Programm- & Systemverifikation

Propositional and First-Order Logic

Georg Weissenbacher

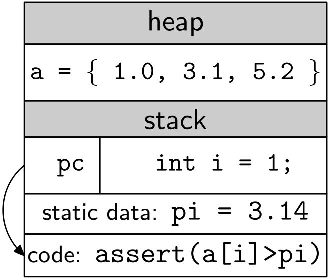
184.741

What happened so far

- How bugs come into being:
  - Fault – cause of an error (e.g., mistake in coding)
  - Error – *incorrect* state that may lead to failure
  - Failure – deviation from *desired* behaviour
- We specified *intended* behaviour using assertions
- We proved our programs correct (inductive invariants).
- We learned how to test programs.
- We generated test cases.

Assertion Violations

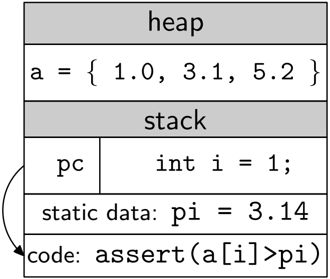
How do we know what (`a[i]` > `pi`) means?



Assertion Violations

How do we know what (a[i] > pi) means?

- Programming Language Semantics



Use Formal Languages to Specify Semantics

Enables us to…

- unambiguously specify meaning of language constructs
- formally reason about correctness of
  - program transformations/optimisations
  - code generation
  - program correctness

Which Formal Languages Do You Already Know?

Which Formal Languages Do You Already Know?

- Propositional Logic (PL, “Aussagenlogik”)

Which Formal Languages Do You Already Know?

- Propositional Logic (PL, “Aussagenlogik”)
- First-Order Logic (FOL, “Prädikatenlogik”)

Which Formal Languages Do You Already Know?

- Propositional Logic (PL, “Aussagenlogik”)
- First-Order Logic (FOL, “Prädikatenlogik”)

This lecture:

Syntax & Semantics of PL & FOL

(Some of the following slides borrowed with permission from Aaron Bradley)

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
& & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
& & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= \textit{formula} \Rightarrow \textit{formula}$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\mathit{formula} & ::= & \mathit{formula} \wedge \mathit{formula} \mid \mathit{formula} \vee \mathit{formula} \mid \\
 & & \mathit{formula} \Rightarrow \mathit{formula} \mid \mathit{formula} \Leftrightarrow \mathit{formula} \mid \\
 & & \neg \mathit{formula} \mid (\mathit{formula}) \mid \mathit{atom} \\
\mathit{atom} & ::= & \mathit{identifier} \mid \mathit{constant} \\
\mathit{constant} & ::= & \text{true} \mid \text{false} \\
\mathit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\mathit{formula} ::= \mathit{formula} \Rightarrow \text{atom}$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \land \textit{formula} \mid \textit{formula} \lor \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- *formula* ::= *formula* $\Rightarrow$ identifier

Syntax of Propositional Logic

$$\begin{array}{rcl} \textit{formula} & ::= & \textit{formula} \land \textit{formula} \mid \textit{formula} \lor \textit{formula} \mid \\ & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\ & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\ \textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\ \textit{constant} & ::= & \text{true} \mid \text{false} \\ \textit{identifier} & \in & \{P, Q, R, \ldots\} \end{array}$$

Formulas built recursively from syntax:

- $\textit{formula} ::= \textit{formula} \Rightarrow Q$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- *formula* ::= (*formula*) $\Rightarrow Q$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (\textit{formula} \wedge \textit{formula}) \Rightarrow Q$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (\textit{atom} \wedge \textit{atom}) \Rightarrow Q$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \land \textit{formula} \mid \textit{formula} \lor \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (\textit{identifier} \land \textit{constant}) \Rightarrow Q$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (P \wedge \textit{true}) \Rightarrow Q$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (P \wedge \textit{true}) \Rightarrow Q$

Can also be *parsed* recursively:

- $\textit{formula} ::= \underbrace{(P \ \wedge \ Q) \ \Rightarrow \ (\text{true} \ \vee \ \neg Q)}_{\text{formula}}$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
& & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
& & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- *formula* ::= $(P \wedge \textit{true}) \Rightarrow Q$

Can also be *parsed* recursively:

- *formula* ::= $\underbrace{(P \;\wedge\; Q)}_{\text{formula}} \;\Rightarrow\; \underbrace{(\text{true} \;\vee\; \neg Q)}_{\text{formula}}$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg \textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (P \wedge \textit{true}) \Rightarrow Q$

Can also be *parsed* recursively:

- $\textit{formula} ::= (\underbrace{P}_{\text{atom}} \wedge \underbrace{Q}_{\text{atom}}) \Rightarrow (\underbrace{\text{true}}_{\text{atom}} \vee \underbrace{\neg Q}_{\text{formula}})$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (P \wedge \textit{true}) \Rightarrow Q$

Can also be *parsed* recursively:

- $\textit{formula} ::= (\underbrace{P}_{\text{identifier}} \wedge \underbrace{Q}_{\text{identifier}}) \Rightarrow (\underbrace{\text{true}}_{\text{constant}} \vee \neg \underbrace{Q}_{\text{atom}})$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg \textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (P \wedge \textit{true}) \Rightarrow Q$

Can also be *parsed* recursively:

- $\textit{formula} ::= (\underbrace{P}_{\text{identifier}} \wedge \underbrace{Q}_{\text{identifier}}) \Rightarrow (\underbrace{\text{true}}_{\text{constant}} \vee \neg \underbrace{Q}_{\text{identifier}})$

Syntax of Propositional Logic

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
& & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
& & \neg \textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false} \\
\textit{identifier} & \in & \{P, Q, R, \ldots\}
\end{array}
$$

Formulas built recursively from syntax:

- $\textit{formula} ::= (P \wedge \textit{true}) \Rightarrow Q$

Can also be *parsed* recursively:

- $\textit{formula} ::=$

- Syntax only specifies *structure*
- Formula adheres to syntax $\Leftrightarrow$ Formula is *well-formed*

Semantics (Meaning) of Propositional Logic

Syntax does *not* tell us how to *interpret* formulas

Propositional Identifiers:

- P, Q, R, ...
- represent "propositions":
  - "it is raining"
  - "the least significant bit of `x` is 1"

Semantics (Meaning) of Propositional Logic

Syntax does *not* tell us how to *interpret* formulas

Propositional Identifiers:

- P, Q, R, ...
- represent "propositions":
  - "it is raining"
  - "the least significant bit of `x` is 1"
- We *ignore* the underlying meaning of propositions
  - Propositions take the values *true* or *false*

Propositional Logic: Interpretations

An *interpretation* I assigns truth values to identifiers

$$I = \{P \mapsto \text{true}, Q \mapsto \text{false}, R \mapsto \text{true}, \ldots\}$$

Meaning of Boolean operations is defined via truth table:

F_1	F_2	$\neg F_1$	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \Rightarrow F_2$	$F_1 \Leftrightarrow F_2$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Evaluating Formulas

Evaluation of a formula using a truth table:

P	Q	$P \wedge Q$	$\neg Q$	$\text{true} \vee \neg Q$	$(P \wedge Q) \Rightarrow (\text{true} \vee \neg Q)$
true	false	false	true	true	true

Evaluating Formulas

Evaluation of a formula using a truth table:

P	Q	$P \wedge Q$	$\neg Q$	true $\vee \neg Q$	$(P \wedge Q) \Rightarrow$ (true $\vee \neg Q$)
true	false	false	true	true	true

How many Boolean operations over n propositions are there?

Inductive Definition of Semantics

$I \models F$	if F evaluates to	true	under I
$I \not\models F$	if F evaluates to	false	under I

Base case

$$I \models P \quad \text{iff} \quad I(P) = \text{true}$$
$$I \not\models P \quad \text{iff} \quad I(P) = \text{false}$$

Inductive case

$$\begin{array}{lll} I \models \neg F & \text{iff} & I \not\models F \\ I \models F_1 \wedge F_2 & \text{iff} & I \models F_1 \text{ and } I \models F_2 \\ I \models F_1 \vee F_2 & \text{iff} & I \models F_1 \text{ or } I \models F_2 \\ I \models F_1 \Rightarrow F_2 & \text{iff} & I \not\models F_1 \text{ or } I \models F_2 \\ I \models F_1 \Leftrightarrow F_2 & \text{iff} & I \models F_1 \text{ and } I \models F_2 \\ & & \text{or } I \not\models F_1 \text{ and } I \not\models F_2 \end{array}$$

Inductive Definition of Semantics

Given an interpretation I, we can *evaluate* a formula recursively:

- $I = \{P \mapsto \text{true}, Q \mapsto \text{false}\}$
- *formula* ::= $(\underbrace{P}_{\text{true}} \land \underbrace{Q}_{\text{false}}) \Rightarrow (\text{true} \lor \neg \underbrace{Q}_{\text{false}})$

Inductive Definition of Semantics

Given an interpretation I, we can *evaluate* a formula recursively:

- $I = \{P \mapsto \text{true}, Q \mapsto \text{false}\}$
- *formula* ::= $(\underbrace{P}_{\text{true}} \wedge \underbrace{Q}_{\text{false}}) \Rightarrow (\text{true} \vee \underbrace{\neg Q}_{\text{true}})$

Inductive Definition of Semantics

Given an interpretation I, we can *evaluate* a formula recursively:

- $I = \{P \mapsto \text{true}, Q \mapsto \text{false}\}$
- *formula* ::= $\underbrace{(P \ \wedge \ Q)}_{\text{false}} \ \Rightarrow \ \underbrace{(\text{true} \ \vee \ \neg Q)}_{\text{true}}$

Inductive Definition of Semantics

Given an interpretation I, we can *evaluate* a formula recursively:

- $I = \{P \mapsto \text{true}, Q \mapsto \text{false}\}$
- *formula* ::= $\underbrace{(P \ \land \ Q) \ \Rightarrow \ (\text{true} \ \lor \ \neg Q)}_{\text{true}}$

Satisfiability and Validity

F is satisfiable iff there exists an interpretation I such that $I \models F$.
F is valid iff for all interpretations I it holds that $I \models F$.

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable.}}$$

Example:

$$F : P \wedge Q \Rightarrow P \vee \neg Q$$

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
false	false	false	true	true	true
false	true	false	false	false	true
true	false	false	true	true	true
true	true	true	false	true	true

Equivalence and Semantic Consequence

F_1 and F_2 are equivalent ($F_1 \equiv F_2$)
if and only if

$$\{I \mid I \models F_1\} \quad = \quad \{I \mid I \models F_2\}$$

F_1 entails F_2 ($F_1 \models F_2$)
if and only if

$$\{I \mid I \models F_1\} \quad \subseteq \quad \{I \mid I \models F_2\}$$

$\equiv$ and $\models$ are symbols of the *meta*-language:
$F_1 \equiv F_2$ and $F_1 \models F_2$ are *not* formulas!

Equi-satisfiability

Two formulas with different propositional identifiers

- have incomparable interpretations
- can therefore not be equivalent

F_1 and F_2 are equi-satisfiable
if and only if
F_1 is satisfiable iff F_2 is satisfiable

Equi-satisfiability

Two formulas with different propositional identifiers

- have incomparable interpretations
- can therefore not be equivalent

F_1 and F_2 are equi-satisfiable
if and only if
F_1 is satisfiable iff F_2 is satisfiable

Example: $P \wedge Q$ and $R \wedge (R \Leftrightarrow (P \wedge Q))$

Data Structures for Representing Formulas

Truth tables are just one (inefficient) way of representing Boolean functions. Other forms are…

- Propositional Logic / (Quantified) Boolean Formulas
- Negation Normal Form, Conjunctive Normal Form
- Polynomials in GF(2)
- Boolean Circuits
- Binary Decision Trees/Diagrams (BDDs)

Functional Completeness

- A Boolean data structure is functionally complete if all Boolean functions can be expressed in this data structure.
- All data structures mentioned above are functionally complete

Functional Completeness

- A Boolean data structure is functionally complete if all Boolean functions can be expressed in this data structure.
- All data structures mentioned above are functionally complete

Which Boolean operators result in a functionally complete logic?

Functional Completeness

- A Boolean data structure is functionally complete if all Boolean functions can be expressed in this data structure.
- All data structures mentioned above are functionally complete

Which Boolean operators result in a functionally complete logic?

- $\vee$ and $\neg$, or $\wedge$ and $\neg$

Canonicity of Boolean Data Structures

Data structures for F and G are identical
if and only if
F and G are identical

Canonicity of Boolean Data Structures

Data structures for F and G are identical
if and only if
F and G are identical

Examples:

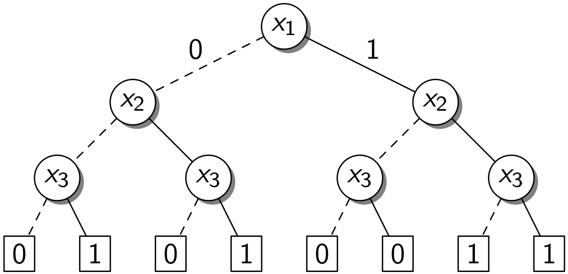
- Truth tables
- (Ordered) Binary Decision Diagrams
- Conjunctive Normal Form (with maxterms)
- ...

Binary Decision Diagrams (Bryant 86)

- Store formulas as directed acyclic graphs
  - Nodes represent variables
  - Edges represent assignments
- Assignments can be derived in $O(\#\text{variables})$
- Representation is canonical
  - if order of variables fixed for all paths in graph

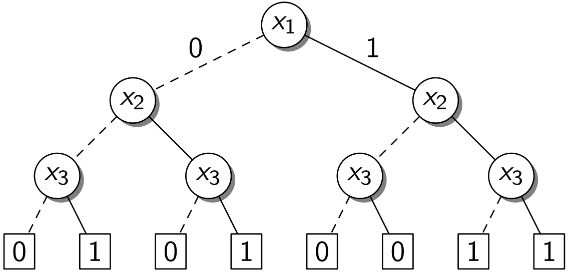
Binary Decision Tree

- Encode decisions and outcome in tree
  - Satisfying assignment can be found efficiently
- Wasteful, lot of redundancy
  - Not much better than truth table



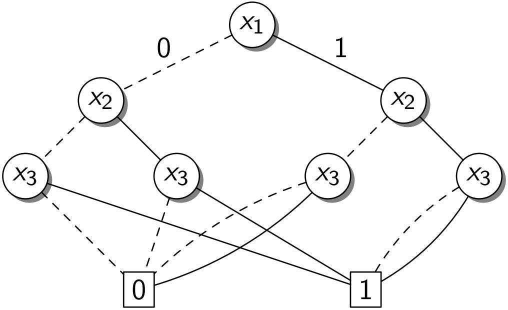
$$((x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3))$$

Binary Decision Tree: Reductions



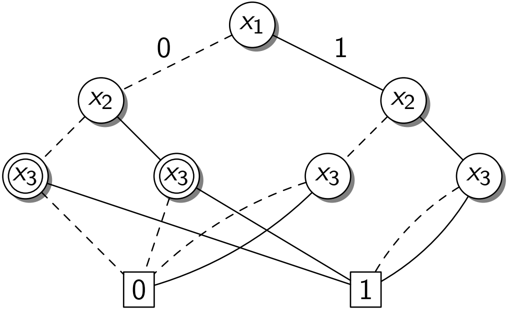
Binary Decision Tree: Reductions

- Merge leaf nodes



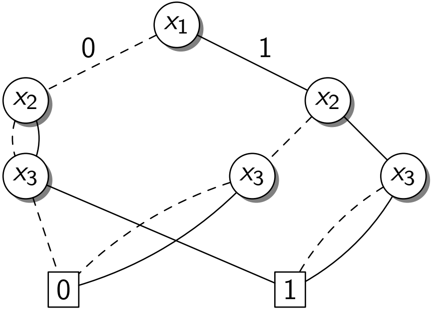
Binary Decision Tree: Reductions

- Merge leaf nodes
- Merge isomorphic subtrees



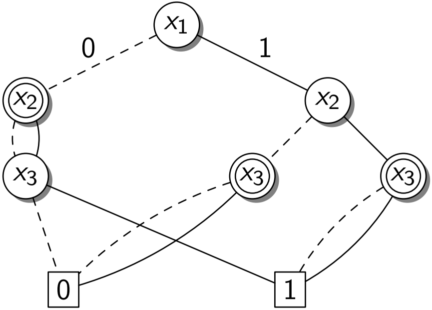
Binary Decision Tree: Reductions

- Merge leaf nodes
- Merge isomorphic subtrees



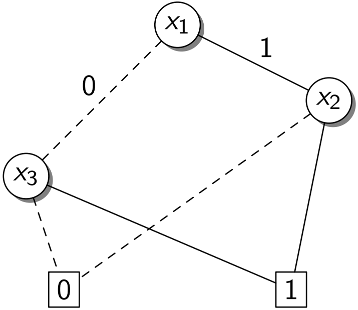
Binary Decision Tree: Reductions

- Merge leaf nodes
- Merge isomorphic subtrees
- Remove redundant nodes (introduce *don't cares*)



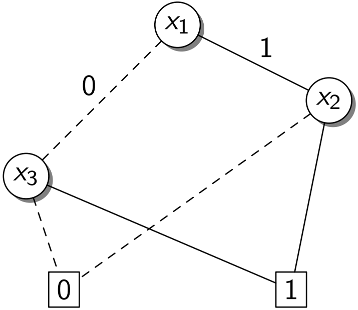
Binary Decision Tree: Reductions

- Merge leaf nodes
- Merge isomorphic subtrees
- Remove redundant nodes (introduce *don't cares*)



Binary Decision Tree: Reductions

- Merge leaf nodes
- Merge isomorphic subtrees
- Remove redundant nodes (introduce *don't cares*)



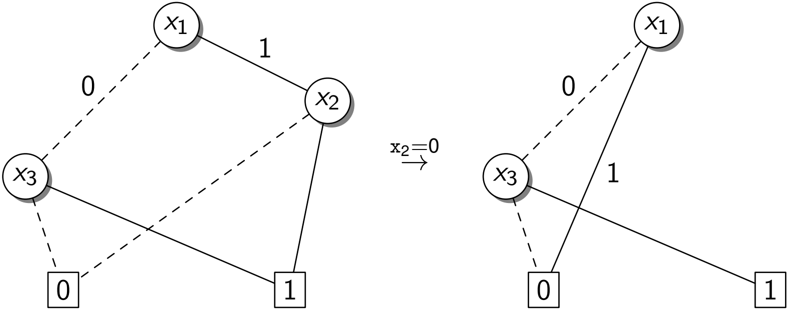
- Repeat reductions as long as possible

Constructing Binary Decision Diagrams

- Construction follows structure of formula
- $\mathcal{B}_1$ and $\mathcal{B}_2$ represent F_1 and F_2
  then $\mathcal{B}_1 \star \mathcal{B}_2$ represents $F_1 \star F_2$ (where $\star \in \{\wedge, \vee, \ldots\}$)
- Complexity of $\mathcal{B}_1 \star \mathcal{B}_2$ bounded by $|\mathcal{B}_1| \cdot |\mathcal{B}_2|$

Constructing Binary Decision Diagrams: Restrict

$$F|_{\mathtt{x}=0} \quad \equiv \quad F[\mathtt{x}/0]$$



$((\mathtt{x}_1 \wedge \mathtt{x}_2) \vee (\neg \mathtt{x}_1 \wedge \mathtt{x}_3))$

$(\neg \mathtt{x}_1 \wedge \mathtt{x}_3)$

Constructing BDDs: Shannon Expansion

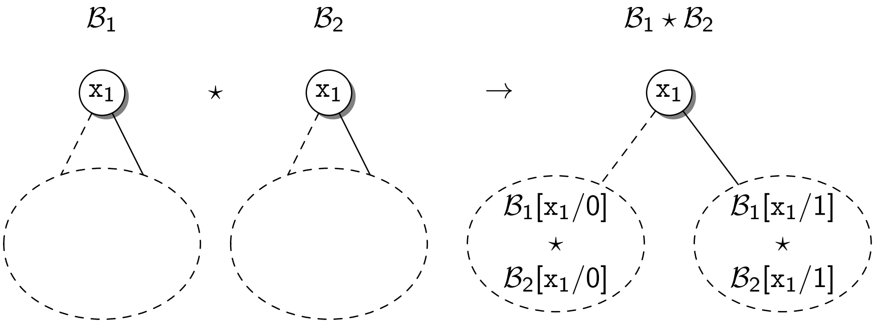
Definition (Shannon Expansion)

$$F \quad \equiv \quad (\neg \mathtt{x} \land F[\mathtt{x}/0]) \lor (\mathtt{x} \land F[\mathtt{x}/1])$$

Constructing BDDs: Shannon Expansion

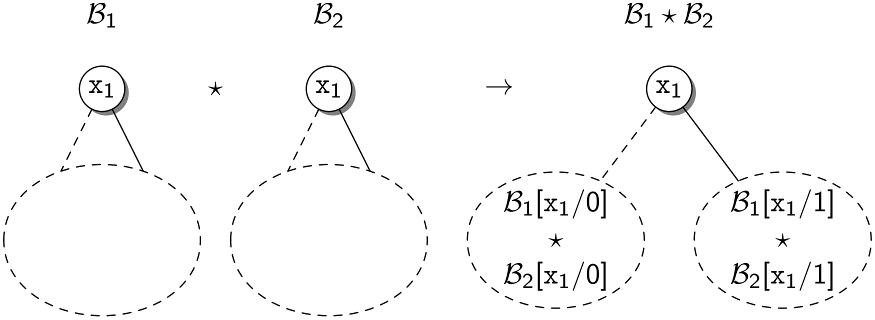
Definition (Shannon Expansion)

$$F \quad \equiv \quad (\neg \mathtt{x} \wedge F[\mathtt{x}/0]) \ \vee \ (\mathtt{x} \wedge F[\mathtt{x}/1])$$



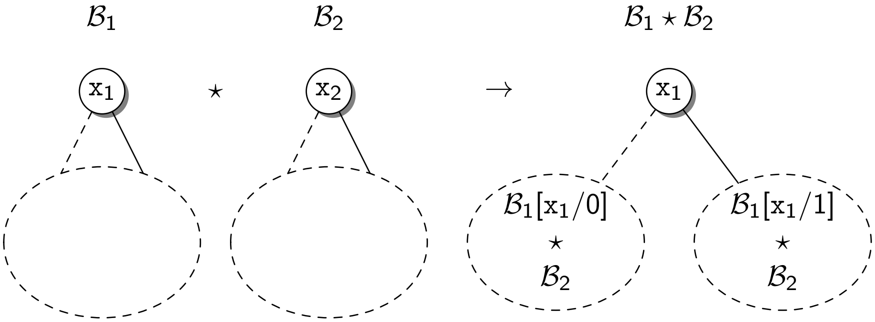
Combining two BDDs $\mathcal{B}_1 \star \mathcal{B}_2$

- Requirement: Same variable order!
- Start from root nodes v_1 and v_2
- Case 1: $\mathrm{var}(v_1) = \mathrm{var}(v_2) = \mathtt{x}_1$



Combining two BDDs $\mathcal{B}_1 \star \mathcal{B}_2$

- Case 2: $\mathrm{var}(v_1) \neq \mathrm{var}(v_2)$
  - $\mathrm{var}(v_1) = \mathtt{x}_1$, $\mathrm{var}(v_2) = \mathtt{x}_2$
  - $\mathtt{x}_1$ precedes $\mathtt{x}_2$ in variable order
  - Therefore, $\mathtt{x}_1$ *does not occur in* $\mathcal{B}_2$!



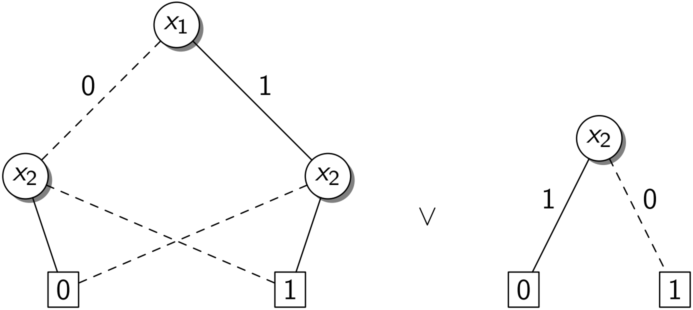
Combining two BDDs $\mathcal{B}_1 \star \mathcal{B}_2$

- Case 3: v_1 and v_2 are terminal nodes $\boxed{1}$ or $\boxed{0}$

$$\mathcal{B}_1 \star \mathcal{B}_2 \qquad \equiv \qquad \mathrm{val}(v_1) \star \mathrm{val}(v_1)$$

$$\boxed{0} \star \boxed{1} \qquad \rightarrow \qquad \boxed{0 \star 1}$$

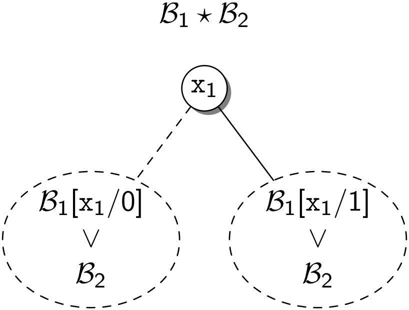
Constructing BDDs: Example



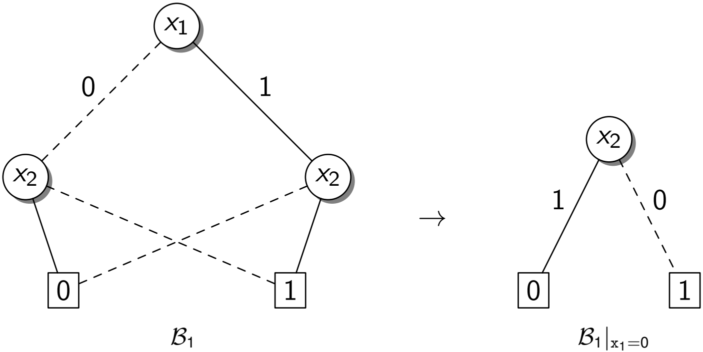
$\mathcal{B}_1 : \mathtt{x}_1 \Leftrightarrow \mathtt{x}_2$

$\mathcal{B}_2 : \neg\mathtt{x}_2$

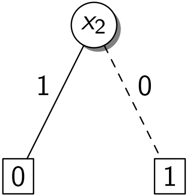
Constructing BDDs: Example



Constructing BDDs: Example

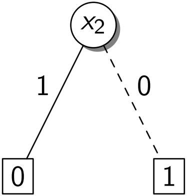


Constructing BDDs: Example



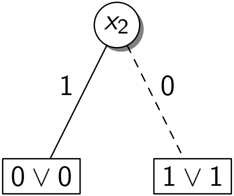
$\mathcal{B}_1|_{x_1=0}$

$\vee$



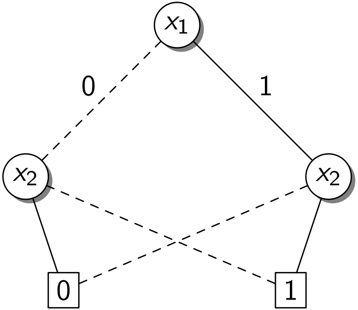
$\mathcal{B}_2$

$=$



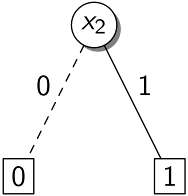
$\mathcal{B}_1|_{x_1=0} \vee \mathcal{B}_2$

Constructing BDDs: Example



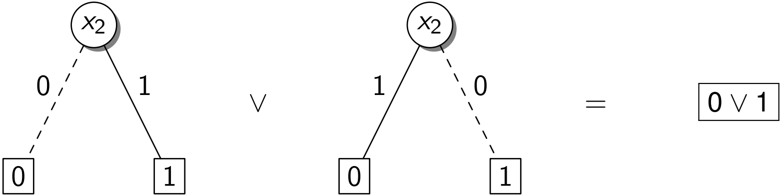
$\mathcal{B}_1$

$\rightarrow$



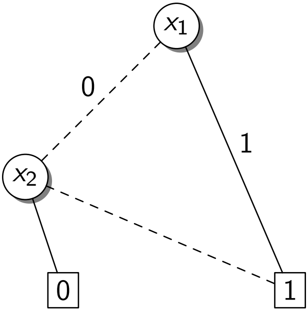
$\mathcal{B}_1|_{\mathtt{x}_1=1}$

Constructing BDDs: Example



$\mathcal{B}_1|_{x_1=1}$ $\mathcal{B}_2$ $\mathcal{B}_1|_{x_1=1} \vee \mathcal{B}_2$

Constructing BDDs: Example



$$x_1 \vee (\neg x_1 \vee \neg x_2)$$

Constructing BDDs: Variable Order

$$(\mathtt{x}_1 \Leftrightarrow \mathtt{y}_1) \wedge \ldots \wedge (\mathtt{x}_n \Leftrightarrow \mathtt{y}_n)$$

- $\mathtt{x}_1, \mathtt{y}_1, \ldots, \mathtt{x}_n, \mathtt{y}_n$: size $3n + 2$
- $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{y}_1, \mathtt{y}_2, \ldots$: size $3 \cdot 2^n - 1$

- There are functions s.t. number of nodes can't be polynomial
  - For instance: Multiplication of bit-vectors

Constructing BDDs: Complexity

- Quantification:

$$\begin{aligned} \forall \mathrm{x} \,.\, F \quad &\equiv \quad F[\mathrm{x}/0] \wedge F[\mathrm{x}/1] \\ \exists \mathrm{x} \,.\, F \quad &\equiv \quad F[\mathrm{x}/0] \vee F[\mathrm{x}/1] \end{aligned}$$

- Furthermore: If $F \equiv$ true then BDD is $\boxed{1}$
  - Follows immediately, because representation is *canonical*
- What does that mean for complexity?

Constructing BDDs: Complexity

- Quantification:

$$\begin{aligned} \forall \mathrm{x} \,.\, F &\equiv F[\mathrm{x}/0] \wedge F[\mathrm{x}/1] \\ \exists \mathrm{x} \,.\, F &\equiv F[\mathrm{x}/0] \vee F[\mathrm{x}/1] \end{aligned}$$

- Furthermore: If $F \equiv$ true then BDD is $\boxed{1}$
  - Follows immediately, because representation is *canonical*
- What does that mean for complexity?
  - Can solve **TQBF**, *the* prototypical PSPACE-complete problem

Compression of Boolean Data Structures

Remember:

- Number of Boolean functions: 2^{2^n}

Which representation is ”compact”?

- No data structure with good average compression

In practice:

QBF $>$ prop. logic $>$ BDDs $>$ Binary Decision Trees $>$ truth tables

Normal Forms: Negation Normal Form

Negations appear in *literals* only:

$$\begin{array}{rcl} \textit{formula} & ::= & \textit{formula} \land \textit{formula} \mid \textit{formula} \lor \textit{formula} \mid \textit{literal} \\ \textit{literal} & ::= & \textit{atom} \mid \neg\textit{atom} \end{array}$$

Transformation into NNF:

- Eliminate implication and bi-implication

$$\begin{array}{rcl} F_1 \Rightarrow F_2 & \equiv & \neg F_1 \lor F_2 \\ F_1 \Leftrightarrow F_2 & \equiv & (F_1 \Rightarrow F_2) \land (F_2 \Rightarrow F_1) \end{array}$$

- Eliminate (double) negation:

$$\neg\neg F \equiv F \qquad \neg\text{true} \equiv \text{false} \qquad \neg\text{false} \equiv \text{true}$$

- “Push” negation inwards:

$$\left.\begin{array}{rcl} \neg(F_1 \land F_2) & \equiv & \neg F_1 \lor \neg F_2 \\ \neg(F_1 \lor F_2) & \equiv & \neg F_1 \land \neg F_2 \end{array}\right\} \text{De Morgan's Law}$$

Negation Normal Form: Example

$$\neg(P \Rightarrow \neg(P \wedge Q))$$

Negation Normal Form: Example

$$\neg(P \Rightarrow \neg(P \land Q))$$

$$\neg(\neg P \lor \neg(P \land Q)) \quad \Rightarrow \text{ to } \lor$$

Negation Normal Form: Example

$$\neg(P \Rightarrow \neg(P \land Q))$$

$$\begin{array}{ll} \neg(\neg P \lor \neg(P \land Q)) & \Rightarrow \text{ to } \lor \\ \neg\neg P \land \neg\neg(P \land Q) & \text{De Morgan} \end{array}$$

Negation Normal Form: Example

$$\neg(P \Rightarrow \neg(P \wedge Q))$$

$$\begin{array}{ll}
\neg(\neg P \vee \neg(P \wedge Q)) & \Rightarrow \text{ to } \vee \\
\neg\neg P \wedge \neg\neg(P \wedge Q) & \text{De Morgan} \\
P \wedge P \wedge Q & \neg\neg
\end{array}$$

Conjunctive Normal Form

$$
\begin{aligned}
\textit{formula} \quad &::= \quad \textit{formula} \wedge \textit{formula} \mid (\textit{clause}) \\
\textit{clause} \quad &::= \quad \textit{literal} \vee \textit{clause} \mid \textit{literal} \\
\textit{literal} \quad &::= \quad \textit{atom} \mid \neg \textit{atom}
\end{aligned}
$$

- CNF formula: A conjunction of clauses (product of sums)

$$\bigwedge_i \bigvee_j \ell_{i,j}\,, \qquad \ell_{i,j} \in \{P, \neg P \mid P \in \text{Identifiers}\}$$

  e.g.,

$$\neg P \wedge (P_1 \vee \neg Q) \wedge (\neg P \vee Q) \wedge P$$

- Remember:
  - $\bigvee_{\ell \in \emptyset} \ell \;\equiv\;$ false (we use $\Box$ to denote the empty clause)
- Alternative (more compact) notation:

$$(\overline{P})\ (P\,\overline{Q})\ (\overline{P}\,Q)\ (P)$$

Tseitin Encoding

- If we use propositional logic rewrite rules:

$$(P \wedge Q) \vee (R \wedge S) \quad \equiv \quad (P \vee R) \wedge (P \vee S) \wedge (Q \vee R) \wedge (Q \vee S)$$

  Blowup if applied repeatedly!
- Idea: Construct *satisfiability-equivalent* formula
- Introduce a fresh symbol for each subterm:

$$\begin{gathered} (P \wedge Q) \vee (R \wedge S) \\ \longrightarrow \\ (O_1 \vee O_2) \;\wedge\; (O_1 \Leftrightarrow (P \wedge Q)) \;\wedge\; (O_2 \Leftrightarrow (R \wedge S)) \end{gathered}$$

- But this is still not CNF!

Tseitin Encoding

$$(O_1 \lor O_2) \land (O_1 \Leftrightarrow (P \land Q)) \land (O_2 \Leftrightarrow (R \land S))$$

- $(O_1 \Leftrightarrow (P \land Q)) \qquad \equiv$

Tseitin Encoding

$$(O_1 \lor O_2) \land (O_1 \Leftrightarrow (P \land Q)) \land (O_2 \Leftrightarrow (R \land S))$$

- $(O_1 \Leftrightarrow (P \land Q)) \qquad \equiv$

  $(O_1 \Rightarrow P) \land (O_1 \Rightarrow Q) \land ((P \land Q) \Rightarrow O_1) \qquad \equiv$

Tseitin Encoding

$$(O_1 \vee O_2) \;\wedge\; (O_1 \Leftrightarrow (P \wedge Q)) \;\wedge\; (O_2 \Leftrightarrow (R \wedge S))$$

- $(O_1 \Leftrightarrow (P \wedge Q)) \qquad \equiv$

  $(O_1 \Rightarrow P) \wedge (O_1 \Rightarrow Q) \wedge ((P \wedge Q) \Rightarrow O_1) \qquad \equiv$

  $(P \vee \neg O_1) \wedge (Q \vee \neg O_1) \wedge (O \vee \neg P \vee \neg Q)$

Tseitin Encoding

$$(O_1 \vee O_2) \wedge (O_1 \Leftrightarrow (P \wedge Q)) \wedge (O_2 \Leftrightarrow (R \wedge S))$$

- $(O_1 \Leftrightarrow (P \wedge Q)) \quad \equiv$

  $(O_1 \Rightarrow P) \wedge (O_1 \Rightarrow Q) \wedge ((P \wedge Q) \Rightarrow O_1) \quad \equiv$

  $(P \vee \neg O_1) \wedge (Q \vee \neg O_1) \wedge (O \vee \neg P \vee \neg Q)$
- Constant blowup

Tseitin Encoding

Negation:

$$
\begin{aligned}
P \Leftrightarrow \neg Q &\equiv (P \Rightarrow \neg Q) \wedge (\neg Q \Rightarrow P) \\
&\equiv (\neg P \vee \neg Q) \wedge (Q \vee P)
\end{aligned}
$$

Disjunction:

$$
\begin{aligned}
P \Leftrightarrow (Q \vee R) &\equiv (Q \Rightarrow P) \wedge (R \Rightarrow P) \wedge (P \Rightarrow (Q \vee R)) \\
&\equiv (\neg Q \vee P) \wedge (\neg R \vee P) \wedge (\neg P \vee Q \vee R)
\end{aligned}
$$

Conjunction:

$$
\begin{aligned}
P \Leftrightarrow (Q \wedge R) &\equiv (P \Rightarrow Q) \wedge (P \Rightarrow R) \wedge ((Q \wedge R) \Rightarrow P) \\
&\equiv (\neg P \vee Q) \wedge (\neg P \vee R) \wedge (\neg (Q \wedge R) \vee P) \\
&\equiv (\neg P \vee Q) \wedge (\neg P \vee R) \wedge (\neg Q \vee \neg R \vee P)
\end{aligned}
$$

Tseitin Encoding

Equivalence:

$$
\begin{aligned}
& P \Leftrightarrow (Q \Leftrightarrow R) \\
\equiv\quad & (P \Rightarrow (Q \Leftrightarrow R)) \land ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv\quad & (P \Rightarrow ((Q \Rightarrow R) \land (R \Rightarrow Q)) \land ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv\quad & (P \Rightarrow (Q \Rightarrow R)) \land (P \Rightarrow (R \Rightarrow Q)) \land ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv\quad & (\neg P \lor \neg Q \lor R) \land (\neg P \lor \neg R \lor Q) \land ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv\quad & (\neg P \lor \neg Q \lor R) \land (\neg P \lor \neg R \lor Q) \land (((Q \land R) \lor (\neg Q \land \neg R)) \Rightarrow P) \\
\equiv\quad & (\neg P \lor \neg Q \lor R) \land (\neg P \lor \neg R \lor Q) \land ((Q \land R) \Rightarrow P) \land ((\neg Q \land \neg R) \Rightarrow P) \\
\equiv\quad & (\neg P \lor \neg Q \lor R) \land (\neg P \lor \neg R \lor Q) \land (\neg Q \lor \neg R \lor P) \land (Q \lor R \lor P)
\end{aligned}
$$

Tseitin Encoding

Equivalence:

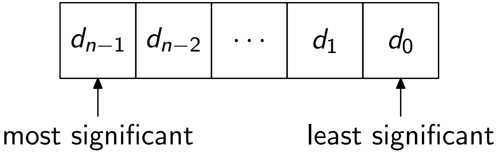
$$
\begin{array}{rl}
& P \Leftrightarrow (Q \Leftrightarrow R) \\
\equiv & (P \Rightarrow (Q \Leftrightarrow R)) \wedge ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv & (P \Rightarrow ((Q \Rightarrow R) \wedge (R \Rightarrow Q)) \wedge ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv & (P \Rightarrow (Q \Rightarrow R)) \wedge (P \Rightarrow (R \Rightarrow Q)) \wedge ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv & (\neg P \vee \neg Q \vee R) \wedge (\neg P \vee \neg R \vee Q) \wedge ((Q \Leftrightarrow R) \Rightarrow P) \\
\equiv & (\neg P \vee \neg Q \vee R) \wedge (\neg P \vee \neg R \vee Q) \wedge (((Q \wedge R) \vee (\neg Q \wedge \neg R)) \Rightarrow P) \\
\equiv & (\neg P \vee \neg Q \vee R) \wedge (\neg P \vee \neg R \vee Q) \wedge ((Q \wedge R) \Rightarrow P) \wedge ((\neg Q \wedge \neg R) \Rightarrow P) \\
\equiv & (\neg P \vee \neg Q \vee R) \wedge (\neg P \vee \neg R \vee Q) \wedge (\neg Q \vee \neg R \vee P) \wedge (Q \vee R \vee P)
\end{array}
$$

- Blowup by *constant* factor of 4
- Resulting formula satisfiable iff initial formula is

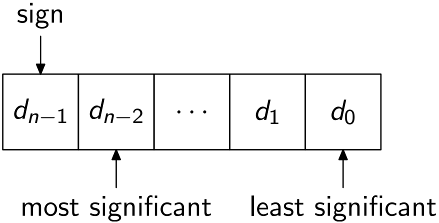
Expressing Bit-Vector Arithmetic in PL

At first sight, PL is not very expressive. . .

- Remember: C integers are bit-vectors $d_{n-1} \ldots d_0$ ($d_i \in \mathbb{B}$, $0 \leq i < n$)
- n is *width* of bit-vector.
- Unsigned:



- Signed:



Bit-Vectors: Interpretations

Interpretation function which maps $d_{n-1} \ldots d_0$ to finite sub-domain of $\mathbb{N}_0$ and $\mathbb{Z}$:

$$(d_{n-1} \ldots d_0)^{\mathcal{M}} \stackrel{\text{def}}{=} \begin{cases} \sum_{i=0}^{n-1} d_i \cdot 2^i & \text{unsigned} \\ -2^{n-1} \cdot d_{n-1} + \sum_{i=0}^{n-2} d_i \cdot 2^i & \text{signed} \end{cases}$$

- Accordingly, $=$, $\neq$, $\geq$, and $>$ take standard meaning in $\mathbb{Z}$.

Encoding Bit-Vector Operations

Equality $\mathtt{x} = \mathtt{y}$ is straight-forward:

$$\bigwedge_{i=0}^{n-1} (\mathtt{x}_i \Leftrightarrow \mathtt{y}_i)$$

Encoding Bit-Vector Operations

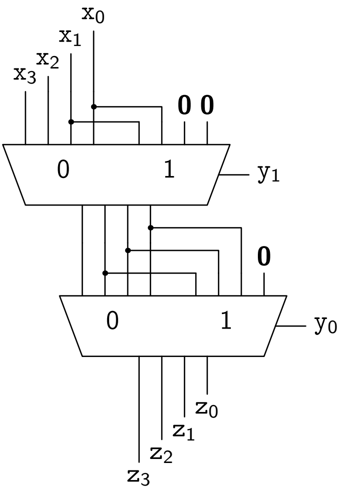
$$\mathtt{z} = \mathtt{x} \,\&\, \mathtt{y} \quad \ldots \quad \bigwedge_{i=0}^{n-1} (\mathtt{z}_i \Leftrightarrow (\mathtt{x}_i \wedge \mathtt{y}_i))$$

$$\mathtt{z} = \mathtt{x} \mid \mathtt{y} \quad \ldots \quad \bigwedge_{i=0}^{n-1} (\mathtt{z}_i \Leftrightarrow (\mathtt{x}_i \vee \mathtt{y}_i))$$

$$\mathtt{z} = \mathtt{x} \oplus \mathtt{y} \quad \ldots \quad \bigwedge_{i=0}^{n-1} \mathtt{z}_i \Leftrightarrow ((\mathtt{x}_i \vee \mathtt{y}_i) \wedge (\neg\mathtt{x}_i \vee \neg\mathtt{y}_i))$$

Encoding Bit-Vector Operations

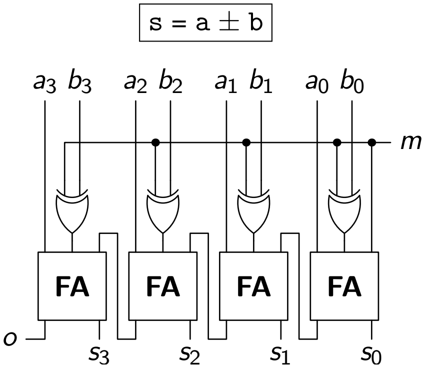
Shift operations implemented by means of a cascade of parallel multiplexers known as barrel shifter.

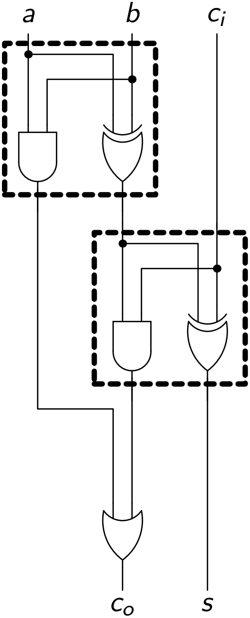


4-bit barrel shifter implementing $z = x \ll y$

i^{th} stage performs shift by 2^i positions if y_i is true.

Encoding Bit-Vector Operations





Encoding Bit-Vector Operations

- $\mathtt{x} < \mathtt{y}$ can be expressed using of subtraction
- If $\mathtt{x} < \mathtt{y}$, then $\mathtt{x} - \mathtt{y}$ yields *overflow* (can be detected by checking the signals c_o
  - Unsigned operands, overflow if $c_o =$ true.
  - Signed operands, $(c_o \oplus c_{o-1})$ indicates overflow

Encoding Bit-Vector Operations

- Multiplication uses shift-and-add circuit
- i.e., multiplication of 2-bit parameters $\mathtt{x}$ and $\mathtt{y}$ ($[\mathtt{x}_1\ \mathtt{x}_0]$ and $[\mathtt{y}_1\ \mathtt{y}_0]$) is

$$[\mathtt{z}_2\ \mathtt{z}_1\ \mathtt{z}_0] = ([0\ \mathtt{x}_1\ \mathtt{x}_0] \& [\mathtt{y}_0\ \mathtt{y}_0\ \mathtt{y}_0]) + (([0\ \mathtt{x}_1\ \mathtt{x}_0] \ll 1) \& [\mathtt{y}_1\ \mathtt{y}_1\ \mathtt{y}_1])\ .$$

Encoding Bit-Vector Operations

- Integer division $\mathtt{z} = \frac{\mathtt{x}}{\mathtt{y}}$ (for $\mathtt{y} \neq 0$)

$$(\mathtt{z} \cdot \mathtt{y} + \mathtt{r} = \mathtt{x}) \wedge (\mathtt{r} < \mathtt{y})$$

- where $\mathtt{r}$ denotes the remainder

Restrictions of Propositional Logic

- Sufficient to encode bit-vector operations
- What about infinite domains
  - $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\ldots$
  - data-structures like arrays, maps, lists?

Syntax

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \textit{formula} \Rightarrow \textit{formula} \mid \textit{formula} \Leftrightarrow \textit{formula} \mid \\
 & & \neg\textit{formula} \mid (\textit{formula}) \mid \\
 & & \textit{predicate}\,(\textit{term},\ldots,\textit{term}) \mid \textit{term} = \textit{term} \\
 & & \forall\ \textit{variable}\ .\ \textit{formula} \mid \exists\ \textit{variable}\ .\ \textit{formula} \\
\textit{term} & ::= & \textit{variable} \mid \textit{constant} \mid \textit{function}\,(\textit{term},\ldots\textit{term})
\end{array}
$$

- *variables*, *functions*, *predicates*, and *constants* are represented by unique identifiers
- each *function* and *predicate* has a fixed arity
- $\forall$, $\exists$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$, and *variables* are logical symbols
- *predicates*, *constants*, *functions* are non-logical symbols

First Order Logic: Quantifiers

$$\forall x \,.\, \underbrace{p(f(x), x) \Rightarrow (\exists y \,.\, \underbrace{p(f(g(x,y)), g(x,y)))}_{G} \wedge q(x, f(x))}_{F}$$

- The scope of $\forall x$ is F.
- The scope of $\exists y$ is G.
- x and y are *bound*.
- The formula reads:

  "For all x, if $p(f(x), x)$ then there exists a y such that $p(f(g(x,y)), g(x,y))$ and $q(x, f(x))$"

Examples

- $\forall x\,.\,(\text{even}(x) \vee \text{odd}(x)) \wedge \forall x\,.\,(\text{even}(x) \Leftrightarrow \neg\,\text{odd}(x))$
- $\forall x\,.\,\forall y\,.\,(x = y) \quad \Rightarrow \quad (f(x) = f(y))$
- $\forall z\,.\,\exists y\,.\, +(z, y) = 1$
- $\forall x, y, z\,.\,\text{triangle}(x, y, z) \Rightarrow \text{length}(x) < \text{length}(y) + \text{length}(z)$

Note:

- *even*, *odd*, *triangle*, and $<$ are identifiers representing arbitrary predicates
- f, $+$, and *length* are just identifiers representing some arbitrary functions
- 1 is just an identifier representing some arbitrary constant
- $x < y + z$ is *infix* notation for $<(x, +(y, z))$

Semantics

Definition (Model)

A model $\mathcal{M}$ of a formula F comprises

- a (non-empty) domain $\mathcal{D}$, and
- an *interpretation* function assigning meaning to non-logical symbols in F.

Semantics

Definition (Model)

A model $\mathcal{M}$ of a formula F comprises

- a (non-empty) domain $\mathcal{D}$, and
- an *interpretation* function assigning meaning to non-logical symbols in F.

For example:

- If c is a constant, then $c^{\mathcal{M}} \in \mathcal{D}$
- If f is a function of arity n, then $f^{\mathcal{M}} \in \mathcal{D}^n \to \mathcal{D}$
- If P is a predicate of arity n, then $P^{\mathcal{M}} \in \mathcal{D}^n \to \mathbb{B}$
- Note: $(f(t_1, \ldots, t_n))^{\mathcal{M}} = f^{\mathcal{M}}(t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}})$

Semantics

$\mathcal{M} \models F$ if and only if F is true in $\mathcal{M}$

- $\mathcal{M} \models R(t_1, \ldots, t_n)$ if and only if $R^{\mathcal{M}}(t_1^{\mathcal{M}}, \ldots t_n^{\mathcal{M}})$
- $\mathcal{M} \models (t_1 = t_2)$ if and only if $(t_1^{\mathcal{M}} = t_2^{\mathcal{M}})$
- $\mathcal{M} \models \neg F$ if and only if not $\mathcal{M} \models F$
- $\mathcal{M} \models F \wedge G$ if and only if $\mathcal{M} \models F$ *and* $\mathcal{M} \models G$
- $\mathcal{M} \models F \vee G$ if and only if $\mathcal{M} \models F$ *or* $\mathcal{M} \models G$
- $\mathcal{M} \models F \Rightarrow G$ if and only if $\mathcal{M} \models \neg F \vee G$
- $\mathcal{M} \models F \Leftrightarrow G$ if and only if $\mathcal{M} \models (F \Rightarrow G) \wedge (G \Rightarrow F)$

First Oder Logic Semantics: Example

$$F : p(f(x, y)z) \Rightarrow p(y, g(z, x))$$

$$\mathcal{D} = \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$$
$$f^{\mathcal{M}} = +, g^{\mathcal{M}} = -, p^{\mathcal{M}} = >$$

Therefore,

$$F^{\mathcal{M}} = (x + y) > z \Rightarrow (y > z - x)$$

- The variables x, y, and z are *free* in F

Semantics

- We can't determine the truth of a formula unless all variables are quantified
  - Un-quantified variables are *free*
  - Formulas in which all variables are quantified are *closed*
  - Closed formulas have no free variables

Semantics

$$\mathcal{M} \models \forall x \,.\, F(x)$$

- if and only if for every $m \in \mathcal{D}$, if we add a constant c to our language and extend $\mathcal{M}$ such that $c^{\mathcal{M}} = m$, then $\mathcal{M} \models F(c)$

Semantics

$$\mathcal{M} \models \forall x \,.\, F(x)$$

- if and only if for every $m \in \mathcal{D}$, if we add a constant c to our language and extend $\mathcal{M}$ such that $c^{\mathcal{M}} = m$, then $\mathcal{M} \models F(c)$
  - This trick is necessary since we can't refer to m directly

Semantics

$$\mathcal{M} \models \forall x \,.\, F(x)$$

- if and only if for every $m \in \mathcal{D}$, if we add a constant c to our language and extend $\mathcal{M}$ such that $c^{\mathcal{M}} = m$, then $\mathcal{M} \models F(c)$
  - This trick is necessary since we can't refer to m directly
- $\mathcal{M} \models \exists x \,.\, F(x)$ if and only if $\mathcal{M} \models \neg\forall x \,.\, \neg F(x)$
- Whether a closed formula F is true depends solely on $\mathcal{D}$ and the denotations of the non-logical symbols in F

Closed Formula: Example

Let $\mathcal{D} = \mathbb{Q}$, the set of rational numbers, and let $\times^{\mathcal{M}}$ be multiplication

$$\forall x \,.\, \exists y \,.\, 2 \times y = x$$

- Let $\mathcal{M}_1$ be $\mathcal{M}$ augmented with $c^{\mathcal{M}_1} = v$, $v \in \mathbb{Q}$
- Let $\mathcal{M}_2$ be $\mathcal{M}_1$ augmented with $d^{\mathcal{M}_2} = \frac{v}{2}$

Closed Formula: Example

Let $\mathcal{D} = \mathbb{Q}$, the set of rational numbers, and let $\times^{\mathcal{M}}$ be multiplication

$$\forall x \,.\, \exists y \,.\, 2 \times y = x$$

- Let $\mathcal{M}_1$ be $\mathcal{M}$ augmented with $c^{\mathcal{M}_1} = v$, $v \in \mathbb{Q}$
- Let $\mathcal{M}_2$ be $\mathcal{M}_1$ augmented with $d^{\mathcal{M}_2} = \frac{v}{2}$
- Then $\mathcal{M}_2 \models 2 \times d = c$

Closed Formula: Example

Let $\mathcal{D} = \mathbb{Q}$, the set of rational numbers, and let $\times^{\mathcal{M}}$ be multiplication

$$\forall x \,.\, \exists y \,.\, 2 \times y = x$$

- Let $\mathcal{M}_1$ be $\mathcal{M}$ augmented with $c^{\mathcal{M}_1} = v$, $v \in \mathbb{Q}$
- Let $\mathcal{M}_2$ be $\mathcal{M}_1$ augmented with $d^{\mathcal{M}_2} = \frac{v}{2}$
- Then $\mathcal{M}_2 \models 2 \times d = c$
- Therefore $\mathcal{M}_1 \models \exists y \,.\, 2 \times y = c$
- Therefore $\mathcal{M} \models \forall x \,.\, \exists y \,.\, 2 \times y = x$ (since $v \in \mathbb{Q}$ is arbitrary)

Satisfiability and Validity

- F is satisfiable iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is valid iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Satisfiability and Validity

- F is <u>satisfiable</u> iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is <u>valid</u> iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg \exists x \,.\, \neg P(x))$

Satisfiability and Validity

- F is satisfiable iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is valid iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg \exists x \,.\, \neg P(x))$
Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.P(x)) \Leftrightarrow (\neg \exists x \,.\neg P(x))$$

Satisfiability and Validity

- F is satisfiable iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is valid iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg \exists x \,.\, \neg P(x))$
Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.\, P(x)) \Leftrightarrow (\neg \exists x \,.\, \neg P(x))$$

Case 1:

1	$\mathcal{M} \models \forall x \,.\, P(x)$	(assumption)

Satisfiability and Validity

- F is satisfiable iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is valid iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg\exists x \,.\, \neg P(x))$
Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,. P(x)) \Leftrightarrow (\neg\exists x \,. \neg P(x))$$

Case 1:

1	$\mathcal{M} \models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \not\models \neg\exists x \,.\, \neg P(x)$	(assumption)

Satisfiability and Validity

- F is satisfiable iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is valid iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg\exists x \,.\, \neg P(x))$
Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.\, P(x)) \Leftrightarrow (\neg\exists x \,.\, \neg P(x))$$

Case 1:

1	$\mathcal{M} \models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \not\models \neg\exists x \,.\, \neg P(x)$	(assumption)
3	$\mathcal{M} \models \exists x \,.\, \neg P(x)$	(2 and $\neg$)

Satisfiability and Validity

- F is <u>satisfiable</u> iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is <u>valid</u> iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg \exists x \,.\, \neg P(x))$
Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.P(x)) \Leftrightarrow (\neg \exists x \,.\neg P(x))$$

<u>Case 1:</u>

1	$\mathcal{M} \models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \not\models \neg \exists x \,.\, \neg P(x)$	(assumption)
3	$\mathcal{M} \models \exists x \,.\, \neg P(x)$	(2 and $\neg$)
4	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \models \neg P(c)$	(3 and $\exists$ for some $v \in \mathcal{D}$)

Satisfiability and Validity

- F is <u>satisfiable</u> iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is <u>valid</u> iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg\exists x \,.\, \neg P(x))$
Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,. P(x)) \Leftrightarrow (\neg\exists x \,.\neg P(x))$$

<u>Case 1:</u>

1	$\mathcal{M} \models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \not\models \neg\exists x \,.\, \neg P(x)$	(assumption)
3	$\mathcal{M} \models \exists x \,.\, \neg P(x)$	(2 and $\neg$)
4	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \models \neg P(c)$	(3 and $\exists$ for some $v \in \mathcal{D}$)
5	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \models P(c)$	(1 and $\forall$)

Satisfiability and Validity

- F is satisfiable iff there exists $\mathcal{M}$ s.t. $\mathcal{M} \models F$
- F is valid iff for all $\mathcal{M}$ it holds that $\mathcal{M} \models F$

$$\boxed{F \text{ is valid iff } \neg F \text{ is unsatisfiable}}$$

Example: $F : (\forall x \,.\, P(x)) \Leftrightarrow (\neg\exists x \,.\, \neg P(x))$
Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.\, P(x)) \Leftrightarrow (\neg\exists x \,.\, \neg P(x))$$

Case 1:

1	$\mathcal{M} \models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \not\models \neg\exists x \,.\, \neg P(x)$	(assumption)
3	$\mathcal{M} \models \exists x \,.\, \neg P(x)$	(2 and $\neg$)
4	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \models \neg P(c)$	(3 and $\exists$ for some $v \in \mathcal{D}$)
5	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \models P(c)$	(1 and $\forall$)

4 & 5 are contradictory.

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.\, P(x)) \Leftrightarrow (\neg \exists x \,.\, \neg P(x))$$

<u>Case 2:</u>

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \, . P(x)) \Leftrightarrow (\neg \exists x \, . \neg P(x))$$

Case 2:

1 $\quad \mathcal{M} \not\models \forall x \, . \, P(x)$ (assumption)

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x\,.P(x)) \Leftrightarrow (\neg\exists x\,.\neg P(x))$$

<u>Case 2:</u>

1	$\mathcal{M} \not\models \forall x\,.\,P(x)$	(assumption)
2	$\mathcal{M} \models \neg\exists x\,.\,\neg P(x)$	(assumption)

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x\,.P(x)) \Leftrightarrow (\neg\exists x\,.\neg P(x))$$

<u>Case 2:</u>

1	$\mathcal{M} \not\models \forall x\,.\,P(x)$	(assumption)
2	$\mathcal{M} \models \neg\exists x\,.\,\neg P(x)$	(assumption)
3	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models P(c)$	(1 and $\forall$, for some $v \in \mathcal{D}$)

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x\,.P(x)) \Leftrightarrow (\neg\exists x\,.\neg P(x))$$

<u>Case 2:</u>

1	$\mathcal{M} \not\models \forall x\,.\,P(x)$	(assumption)
2	$\mathcal{M} \models \neg\exists x\,.\,\neg P(x)$	(assumption)
3	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models P(c)$	(1 and $\forall$, for some $v \in \mathcal{D}$)
4	$\mathcal{M} \not\models \exists x\,.\,\neg P(x)$	(2 and $\neg$)

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.P(x)) \Leftrightarrow (\neg\exists x \,.\neg P(x))$$

<u>Case 2:</u>

1	$\mathcal{M} \not\models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \models \neg\exists x \,.\, \neg P(x)$	(assumption)
3	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models P(c)$	(1 and $\forall$, for some $v \in \mathcal{D}$)
4	$\mathcal{M} \not\models \exists x \,.\, \neg P(x)$	(2 and $\neg$)
5	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models \neg P(c)$	(4 and $\exists$)

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.P(x)) \Leftrightarrow (\neg\exists x \,.\neg P(x))$$

<u>Case 2:</u>

1	$\mathcal{M} \not\models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \models \neg\exists x \,.\, \neg P(x)$	(assumption)
3	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models P(c)$	(1 and $\forall$, for some $v \in \mathcal{D}$)
4	$\mathcal{M} \not\models \exists x \,.\, \neg P(x)$	(2 and $\neg$)
5	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models \neg P(c)$	(4 and $\exists$)
6	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \models P(c)$	(5 and $\neg$)

Validity: Example continued

Suppose not. Then there is $\mathcal{M}$ such that

$$\mathcal{M} \not\models (\forall x \,.P(x)) \Leftrightarrow (\neg\exists x \,.\neg P(x))$$

<u>Case 2:</u>

1	$\mathcal{M} \not\models \forall x \,.\, P(x)$	(assumption)
2	$\mathcal{M} \models \neg\exists x \,.\, \neg P(x)$	(assumption)
3	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models P(c)$	(1 and $\forall$, for some $v \in \mathcal{D}$)
4	$\mathcal{M} \not\models \exists x \,.\, \neg P(x)$	(2 and $\neg$)
5	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \not\models \neg P(c)$	(4 and $\exists$)
6	$\mathcal{M} \cup \{c^{\mathcal{M}} \mapsto v\} \models P(c)$	(5 and $\neg$)

3 & 6 are contradictory.

Invalid Formula: Example

Show that the following formula is <u>not</u> valid:

$$(\forall x \,.\, P(x, x)) \Rightarrow (\exists x \,.\, \forall y \,.\, P(x, y))$$

Invalid Formula: Example

Show that the following formula is <u>not</u> valid:

$$(\forall x \,.\, P(x, x)) \Rightarrow (\exists x \,.\, \forall y \,.\, P(x, y))$$

Find model such $\mathcal{M}$ such that

$$\mathcal{M} \models \neg\,((\forall x \,.\, P(x, x)) \Rightarrow (\exists x \,.\, \forall y \,.\, P(x, y)))$$

Invalid Formula: Example

Show that the following formula is <u>not</u> valid:

$$(\forall x \,.\, P(x, x)) \Rightarrow (\exists x \,.\, \forall y \,.\, P(x, y))$$

Find model such $\mathcal{M}$ such that

$$\mathcal{M} \models \neg\,((\forall x \,.\, P(x, x)) \Rightarrow (\exists x \,.\, \forall y \,.\, P(x, y)))$$

i.e.,

$$\mathcal{M} \models (\forall x \,.\, P(x, x)) \wedge \neg(\exists x \,.\, \forall y \,.\, P(x, y))$$

Invalid Formula: Example

Show that the following formula is <u>not</u> valid:

$$(\forall x \,.\, P(x,x)) \Rightarrow (\exists x \,.\, \forall y \,.\, P(x,y))$$

Find model such $\mathcal{M}$ such that

$$\mathcal{M} \models \neg\,((\forall x \,.\, P(x,x)) \Rightarrow (\exists x \,.\, \forall y \,.\, P(x,y)))$$

i.e.,

$$\mathcal{M} \models (\forall x \,.\, P(x,x)) \land \neg(\exists x \,.\, \forall y \,.\, P(x,y))$$

Choose:

$$\begin{array}{lcl} \mathcal{D} & = & \{0,1\} \\ P^{\mathcal{M}} & = & \{(0,0),(1,1)\} \end{array}$$

First Order Logic: Safe Substitution

Example:

$$(\forall x \,.\, \overbrace{P(\underbrace{x}_{\text{bound}}, \underbrace{y}_{\text{free}})}^{\text{scope of } \forall x}) \Rightarrow Q(f(\underbrace{y}_{\text{free}}), \underbrace{x}_{\text{free}})$$

First Order Logic: Safe Substitution

Example:

$$(\forall x \,.\, \overbrace{P(\underbrace{x}_{\text{bound}}, \underbrace{y}_{\text{free}})}^{\text{scope of } \forall x}) \Rightarrow Q(f(\underbrace{y}_{\text{free}}), \underbrace{x}_{\text{free}})$$

Substitution:

$$\sigma : \{x \mapsto g(x), y \mapsto f(x), Q(f(y), x) \mapsto \exists x \,.\, H(x, y)\}$$

First Order Logic: Safe Substitution

Example:

$$(\forall x \,.\, \overbrace{P(\underbrace{x}_{\text{bound}}, \underbrace{y}_{\text{free}})}^{\text{scope of } \forall x}) \Rightarrow Q(f(\underbrace{y}_{\text{free}}), \underbrace{x}_{\text{free}})$$

Substitution:

$$\sigma : \{x \mapsto g(x), y \mapsto f(x), Q(f(y), x) \mapsto \exists x \,.\, H(x, y)\}$$

- Rename bound variable using fresh variable z:

$$(\forall z \,.\, P(z, y)) \Rightarrow Q(f(y), x)$$

First Order Logic: Safe Substitution

Example:

$$(\forall x \,.\, \overbrace{P(\underbrace{x}_{\text{bound}}, \underbrace{y}_{\text{free}})}^{\text{scope of } \forall x}) \Rightarrow Q(f(\underbrace{y}_{\text{free}}), \underbrace{x}_{\text{free}})$$

Substitution:

$$\sigma : \{x \mapsto g(x), y \mapsto f(x), Q(f(y), x) \mapsto \exists x \,.\, H(x, y)\}$$

- Rename bound variable using fresh variable z:

$$(\forall z \,.\, P(z, y)) \Rightarrow Q(f(y), x)$$

- Perform substitution:

$$(\forall z \,.\, P(z, f(x))) \Rightarrow \exists x \,.\, H(x, y)$$

First Order Logic: Safe Substitution

Example:

$$(\forall x \,.\, \overbrace{P(\underbrace{x}_{\text{bound}}, \underbrace{y}_{\text{free}})}^{\text{scope of } \forall x}) \Rightarrow Q(f(\underbrace{y}_{\text{free}}), \underbrace{x}_{\text{free}})$$

Substitution:

$$\sigma : \{x \mapsto g(x), y \mapsto f(x), Q(f(y), x) \mapsto \exists x \,.\, H(x, y)\}$$

- Rename bound variable using fresh variable z:

$$(\forall z \,.\, P(z, y)) \Rightarrow Q(f(y), x)$$

- Perform substitution:

$$(\forall z \,.\, P(z, f(x))) \Rightarrow \exists x \,.\, H(x, y)$$

- No free variable becomes bound during substitution!

First Order Logic: Substitution

Let σ be the substitution

$$\{F_1 \mapsto G_1, \ldots, F_n \mapsto G_n\}$$

such that $F_i \equiv G_i$ for $1 \leq i \leq n$.

If $F\sigma$ is a safe substitution, then $F \equiv F\sigma$

Formula Scheme

Substitution allows us to define *formula schemes*:

$$(\forall x \,.\, F) \Leftrightarrow (\neg\exists x \,.\neg F)$$

Here, F is a place holder!

Formula Scheme

Substitution allows us to define *formula schemes*:

$$(\forall x\,.\,F) \Leftrightarrow (\neg\exists x\,.\neg F)$$

Here, F is a place holder!
Formula scheme with side condition:

$$(\forall x\,.\,F) \Leftrightarrow F \qquad \text{provided } x \text{ not free in } F$$

Formula Scheme

Substitution allows us to define *formula schemes*:

$$(\forall x \,.\, F) \Leftrightarrow (\neg \exists x \,. \neg F)$$

Here, F is a place holder!
Formula scheme with side condition:

$$(\forall x \,.\, F) \Leftrightarrow F \qquad \text{provided } x \text{ not free in } F$$

A formula scheme is *valid*
if and only if
it is valid for *any* FOL formula (obeying the side conditions)

First Order Logic: Inference Rules

- *Inference rules* provide means to reason in FOL:

$$\frac{\text{premises}}{\text{conclusion}}$$

- For instance, for arbitrary formulas P, Q, R:

$$\frac{\neg\neg P}{P} \qquad \frac{P}{\neg\neg P} \qquad \frac{P \qquad Q}{P \wedge Q} \qquad \frac{P \wedge Q}{P} \qquad \frac{P \wedge Q}{Q \wedge P} \qquad \frac{P}{P \vee Q}$$

$$\frac{P \vee Q \qquad \neg P \vee R}{Q \vee R} \qquad \frac{P \Leftrightarrow Q \qquad Q}{P} \qquad \frac{P \Rightarrow Q \qquad Q \Rightarrow P}{P \Leftrightarrow Q}$$

First Order Logic: Derivations

- For instance:

$$\frac{\forall x\,.\,P(x) \vee \neg\forall y\,.\,Q(y) \qquad \forall y\,.\,Q(y)}{\forall x\,.\,P(x)}$$

- A *derivation* comprises a number of inference steps, e.g.:

$$\frac{\dfrac{\neg\neg P}{P} \qquad \dfrac{\neg R \wedge Q}{Q}}{P \wedge Q}$$

- We write $P \vdash Q$ if Q can be *derived* from P

First Order Logic: Derivations

- We can also use derivations in premises:

$$\frac{P \vdash Q \qquad P \vdash \neg Q}{\neg P} \qquad \text{(reductio ad absurdum)}$$

$$\frac{P \vdash Q}{P \Rightarrow Q} \qquad \text{(Deduction theorem)}$$

$$\frac{P \vee Q \quad P \vdash R \quad Q \vdash R}{R} \qquad \text{(Case analysis)}$$

First Order Logic: Substitution

- We use $P[t/x]$ to denote the replacement of *all free occurrences* of x in P by term t. Then

$$\frac{\forall x\, .P}{P[t/x]} \qquad \text{(universal instantiation)}$$

  if no *free* variable of t becomes bound during the substitution

First Order Logic: Substitution

- We use $P[t/x]$ to denote the replacement of *all free occurrences* of x in P by term t. Then

$$\frac{\forall x\,.P}{P[t/x]} \qquad \text{(universal instantiation)}$$

  if no *free* variable of t becomes bound during the substitution
- For instance:

$$\frac{\forall x\,.\ \mathsf{even}(x) \vee \mathsf{odd}(x)}{\mathsf{even}(1) \vee \mathsf{odd}(1)}$$

First Order Logic: Substitution

- We use $P[t/x]$ to denote the replacement of *all free occurrences* of x in P by term t. Then

$$\frac{\forall x\,.P}{P[t/x]} \qquad \text{(universal instantiation)}$$

  if no *free* variable of t becomes bound during the substitution
- For instance:

$$\frac{\forall x\,.\ \mathsf{even}(x) \lor \mathsf{odd}(x)}{\mathsf{even}(1) \lor \mathsf{odd}(1)}$$

- But <u>not</u>:

$$\frac{\forall x\,.\,\exists y\,.\,x = y}{(\exists y\,.\,x = y)[y + 1/x]}$$

First Order Logic: Substitution

- We use $P[t/x]$ to denote the replacement of *all free occurrences* of x in P by term t. Then

$$\frac{\forall x\,.P}{P[t/x]} \qquad \text{(universal instantiation)}$$

  if no *free* variable of t becomes bound during the substitution

- For instance:

$$\frac{\forall x\,.\ \mathsf{even}(x) \vee \mathsf{odd}(x)}{\mathsf{even}(1) \vee \mathsf{odd}(1)}$$

- But <u>not</u>:

$$\frac{\forall x\,.\,\exists y\,.\,x = y}{\exists y\,.\,y + 1 = y}$$

First Order Logic: Substitution

- Substitutions can also occur in the premise:

$$\frac{P[c/x]}{\exists x \,.\, P} \qquad \text{(existential generalization)}$$

where c is a constant and x must not occur free in $P[c/x]$

First Order Logic: Axioms

- An *axiom* is an inference rule without a premise:

$$\overline{\ P\ }$$

(We will omit the bar if it's clear that P is an axiom)

First Order Logic: Axioms

- An *axiom* is an inference rule without a premise:

$$\frac{}{P}$$

  (We will omit the bar if it's clear that P is an axiom)

- *Axioms* denote tautologies in a given theory, e.g.:

  $\forall x, y \,.\, (x + y) = (y + x)$

  $\forall x \,.\, \mathsf{even}(x) \vee \mathsf{odd}(x)$

  $\forall x \,.\, prime(x) \Leftrightarrow ((x > 1) \wedge \nexists i, j \,.(x = i \cdot j) \wedge (i > 1) \wedge (j > 1)))$

First Order Logic: Axioms

- An *axiom* is an inference rule without a premise:

$$\overline{P}$$

  (We will omit the bar if it's clear that P is an axiom)

- *Axioms* denote tautologies in a given theory, e.g.:

  $\forall x, y \,.\, (x + y) = (y + x)$
  $\forall x \,.\, \text{even}(x) \vee \text{odd}(x)$
  $\forall x \,.\, \textit{prime}(x) \Leftrightarrow ((x > 1) \wedge \nexists i, j \,.(x = i \cdot j) \wedge (i > 1) \wedge (j > 1)))$

- Can use axioms to determine the denotation of non-logical symbols

Axiom for Arrays

Array operations:

- *select(a,i)*
- *store(a,i,v)*

$$\forall i, j, a, v \,.\, \left(\begin{array}{c} (i = j) \wedge \mathrm{select}(\mathrm{store}(a, i, v), j) = v \\ \vee \\ \neg(i = j) \wedge \mathrm{select}(\mathrm{store}(a, i, v), j) = \mathrm{select}(a, j) \end{array} \right)$$

Axioms for Peano Arithmetic

Predicates and Functions:

- *N(x)* denotes $x \in \mathbb{N}$
  - "Syntactic sugar": $(\forall x \in \mathbb{N} \,.\, F)$ short for $(\forall x \,.\, N(x) \Rightarrow F)$
- *S(x)* denotes successor of x (i.e., $x + 1$)

Axioms for Peano Arithmetic

Predicates and Functions:

- *N(x)* denotes $x \in \mathbb{N}$
  - "Syntactic sugar": $(\forall x \in \mathbb{N} \,.\, F)$ short for $(\forall x \,.\, N(x) \Rightarrow F)$
- *S(x)* denotes successor of x (i.e., $x + 1$)

$$\forall x \in \mathbb{N} \,.\, 0 \neq S(x)$$

$$\forall x, y \in \mathbb{N} \,.\, (S(x) = S(y)) \Rightarrow (x = y)$$

$$\forall x \in \mathbb{N} \,.\, x + 0 = x$$

$$\forall x, y \in \mathbb{N} \,.\, (x + S(y)) = S(x + y)$$

$$\forall x \in \mathbb{N} \,.\, 0 \cdot x = 0$$

$$\forall x, y \in \mathbb{N} \,.\, (x \cdot S(y)) = x \cdot y + x$$

Peano Arithmetic: Natural Induction

- Quantification over sets of numbers impossible in FOL!
- Induction requires (countably) infinitely many axioms

Induction schema: For each formula F

$$\forall y_0, \ldots, y_n \in \mathbb{N} \,.\, \left(\begin{array}{c} F(0, y_0, \ldots, y_n) \\ \wedge \\ \forall x \in \mathbb{N} \,.\, (F(x, y_0, \ldots, y_n) \Rightarrow F(S(x), y_0, \ldots, y_n)) \end{array} \right) \Rightarrow \forall x \in \mathbb{N} \,.\, F(x, y_0, \ldots, y_n)$$

Ordered Semi-rings

$\forall x, y, z \in \mathbb{N} \,.\, (x + y) + z = x + (y + z)$	Addition associative
$\forall x, y \in \mathbb{N} \,.\, x + y = y + x$	Addition commutative
$\forall x, y, z \in \mathbb{N} \,.\, (x \cdot y) \cdot z = x \cdot (y \cdot z)$	Multiplication associative
$\forall x, y \in \mathbb{N} \,.\, x \cdot y = y \cdot x$	Multiplication commutative
$\forall x, y, z \in \mathbb{N} \,.\, x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	Distributive law
$\forall x \in \mathbb{N} \,.\, x + 0 = x \wedge x \cdot 0 = 0$	Identity for addition
$\forall x \in \mathbb{N} \,.\, x \cdot 1 = x$	Identity for multiplication
$\forall x, y, z \in \mathbb{N} \,.\, x < y \wedge y < z \Rightarrow x < z$	Transitivity of $<$
$\forall x \in \mathbb{N} \,.\, \neg(x < x)$	$<$ is irreflexive
$\forall x, y \in \mathbb{N} \,.\, (x < y) \vee (y < x)$	Total order
$\forall x, y, z \in \mathbb{N} \,.\, (x < y) \Rightarrow (x + z < y + z)$	
$\forall x, y, z \in \mathbb{N} \,.\, (0 < z \wedge x < y) \Rightarrow (x \cdot z < y \cdot z)$	
$\forall x, y \in \mathbb{N} \,.\, (x < y) \Rightarrow \exists z \in \mathbb{N} \,.\, x + z = y$	
$0 < 1 \wedge \forall x \in \mathbb{N} \,.\, (x > 0) \Rightarrow (x \geq 1)$	
$\forall x \in \mathbb{N} \,.\, x \geq 0$	

Example Revisited

$$\forall x \,.\, (x + 1) > x$$

- Valid in the theory of arithmetic
- Not valid in the theory of bit-vectors
- *Undefined* in the C++ language

Formal Specifications

- First-Order Logic allows for unambiguous specifications.
- Recall *coverage*:
  - Can axiomatize $\mathsf{defs}(\mathsf{x})$, $\mathsf{p\text{-}use}(\mathsf{x})$, $\mathsf{c\text{-}use}(\mathsf{x})$, $\mathsf{path}(p, \ell, \ell')$, $\mathsf{def\text{-}clear}(p, \mathsf{x})$, $\mathsf{dpu}(\ell, \mathsf{x})$, $\mathsf{dcu}(\ell, \mathsf{x})$, ...
  - Paths sufficient to achieve all-c-uses:

$$\forall \mathsf{x} \,.\, \forall \ell \in \mathsf{defs}(\mathsf{x}) \,.\, \forall \ell' \in \mathsf{dcu}(\ell, \mathsf{x}) \,.\, \exists p \in \mathsf{Paths} \,.\\ \mathsf{path}(p, \ell, \ell') \wedge \mathsf{def\text{-}clear}(p, \mathsf{x})$$

Summary

- Logic enables unambiguous specifications
- Next time: how to reason about programs!