# Formal Specification and Verification of Object-Oriented Programs

## Syntax and Semantics of First-Order Logics

TECHNISCHE UNIVERSITÄT DARMSTADT

## Motivation



### JML combines

- JAVA expressions
- First-Order Logic (FOL)

We will verify JAVA programs using Dynamic Logic

### Dynamic Logic combines

- First-Order Logic (FOL)
- JAVA programs

# FOL: Language and Calculus



Providing JML with a formal semantics

Translate JML specifications into Dynamic Logic

First step (this week)

## Formal Semantics of JML Expressions

- FOL as language (syntax)
- Formal semantics of FOL
- Calculus for FOL to prove validity of formulas
- KeY system as FOL prover (to begin with)

# First-Order Logic: Signature



**Type Hierarchy $\mathcal{T}$ = (TNames, $\preceq$) consists of**

- a non-empty set of type names TNames with $\bot, \top \in$ TNames
- a reflexive, transitive subtype relation $\preceq$ such that for any $\mathtt{A} \in$ TNames

$$\bot \preceq \mathtt{A} \preceq \top$$

Two types $T_1$, $T_2$ are called incomparable iff. $T_1 \not\preceq T_2$ and $T_2 \not\preceq T_1$

## Example

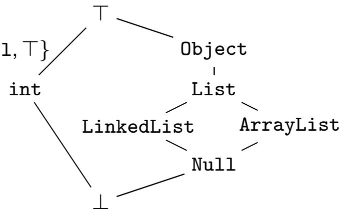
TNames = $\{\bot, \mathtt{int}, \mathtt{Object}, \mathtt{List}, \mathtt{LinkedList}, \mathtt{ArrayList}, \mathtt{Null}, \top\}$

$\bot \preceq \mathtt{int}, \mathtt{int} \preceq \top$

$\bot \preceq \mathtt{Object}$, $\mathtt{Object} \preceq \mathtt{List}$, $\mathtt{List} \preceq \mathtt{LinkedList}$

...

E.g., `int` and `Object` are incomparable types



# First-Order Logic: Signature

TECHNISCHE UNIVERSITÄT DARMSTADT

## Signature $\Sigma_{\mathcal{T}}$ for a Type Hierarchy $\mathcal{T}$

A first-order signature $\Sigma$ for a type hierarchy $\mathcal{T}$ consists of

- a set $F_{\Sigma_{\mathcal{T}}}$ of function symbols
- a set $P_{\Sigma_{\mathcal{T}}}$ of predicate symbols
- a set $V_{\Sigma_{\mathcal{T}}}$ of logic variable symbols

with $F_{\Sigma_{\mathcal{T}}}, P_{\Sigma_{\mathcal{T}}}, V_{\Sigma_{\mathcal{T}}}$ pairwise disjoint. (We omit the subscript $\mathcal{T}$ or $\Sigma_{\mathcal{T}}$ if no ambiguity arises.)

## Definition (Type Declaration of Symbols )

- $T\ f(T_1, \ldots, T_n) \in F_{\Sigma_{\mathcal{T}}}$ with $T, T_i \in \mathcal{T} - \{\perp\}$; arity of $f$ is $n$
  - for constants $T\ c()$, i.e., function symbols with $n = 0$, we write $T\ c$
- $p(T_1, \ldots, T_n) \in P_{\Sigma_{\mathcal{T}}}$ with $T_i \in \mathcal{T} - \{\perp\}$; arity of $p$ is $n$
  - for propositional variables $p()$, i.e., predicate symbols with $n = 0$, we write $p$
- $v : T \in V_{\Sigma_{\mathcal{T}}}$ with $T \in \mathcal{T} - \{\perp\}$; arity of $v$ is 0

## Signature $\Sigma_{\Sigma_{\mathcal{T}_1}}$

TECHNISCHE UNIVERSITÄT DARMSTADT

### Example (Signature $\Sigma_{\mathcal{T}_1}$)

$\mathcal{T}_{\Sigma_{\mathcal{T}_1}} = \{\bot, \texttt{int}, \top\}$ with $\bot \preceq \texttt{int} \preceq \top$

$F_{\Sigma_{\mathcal{T}_1}} = \{\texttt{int +(int,int)}, \texttt{int -(int,int)}\} \cup \{\ldots, \texttt{int -2}, \texttt{int -1}, \texttt{int 0}, \texttt{int 1}, \texttt{int 2}, \ldots\}$

$P_{\Sigma_{\mathcal{T}_1}} = \{\texttt{<(int, int)}\}$

In signature $\Sigma_{\mathcal{T}_1}$ we have

- two function symbols + and - each of arity 2
- infinitely many constant symbols ..., $\texttt{-2}, \texttt{-1}, \texttt{0}, \texttt{1}, \texttt{2}, \ldots$
- one predicate symbol < of arity 2

# Signature $\Sigma_{\mathcal{T}_2}$

TECHNISCHE UNIVERSITÄT DARMSTADT

## Example (Signature $\Sigma_{\mathcal{T}_2}$)

$\mathcal{T}_2 = \{\perp, \texttt{int}, \texttt{List}, \texttt{Heap}, \texttt{Field}, \top\}$ (all types incomparable),

$F_{\Sigma_{\mathcal{T}_2}} = \{\texttt{null}, \texttt{o}, \texttt{store}, \texttt{select}, \texttt{next}\} \cup \{\texttt{...,-2,-1,0,1,2,...}\}$

(type declarations see below)

$P_{\Sigma_{\mathcal{T}_2}} = \{\}$

Intuition: `select`/`store` model program heap; `next` models field of `List` objects

## Example (Type declarations)

$\texttt{List null, o;}$

$\texttt{Heap store(Heap, }\overbrace{\texttt{List, Field, List}}^{\texttt{o.next := o}'}\texttt{);}$

$\texttt{List select(Heap, }\overbrace{\texttt{List, Field}}^{\texttt{o.next}}\texttt{);}$

$\texttt{Field next;} \quad \texttt{int 0; int 1; int -1; ...}$

# First-Order Terms

TECHNISCHE UNIVERSITÄT DARMSTADT

## Definition (Terms, inductive definition)

A first-order term of type $T \in \mathcal{T} - \{\perp\}$

- is a logic variable of type $T$, or
- has the form $f(t_1, \dots, t_n)$,

  where $T\ f(T_1, \dots, T_n) \in F_{\Sigma_\mathcal{T}}$ and each $t_i$ is a term of type $T_i'$ with $T_i' \preceq T_i$, or

Think of first-order terms as side effect-free expressions in programs

# Terms over $\Sigma_{\mathcal{T}_1}$

TECHNISCHE UNIVERSITÄT DARMSTADT

## Example (Terms over $\Sigma_{\mathcal{T}_1}$)

(assuming logic variable declarations `int` $v_1$`;` `int` $v_2$`;`)

- `-7`
- `+(-2, 99)`
- `-(7, 8)`
- `+(-(7, 8), 1)`
- `+(-(`$v_1$`, 8), `$v_2$`)`

## Example (Using infix notation of functions)

- `-2 + 99`
- `7 - 8`
- `(7 - 8) + 1`
- `(`$v_1$` - 8) + `$v_2$

# Terms over $\Sigma_{\mathcal{T}_2}$

## Example (Terms over $\Sigma_{\mathcal{T}_2}$)

(assuming logic variable declarations `List` $u$; `Heap` $h$;)

- `-7` has type `int`
- `null` has type `List`
- `store(`$h$`, o, next, null)` has type `Heap`
- `select(store(`$h$`, o, next, null), `$u$`, next)` has type `List`

## Example (Intuition behind functions modeling object fields)

- `o.next = null;`
- $u$`.next` in an execution state, where `o.next == null`

## Atomic Formulas

TECHNISCHE UNIVERSITÄT DARMSTADT

### Definition (Logical Atoms)

Given a signature $\Sigma$, a logical atom has one of the forms

- true
- false
- $t_1 \doteq t_2$ ("equality"),
  where $t_1$ and $t_2$ have a common subtype $\neq \perp$
- $p(t_1, \ldots, t_n)$ ("predicate"),

  where $p \in P_{\Sigma_\mathcal{T}}$, and each $t_i$ is term of the correct type following the typing $p$

# Atomic Formulas over Signature $\Sigma_{\mathcal{T}_1}$

## Example (Atomic formulas over $\Sigma_{\mathcal{T}_1}$)

(assuming variable declaration `int` $v$;)

- $7 \doteq 8$
- `7 < 8`
- `-2 - ` $v$ `< 99`
- $v$ `< (`$v$ `+ 1)`

# Atomic Formulas over Signature $\Sigma_{\mathcal{T}_2}$

TECHNISCHE UNIVERSITÄT DARMSTADT

## Example (Atomic formulas over $\Sigma_{\mathcal{T}_2}$)

(assuming variable declarations $\texttt{List}\ u;\ \texttt{Heap}\ h;$)

- $\texttt{store}(h, \texttt{o}, \texttt{next}, \texttt{null}) \doteq h$
- $\texttt{select}(\texttt{store}(h, u, \texttt{next}, \texttt{null}), u, \texttt{next}) \doteq \texttt{null}$

# General Formulas

TECHNISCHE UNIVERSITÄT DARMSTADT

## Definition (FO Formulas, inductive definition)

- Any atomic formula is a formula
- With $\phi$ and $\psi$ formulas, *x* a logic variable of type $T \in \mathcal{T} - \{\bot\}$, the following are also formulas:
  - $\neg\phi$ "not $\phi$"
  - $\phi \wedge \psi$ "$\phi$ and $\psi$"
  - $\phi \vee \psi$ "$\phi$ or $\psi$"
  - $\phi \rightarrow \psi$ "$\phi$ implies $\psi$"
  - $\phi \leftrightarrow \psi$ "$\phi$ is equivalent to $\psi$"
  - $\forall\ T\ x;\ \phi$ "for all $x$ of type $T$ formula $\phi$ holds"
  - $\exists\ T\ x;\ \phi$ "there exists an $x$ of type $T$ such that $\phi$ holds"

In $\forall\ T\ x;\ \phi$ and $\exists\ T\ x;\ \phi$ the logic variable $x$ is bound (i.e., not free)

Formulas with no free logic variables are called closed

# General Formulas: Examples

TECHNISCHE UNIVERSITÄT DARMSTADT

(signatures obvious, typing irrelevant: left out)

**Example (There are at least two distinct elements)**

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

**Example (Axioms of strict partial order)**

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

Asymmetry $\forall x; \forall y; (x < y \rightarrow \neg(y < x))$

Transitivity $\forall x; \forall y; \forall z;$
$$(x < y \wedge y < z \rightarrow x < z)$$

(is any of the three formulas redundant?)

# General Formulas: Examples

TECHNISCHE UNIVERSITÄT DARMSTADT

(signatures obvious, typing irrelevant: left out)

**Example (There must be an $\infty$ number of elements)**

Signature and axioms of strict partial order plus

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

# Remark on Concrete Syntax



| | Text book | KeY |
|---|---|---|
| Negation | $\neg$ | ! |
| Conjunction | $\wedge$ | & |
| Disjunction | $\vee$ | \| |
| Implication | $\rightarrow$, $\supset$ | -> |
| Equivalence | $\leftrightarrow$ | <-> |
| Universal Quantifier | $\forall\, x; \phi$ | `\forall` $T\ x; \phi$ |
| Existential Quantifier | $\exists\, x; \phi$ | `\exists` $T\ x; \phi$ |
| Value equality | $\doteq$ | = |

# FO Syntax in KeY

**Demo: `order.key`**

TECHNISCHE UNIVERSITÄT DARMSTADT

```
\sorts { // sorts = types
  any;
}
\predicates { // use symbol p for order predicate
  p(any,any);
}
\problem { // a formula
 (
   \forall any x; !p(x,x) &
   \forall any x; \forall any y; \forall any z;
     (p(x,y) & p(y,z) -> p(x,z))
   ->
   \forall any x; \forall any y; (p(x,y) -> !p(y,x))
 )
}
```

# Why Semantics?



How do we know that a formula like

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

expresses our intention?

# First-Order Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

## From propositional to first-order semantics

- In prop. logic, an interpretation of prop. variables with $\{tt, ff\}$ suffices
- 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

## Need to interpret a first-order formula relative to a signature $\Sigma$

1. A typed universe (domain) of objects
2. A mapping from variables to objects of suitable type
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 typed universe of objects for $\mathcal{T}$

## Definition (Universe/Domain)

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

- Notation for the domain elements of type $T \in \mathcal{T} - \{\perp\}$:
  $\mathcal{D}^T = \{d \in \mathcal{D} \mid \delta(d) \preceq T\}$ and $\mathcal{D}^\perp = \emptyset$
- Obviously, $\mathcal{D} = \bigcup_{T \in \mathcal{T}} D^T$
- Each type $T \in \mathcal{T} - \{\perp\}$ must "contain" at least one domain element: $\mathcal{D}^T \neq \emptyset$

## First-Order States (Models)

TECHNISCHE UNIVERSITÄT DARMSTADT

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

### Definition (First-Order State/Model)

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

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

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

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

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

Then $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ is a first-order state (or model).

# First-Order States Cont'd

TECHNISCHE UNIVERSITÄT DARMSTADT

## Example

Signature: `int i; int j; int f(int); Object obj; <(int,int);`

Domain: $\mathcal{D} = \{17, 2, o\}$ with obvious typing ($|\mathcal{D}^{\textbf{int}}| = 2$ !)

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

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

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

One of uncountably many possible first-order states!

## Semantics of Reserved Signature Symbols



### Definition

Equality symbol $\doteq$ declared as $\doteq(T, T)$ for any type $T \in \mathcal{T} - \{\perp\}$

Interpretation 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 of $\doteq$ for example domain

# Signature Symbols vs. Domain Elements

TECHNISCHE UNIVERSITÄT DARMSTADT

- Domain elements different from the terms representing them
- First-order formulas and terms have no access to domain
  - $\mathcal{D}$, $\delta$, $\mathcal{D}^T$ are not part of FOL signature
  - Cf. languages (e.g., JAVA) without access to heap representation

## Example

Signature: `Object obj1, obj2;`
Does `obj1` $\doteq$ `obj2` hold in a state?

We have no idea what the elements of $\mathcal{D} = \mathcal{D}^{\texttt{Object}}$ look like

- Holds always, if $|\mathcal{D}| = 1$
- Maybe, otherwise
- How to establish that there are exactly 42 elements in $\mathcal{D}^{\texttt{Object}}$?
  - How to do that in JAVA?

## Variable Assignments

TECHNISCHE UNIVERSITÄT DARMSTADT

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 state (model) $\mathcal{S}$ and a variable assignment $\beta$:
it is possible to evaluate first-order terms under $\mathcal{S}$ and $\beta$

## Definition (Valuation of Terms)

Let $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$

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

- $val_{\mathcal{S},\beta}(x) = \beta(x)$
- $val_{\mathcal{S},\beta}(f(t_1, \dots, t_r)) = \mathcal{I}(f)(val_{\mathcal{S},\beta}(t_1), \dots, val_{\mathcal{S},\beta}(t_r))$

## Semantic Evaluation of Terms Cont'd

TECHNISCHE UNIVERSITÄT DARMSTADT

### Example

Signature: `int i; int j; int f(int); int hashcode(Object);`
Domain: $\mathcal{D} = \{0, 1, 2, 17, o, nil\}$
Variables: `Object obj; int x;`

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

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

| $\mathcal{D}^{\mathtt{Object}}$ | $\mathcal{I}(\mathtt{hashcode})$ |
|---|---|
| $o$ | 1 |
| $nil$ | 0 |

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

1. $val_{\mathcal{S},\beta}(\mathtt{f(f(i))})$ ?
2. $val_{\mathcal{S},\beta}(x)$ ?
3. $val_{\mathcal{S},\beta}(\mathtt{hashcode(obj)})$ ?

# Semantic Evaluation of Terms: Example

TECHNISCHE UNIVERSITÄT DARMSTADT

1. $$\begin{aligned} val_{\mathcal{S},\beta}(\texttt{f(f(i))}) &= \mathcal{I}(\texttt{f})(val_{\mathcal{S},\beta}(\texttt{f(i)})) \\ &= \mathcal{I}(\texttt{f})(\mathcal{I}(\texttt{f})(val_{\mathcal{S},\beta}(\texttt{i}))) \\ &= \mathcal{I}(\texttt{f})(\mathcal{I}(\texttt{f})(\mathcal{I}(\texttt{i}))) \\ &= \mathcal{I}(\texttt{f})(\mathcal{I}(\texttt{f})(17)) \\ &= \mathcal{I}(\texttt{f})(2) \\ &= 17 \end{aligned}$$
2. $val_{\mathcal{S},\beta}(x) = \beta(x) = 17$
3. $$\begin{aligned} val_{\mathcal{S},\beta}(\texttt{hashcode(obj)}) &= \mathcal{I}(\texttt{hashcode})(val_{\mathcal{S},\beta}(\texttt{obj})) \\ &= \mathcal{I}(\texttt{hashcode})(\beta(\texttt{obj})) \\ &= \mathcal{I}(\texttt{hashcode})(o) \\ &= 1 \end{aligned}$$

# Semantic Evaluation of Formulas



## Definition (Valuation of Formulas)

$val_{\mathcal{S},\beta}(\phi)$ for $\phi \in For$

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

## Semantic Evaluation of Formulas Cont'd

TECHNISCHE UNIVERSITÄT DARMSTADT

### Example

Signature: `int j; int f(int); Object obj; <(int,int);`

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

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

Domain: $\mathcal{D} = \{2,\ 17,\ o\}$

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

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

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

# Semantic Evaluation of Formulas: Example

1. $$\begin{aligned} val_{\mathcal{S},\beta}(\mathtt{f}(\mathtt{j}) < \mathtt{j}) &= \mathcal{I}(<)(val_{\mathcal{S},\beta}(\mathtt{f}(\mathtt{j})), val_{\mathcal{S},\beta}(\mathtt{j})) \\ &= \mathcal{I}(<)(\mathcal{I}(\mathtt{f})(val_{\mathcal{S},\beta}(\mathtt{j})), \mathcal{I}(\mathtt{j})) \\ &= \mathcal{I}(<)(\mathcal{I}(\mathtt{f})(17), 17) \\ &= \mathcal{I}(<)(2, 17) \\ &= tt \end{aligned}$$

2. $$\begin{aligned} val_{\mathcal{S},\beta}(\exists\, \mathbf{int}\ x; \mathtt{f}(x) \doteq x) = tt \textit{ iff } & val_{\mathcal{S},\beta_x^d}(\mathtt{f}(x) \doteq x) = T \text{ for } d \in \mathcal{D}^{\mathbf{int}} \\ \textit{iff } & \mathcal{I}(\doteq)(val_{\mathcal{S},\beta_x^d}(\mathtt{f}(x)), val_{\mathcal{S},\beta_x^d}(x)) = tt \\ \textit{if } & \mathcal{I}(\doteq)(val_{\mathcal{S},\beta_x^2}(\mathtt{f}(x)), val_{\mathcal{S},\beta_x^2}(x)) = tt \\ \textit{iff } & \mathcal{I}(\doteq)(2, 2) = tt \checkmark \end{aligned}$$

# Semantic Evaluation of Formulas: Example

TECHNISCHE UNIVERSITÄT DARMSTADT

3. $val_{\mathcal{S},\beta}(\forall\, \texttt{Object}\ o1;\ \forall\, \texttt{Object}\ o2;\ o1 \doteq o2) = tt$

   *iff* $val_{\mathcal{S},\beta^{d_1,d_2}_{o1,o2}}(o1 \doteq o2) = tt$ f.a. $d_1, d_2 \in \mathcal{D}^{\texttt{Object}} = \{o\}$

   *iff* $val_{\mathcal{S},\beta^{o,o}_{o1,o2}}(o1 \doteq o2) = tt$

   *iff* $\mathcal{I}(\doteq)(val_{\mathcal{S},\beta^{o,o}_{o1,o2}}(o1), val_{\mathcal{S},\beta^{o,o}_{o1,o2}}(o2)) = tt$

   *iff* $\mathcal{I}(\doteq)(\beta^{o,o}_{o1,o2}(o1), \beta^{o,o}_{o1,o2}(o2)) = tt$

   *iff* $\mathcal{I}(\doteq)(o, o) = tt$ ✓

## Definition (Satisfiability, Truth, Validity)

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

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

## Example

- $\texttt{f(j)} < \texttt{j}$ is true in $\mathcal{S}$ of previous slide
- $\exists\,$**int** $x$; $\texttt{i} \doteq x$? Valid: can always choose $d = \mathcal{I}(\texttt{i})$ in $val_{\mathcal{S},\beta_x^d}$
- $\exists\,$**int** $x$; $\neg(x \doteq x)$? Not satisfiable

## Useful Valid Formulas (Propositional Logic)



Let $\phi$ and $\psi$ be arbitrary, closed formulas (whether valid of not)

Then the following formulas are valid:

- $\neg(\phi \wedge \psi) \leftrightarrow (\neg\phi \vee \neg\psi)$
- $\neg(\phi \vee \psi) \leftrightarrow (\neg\phi \wedge \neg\psi)$
- $(\mathit{true} \wedge \phi) \leftrightarrow \phi$
- $(\mathit{false} \vee \phi) \leftrightarrow \phi$
- $\mathit{true} \vee \phi$
- $\neg(\mathit{false} \wedge \phi)$
- $(\phi \rightarrow \psi) \leftrightarrow (\neg\phi \vee \psi)$
- $\phi \rightarrow \mathit{true}$
- $\mathit{false} \rightarrow \phi$
- $(\mathit{true} \rightarrow \phi) \leftrightarrow \phi$
- $(\phi \rightarrow \mathit{false}) \leftrightarrow \neg\phi$

# Useful Valid Formulas (FO Logic)

Assume that $x$ is the only variable which may appear freely in $\phi$ or $\psi$

Then the following formulas are valid:

- $\neg(\exists\ T\ x;\ \phi) \leftrightarrow \forall\ T\ x;\ \neg\phi$
- $\neg(\forall\ T\ x;\ \phi) \leftrightarrow \exists\ T\ x;\ \neg\phi$
- $(\forall\ T\ x;\ \phi \wedge \psi) \leftrightarrow (\forall\ T\ x;\ \phi) \wedge (\forall\ T\ x;\ \psi)$
- $(\exists\ T\ x;\ \phi \vee \psi) \leftrightarrow (\exists\ T\ x;\ \phi) \vee (\exists\ T\ x;\ \psi)$

Are the following formulas also valid? (Exercise)

- $(\forall\ T\ x;\ \phi \vee \psi) \leftrightarrow (\forall\ T\ x;\ \phi) \vee (\forall\ T\ x;\ \psi)$
- $(\exists\ T\ x;\ \phi \wedge \psi) \leftrightarrow (\exists\ T\ x;\ \phi) \wedge (\exists\ T\ x;\ \psi)$

## Impracticality of Semantic Arguments

TECHNISCHE UNIVERSITÄT DARMSTADT

### Showing Validity by Computation of $val_{\mathcal{S},\beta}$ Impractical

- There are uncountably many FO states
- Even a single state may have infinite domain
- Even when domain finite: recursion $\Rightarrow$ exponential number of cases

Need syntactic proof method:

- Use only symbols occurring in a formula already
- Avoid full recursive evaluation whenever possible