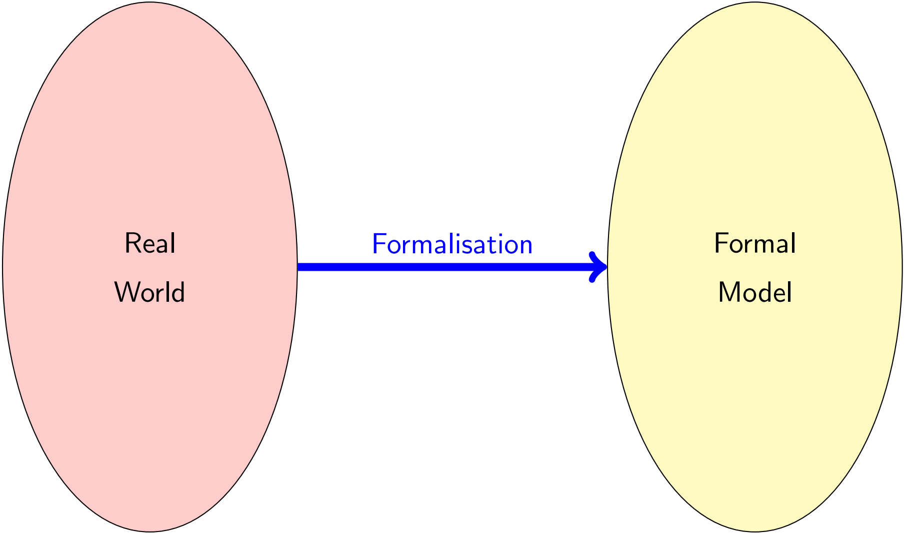
# Formal Specification and Verification

## First-Order Logic

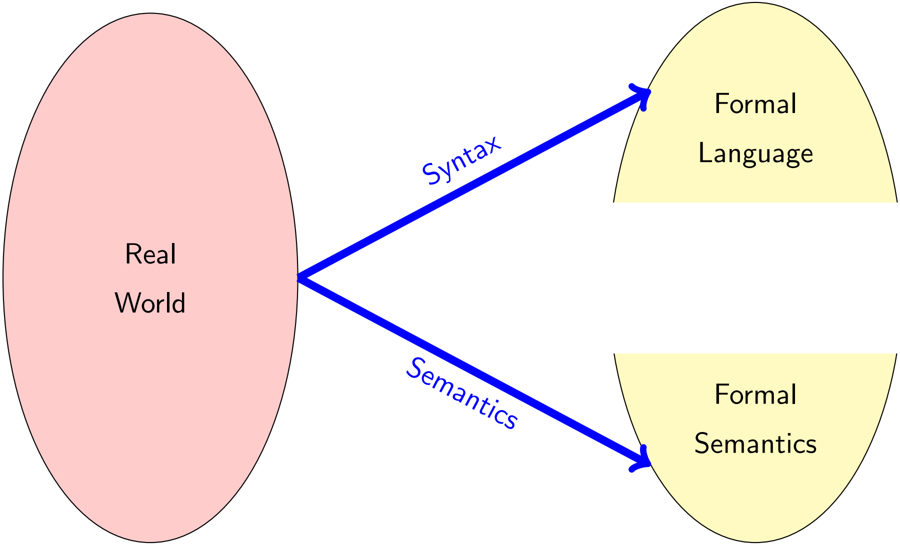
Bernhard Beckert

Based on a lecture by Wolfgang Ahrendt and Reiner Hähnle at Chalmers University, Göteborg

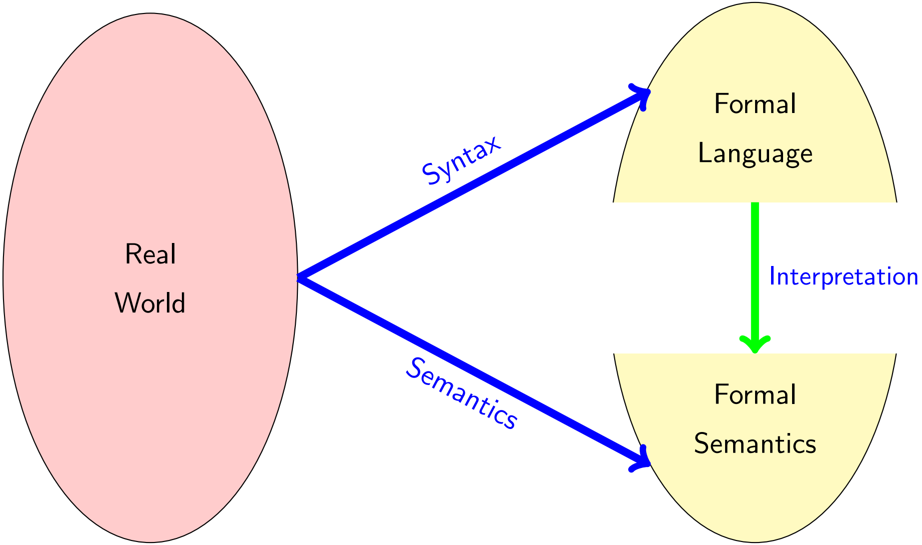
# Formalisation



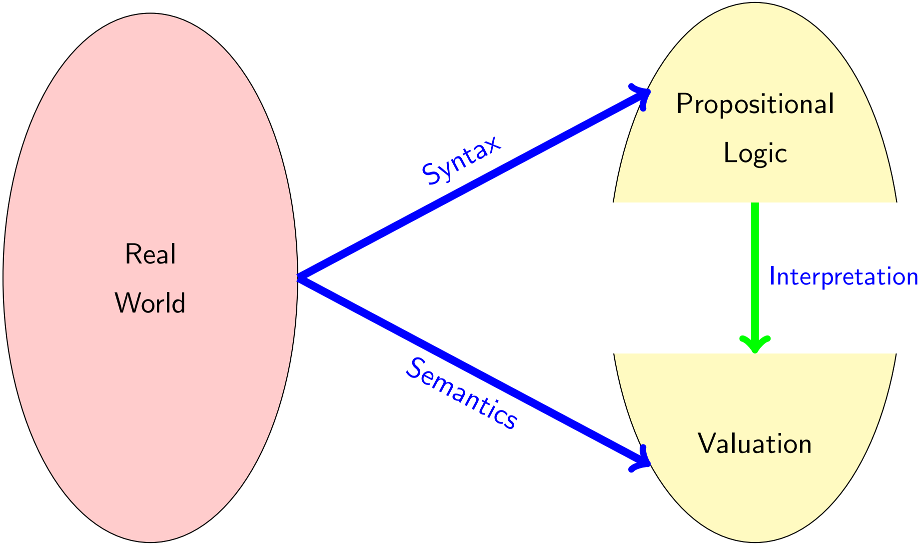
# Formalisation: Syntax, Semantics



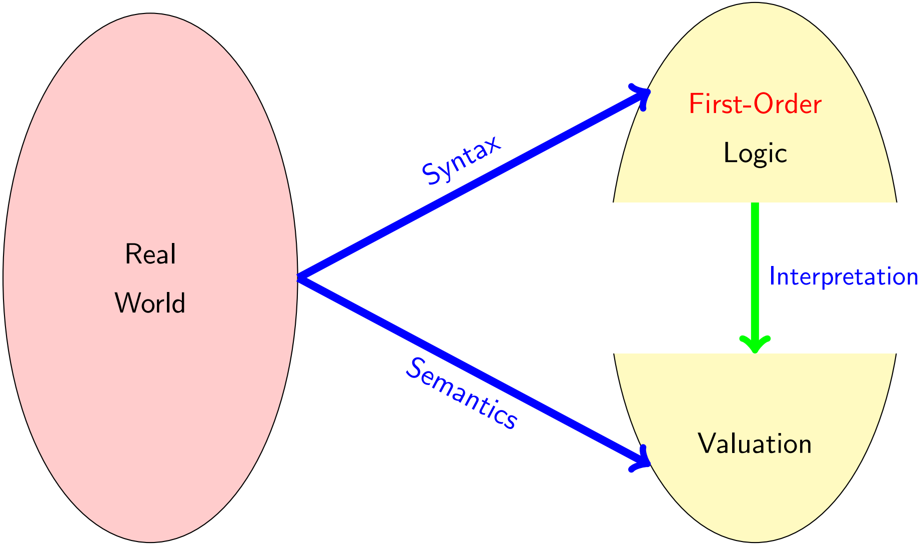
# Formalisation: Syntax, Semantics



# Formalisation: Syntax, Semantics



# Formalisation: Syntax, Semantics



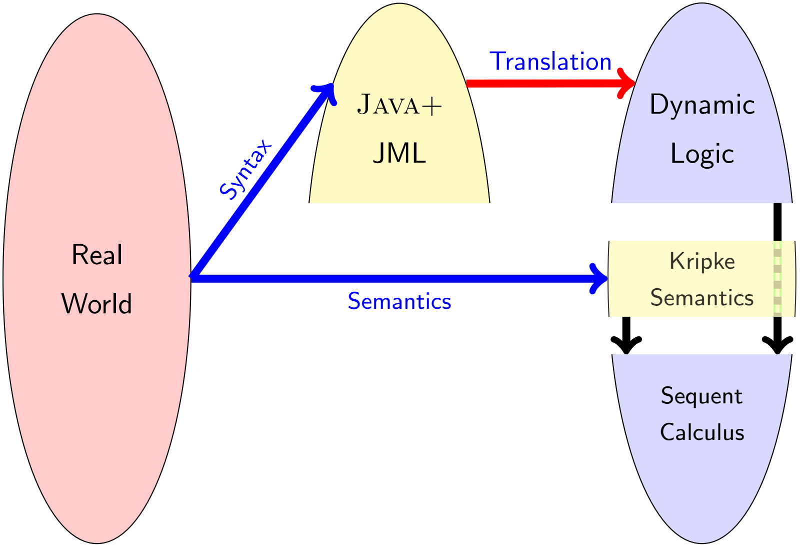
# Approaches to Formal Software Verification

KeY
2nd part
of course

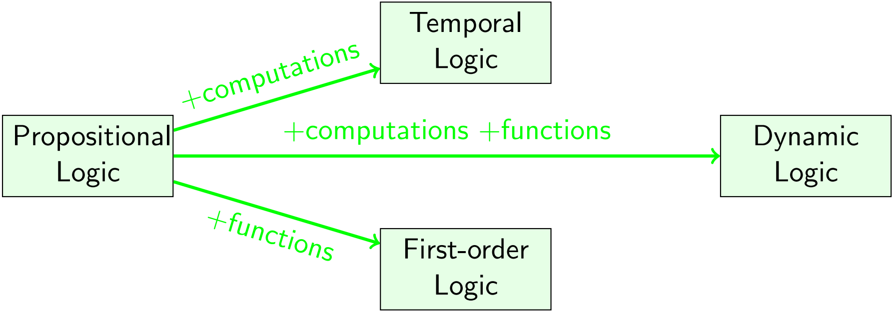
| | |
|---|---|
| Concrete programs, Complex properties | Concrete programs, Simple properties |
| Abstract programs, Complex properties | Abstract programs, Simple properties |

Spin
1st part
of course

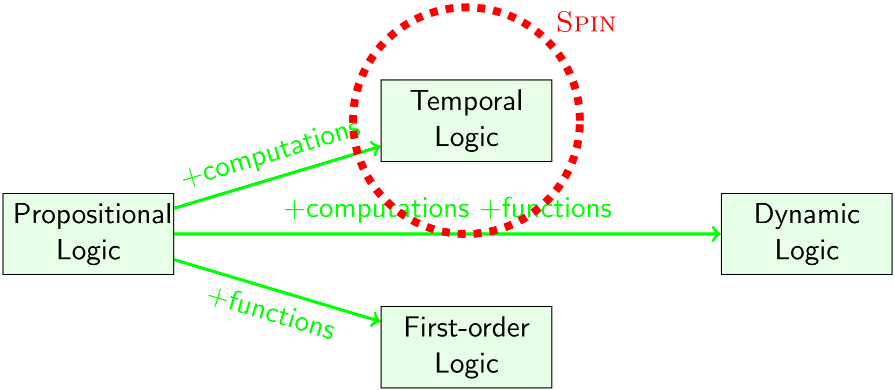
# Formal Verification: Deduction



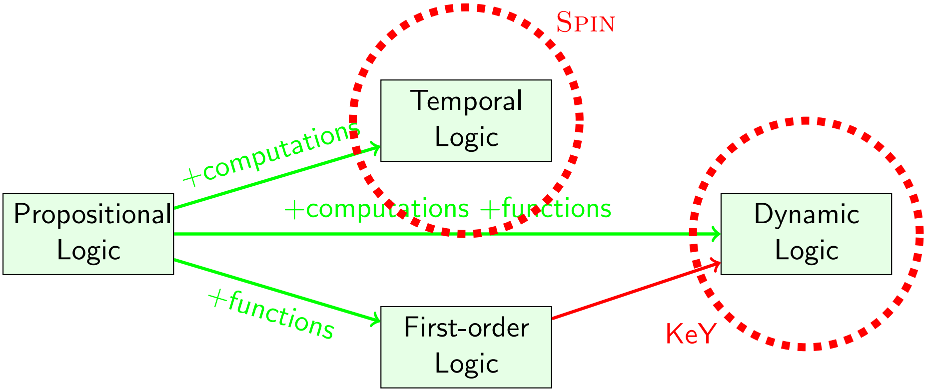
# Beyond Propositional Logic



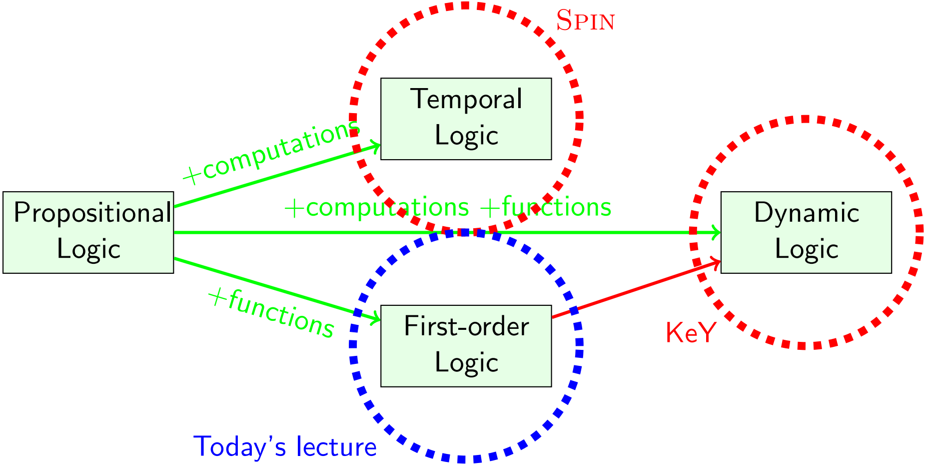
# Beyond Propositional Logic



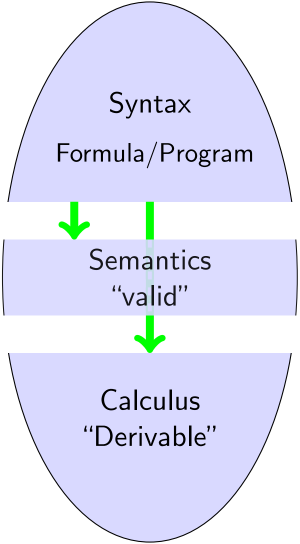
# Beyond Propositional Logic



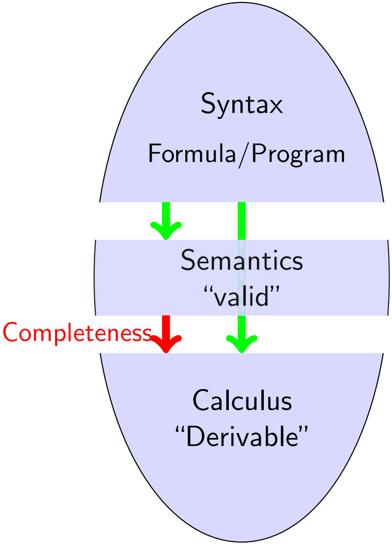
# Beyond Propositional Logic



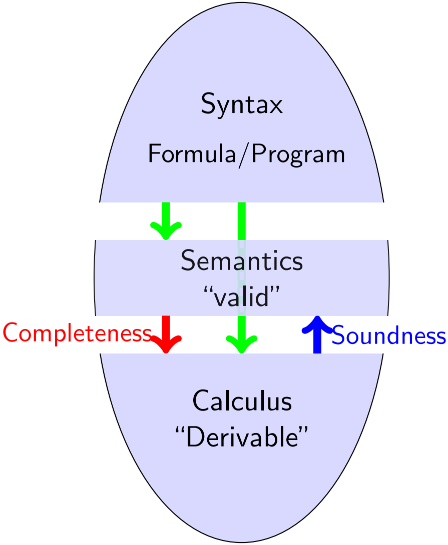
# Syntax, Semantics, Calculus



# Syntax, Semantics, Calculus



# Syntax, Semantics, Calculus



# Limitations of Propositional Logic

**Fixed, finite number of objects**

Cannot express: let $g$ be group with arbitrary number of elements

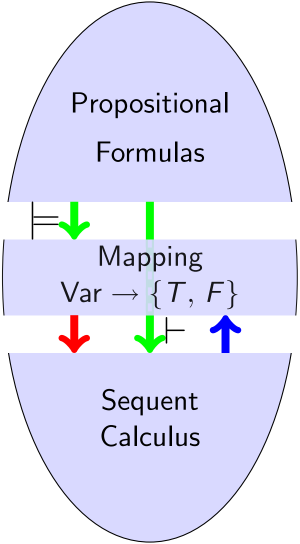
**No functions or relations with arguments**

Can express: finite function/relation table with indexed variables $p_{ij}$
Cannot express:
properties of function/relation on all arguments, e.g., "+" is associative

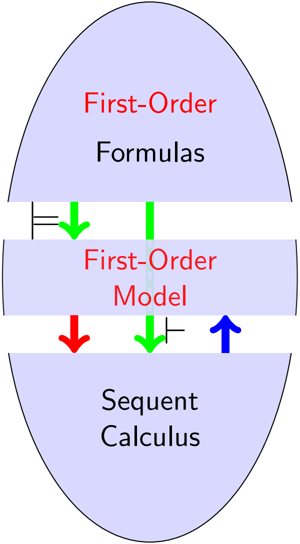
**Static interpretation**

Programs change value of their variables, e.g., via assignment, call, etc.
Propositional formulas look at one single interpretation at a time

# Propositional Logic



# First-Order Logic



# Syntax of First-Order Logic: Signature

**Definition (First-Order Signature)**

First-order signature $\Sigma = (\mathrm{PSym}, \mathrm{FSym}, \alpha)$

| | |
|---|---|
| Predicate or Relation Symbols | $\mathrm{PSym} = \{p_i \mid i \in \mathbb{N}\}$ |
| Function Symbols | $\mathrm{FSym} = \{f_i \mid i \in \mathbb{N}\}$ |

Typing function $\alpha$, set of types $\mathcal{T}$

- $\alpha(p) \in \mathcal{T}^*$ for all $p \in \mathrm{PSym}$
- $\alpha(f) \in \mathcal{T}^* \times \mathcal{T}$ for all $f \in \mathrm{FSym}$

**Definition (Variables)**

$\mathrm{VSym} = \{x_i \mid i \in \mathbb{N}\}$ set of typed variables

- In contrast to "standard" FOL, our symbols are typed
  Necessary to model a typed programming language such as JAVA!
- Allow any non-reserved name for symbols, not merely $p_3$, $f_{17}, \ldots$

# Syntax of First-Order Logic: Signature Cont'd

**Declaration of signature symbols**

- Write $T\ x$; to declare variable $x$ of type $T$
- Write $p(T_1, \ldots, T_r)$; for $\alpha(p) = (T_1, \ldots, T_r)$
- Write $T\ f(T_1, \ldots, T_r)$; for $\alpha(f) = ((T_1, \ldots, T_r),\ T)$

Similar convention as in JAVA, no overloading of symbols
Case $r = 0$ is allowed, then write $p$ instead of $p()$, etc.

# Syntax of First-Order Logic: Signature Cont'd

**Declaration of signature symbols**

- Write $T\ x$; to declare variable $x$ of type $T$
- Write $p(T_1, \ldots, T_r)$; for $\alpha(p) = (T_1, \ldots, T_r)$
- Write $T\ f(T_1, \ldots, T_r)$; for $\alpha(f) = ((T_1, \ldots, T_r),\ T)$

Similar convention as in JAVA, no overloading of symbols
Case $r = 0$ is allowed, then write $p$ instead of $p()$, etc.

**Example**

**Variables** `integerArray a; int i;`

**Predicates** `isEmpty(List); alertOn;`

**Functions** `int arrayLookup(int); java.lang.Object o;`

# OO Type Hierarchy

We want to model the behaviour of JAVA programs
Admissible types $\mathcal{T}$ form object-oriented type hierarchy

## OO Type Hierarchy

We want to model the behaviour of JAVA programs
Admissible types $\mathcal{T}$ form object-oriented type hierarchy

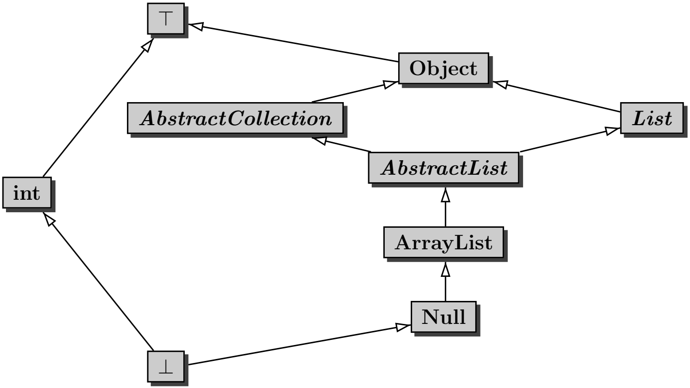
**Definition (OO Type Hierarchy)**

- $\mathcal{T}$ is finite set of types (not parameterized)
- Given subtype relation $\sqsubseteq$, assume $\mathcal{T}$ $\sqcap$-closed
- Dynamic types $\mathcal{T}_d \subseteq \mathcal{T}$, where $\top \in \mathcal{T}_d$
- Abstract types $\mathcal{T}_a \subseteq \mathcal{T}$, where $\bot \in \mathcal{T}_a$
- $\mathcal{T}_d \cap \mathcal{T}_a = \emptyset$
- $\mathcal{T}_d \cup \mathcal{T}_a = \mathcal{T}$
- $\bot \sqsubseteq T \sqsubseteq \top$ for all $T \in \mathcal{T}$

OO Type Hierarchy Cont'd

**Example**

Using UML notation



## OO Type Hierarchy Cont'd

- Dynamic types are those with direct elements
- Abstract types for abstract classes and interfaces
- In Java primitive (value) and object types incomparable
- $\perp$ is abstract and hence no object ever can have this type
  $\perp$ cannot occur in declaration of signature symbols
- Each abstract type except $\perp$ has a non-empty dynamic subtype
- In Java $\top$ is chosen to have no direct elements
- Java has infinitely many types: **int**[], **int**[][],...
  Restrict $\mathcal{T}$ to the finitely many types that occur in a given program

## OO Type Hierarchy Cont'd

- Dynamic types are those with direct elements
- Abstract types for abstract classes and interfaces
- In JAVA primitive (value) and object types incomparable
- $\bot$ is abstract and hence no object ever can have this type
  $\bot$ cannot occur in declaration of signature symbols
- Each abstract type except $\bot$ has a non-empty dynamic subtype
- In JAVA $\top$ is chosen to have no direct elements
- JAVA has infinitely many types: **int**[], **int**[][],...
  Restrict $\mathcal{T}$ to the finitely many types that occur in a given program

**Example (The Minimal Type Hierarchy)**

$\mathcal{T} = \{\bot, \top\}$
All signature symbols have same type $\top$: drop type, untyped logic

## Reserved Signature Symbols

**Reserved signature symbols**

- Equality symbol $\doteq \in \mathrm{PSym}$ declared as $\doteq(\top, \top)$

  Written infix: $x \doteq 0$
- Type predicate symbol $\sqsubseteq T \in \mathrm{PSym}$ for each $T \in \mathcal{T}$

  Declared as $\sqsubseteq T(\top)$

  Written postfix: $\mathtt{i}\sqsubseteq\mathbf{int}$ — read "instance of"
- Type cast symbol $(T) \in \mathrm{FSym}$ for each $T \in \mathcal{T}$

  Declared as $T\ (T)(\top)$

## Reserved Signature Symbols

**Reserved signature symbols**

- Equality symbol $\doteq \in \mathrm{PSym}$ declared as $\doteq(\top, \top)$

  Written infix: $x \doteq 0$
- Type predicate symbol $\sqsubseteq T \in \mathrm{PSym}$ for each $T \in \mathcal{T}$

  Declared as $\sqsubseteq T(\top)$

  Written postfix: $\mathtt{i} \sqsubseteq \mathbf{int}$ — read "instance of"
- Type cast symbol $(T) \in \mathrm{FSym}$ for each $T \in \mathcal{T}$

  Declared as $T\ (T)(\top)$

So far, we have a type system and a signature — where is the logic?

# Terms

### First-order terms, informally

- Think of first-order terms as expressions in a programming language
  Built up from variables, constants, function symbols
- First-order terms have no side effects (like PROMELA, unlike JAVA)
- First-order terms have a type and must respect type hierarchy
  - type of $f(g(x))$ is result type in declaration of function $f$
  - in $f(g(x))$ the result type of $g$ is subtype of argument type of $f$, etc.

# Terms

## First-order terms, informally

- Think of first-order terms as expressions in a programming language
  Built up from variables, constants, function symbols
- First-order terms have no side effects (like PROMELA, unlike JAVA)
- First-order terms have a type and must respect type hierarchy
  - type of $f(g(x))$ is result type in declaration of function $f$
  - in $f(g(x))$ the result type of $g$ is subtype of argument type of $f$, etc.

## Definition (First-Order Terms $\{\mathbf{Term}_T\}_{T \in \mathcal{T}}$ with type $T \in \mathcal{T}$)

- $x$ is term of type $T$ for variable declared as $T\ x$;
- $f(t_1, \ldots, t_r)$ is term of type $T$ for
  - function symbol declared as $T\ f(T_1, \ldots, T_r)$; and
  - terms $t_i$ of type $T_i' \sqsubseteq T_i$ for $1 \leq i \leq r$
- There are no other terms (inductive definition)

# Terms, Cont'd

**Example**

Signature: **int** `i`; **short** `j`; `List l`; **int** `f(`**int**`)`;

- `f(i)` has result type **int** and is contained in $\text{Term}_{\textbf{int}}$
- `f(j)` has result type **int** (when **short** $\sqsubseteq$ **int**)
- `f(l)` is ill-typed (when **int**, `List` incomparable)
- `f(i,i)` is not a term (doesn't match declaration)
- (**int**)`j` is term of type **int**
- even (**int**)`l` is term of type **int** (type cast always well-formed)

# Terms, Cont'd

**Example**

Signature: **int** `i`; **short** `j`; `List l`; **int** `f(`**int**`)`;

- `f(i)` has result type **int** and is contained in $\text{Term}_{\mathbf{int}}$
- `f(j)` has result type **int** (when **short** $\sqsubseteq$ **int**)
- `f(l)` is ill-typed (when **int**, `List` incomparable)
- `f(i,i)` is not a term (doesn't match declaration)
- `(`**int**`)j` is term of type **int**
- even `(`**int**`)l` is term of type **int** (type cast always well-formed)

- If $f$ is constant ($r = 0$) write $f$ instead of $f()$
- Use infix notation liberally, where appropriate:
  declare **int** `+(`**int**`,` **int**`)`; then write `i+j`, etc.
- Use brackets to disambiguiate parsing:
  `(i+j)*i`

# First-Order Atomic Formulas

**Definition (Atomic First-Order Formulas)**

$p(t_1, \ldots, t_r)$ is atomic first-order formula for

- predicate symbol declared as $p(T_1, \ldots, T_r)$; and
- terms $t_i$ of type $T'_i \sqsubseteq T_i$ for $1 \leq i \leq r$

# First-Order Atomic Formulas

**Definition (Atomic First-Order Formulas)**

$p(t_1, \dots, t_r)$ is atomic first-order formula for

- predicate symbol declared as $p(T_1, \dots, T_r)$; and
- terms $t_i$ of type $T'_i \sqsubseteq T_i$ for $1 \leq i \leq r$

**Example**

Signature: $\mathbf{int}$ `i;` $\mathbf{short}$ `j;` `List l;` `<(`$\mathbf{int}$`,` $\mathbf{int}$`);`

- `i < i` is an atomic first-order formula
- `i < j` is an atomic first-order formula (when $\mathbf{short} \sqsubseteq \mathbf{int}$)
- `i < l` is ill-typed (when $\mathbf{int}$, `List` incomparable)
- `i`$\doteq$`j` and even `i`$\doteq$`l` are atomic first-order formulas
- `i`$\sqsubseteq$$\mathbf{short}$ is an atomic first-order formula

# First-Order Formulas

**Definition (Set of First-Order Formulas *For*)**

- Truth constants true, false and all first-order atomic formulas are first-order formulas
- If $\phi$ and $\psi$ are first-order formulas then

$$!\,\phi, \quad (\phi \;\&\; \psi), \quad (\phi \mid \psi), \quad (\phi \;\texttt{->}\; \psi), \quad (\phi \;\texttt{<->}\; \psi)$$

  are also first-order formulas
- If $T\ x$ is a variable declaration, $\phi$ a first-order formula, then $\forall\ T\ x;\ \phi$ and $\exists\ T\ x;\ \phi$ are first-order formulas
  Any occurrence of $x$ in $\phi$ must be well-typed

- $\forall\ T\ x;\ \phi$ called universally quantified formula
- $\exists\ T\ x;\ \phi$ called existentially quantified formula

# First-Order Formulas Cont'd

- In $\forall\, T\ x;\ \phi$ and $\exists\, T\ x;\ \phi$ call $\phi$ the scope of $x$ bound by $\forall/\exists$
- Analogy between variables bound in quantified formulas and program locations declared as local variables/formal parameters

We require that all variables occur bound
$\Rightarrow$ All variable declarations are quantifier-local

**Example**

- $\forall\, \mathbf{int}\ i;\ \exists\, \mathbf{int}\ j;\ i < j$ is a first-order formula
- $\forall\, \mathbf{int}\ i;\ \exists\, \texttt{List}\ l;\ i < l$ is ill-typed
- $\forall\, \mathbf{int}\ i;\ i < j$ is a first-order formula
  if $j$ is a constant compatible with **int**
- $(\forall\, \mathbf{int}\ i;\ \forall\, \mathbf{int}\ j;\ i < j)\ \mid\ (\forall\, \mathbf{int}\ i;\ \forall\, \mathbf{int}\ j;\ i > j)$
  is a first-order formula

# Remark on Concrete Syntax

<table>
<tr><th></th><th>Text book</th><th>SPIN</th><th>KeY</th><th>JAVA</th></tr>
<tr><td>Negation</td><td>$\neg$</td><td>!</td><td>!</td><td>!</td></tr>
<tr><td>Conjunction</td><td>$\wedge$</td><td>&&</td><td>&</td><td>&&</td></tr>
<tr><td>Disjunction</td><td>$\vee$</td><td>||</td><td>|</td><td>||</td></tr>
<tr><td>Implication</td><td>$\rightarrow, \supset$</td><td>-></td><td>-></td><td>n/a</td></tr>
<tr><td>Equivalence</td><td>$\leftrightarrow$</td><td><-></td><td><-></td><td>n/a</td></tr>
<tr><td>Universal Quantifier</td><td>$\forall x; \phi$</td><td>n/a</td><td>\forall $T$ $x$; $\phi$</td><td>n/a</td></tr>
<tr><td>Existential Quantifier</td><td>$\exists x; \phi$</td><td>n/a</td><td>\exists $T$ $x$; $\phi$</td><td>n/a</td></tr>
<tr><td>Value equality</td><td>$\doteq$</td><td>==</td><td>=</td><td>==</td></tr>
</table>

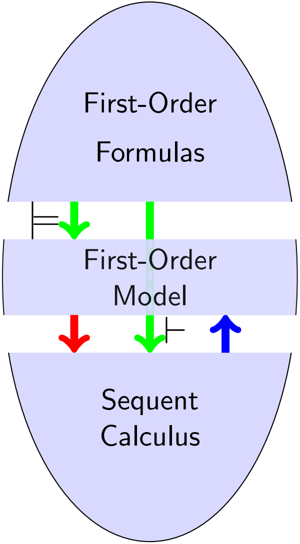
# Remark on Concrete Syntax

<table>
<tr><th></th><th>Text book</th><th>SPIN</th><th>KeY</th><th>JAVA</th></tr>
<tr><td>Negation</td><td>$\neg$</td><td>!</td><td>!</td><td>!</td></tr>
<tr><td>Conjunction</td><td>$\wedge$</td><td>&&</td><td>&</td><td>&&</td></tr>
<tr><td>Disjunction</td><td>$\vee$</td><td>||</td><td>|</td><td>||</td></tr>
<tr><td>Implication</td><td>$\rightarrow$, $\supset$</td><td>-></td><td>-></td><td>n/a</td></tr>
<tr><td>Equivalence</td><td>$\leftrightarrow$</td><td><-></td><td><-></td><td>n/a</td></tr>
<tr><td>Universal Quantifier</td><td>$\forall x; \phi$</td><td>n/a</td><td>\forall $T$ $x$; $\phi$</td><td>n/a</td></tr>
<tr><td>Existential Quantifier</td><td>$\exists x; \phi$</td><td>n/a</td><td>\exists $T$ $x$; $\phi$</td><td>n/a</td></tr>
<tr><td>Value equality</td><td>$\doteq$</td><td>==</td><td>=</td><td>==</td></tr>
</table>

For quantifiers we normally use textbook syntax and suppress type information to ease readability

For propositional connectives we use KeY syntax

# First-Order Semantics



# First-Order Semantics

**From propositional to first-order semantics**

- In prop. logic, an interpretation of variables with $\{T, F\}$ sufficed
- In first-order logic we must assign meaning to:
  - variables bound in quantifiers
  - constant and function symbols
  - predicate symbols
- Each variable or function value may denote a different object
- Respect typing: `int i`, `List l` must denote different objects

**What we need (to interpret a first-order formula)**

1. A collection of typed universes of objects (akin to heap objects)
2. A mapping from variables to objects
3. A mapping from function arguments to function values
4. The set of argument tuples where a predicate is true

# First-Order Domains/Universes

1. A collection of typed universes of objects

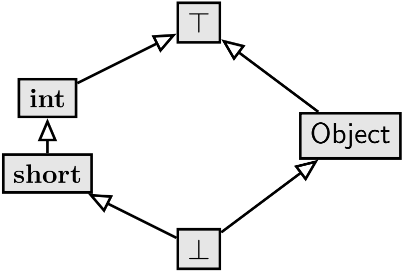
**Definition (Universe/Domain)**

A non-empty set $\mathcal{D}$ of objects is a universe or domain
Each element of $\mathcal{D}$ has a fixed type given by $\delta : \mathcal{D} \rightarrow \mathcal{T}_d$

- Like heap objects and values in JAVA
- Notation for the domain elements type-compatible with $T \in \mathcal{T}$: $\mathcal{D}^T = \{d \in \mathcal{D} \mid \delta(d) \sqsubseteq T\}$
- For each dynamic type $T \in \mathcal{T}_d$ there must be at least one domain element type-compatible with it: $\mathcal{D}^T \neq \emptyset$

# First-Order Universes Cont'd

## Example




- $\mathcal{D} = \{17, o\}$
- $\delta(17) = \mathbf{short}$, $\delta(o) = \mathsf{Object}$
- Then $\mathcal{D}^{\mathbf{short}} = \mathcal{D}^{\mathbf{int}} = \{17\}$, $\mathcal{D}^{\mathsf{Object}} = \{o\}$, $\mathcal{D}^{\top} = \mathcal{D} = \{17, o\}$, and $\mathcal{D}^{\perp} = \{\}$

# First-Order Models

3. A mapping from function arguments to function values
4. The set of argument tuples where a predicate is true

**Definition (First-Order Model)**

Let $\mathcal{D}$ be a domain with typing function $\delta$

Let $f$ be declared as $T\ f(T_1, \ldots, T_r)$;

Let $p$ be declared as $p(T_1, \ldots, T_r)$;

Let $\mathcal{I}(f) : \mathcal{D}^{T_1} \times \cdots \times \mathcal{D}^{T_r} \to \mathcal{D}^{T}$

Let $\mathcal{I}(p) \subseteq \mathcal{D}^{T_1} \times \cdots \times \mathcal{D}^{T_r}$

Then $\mathcal{M} = (\mathcal{D}, \delta, \mathcal{I})$ is a first-order model

# First-Order Models Cont'd

## Example

Signature: `int i; short j; int f(int); Object obj; <(int,int);`
$\mathcal{D} = \{17, 2, o\}$ where all numbers are short

$$\mathcal{I}(i) = 17$$
$$\mathcal{I}(j) = 17$$
$$\mathcal{I}(\texttt{obj}) = o$$

| $\mathcal{D}^{\textbf{int}}$ | $\mathcal{I}(f)$ |
|---|---|
| 2 | 2 |
| 17 | 2 |

| $\mathcal{D}^{\textbf{int}} \times \mathcal{D}^{\textbf{int}}$ | in $\mathcal{I}(<)$? |
|---|---|
| $(2, 2)$ | $F$ |
| $(2, 17)$ | $T$ |
| $(17, 2)$ | $F$ |
| $(17, 17)$ | $F$ |

One of uncountably many possible first-order models!

## Semantics of Reserved Signature Symbols

**Definition**

- Equality symbol $\doteq$ declared as $\doteq(\top, \top)$

  Model is fixed as $\mathcal{I}(\doteq) = \{(d, d) \mid d \in \mathcal{D}\}$
  "Referential Equality" (holds if arguments refer to identical object)

  Exercise: write down the predicate table for example domain

- Type predicate symbol $\sqsubseteq T$ for any $T$, declared as $\sqsubseteq T(\top)$

  $\mathcal{I}(\sqsubseteq T) = \mathcal{D}^T$

  Exercise: what is $\mathcal{I}(\sqsubseteq\text{Object})$?

- Type cast symbol $(T)$ for each $T$, declared as $T\ (T)(\top)$

  Casts that succeed ($\delta(x) \sqsubseteq T$): $\mathcal{I}((T))(x) = x$ identity
  Casts that do not succeed: $\mathcal{I}((T))(x) = d$ arb. fixed $d \in \mathcal{D}^T$

  Exercise: what is $\mathcal{I}((\mathbf{int}))(17)$?

## Signature Symbols vs. Domain Elements

- Domain elements different from the terms representing them
- First-order formulas and terms have no access to domain
- As in JAVA: identity and memory layout of values/objects hidden
- Think of a first-order model as a “heap” of first-order logic

**Example**

Signature: `Object obj1, obj2;`
Domain: $\mathcal{D} = \{o\}$

In this model, necessarily $\mathcal{I}(\texttt{obj1}) = \mathcal{I}(\texttt{obj2}) = o$

Effect similar to aliasing in JAVA with reference types

## Variable Assignments

**2.** A mapping from variables to objects

Think of variable assignment as environment for storage of local variables

**Definition (Variable Assignment)**

A variable assignment $\beta$ maps variables to domain elements
It respects the variable type, i.e., if $x$ has type $T$ then $\beta(x) \in \mathcal{D}^T$

**Definition (Modified Variable Assignment)**

Let $y$ be variable of type $T$, $\beta$ variable assignment, $d \in \mathcal{D}^T$:

$$\beta_y^d(x) := \begin{cases} \beta(x) & x \neq y \\ d & x = y \end{cases}$$

## Semantic Evaluation of Terms

Given a first-order model $\mathcal{M}$ and a variable assignment $\beta$
it is possible to evaluate first-order terms under $\mathcal{M}$ and $\beta$

**Analogy**

Evaluating an expression in a programming language
with respect to a given heap ($\mathcal{M}$) and binding of local variables ($\beta$)

**Definition (Valuation of Terms)**

$val_{\mathcal{M},\beta} : \text{Term} \to \mathcal{D}$ such that $val_{\mathcal{M},\beta}(t) \in \mathcal{D}^T$ for $t \in \text{Term}_T$:

- $val_{\mathcal{M},\beta}(x) = \beta(x)$ (recall that $\beta$ respects typing)
- $val_{\mathcal{M},\beta}(f(t_1, \ldots, t_r)) = \mathcal{I}(f)(val_{\mathcal{M},\beta}(t_1), \ldots, val_{\mathcal{M},\beta}(t_r))$

## Semantic Evaluation of Terms Cont'd

### Example

Signature: **int** `i`; **short** `j`; **int** `f`(**int**);

$\mathcal{D} = \{17, 2, o\}$ where all numbers are short

Variables: `Object obj`; **int** `x`;

$\mathcal{I}(\mathtt{i}) = 17$

$\mathcal{I}(\mathtt{j}) = 17$

| $\mathcal{D}^{\mathbf{int}}$ | $\mathcal{I}(\mathtt{f})$ |
|---|---|
| 2 | 17 |
| 17 | 2 |

| Var | $\beta$ |
|---|---|
| `obj` | $o$ |
| `x` | 17 |

- $val_{\mathcal{M},\beta}(\mathtt{f}(\mathtt{f}(\mathtt{i})))$ ?
- $val_{\mathcal{M},\beta}(x)$ ?
- $val_{\mathcal{M},\beta}((\mathbf{int})\mathtt{obj})$ ?

## Semantic Evaluation of Formulas

Formulas are true or false

A validity relation is more convenient than a function

**Definition (Validity Relation for Formulas)**

$\mathcal{M}, \beta \models \phi$ for $\phi \in For$ "$\mathcal{M}, \beta$ models $\phi$"

- $\mathcal{M}, \beta \models p(t_1, \ldots, t_r)$ iff $(val_{\mathcal{M},\beta}(t_1), \ldots, val_{\mathcal{M},\beta}(t_r)) \in \mathcal{I}(p)$
- $\mathcal{M}, \beta \models \phi \,\&\, \psi$ iff $\mathcal{M}, \beta \models \phi$ and $\mathcal{M}, \beta \models \psi$
- ... as in propositional logic
- $\mathcal{M}, \beta \models \forall\, T\, x; \phi$ iff $\mathcal{M}, \beta_x^d \models \phi$ for all $d \in \mathcal{D}^T$
- $\mathcal{M}, \beta \models \exists\, T\, x; \phi$ iff $\mathcal{M}, \beta_x^d \models \phi$ for at least one $d \in \mathcal{D}^T$

## Semantic Evaluation of Formulas Cont'd

### Example

Signature: `short j; int f(int); Object obj; <(int,int);`
$\mathcal{D} = \{17, 2, o\}$ where all numbers are short

$\mathcal{I}(j) = 17$
$\mathcal{I}(\texttt{obj}) = o$

| $\mathcal{D}^{\textbf{int}}$ | $\mathcal{I}(f)$ |
|---|---|
| 2 | 2 |
| 17 | 2 |

| $\mathcal{D}^{\textbf{int}} \times \mathcal{D}^{\textbf{int}}$ | in $\mathcal{I}(<)$? |
|---|---|
| $(2,2)$ | $F$ |
| $(2,17)$ | $T$ |
| $(17,2)$ | $F$ |
| $(17,17)$ | $F$ |

- $\mathcal{M}, \beta \models f(j) < j$ ?
- $\mathcal{M}, \beta \models \exists\, \textbf{int}\ x;\ f(x) \doteq x$ ?
- $\mathcal{M}, \beta \models \forall\, \texttt{Object}\ o1;\ \forall\, \texttt{Object}\ o2;\ o1 \doteq o2$ ?

## Semantic Notions

**Definition (Satisfiability, Truth, Validity)**

| | | | | |
|---|---|---|---|---|
| $\mathcal{M}, \beta \models \phi$ | | | | ($\phi$ is **satisfiable**) |
| $\mathcal{M} \models \phi$ | iff | for all $\beta$ : | $\mathcal{M}, \beta \models \phi$ | ($\phi$ is **true** in $\mathcal{M}$) |
| $\models \phi$ | iff | for all $\mathcal{M}$ : | $\mathcal{M} \models \phi$ | ($\phi$ is **valid**) |

Closed formulas that are satisfiable are also true: one top-level notion

## Semantic Notions

**Definition (Satisfiability, Truth, Validity)**

| | | | | | |
|---|---|---|---|---|---|
| $\mathcal{M}, \beta \models \phi$ | | | | | ($\phi$ is **satisfiable**) |
| $\mathcal{M}$ | $\models \phi$ | iff | for all $\beta$ : | $\mathcal{M}, \beta \models \phi$ | ($\phi$ is **true** in $\mathcal{M}$) |
| | $\models \phi$ | iff | for all $\mathcal{M}$ : | $\mathcal{M} \models \phi$ | ($\phi$ is **valid**) |

Closed formulas that are satisfiable are also true: one top-level notion

**Example**

- $f(j) < j$ is true in $\mathcal{M}$
- $\exists\,\mathbf{int}\ x;\ i \doteq x$ is valid
- $\exists\,\mathbf{int}\ x;\ !(x \doteq x)$ is not satisfiable

## Untyped First-Order Logic

**Most logic textbooks introduce untyped logic**

### How to obtain untyped logic as a special case

- Minimal Type Hierarchy: $\mathcal{T} = \{\bot, \top\}$
- $\mathcal{D} = \mathcal{D}^{\top} \neq \emptyset$: only one populated type $\top$, drop all typing info
- Signature merely specifies arity of functions and predicates: Write $f/1$, $</2$, $i/0$, etc.
- Untyped logic is suitable whenever we model a uniform domain
- Typical applications: pure mathematics such as algebra

## Untyped First-Order Logic Cont'd

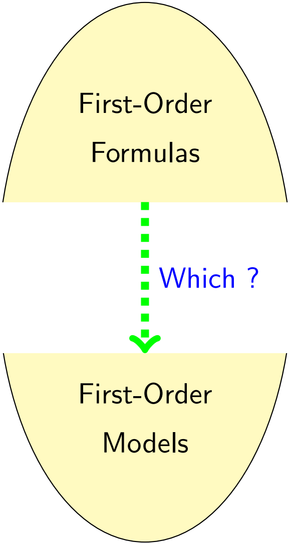
**Example (Axiomatization of a group in first-order logic)**

Signature $\Sigma_G$: FSym $= \{\circ/2, \mathbf{e}/0\}$, PSym $= \{\doteq /2\}$
Let $G$ be the following formulas:

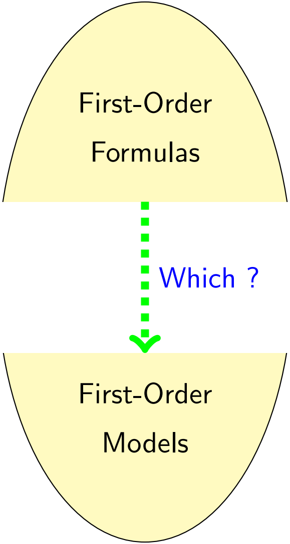
| | |
|---|---|
| Left identity | $\forall x;\ \mathbf{e} \circ x \doteq x$ |
| Left inverse | $\forall x;\ \exists y;\ y \circ x \doteq \mathbf{e}$ |
| Associativity | $\forall x;\ \forall y;\ \forall z;\ (x \circ y) \circ z \doteq x \circ (y \circ z)$ |

Let $\phi$ be $\Sigma_G$-formula.
Whenever $\models G \mathrel{-\!\!>} \phi$, then $\phi$ is a theorem of group theory

# Modeling with First-Order Logic



# Modeling with First-Order Logic

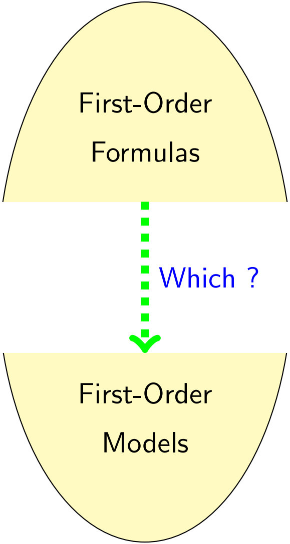



**Example (At least two elements)**

$\exists\, x;\ \exists\, y;\ !(x \doteq y)$

How to do this without built-in equality?

# Modeling with First-Order Logic




**Example (Strict partial order)**

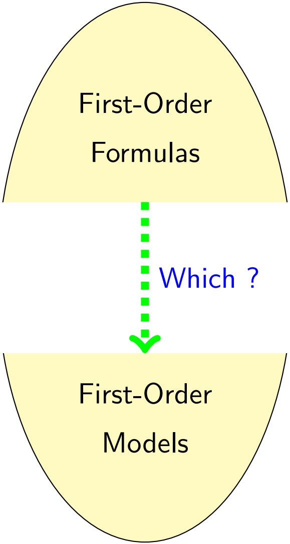
$\mathrm{PSym} = \{< /2\}$

Irreflexivity $\forall x; !(x < x)$

Asymmetry $\forall x; \forall y; (x < y \text{ -> } !(y < x))$

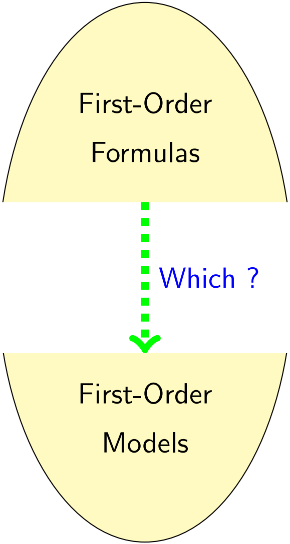
Transitivity $\forall x; \forall y; \forall z;$
$(x < y \;\&\; y < z \text{ -> } x < z)$

# Modeling with First-Order Logic




**Example (All models have infinite domain)**

# Modeling with First-Order Logic

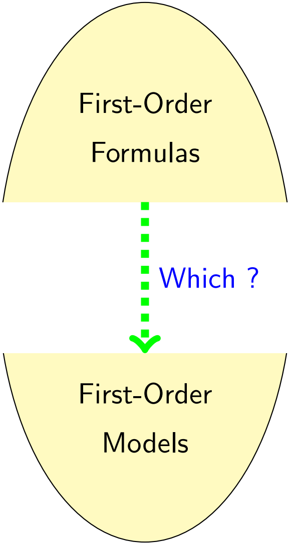



**Example (All models have infinite domain)**

Signature and axioms of irreflexive order plus

Existence Successor $\forall x; \exists y; x < y$

# Modeling with First-Order Logic




**Example (Abstract data types)**

$$\text{FSym} = \{\ \text{Stack push}(\mathbf{int}, \text{Stack}); \quad \mathbf{int}\ \text{pop}(\text{Stack}); \quad \text{Stack nil}; \}$$

$$\forall\, \mathbf{int}\ i;\ \forall\, \text{Stack}\ s;\ \text{pop}(\text{push}(i, s)) \doteq s$$

$\ldots$

## Summary and Outlook

### Summary

- First-order formulas defined over a signature of typed symbols
- Hierarchical OO type system with abstract and dynamic types
- Quantification over variables, no "free" variables in formulas
- Semantic domain like objects in a JAVA heap
- First-order model assigns semantic value to terms and formulas
- Semantic notions satisfiability and validity

## Summary and Outlook

### Summary

- First-order formulas defined over a signature of typed symbols
- Hierarchical OO type system with abstract and dynamic types
- Quantification over variables, no "free" variables in formulas
- Semantic domain like objects in a JAVA heap
- First-order model assigns semantic value to terms and formulas
- Semantic notions satisfiability and validity

### Semantic evaluation is not feasible in practice

- There is an $\infty$ (even: uncountable) number of first-order models
- Evaluation of quantified formula may involve $\infty$ number of cases
- Next goal: a syntactic calculus allowing mechanical validity checking

# Literature for this Lecture

**KeY Book** Verification of Object-Oriented Software (see course web page), Chapter 2: First-Order Logic

**Fitting** First-Order Logic and Automated Theorem Proving, 2nd edn., Springer 1996

**Huth & Ryan** Logic in Computer Science, 2nd edn., Cambridge University Press, 2004