Alcino Cunha

SPECIFICATION AND MODELING

FIRST-ORDER LOGIC

Universidade do Minho & INESC TEC

2019/20

FROM PROPOSITIONAL TO FIRST-ORDER LOGIC

- Introduces a *domain of discourse*
- Generalize propositional symbols to *predicates*
- Allows *quantifiers* and *variables* ranging over the domain

Propositional logic

$\texttt{File1\_is\_in\_trash} \land \texttt{File2\_is\_in\_trash}$

$\texttt{File2\_has\_name\_Name1}$

First-order logic

$\texttt{is\_in\_trash}(\texttt{File1}) \land \texttt{is\_in\_trash}(\texttt{File2})$

$\texttt{has\_name}(\texttt{File2}, \texttt{Name1})$

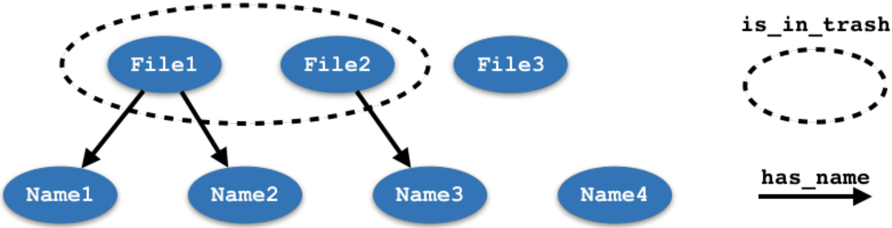
$\forall x . \texttt{is\_in\_trash}(x)$

PREDICATES (AKA SETS AND RELATIONS)

$$
\begin{aligned}
\texttt{is\_in\_trash}(\texttt{File1}) &= \top \\
\texttt{is\_in\_trash}(\texttt{File2}) &= \top \\
\texttt{is\_in\_trash}(\_) &= \bot \\
\texttt{has\_name}(\texttt{File1}, \texttt{Name1}) &= \top \\
\texttt{has\_name}(\texttt{File1}, \texttt{Name2}) &= \top \\
\texttt{has\_name}(\texttt{File2}, \texttt{Name3}) &= \top \\
\texttt{has\_name}(\_, \_) &= \bot
\end{aligned}
$$

$$
\begin{aligned}
\texttt{is\_in\_trash} &= \{(\texttt{File1}), (\texttt{File2})\} \\
\texttt{has\_name} &= \{(\texttt{File1}, \texttt{Name1}), (\texttt{File1}, \texttt{Name2}), (\texttt{File2}, \texttt{Name3})\}
\end{aligned}
$$

VISUALISING SETS AND RELATIONS



SYNTAX

Category	Identifier
Variables	$x, y, z, \ldots$
Constants	$a, b, c, \ldots$
Functions	$f, g, h, \ldots$
Predicates	$P, Q, R, \ldots$
Terms	$t, u, v, \ldots$
Formulas	$\phi, \varphi, \psi, \ldots$

SYNTAX

$$
\begin{array}{rcl}
t & ::= & x \\
  & | & c \\
  & | & f(t_1, \ldots, t_{\mathrm{ar}(f)}) \\
  & & \\
\phi & ::= & P(t_1, \ldots, t_{\mathrm{ar}(P)}) \\
  & | & t = u \\
  & | & \top \\
  & | & \bot \\
  & | & \neg\phi \\
  & | & \phi_1 \wedge \phi_2 \\
  & | & \phi_1 \vee \phi_2 \\
  & | & \phi_1 \rightarrow \phi_2 \\
  & | & \forall x.\phi \\
  & | & \exists x.\phi
\end{array}
$$

FIRST-ORDER STRUCTURES AND VARIABLE ASSIGNMENTS

- The semantics of a first-order formula is defined over a *first-order structure* (aka *model*) $\mathcal{M} = (\mathcal{D}, \mathcal{I})$
  - $\mathcal{D}$ is a non-empty domain of interpretation (or discourse) with equality
  - $\mathcal{I}$ is the interpretation constants, functions, and predicates:
    - $\mathcal{I}(c) \in \mathcal{D}$
    - $\mathcal{I}(f) \in \mathcal{D}^{\mathrm{ar}(f)} \to \mathcal{D}$
    - $\mathcal{I}(P) \subseteq \mathcal{D}^{\mathrm{ar}(P)}$
- For interpreting (free) variables we also need an assignment $\mathcal{A}$:
  - $\mathcal{A}(x) \in \mathcal{D}$
- The fact that a formula ϕ is valid in a model $\mathcal{M}$ with assignment $\mathcal{A}$ is denoted by $\mathcal{M}, \mathcal{A} \models \phi$
- If the formula is closed we write just $\mathcal{M} \models \phi$, assuming $\mathcal{A}$ to be the empty assignment

EXAMPLE

- Given $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ with:
  - $\mathcal{D} = \{\texttt{File1}, \texttt{File2}, \texttt{Name1}, \texttt{Name2}\}$
  - $\mathcal{I}(\texttt{is\_a\_file}) = \{(\texttt{File1}), (\texttt{File2})\}$
  - $\mathcal{I}(\texttt{is\_a\_name}) = \{(\texttt{Name1}), (\texttt{Name2})\}$
  - $\mathcal{I}(\texttt{is\_in\_trash}) = \{(\texttt{File1}), (\texttt{File2})\}$
  - $\mathcal{I}(\texttt{has\_name}) = \{(\texttt{File2}, \texttt{Name1})\}$
- We have:

$$\mathcal{M} \models \forall x.\texttt{is\_a\_file}(x) \vee \texttt{is\_a\_name}(x)$$
$$\mathcal{M} \models \forall x.\texttt{is\_a\_file}(x) \longrightarrow \texttt{is\_in\_trash}(x)$$
$$\mathcal{M} \not\models \forall x.\texttt{is\_a\_file}(x) \longrightarrow \exists y.\texttt{has\_name}(x, y)$$

SEMANTICS

$$
\begin{array}{rcl}
[\![x]\!]_{\mathcal{M},\mathcal{A}} & = & \mathcal{A}(x) \\
[\![c]\!]_{\mathcal{M},\mathcal{A}} & = & \mathcal{I}(c) \\
[\![f(t_1, \ldots, t_n)]\!]_{\mathcal{M},\mathcal{A}} & = & \mathcal{I}(f)([\![t_1]\!]_{\mathcal{M},\mathcal{A}}, \ldots, [\![t_n]\!]_{\mathcal{M},\mathcal{A}})
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{M}, \mathcal{A} \models P(t_1, \ldots, t_n) & \text{iff} & ([\![t_1]\!]_{\mathcal{M},\mathcal{A}}, \ldots, [\![t_n]\!]_{\mathcal{M},\mathcal{A}}) \in \mathcal{I}(P) \\
\mathcal{M}, \mathcal{A} \models t = u & \text{iff} & [\![t]\!]_{\mathcal{M},\mathcal{A}} = [\![u]\!]_{\mathcal{M},\mathcal{A}} \\
\mathcal{M}, \mathcal{A} \models \top & & \\
\mathcal{M}, \mathcal{A} \not\models \bot & & \\
\mathcal{M}, \mathcal{A} \models \neg\phi & \text{iff} & \mathcal{M}, \mathcal{A} \not\models \phi \\
\mathcal{M}, \mathcal{A} \models \phi_1 \wedge \phi_2 & \text{iff} & \mathcal{M}, \mathcal{A} \models \phi_1 \text{ and } \mathcal{M}, \mathcal{A} \models \phi_2 \\
\mathcal{M}, \mathcal{A} \models \phi_1 \vee \phi_2 & \text{iff} & \mathcal{M}, \mathcal{A} \models \phi_1 \text{ or } \mathcal{M}, \mathcal{A} \models \phi_2 \\
\mathcal{M}, \mathcal{A} \models \phi_1 \rightarrow \phi_2 & \text{iff} & \mathcal{M}, \mathcal{A} \not\models \phi \text{ or } \mathcal{M}, \mathcal{A} \models \phi_2 \\
\mathcal{M}, \mathcal{A} \models \forall x.\phi & \text{iff} & \mathcal{M}, \mathcal{A}[x \mapsto a] \models \phi \text{ for all } a \in \mathcal{D} \\
\mathcal{M}, \mathcal{A} \models \exists x.\phi & \text{iff} & \mathcal{M}, \mathcal{A}[x \mapsto a] \models \phi \text{ for some } a \in \mathcal{D}
\end{array}
$$

FIRST-ORDER LOGIC SYNTAX IN ALLOY

Alloy	Math
x_1 `->` … `->` x_n **`in`** P	$P(x_1, \ldots, x_n)$
x_1 `->` … `->` x_n **`not in`** P	$\neg P(x_1, \ldots, x_n)$
x `=` y	$x = y$
x `!=` y	$\neg(x = y)$
`not` ϕ	$\neg\phi$
ϕ **`and`** ψ	$\phi \wedge \psi$
ϕ **`or`** ψ	$\phi \vee \psi$
ϕ **`implies`** ψ	$\phi \longrightarrow \psi$
`all` x `:` P `|` ϕ	$\forall x \cdot P(x) \longrightarrow \phi$
`some` x `:` P `|` ϕ	$\exists x \cdot P(x) \wedge \phi$

PREDICATE DECLARATIONS IN ALLOY

- Unary predicates are known as *signatures* or *sets*
  - Declared with the **sig** keyword
  - Sub-set signatures are declared with the **in** keyword
- Predicates of higher arity are known as *relations*
  - Declared inside signatures

```
sig Name {}
sig File {
  name : set Name,
  link : set File
}
sig Trash in File {}
sig Protected in File {}
```

PREDICATE DECLARATIONS IN ALLOY

- Declarations induce a set of implicit “typing” constraints
  - *Top-level* (non sub-set) signatures are disjoint
  - Sub-set signatures are indeed sub-sets of the parent signature
  - Relations only contain tuples of the correct signatures
- Some special predicates are pre-defined
  - **`univ`** is the union of all top-level signatures
  - **`none`** is the empty set
  - **`iden`** is the identity binary relation over **`univ`**

FORMULA EXAMPLES

```
-- The trash is empty
all f : File | f not in Trash

-- Every file is either in the trash or protected
all f : File | f in Trash or f in Protected
all f : File | f in Trash implies f not in Protected

-- There are no links
all x,y : File | x->y not in link

-- Every file has at least one name
all x : File | some y : Name | x->y in name

-- Every file has at most one name
all x : File, y,z : Name | x->y in name and x->z in name implies y=z
```

WHAT ABOUT SET INCLUSION AND SET OPERATORS?

- Set inclusion and set operators can be defined in first-order logic
- Set operators act like combinators that build more complex (unary) predicates out of simpler ones

$$A \subseteq B \equiv \forall x . A(x) \longrightarrow B(x)$$

$$(A \cup B)(x) \equiv A(x) \vee B(x)$$

- These (and other) combinators simplify the specification of constraints
- They will be the subject of our next class about *relational logic*, the logic of Alloy!