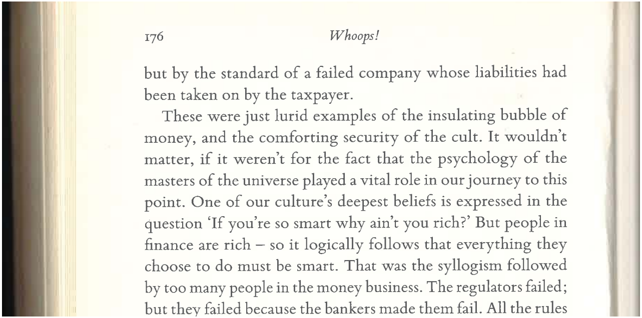

176 *Whoops!*

but by the standard of a failed company whose liabilities had been taken on by the taxpayer.

These were just lurid examples of the insulating bubble of money, and the comforting security of the cult. It wouldn't matter, if it weren't for the fact that the psychology of the masters of the universe played a vital role in our journey to this point. One of our culture's deepest beliefs is expressed in the question 'If you're so smart why ain't you rich?' But people in finance are rich – so it logically follows that everything they choose to do must be smart. That was the syllogism followed by too many people in the money business. The regulators failed; but they failed because the bankers made them fail. All the rules

Language and logic





176 *Whoops!*

but by the standard of a failed company whose liabilities had been taken on by the taxpayer.

These were just lurid examples of the insulating bubble of money, and the comforting security of the cult. It wouldn't matter, if it weren't for the fact that the psychology of the masters of the universe played a vital role in our journey to this point. One of our culture's deepest beliefs is expressed in the question 'If you're so smart why ain't you rich?' But people in finance are rich – so it logically follows that everything they choose to do must be smart. That was the syllogism followed by too many people in the money business. The regulators failed; but they failed because the bankers made them fail. All the rules

We may think that logic is "built into" our brains, but not really. Our brains often make logically incorrect deductions.

What is logic?

Logic = Syntax + Semantics + Proofs

Proofs

- Manual, or
- Automated: Proofs = Computations

Example:

- Syntax: boolean formulas
- Semantics: boolean functions
- Proofs: is a formula satisfiable? valid (a tautology)?
  - E.g., for boolean logic: an NP-complete problem (a representative for many combinatorial problems).

BOOLEAN LOGIC

(a.k.a. Propositional Logic or Propositional Calculus)

Syntax

Symbols:

- Constants: "false" and "true", or $0, 1$, or $\bot, \top$
- Variable symbols (*atomic propositions*): $p, q, ..., x, y, ...$
- Boolean connectives: $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (implies), $\equiv$ or $\leftrightarrow$ (is equivalent to)
- Parentheses $()$: used to make syntax unambiguous

Expressions (formulas):

$$
\begin{aligned}
\phi \quad ::= \quad & 0 \mid 1 \mid p \mid q \mid ... \mid x \mid y \mid ... \\
& \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
& \mid \neg \phi' \\
& \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2
\end{aligned}
$$

Syntax

Examples:

$$x \vee \neg x$$

$$x \rightarrow y \rightarrow z \text{ (ambiguous)}$$

$$x \rightarrow (y \rightarrow z)$$

$$(x \rightarrow y) \rightarrow z$$

$$(p \rightarrow q) \leftrightarrow (0 \vee \neg p \vee q)$$

Syntax

Examples:

$$x \vee \neg x$$

$$x \rightarrow y \rightarrow z \text{ (ambiguous)}$$

$$x \rightarrow (y \rightarrow z)$$

$$(x \rightarrow y) \rightarrow z$$

$$(p \rightarrow q) \leftrightarrow (0 \vee \neg p \vee q)$$

$\neg$ usually bings stronger, so $\neg p \vee q$ means $(\neg p) \vee q$.

Syntax

Examples:

$$x \vee \neg x$$

$$x \rightarrow y \rightarrow z \text{ (ambiguous)}$$

$$x \rightarrow (y \rightarrow z)$$

$$(x \rightarrow y) \rightarrow z$$

$$(p \rightarrow q) \leftrightarrow (0 \vee \neg p \vee q)$$

$\neg$ usually bings stronger, so $\neg p \vee q$ means $(\neg p) \vee q$.

Similarly, $p \wedge q \vee r$ usually means $(p \wedge q) \vee r$,
$p \wedge q \rightarrow a \vee b$ usually means $(q \wedge q) \rightarrow (a \vee b)$,
etc.

Syntax

Examples:

$$x \vee \neg x$$

$$x \rightarrow y \rightarrow z \text{ (ambiguous)}$$

$$x \rightarrow (y \rightarrow z)$$

$$(x \rightarrow y) \rightarrow z$$

$$(p \rightarrow q) \leftrightarrow (0 \vee \neg p \vee q)$$

$\neg$ usually bings stronger, so $\neg p \vee q$ means $(\neg p) \vee q$.

Similarly, $p \wedge q \vee r$ usually means $(p \wedge q) \vee r$,
$p \wedge q \rightarrow a \vee b$ usually means $(q \wedge q) \rightarrow (a \vee b)$,
etc.

When unsure, better use parentheses!

Alternative syntax

- $\Rightarrow$ instead of $\rightarrow$, but in modern logic notation, $\Rightarrow$ is used for semantical entailment, as in "formula ϕ entails formula ϕ', or $\phi \Rightarrow \phi'$, meaning that ϕ' is true when ϕ is true"
- $\Leftrightarrow$ instead of $\leftrightarrow$
- $+$ instead of $\vee$
- $\cdot$ instead of $\wedge$ (often omitted altogether)
- $\overline{x}$ instead of $\neg x$

E.g.,

$$xy + \overline{z}$$

instead of

$$(x \wedge y) \vee (\neg z)$$

Semantics

The **meaning** of logical formulas.

E.g., what is the semantics of a boolean formula such as $p \rightarrow q$?

Semantics

The **meaning** of logical formulas.

E.g., what is the semantics of a boolean formula such as $p \rightarrow q$?

"If p, then q", of course.

So, why do we even need to talk about semantics?

Semantics

What is the meaning of a boolean formula?

Different views (all equivalent):

- A "truth table".
- A boolean function.
- A set containing the "solutions" ("models") of the formula.

Semantics

What is the meaning of a boolean formula?

Different views (all equivalent):

- A "truth table".
- A boolean function.
- A set containing the "solutions" ("models") of the formula.

Why not consider the syntax itself to be the semantics?

Semantics

Formula:

$$x \wedge (y \vee z)$$

Truth table:

x	y	z	result
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Semantics

Formula:

$$x \wedge (y \vee z)$$

Truth table:

x	y	z	result
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

An equivalent formula (different syntax, same semantics):

$$(x \wedge y) \vee (x \wedge z)$$

Semantics

Boolean function: a function $f : \mathbb{B}^n \to \mathbb{B}^m$, where $\mathbb{B} = \{0, 1\}$.

Formula:

$$x \wedge (y \vee z)$$

defines[1] the boolean function: $f : \mathbb{B}^3 \to \mathbb{B}$ such that:

$$f(0,0,0) = 0$$
$$f(0,0,1) = 0$$
$$\ldots$$

[1] assuming an order on the variables: (1) x, (2) y, (3) z.

Semantics

Boolean function: a function $f : \mathbb{B}^n \to \mathbb{B}^m$, where $\mathbb{B} = \{0, 1\}$.

Formula:

$$x \wedge (y \vee z)$$

defines[1] the boolean function: $f : \mathbb{B}^3 \to \mathbb{B}$ such that:

$$f(0,0,0) = 0$$
$$f(0,0,1) = 0$$
$$\ldots$$

Note: a boolean function $f : A \to \mathbb{B}$ defines a set $S_f \subseteq A$.

$$S_f = \{a \in A \mid f(a) = 1\}$$

f is often called the *characteristic function* of S_f.

[1] assuming an order on the variables: (1) x, (2) y, (3) z.

Semantics

A formula $\phi : x \wedge (y \vee z)$ defines[2] a subset $[\![\phi]\!] \subseteq \mathbb{B}^3$:

$$[\![\phi]\!] = \{(1,0,1),(1,1,0),(1,1,1)\}$$

This is the set of "solutions": all assignments to x, y, z which make the formula true.

[2] assuming an order on the variables: (1) x, (2) y, (3) z.

Semantics

A formula $\phi : x \wedge (y \vee z)$ defines[2] a subset $[\![\phi]\!] \subseteq \mathbb{B}^3$:

$$[\![\phi]\!] = \{(1,0,1),(1,1,0),(1,1,1)\}$$

This is the set of "solutions": all assignments to x, y, z which make the formula true.

To be independent from an implicit order on variables, we can also view $[\![\phi]\!]$ as a set of **minterms**:

$$[\![\phi]\!] = \{x\bar{y}z, xy\bar{z}, xyz\}$$

[2] assuming an order on the variables: (1) x, (2) y, (3) z.

Semantics

A formula $\phi : x \wedge (y \vee z)$ defines[2] a subset $[\![\phi]\!] \subseteq \mathbb{B}^3$:

$$[\![\phi]\!] = \{(1,0,1),(1,1,0),(1,1,1)\}$$

This is the set of "solutions": all assignments to x, y, z which make the formula true.

To be independent from an implicit order on variables, we can also view $[\![\phi]\!]$ as a set of **minterms**:

$$[\![\phi]\!] = \{x\overline{y}z, xy\overline{z}, xyz\}$$

We can also view $[\![\phi]\!]$ as a set of **sets of atomic propositions**:

$$[\![\phi]\!] = \{\{x,z\},\{x,y\},\{x,y,z\}\}$$

[2] assuming an order on the variables: (1) x, (2) y, (3) z.

Semantics

A formula $\phi : x \wedge (y \vee z)$ defines[2] a subset $[\![\phi]\!] \subseteq \mathbb{B}^3$:

$$[\![\phi]\!] = \{(1,0,1),(1,1,0),(1,1,1)\}$$

This is the set of "solutions": all assignments to x, y, z which make the formula true.

To be independent from an implicit order on variables, we can also view $[\![\phi]\!]$ as a set of **minterms**:

$$[\![\phi]\!] = \{x\overline{y}z, xy\overline{z}, xyz\}$$

We can also view $[\![\phi]\!]$ as a set of **sets of atomic propositions**:

$$[\![\phi]\!] = \{\{x,z\},\{x,y\},\{x,y,z\}\}$$

What is the type of $[\![\phi]\!]$ in this last case?

[2] assuming an order on the variables: (1) x, (2) y, (3) z.

Semantics

A formula $\phi : x \wedge (y \vee z)$ defines[2] a subset $[\![\phi]\!] \subseteq \mathbb{B}^3$:

$$[\![\phi]\!] = \{(1,0,1),(1,1,0),(1,1,1)\}$$

This is the set of "solutions": all assignments to x, y, z which make the formula true.

To be independent from an implicit order on variables, we can also view $[\![\phi]\!]$ as a set of **minterms**:

$$[\![\phi]\!] = \{x\overline{y}z, xy\overline{z}, xyz\}$$

We can also view $[\![\phi]\!]$ as a set of **sets of atomic propositions**:

$$[\![\phi]\!] = \{\{x,z\},\{x,y\},\{x,y,z\}\}$$

What is the type of $[\![\phi]\!]$ in this last case?
$[\![\phi]\!] \subseteq \mathbb{B}^P = 2^P$ where P is the set of atomic propositions (= formula variables).

[2] assuming an order on the variables: (1) x, (2) y, (3) z.

Semantics: satisfaction relation

Satisfaction relation:

$$a \models \phi$$

means a is a "solution" (or **model**) of ϕ ("a *satisfies* ϕ").

So

$$a \models \phi \qquad \text{iff} \qquad a \in [\![\phi]\!]$$

Satisfiability and Validity

A formula ϕ is **satisfiable** if $[\![\phi]\!]$ is non-empty, i.e., if there exists $a \models \phi$.

A formula ϕ is **valid** (a **tautology**) if for all a, $a \models \phi$, i.e., if $[\![\phi]\!] = 2^P$.

Satisfiability and Validity

A formula ϕ is **satisfiable** if $[\![\phi]\!]$ is non-empty, i.e., if there exists $a \models \phi$.

A formula ϕ is **valid** (a **tautology**) if for all a, $a \models \phi$, i.e., if $[\![\phi]\!] = 2^P$.

Note: a formula can be one of three things:

- Unsatisfiable
- Valid
- Neither: satisfiable, but not valid

Satisfiability and Validity

Is it decidable to check satisfiability and validity of propositional logic formulas?

Satisfiability and Validity

Is it decidable to check satisfiability and validity of propositional logic formulas?

Yes:

- Brute-force method for satisfiability (SAT): test all possible variable assignments. If the formula has n variables $\Rightarrow 2^n$ possible assignments.
- Can we do better?
- In the worst case, no: 3-SAT (SAT of formulas where each clause has at most 3 literals) is a classic NP-complete problem.
- In practice, modern SAT solvers can handle formulas with thousands of variables or more.

Satisfiability and Validity

Is it decidable to check satisfiability and validity of propositional logic formulas?

Yes:

- Brute-force method for satisfiability (SAT): test all possible variable assignments. If the formula has n variables $\Rightarrow 2^n$ possible assignments.
- Can we do better?
- In the worst case, no: 3-SAT (SAT of formulas where each clause has at most 3 literals) is a classic NP-complete problem.
- In practice, modern SAT solvers can handle formulas with thousands of variables or more.
- What about validity?

Satisfiability and Validity

Is it decidable to check satisfiability and validity of propositional logic formulas?

Yes:

- Brute-force method for satisfiability (SAT): test all possible variable assignments. If the formula has n variables $\Rightarrow 2^n$ possible assignments.
- Can we do better?
- In the worst case, no: 3-SAT (SAT of formulas where each clause has at most 3 literals) is a classic NP-complete problem.
- In practice, modern SAT solvers can handle formulas with thousands of variables or more.
- What about validity?
  Check satisfiability of $\neg\phi$: $\neg\phi$ is unsat iff ϕ is valid.

NORMAL FORMS

CNF and DNF

Literal: a variable x or its negation $\overline{x}$.

Clause: a disjunction of literals. E.g.:

$$\begin{aligned} \text{clause 1} &: \quad x + y \\ \text{clause 2} &: \quad \overline{x} + z + w \end{aligned}$$

CNF (**conjunctive normal form**): conjunction of clauses, i.e., conjunction of disjunctions of literals (also called POS - "product of sums"). E.g.:

$$(x + y) \cdot (\overline{x} + z + w) \cdots$$

DNF (**disjunctive normal form**): disjunction of conjunctions of literals (also called SOP - "sum of products"). E.g.:

$$(xy) + (\overline{x}zw) + \cdots$$

NNF: Negation Normal Form

All negations are "pushed" into literals.
E.g.:

$(x \wedge y) \rightarrow (z \wedge w) \quad \leadsto$

NNF: Negation Normal Form

All negations are "pushed" into literals.
E.g.:

$$(x \wedge y) \rightarrow (z \wedge w) \quad \rightsquigarrow \quad (\neg(x \wedge y)) \vee (z \wedge w) \quad \rightsquigarrow$$

NNF: Negation Normal Form

All negations are "pushed" into literals.
E.g.:

$$(x \wedge y) \rightarrow (z \wedge w) \quad \rightsquigarrow \quad (\neg(x \wedge y)) \vee (z \wedge w) \quad \rightsquigarrow \quad (\neg x \vee \neg y) \vee (z \wedge w)$$

Translation into DNF

Every formula can be trivially transformed into DNF.
How?

Translation into DNF

Every formula can be trivially transformed into DNF.
How?

By taking the disjunction of all the satisfying assignments of the formula (every satisfying assignment is a minterm).
This procedure is not efficient, as a formula may have exponentially many satisfying assignments.

Translation into DNF

Every formula can be trivially transformed into DNF.
How?

By taking the disjunction of all the satisfying assignments of the formula (every satisfying assignment is a minterm).
This procedure is not efficient, as a formula may have exponentially many satisfying assignments.

Are there more efficient ways to transform into DNF? (Hint: how easy is it to check whether a DNF formula is SAT? how hard is SAT?)

Translation into DNF

Every formula can be trivially transformed into DNF.
How?

By taking the disjunction of all the satisfying assignments of the formula (every satisfying assignment is a minterm).
This procedure is not efficient, as a formula may have exponentially many satisfying assignments.

Are there more efficient ways to transform into DNF? (Hint: how easy is it to check whether a DNF formula is SAT? how hard is SAT?)

No, because SAT is NP-hard in general but SAT on DNF formulas is linear (just find one conjunction that can be satisfied).

Translation into CNF

Given a formula in NNF, how to transform it into CNF?

Translation to CNF

CNF(ϕ):
if ϕ is a literal **then**
    return ϕ;
else if ϕ is $\phi_1 \wedge \phi_2$ **then**
    return CNF(ϕ_1) $\wedge$ CNF(ϕ_2);
else if ϕ is $\phi_1 \vee \phi_2$ **then**
    return DistributeOr$\big($CNF(ϕ_1), CNF(ϕ_2)$\big)$;
else
    error: ϕ not in NNF;
end if

DistributeOr(ϕ_1, ϕ_2):
if ϕ_1 is $\phi_{11} \wedge \phi_{12}$ **then**
    return DistributeOr(ϕ_{11}, ϕ_2) $\wedge$ DistributeOr(ϕ_{12}, ϕ_2);
else if ϕ_2 is $\phi_{21} \wedge \phi_{22}$ **then**
    return DistributeOr(ϕ_1, ϕ_{21}) $\wedge$ DistributeOr(ϕ_1, ϕ_{22});
else
    return $\phi_1 \vee \phi_2$;   /* both must be literals or disjunctions at this point */
end if

Translation to CNF

CNF(ϕ):
if ϕ is a literal **then**
  return ϕ;
else if ϕ is $\phi_1 \wedge \phi_2$ **then**
  return CNF(ϕ_1) $\wedge$ CNF(ϕ_2);
else if ϕ is $\phi_1 \vee \phi_2$ **then**
  return DistributeOr(CNF(ϕ_1), CNF(ϕ_2));
else
  error: ϕ not in NNF;
end if

DistributeOr(ϕ_1, ϕ_2):
if ϕ_1 is $\phi_{11} \wedge \phi_{12}$ **then**
  return DistributeOr(ϕ_{11}, ϕ_2) $\wedge$ DistributeOr(ϕ_{12}, ϕ_2);
else if ϕ_2 is $\phi_{21} \wedge \phi_{22}$ **then**
  return DistributeOr(ϕ_1, ϕ_{21}) $\wedge$ DistributeOr(ϕ_1, ϕ_{22});
else
  return $\phi_1 \vee \phi_2$; /* both must be literals or disjunctions at this point */
end if

How large can CNF(ϕ) be in the worst-case?

Translation to CNF

CNF(ϕ):
if ϕ is a literal **then**
  return ϕ;
else if ϕ is $\phi_1 \wedge \phi_2$ **then**
  return CNF(ϕ_1) $\wedge$ CNF(ϕ_2);
else if ϕ is $\phi_1 \vee \phi_2$ **then**
  return DistributeOr(CNF(ϕ_1), CNF(ϕ_2));
else
  error: ϕ not in NNF;
end if

DistributeOr(ϕ_1, ϕ_2):
if ϕ_1 is $\phi_{11} \wedge \phi_{12}$ **then**
  return DistributeOr(ϕ_{11}, ϕ_2) $\wedge$ DistributeOr(ϕ_{12}, ϕ_2);
else if ϕ_2 is $\phi_{21} \wedge \phi_{22}$ **then**
  return DistributeOr(ϕ_1, ϕ_{21}) $\wedge$ DistributeOr(ϕ_1, ϕ_{22});
else
  return $\phi_1 \vee \phi_2$;   /* both must be literals or disjunctions at this point */
end if

How large can CNF(ϕ) be in the worst-case? Exponential, e.g., translate

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee \cdots \vee (a_n \wedge b_n)$$

Translation to CNF

CNF(ϕ):
if ϕ is a literal **then**
    return ϕ;
else if ϕ is $\phi_1 \wedge \phi_2$ **then**
    return CNF(ϕ_1) $\wedge$ CNF(ϕ_2);
else if ϕ is $\phi_1 \vee \phi_2$ **then**
    return DistributeOr(CNF(ϕ_1), CNF(ϕ_2));
else
    error: ϕ not in NNF;
end if

DistributeOr(ϕ_1, ϕ_2):
if ϕ_1 is $\phi_{11} \wedge \phi_{12}$ **then**
    return DistributeOr(ϕ_{11}, ϕ_2) $\wedge$ DistributeOr(ϕ_{12}, ϕ_2);
else if ϕ_2 is $\phi_{21} \wedge \phi_{22}$ **then**
    return DistributeOr(ϕ_1, ϕ_{21}) $\wedge$ DistributeOr(ϕ_1, ϕ_{22});
else
    return $\phi_1 \vee \phi_2$;   /* both must be literals or disjunctions at this point */
end if

How large can CNF(ϕ) be in the worst-case? Exponential, e.g., translate

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee \cdots \vee (a_n \wedge b_n)$$

We'll discuss polynomial translations when we talk about SAT solving.

FIRST-ORDER LOGIC

(also called PREDICATE LOGIC)

Limitations of propositional logic

All humans are mortal.

How to write it in propositional logic?

Limitations of propositional logic

All humans are mortal.

How to write it in propositional logic?

We can associate one proposition p_i for every human i, with the meaning *"human i is mortal"*, and then state:

$$p_1 \wedge p_2 \wedge \cdots \wedge p_{7000000000}$$

Limitations of propositional logic

All humans are mortal.

How to write it in propositional logic?

We can associate one proposition p_i for every human i, with the meaning *"human i is mortal"*, and then state:

$$p_1 \wedge p_2 \wedge \cdots \wedge p_{7000000000}$$

But even this is not enough, since we also want to talk about future generations.

Expressing this in (first-order) predicate logic

$$\forall x : H(x) \rightarrow M(x)$$

x: variable

H, M: predicates (functions that return "true" or "false")

$H(x)$: "x is human".

$M(x)$: "x is mortal".

$\forall$: "for all" quantifier.

First-Order Predicate Logic (FOL) – Syntax

Terms:

$$t \quad ::= \quad x \mid c \mid f(t_1, ..., t_n)$$

where x is any variable symbol, c is any constant symbol,[3] and f is any function symbol of some arity n.

Formulas:

$$\begin{aligned} \phi \quad ::= \quad & P(t_1, ..., t_n) \\ & \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\neg\phi) \mid \cdots \\ & \mid (\forall x : \phi) \mid (\exists x : \phi) \end{aligned}$$

where P is any predicate symbol of some arity n, and t_i are terms.

[3] constants can also be seen as functions of arity 0

FOL – Syntax

Example:

$$\forall x : x > 0 \rightarrow x + 1 > 0$$

FOL – Syntax

Example:

$$\forall x : x > 0 \rightarrow x + 1 > 0$$

or, more pedantically:

$$\forall x : \; >(x, 0) \rightarrow \; >(+(x, 1), 0)$$

FOL – Syntax

Example:

$$\forall x : x > 0 \rightarrow x + 1 > 0$$

or, more pedantically:

$$\forall x : \; >(x, 0) \rightarrow \; >(+(x, 1), 0)$$

- $0, 1$: constants
- x: variable symbol
- $+$: function symbol of arity 2
- $>$: predicate symbol of arity 2

FOL – Syntax

Note:

- This is also a syntactically well-formed formula:

$$x > 0 \rightarrow x + 1 > 0$$

FOL – Syntax

Note:

- This is also a syntactically well-formed formula:

$$x > 0 \rightarrow x + 1 > 0$$

- so is this:

$$\forall x : x > y$$

FOL – Syntax

Note:

- This is also a syntactically well-formed formula:

$$x > 0 \rightarrow x + 1 > 0$$

- so is this:

$$\forall x : x > y$$

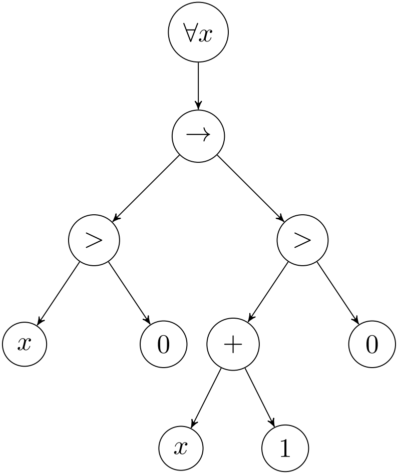
- or this:

$$\forall x : 2z > f(y)$$

Parse Tree of Formula

Formula: $\forall x : x > 0 \rightarrow x + 1 > 0$

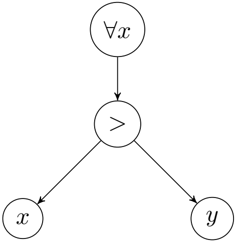
Parse tree:



Free and Bound Variables

Formula: $\forall x : x > y$

Parse tree:



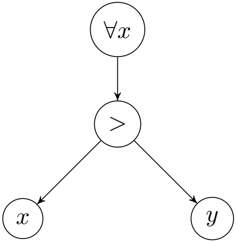
y is **free** in the formula: no ancestor of the leaf node y is a node of the form $\forall y$ or $\exists y$.

x is **bound** in the formula: has ancestor $\forall x$.

Free and Bound Variables

Formula: $\forall x : x > y$

Parse tree:



y is **free** in the formula: no ancestor of the leaf node y is a node of the form $\forall y$ or $\exists y$.

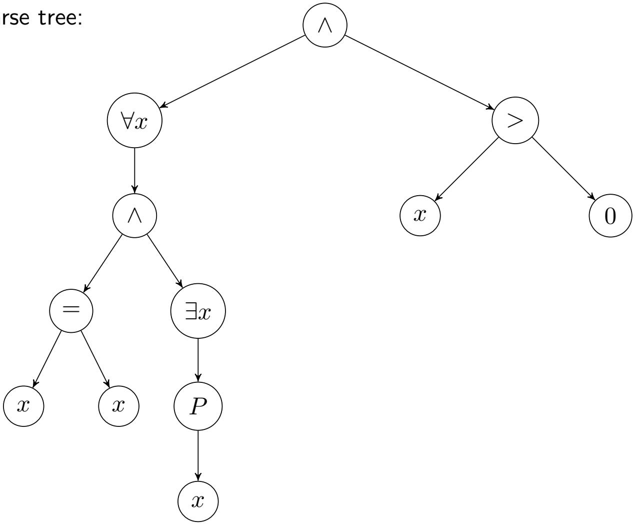
x is **bound** in the formula: has ancestor $\forall x$.

A formula is **closed** if it has no free variables.

Scope of Variables

Formula: $(\forall x : x = x \wedge \exists x : P(x)) \wedge x > 0$

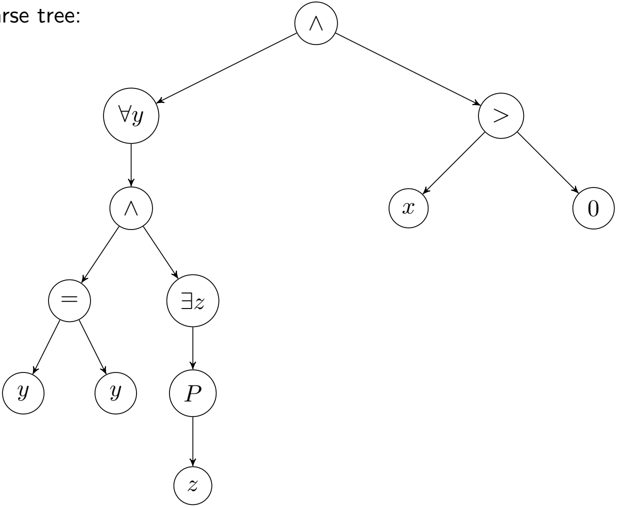
Parse tree:



Renaming

Formula: $(\forall x : x = x \wedge \exists x : P(x)) \wedge x > 0 \;\rightsquigarrow\; (\forall y : y = y \wedge \exists z : P(z)) \wedge x > 0$

Parse tree:



FOL – Semantics

In propositional logic, a “solution” (model) of a formula was simply an assignment of truth values to the propositional variables. E.g.,

$$\underbrace{(p := 1, q := 0)}_{\textit{model}} \models \underbrace{p \lor q}_{\textit{formula}}$$

FOL – Semantics

In propositional logic, a "solution" (model) of a formula was simply an assignment of truth values to the propositional variables. E.g.,

$$\underbrace{(p := 1, q := 0)}_{\textit{model}} \models \underbrace{p \vee q}_{\textit{formula}}$$

What are the "solutions" (models) of predicate logic formulas?

$$\underbrace{???}_{\textit{model}} \models \underbrace{\forall x : P(x) \rightarrow \exists y : Q(x, y)}_{\textit{formula}}$$

FOL – Semantics

In propositional logic, a "solution" (model) of a formula was simply an assignment of truth values to the propositional variables. E.g.,

$$\underbrace{(p := 1, q := 0)}_{\textit{model}} \models \underbrace{p \vee q}_{\textit{formula}}$$

What are the "solutions" (models) of predicate logic formulas?

$$\underbrace{???}_{\textit{model}} \models \underbrace{\forall x : P(x) \rightarrow \exists y : Q(x, y)}_{\textit{formula}}$$

Cannot give meaning to the formula without first giving meaning to P, Q and also specifying where x, y range over.

FOL – Semantics

Let $\mathcal{P}$ and $\mathcal{F}$ be the sets of predicate and function symbols (for simplicity $\mathcal{F}$ also includes the constants).

A *model* $\mathcal{M}$ for the pair $(\mathcal{P}, \mathcal{F})$ consists of the following:

- A non-empty set $\mathcal{U}$, the *universe* of concrete values.
- For each 0-arity symbol $c \in \mathcal{F}$, a concrete value $c_{\mathcal{M}} \in \mathcal{U}$.
- For each $f \in \mathcal{F}$ with arity n, a function $f_{\mathcal{M}} : \mathcal{U}^n \to \mathcal{U}$.
- For each $P \in \mathcal{P}$ with arity n, a set $P_{\mathcal{M}} \subseteq \mathcal{U}^n$.

FOL – Semantics

Let $\mathcal{P}$ and $\mathcal{F}$ be the sets of predicate and function symbols (for simplicity $\mathcal{F}$ also includes the constants).

A *model* $\mathcal{M}$ for the pair $(\mathcal{P}, \mathcal{F})$ consists of the following:

- A non-empty set $\mathcal{U}$, the *universe* of concrete values.
- For each 0-arity symbol $c \in \mathcal{F}$, a concrete value $c_{\mathcal{M}} \in \mathcal{U}$.
- For each $f \in \mathcal{F}$ with arity n, a function $f_{\mathcal{M}} : \mathcal{U}^n \to \mathcal{U}$.
- For each $P \in \mathcal{P}$ with arity n, a set $P_{\mathcal{M}} \subseteq \mathcal{U}^n$.

Note:

- c, f, P are just symbols (syntactic objects).
- $c_{\mathcal{M}}, f_{\mathcal{M}}, P_{\mathcal{M}}$ are semantical objects (values, functions, sets).

FOL – Semantics

Example:

$$\forall x : P(x) \rightarrow \exists y : Q(x, y)$$

Let $\mathcal{M}$ be such that

- $\mathcal{U} = \mathbb{N}$: the set of naturals.
- $P_{\mathcal{M}} = \{0, 2, ...\}$: the set of even naturals.
- $Q_{\mathcal{M}} = \{(0, 1), (1, 2), (2, 3), ...\}$: the set of pairs $(n, n + 1)$, for $n \in \mathbb{N}$.

Then the statement above is true.

FOL – Semantics

Example:

$$\forall x : P(x) \rightarrow \exists y : Q(x, y)$$

Let $\mathcal{M}$ be such that

- $\mathcal{U} = \mathbb{N}$: the set of naturals.
- $P_{\mathcal{M}} = \{0, 2, ...\}$: the set of even naturals.
- $Q_{\mathcal{M}} = \{(0, 1), (1, 2), (2, 3), ...\}$: the set of pairs $(n, n + 1)$, for $n \in \mathbb{N}$.

Then the statement above is true.

Of course, it could have been written "more clearly" (for a human):

$$\forall x : \textit{Even}(x) \rightarrow \exists y : y = x + 1$$

... but a computer (or a person who does not speak English) is equally clueless as to what P or *Even* means ...

FOL – Semantics

Example:

$$\forall x : P(x) \to \exists y : Q(x, y)$$

Let $\mathcal{M}'$ be another model such that

- $\mathcal{U} = \mathbb{N}$: the set of naturals.
- $P_{\mathcal{M}'} = \{0, 2, ...\}$: the set of even naturals.
- $Q_{\mathcal{M}'} = \{(1, 0), (3, 1), (5, 2), ...\}$: the set of pairs $(2n + 1, n)$, for $n \in \mathbb{N}$.

Then the statement above is false.

FOL – Semantics

What is the meaning of $\forall x : x > y$?

Undefined if we know nothing about the value of y.

FOL – Semantics

What is the meaning of $\forall x : x > y$?

Undefined if we know nothing about the value of y.

We need one more thing: *environments* (or "look-up tables" for variables).

Environment:

$$l : \text{VariableSymbols} \rightarrow \mathcal{U}$$

assigns a concrete value to every variable symbol.

FOL – Semantics

What is the meaning of $\forall x : x > y$?

Undefined if we know nothing about the value of y.

We need one more thing: *environments* (or "look-up tables" for variables).

Environment:

$$l : \text{VariableSymbols} \rightarrow \mathcal{U}$$

assigns a concrete value to every variable symbol.

Notation:

$$l[x \rightsquigarrow a]$$

is a new environment l' such that $l'(x) = a$ and $l'(y) = l(y)$ for any other variable y.

FOL – Semantics: Giving concrete values to terms

Once we have $\mathcal{M}$ and l, every term evaluates to a concrete value in $\mathcal{U}$.

Example:

$\mathcal{M}$: $\mathcal{U} = \mathbb{N}$, "0" $= 0$, "1" $= 1$, ..., $+ =$ addition function, ...
l: $x \leadsto 2$, $y \leadsto 1$

term t	value $\mathcal{M}_l(t)$
$x + 1$	3
$x \cdot y$	2
...	

FOL – Semantics: Giving concrete values to terms

Once we have $\mathcal{M}$ and l, every term evaluates to a concrete value in $\mathcal{U}$.

Example:

$\mathcal{M}$: $\mathcal{U} = \mathbb{N}$, "0" $= 0$, "1" $= 1$, ..., $+ =$ addition function, ...
l: $x \rightsquigarrow 2$, $y \rightsquigarrow 1$

term t	value $\mathcal{M}_l(t)$
$x + 1$	3
$x \cdot y$	2
...	

For a term t, we denote this value by $\mathcal{M}_l(t)$.

FOL – Semantics

Finally we can define the satisfaction relation for first-order predicate logic ($\mathcal{M}$: model, l: environment, ϕ: formula):

$$\mathcal{M}, l \models \phi$$

$\mathcal{M}, l \models P(t_1, ..., t_n)$ iff

FOL – Semantics

Finally we can define the satisfaction relation for first-order predicate logic ($\mathcal{M}$: model, l: environment, ϕ: formula):

$$\mathcal{M}, l \models \phi$$

$$\begin{array}{lll}
\mathcal{M}, l \models P(t_1, ..., t_n) & \text{iff} & \big(\mathcal{M}_l(t_1), ..., \mathcal{M}_l(t_n)\big) \in P_{\mathcal{M}} \\
\mathcal{M}, l \models \phi_1 \wedge \phi_2 & \text{iff} & \mathcal{M}, l \models \phi_1 \text{ and } \mathcal{M}, l \models \phi_2 \\
\mathcal{M}, l \models \neg\phi & \text{iff} & \mathcal{M}, l \not\models \phi \\
\mathcal{M}, l \models \forall x : \phi & \text{iff} & \text{for all } a \in \mathcal{U} : \mathcal{M}, l[x \leadsto a] \models \phi \text{ holds} \\
\mathcal{M}, l \models \exists x : \phi & \text{iff} & \text{for some } a \in \mathcal{U} : \mathcal{M}, l[x \leadsto a] \models \phi \text{ holds}
\end{array}$$

FOL – Semantics: Satisfiability, Validity

A FOL formula ϕ is **satisfiable** if there exist $\mathcal{M}, l$ such that $\mathcal{M}, l \models \phi$ holds.

A formula ϕ is **valid** (a **tautology**) if for all $\mathcal{M}, l$, it holds $\mathcal{M}, l \models \phi$.

Satisfiability, Validity

Examples:

1. $\forall x : P(x) \to P(x)$

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \rightarrow P(x)$$

   Valid.

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \rightarrow P(x)$$

   Valid.

2. $$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \rightarrow P(x)$$
   Valid.

2. $$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$
   Satisfiable.

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \rightarrow P(x)$$

   Valid.

2. $$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

   Satisfiable.

   Example model: $\mathcal{U} = \mathbb{N}$, $x \mapsto 0$, $y \mapsto 1$, $f(\_) \mapsto 0$,

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \to P(x)$$
   Valid.

2. $$x \geq 0 \land f(x) \geq 0 \land y \geq 0 \land f(y) \geq 0 \land x \neq y$$
   Satisfiable.

   Example model: $\mathcal{U} = \mathbb{N}$, $x \mapsto 0$, $y \mapsto 1$, $f(\_) \mapsto 0$, $\neq$ is the "not equal to" relation on $\mathbb{N}$: $\neq \mapsto \{(0,1),(0,2),...,(1,0),(1,2),...\}$.

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \rightarrow P(x)$$

   Valid.

2. $$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

   Satisfiable.

   Example model: $\mathcal{U} = \mathbb{N}$, $x \mapsto 0$, $y \mapsto 1$, $f(\_) \mapsto 0$, $\neq$ is the "not equal to" relation on $\mathbb{N}$: $\neq \mapsto \{(0,1),(0,2),...,(1,0),(1,2),...\}$.

3. $$x + 2 = y \wedge f(\texttt{read}(\texttt{write}(A, x, 3), y - 2)) \neq f(y - x + 1)$$

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \rightarrow P(x)$$

   Valid.

2. $$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

   Satisfiable.

   Example model: $\mathcal{U} = \mathbb{N}$, $x \mapsto 0$, $y \mapsto 1$, $f(\_) \mapsto 0$, $\neq$ is the "not equal to" relation on $\mathbb{N}$: $\neq \mapsto \{(0,1),(0,2),...,(1,0),(1,2),...\}$.

3. $$x + 2 = y \wedge f(\mathtt{read}(\mathtt{write}(A, x, 3), y - 2)) \neq f(y - x + 1)$$

   Satisfiable with a non-standard interpretation of $+, -$ or $\mathtt{read}, \mathtt{write}$.

Satisfiability, Validity

Examples:

1. $$\forall x : P(x) \rightarrow P(x)$$
   Valid.

2. $$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$
   Satisfiable.

   Example model: $\mathcal{U} = \mathbb{N}$, $x \mapsto 0$, $y \mapsto 1$, $f(\_) \mapsto 0$, $\neq$ is the "not equal to" relation on $\mathbb{N}$: $\neq \mapsto \{(0,1), (0,2), ..., (1,0), (1,2), ...\}$.

3. $$x + 2 = y \wedge f(\texttt{read}(\texttt{write}(A, x, 3), y - 2)) \neq f(y - x + 1)$$
   Satisfiable with a non-standard interpretation of $+, -$ or $\texttt{read}, \texttt{write}$.

   Unsatisfiable with the standard interpretation of those symbols (theories of arithmetic and arrays). Why?

Normal forms for FOL

- Negation normal form: “push” negation across quantifiers, and then across boolean connectives as in propositional logic

$$\neg\forall x : F[x] \Leftrightarrow \exists x : \neg F[x] \qquad \text{and} \qquad \neg\exists x : F[x] \Leftrightarrow \forall x : \neg F[x]$$

Normal forms for FOL

- Negation normal form: "push" negation across quantifiers, and then across boolean connectives as in propositional logic

$$\neg\forall x : F[x] \Leftrightarrow \exists x : \neg F[x] \qquad \text{and} \qquad \neg\exists x : F[x] \Leftrightarrow \forall x : \neg F[x]$$

- CNF and DNF:
  - First put the formula in **prenex normal form** (PNF), where all quantifiers appear at the beginning of the formula, e.g.,

$$(\forall x : P(x)) \rightarrow (\exists y : R(y)) \quad \rightsquigarrow \quad \exists x : \exists y : \neg P(x) \vee R(y)$$

  - Then convert the "main body" subformula, which is quantifier-free, to CNF or DNF using same methods as for propositional logic.

Prenex normal form

Procedure to convert a formula ϕ in PNF (prenex normal form) [Bradley and Manna, 2007]:

1. Convert ϕ to NNF, to obtain ϕ_1.
2. Rename quantified variables so that there are no such variables that have the same name but are in different scopes, to obtain ϕ_2.
3. Remove quantifiers from ϕ_2 to obtain quantifier-free formula ϕ_3.
4. Add all removed quantifiers at the head of ϕ_3, to obtain ϕ_4:

$$\phi_4 := \mathbf{Q}_1 x_1 : \mathbf{Q}_2 x_2 : \cdots \mathbf{Q}_n x_n : \phi_3$$

so that if quantifier $\mathbf{Q}_j$ is in the scope of $\mathbf{Q}_i$ in ϕ_1, then $i < j$.

Prenex normal form

Procedure to convert a formula ϕ in PNF (prenex normal form) [Bradley and Manna, 2007]:

1. Convert ϕ to NNF, to obtain ϕ_1.
2. Rename quantified variables so that there are no such variables that have the same name but are in different scopes, to obtain ϕ_2.
3. Remove quantifiers from ϕ_2 to obtain quantifier-free formula ϕ_3.
4. Add all removed quantifiers at the head of ϕ_3, to obtain ϕ_4:

$$\phi_4 := \mathbf{Q}_1 x_1 : \mathbf{Q}_2 x_2 : \cdots \mathbf{Q}_n x_n : \phi_3$$

so that if quantifier $\mathbf{Q}_j$ is in the scope of $\mathbf{Q}_i$ in ϕ_1, then $i < j$.

Let's run this on some examples:

$$(\forall x : P(x)) \rightarrow (\exists y : R(y)), \quad \forall x : \neg(\exists y : P(x,y) \wedge P(x,z)) \vee \exists y : P(x,y)$$

THEORIES

First-order theories

- FOL is very general (and also undecidable)
- In practice, we often use restricted subsets, where symbols have the expected meaning, e.g.,
  - Arithmetic formulas: $\forall n : n + 1 > n$
- We formalize this concept as a **theory**, e.g.,
  - Theory of Peano arithmetic (addition, multiplication)
  - Theory of Presburger arithmetic (addition, no multiplication)
  - Theory of arrays
  - Theory of uninterpreted functions with equality
  - ...

First-order theories

A first-order theory is defined by

- its **signature**: the set of constant, function, and predicate symbols
  The signature defines the syntax of the theory.
- its set of **axioms**: these are closed FOL formulas (no free variables) which have symbols only from the theory's signature.
  The axioms define the meaning of the symbols, i.e., the semantics of the theory!

Example: Presburger arithmetic

- Signature: $\Sigma_{\mathbb{N}} = \{0, 1, +, =\}$, where
  - $0, 1$ are constants
  - $+$ is a binary function
  - $=$ is a binary predicate

- Axioms:
  1. $\forall x : x + 0 = x$ (zero is the neutral element for addition)
  2. $\forall x : \neg(x + 1 = 0)$ (no negative numbers)
  3. $\forall x, y : x + 1 = y + 1 \rightarrow x = y$
  4. $F[0] \wedge (\forall x : F[x] \rightarrow F[x + 1]) \rightarrow \forall x : F[x]$ (induction – this is in fact an axiom **schema**, an infinite set of axioms, for any instance of F)
  5. $\forall x, y : x + (y + 1) = (x + y) + 1$

Example: Presburger arithmetic

- Signature: $\Sigma_{\mathbb{N}} = \{0, 1, +, =\}$, where
  - $0, 1$ are constants
  - $+$ is a binary function
  - $=$ is a binary predicate

- Axioms:
  1. $\forall x : x + 0 = x$ (zero is the neutral element for addition)
  2. $\forall x : \neg(x + 1 = 0)$ (no negative numbers)
  3. $\forall x, y : x + 1 = y + 1 \to x = y$
  4. $F[0] \wedge (\forall x : F[x] \to F[x + 1]) \to \forall x : F[x]$ (induction – this is in fact an axiom **schema**, an infinite set of axioms, for any instance of F)
  5. $\forall x, y : x + (y + 1) = (x + y) + 1$

Note: we write $\forall x : x + 0 = x$ for convenience. The legal syntax is $\forall x : \; = (+(x, 0), x)$.

Example: Presburger arithmetic

- Signature: $\Sigma_{\mathbb{N}} = \{0, 1, +, =\}$, where
  - $0, 1$ are constants
  - $+$ is a binary function
  - $=$ is a binary predicate

- Axioms:
  1. $\forall x : x + 0 = x$ (zero is the neutral element for addition)
  2. $\forall x : \neg(x + 1 = 0)$ (no negative numbers)
  3. $\forall x, y : x + 1 = y + 1 \rightarrow x = y$
  4. $F[0] \wedge (\forall x : F[x] \rightarrow F[x + 1]) \rightarrow \forall x : F[x]$ (induction – this is in fact an axiom **schema**, an infinite set of axioms, for any instance of F)
  5. $\forall x, y : x + (y + 1) = (x + y) + 1$

Note: we write $\forall x : x + 0 = x$ for convenience. The legal syntax is $\forall x : \; = (+(x, 0), x)$.

Presburger arithmetic is decidable! (i.e., it is decidable, given a formula, to check whether it is satisfiable, or valid)

Example: Peano arithmetic

- Signature: $\Sigma_{PA} = \{0, 1, +, \cdot, =\}$, where
  - $0, 1$ are constants
  - $+, \cdot$ are binary functions
  - $=$ is a binary predicate

- Axioms: the axioms of Presburger arithmetic, plus
  6. $\forall x : x \cdot 0 = 0$
  7. $\forall x, y : x \cdot (y + 1) = (x \cdot y) + x$

Example: Peano arithmetic

- Signature: $\Sigma_{PA} = \{0, 1, +, \cdot, =\}$, where
  - $0, 1$ are constants
  - $+, \cdot$ are binary functions
  - $=$ is a binary predicate
- Axioms: the axioms of Presburger arithmetic, plus
  6. $\forall x : x \cdot 0 = 0$
  7. $\forall x, y : x \cdot (y + 1) = (x \cdot y) + x$

Peano arithmetic is undecidable.

PROOFS

Proofs

Suppose we want to prove that a given formula is valid.
How to do it?

- We can use the brute-force method, but this only applies to propositional logic formulas, and even there, is intractable.
- We can try to reason in natural language, as in

  $(p \wedge p \rightarrow q) \rightarrow q$ *is valid, because assuming both* p *and* $p \rightarrow q$ *to be true, since* p *is true, and* p *implies* q *by* $p \rightarrow q$*, we can conclude that* q *must also be true.*

  Not very satisfactory ...
- We can try a more systematic and rigorous method (which we can also hope to automate, either fully or partially).

Proof rules for propositional logic

Suppose we want to prove that propositional formula ϕ is valid.
Let's try to reason by contradiction, and attempt to find an assignment a such that $a \not\models \phi$. If we succeed, then ϕ is invalid (not valid). If we reach a contradiction, ϕ is valid.

Proof rules for propositional logic

Suppose we want to prove that propositional formula ϕ is valid.
Let's try to reason by contradiction, and attempt to find an assignment a such that $a \not\models \phi$. If we succeed, then ϕ is invalid (not valid). If we reach a contradiction, ϕ is valid.

We use the following **proof rules** (or **deduction rules**), based on the syntax of ϕ:

- For negation:

$$\frac{a \not\models \neg\phi}{a \models \phi}\ \mathrm{NEG}_1 \qquad\qquad \frac{a \models \neg\phi}{a \not\models \phi}\ \mathrm{NEG}_2$$

The way you read such a rule, e.g., NEG_1, is: *if we assume $a \not\models \neg\phi$, then we can deduce $a \models \phi$.*

Proof rules for propositional logic

- For conjunction:

$$\frac{a \not\models \phi_1 \wedge \phi_2}{a \not\models \phi_1 \mid a \not\models \phi_2} \; \text{AND}_1$$

(proof generates 2 "or" branches)

$$\frac{a \models \phi_1 \wedge \phi_2}{\begin{array}{c} a \models \phi_1 \\ a \models \phi_2 \end{array}} \; \text{AND}_2$$

(proof generates 2 deductions)

Note: here we are going downwards; often proofs are written in the opposite way, going upwards.

Proof rules for propositional logic

- For disjunction:

Proof rules for propositional logic

- For disjunction:

$$\frac{a \not\models \phi_1 \vee \phi_2}{\begin{array}{c} a \not\models \phi_1 \\ a \not\models \phi_2 \end{array}} \ \mathrm{OR}_1 \qquad \frac{a \models \phi_1 \vee \phi_2}{a \models \phi_1 \mid a \models \phi_2} \ \mathrm{OR}_2$$

Proof rules for propositional logic

- For implication:

Proof rules for propositional logic

- For implication:

$$\frac{a \not\models \phi_1 \rightarrow \phi_2}{\begin{array}{c} a \models \phi_1 \\ a \not\models \phi_2 \end{array}} \text{ IMPL}_1 \qquad \frac{a \models \phi_1 \rightarrow \phi_2}{a \not\models \phi_1 \mid a \models \phi_2} \text{ IMPL}_2$$

Proof rules for propositional logic

- For implication:

$$\frac{a \not\models \phi_1 \to \phi_2}{\begin{array}{c} a \models \phi_1 \\ a \not\models \phi_2 \end{array}} \text{ IMPL}_1 \qquad \frac{a \models \phi_1 \to \phi_2}{a \not\models \phi_1 \mid a \models \phi_2} \text{ IMPL}_2$$

- For equivalence:

Proof rules for propositional logic

- For implication:

$$\frac{a \not\models \phi_1 \to \phi_2}{\begin{array}{c} a \models \phi_1 \\ a \not\models \phi_2 \end{array}} \text{ IMPL}_1 \qquad \frac{a \models \phi_1 \to \phi_2}{a \not\models \phi_1 \mid a \models \phi_2} \text{ IMPL}_2$$

- For equivalence:

$$\frac{a \not\models \phi_1 \leftrightarrow \phi_2}{a \models \phi_1 \land \neg\phi_2 \mid a \models \neg\phi_1 \land \phi_2} \text{ EQUIV}_1$$

$$\frac{a \models \phi_1 \leftrightarrow \phi_2}{a \models \phi_1 \land \phi_2 \mid a \not\models \phi_1 \lor \phi_2} \text{ EQUIV}_2$$

Proof rules for propositional logic

When do we reach a contradiction?

- Contradiction rule:

$$\frac{\begin{array}{c} a \not\models \phi \\ a \models \phi \end{array}}{a \models \bot}\ \textsc{Contra}$$

Proof rules for propositional logic

When do we reach a contradiction?

- Contradiction rule:

$$\frac{a \not\models \phi \qquad a \models \phi}{a \models \bot}\ \textsc{Contra}$$

Let's try to prove this using our proof system:

$$(p \to q) \land (q \to r) \to (p \to r)$$

Bibliography

Bradley, A. R. and Manna, Z. (2007).
The calculus of computation - decision procedures with applications to verification.
Springer.

Doxiadis, A., Papadimitriou, C., Papadatos, A., and Di Donna, A. (2009).
Logicomix: An Epic Search for Truth.
Bloomsbury USA.

Huth, M. and Ryan, M. (2004).
Logic in Computer Science: Modelling and Reasoning about Systems.
Cambridge University Press.

Tourlakis, G. (2008).
Mathematical Logic.
Wiley.