Theme 1: Abstract Reasoning

Lecture 2: Logic-based Program Specification

Ahmed Bouajjani

Paris Diderot University, Paris 7

January 2014
Abstract Specification of a Function

Consider a function

\[ f : \text{Dom} \rightarrow \text{CoDom} \]

How to describe in an abstract way its behavior?

Abstraction: No implementation details.

Specification: A relation \( \text{Spec}_f \) between inputs and outputs of \( f \)

\[ \text{Spec}_f(\text{In}, \text{Out}) \subseteq \text{Dom} \times \text{CoDom} \]

What is a suitable (natural) formalism for describing such a relation?
Logic-based Specification Language

- Example: Specification of the Append function:

\[
\text{Spec}_{\text{Append}}(\ell_1, \ell_2, \ell) =
\]

\[
|\ell| = |\ell_1| + |\ell_2| \land
\forall i \in \text{Nat}. (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i] \land
\forall i \in \text{Nat}. (0 \leq i < |\ell_2|) \Rightarrow \ell[|\ell_1| + i] = \ell_2[i]
\]

where:

\[
\forall \ell \in \text{List}[\star]. \forall i \in \text{Nat}. \forall e \in \star. \ell[i] = e \iff
\]

\[
(i < |\ell|) \land
\exists \ell'. (\ell = a \cdot \ell' \land
\]

\[
((i = 0 \land e = a) \lor (i > 0 \land e = \ell'[i - 1]))
\]
Logic-based Specification Language

- **Example:** Specification of the Append function:

\[
\text{Spec\_Append}(\ell_1, \ell_2, \ell) = \\
|\ell| = |\ell_1| + |\ell_2| \land \\
\forall i \in \text{Nat}. (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i] \land \\
\forall i \in \text{Nat}. (0 \leq i < |\ell_2|) \Rightarrow \ell[|\ell_1| + i] = \ell_2[i]
\]

where:

\[
\forall \ell \in \text{List}[\star]. \forall i \in \text{Nat}. \forall e \in \star. \ \ell[i] = e \iff \\
(i < |\ell|) \land \\
\exists \ell'. (\ell = a \cdot \ell' \land \\
((i = 0 \land e = a) \lor (i > 0 \land e = \ell'[i - 1])))
\]

- ⇒ First-order logic over data domains (natural numbers, lists, etc.)
Domains of Interpretation

- Data domain with a set of operations and predicates
  - Consider a data domain $D$
  - Let $Op$ be a set of operations interpreted as functions over $D$
  - Let $Pred$ be a set of predicates interpreted as relations over $D$

- Remark:
  Here the set $Op$ may include constants, seen as operators or arity 0.
Domains of Interpretation

- Data domain with a set of operations and predicates
  - Consider a data domain $D$
  - Let $Op$ be a set of operations interpreted as functions over $D$
  - Let $Pred$ be a set of predicates interpreted as relations over $D$

- Remark:
  *Here the set $Op$ may include constants, seen as operators or arity 0.*

- Domain of interpretation is a triple $(D, Op, Rel)$. 
Domains of Interpretation

- Data domain with a set of operations and predicates
  - Consider a data domain $D$
  - Let $Op$ be a set of operations interpreted as functions over $D$
  - Let $Pred$ be a set of predicates interpreted as relations over $D$

- Remark:
  
  *Here the set $Op$ may include constants, seen as operators or arity 0.*

- Domain of interpretation is a triple $(D, Op, Rel)$.

- Examples of domains of interpretation:
  - $(\text{Bool}, \{tt, ff, not, or, and\}, \{=\})$
  - $(\text{Nat}, \{0, s, +\}, \{\leq\})$
  - $(\text{List}[*], \{[]\}, \cdot, \@\}, \{=\})$
First Order Logic over a Data Domain

- Let \((D, Op, Pred)\) be a domain of interpretation.
- Let \(Var\) be a set of variables.
- Terms:
  \[
t ::= v \in Var \mid op(t, \ldots, t)
\]
  where \(v \in Var\) and \(op \in Op\).
First Order Logic over a Data Domain

- Let \((D, Op, Pred)\) be a domain of interpretation.
- Let \(Var\) be a set of variables.
- Terms:
  \[ t ::= v \in Var \mid op(t, \ldots, t) \]
  where \(v \in Var\) and \(op \in Op\).
- Examples: \(x, 2, x + 2, x + y + 3,\) and \(2x\) as an abbreviation of \(x + x\).
First Order Logic over a Data Domain

- Let \((D, Op, Pred)\) be a domain of interpretation.
- Let \(Var\) be a set of variables.
- Terms:
  \[
  t ::= v \in Var \mid op(t, \ldots, t)
  \]
  where \(v \in Var\) and \(op \in Op\).
- Examples: \(x, 2, x + 2, x + y + 3,\) and \(2x\) as an abbreviation of \(x + x\).
- Terms are interpreted as elements of the domain \(D\):
  - Let \(\nu : Var \rightarrow D\) be a valuation of the variables.
  - Then, \(\langle t \rangle_\nu\) is the value in \(D\) obtained by the evaluation of \(t\), using \(\nu\) as valuation of the variables.
First Order Logic over a Data Domain

- Let \((D, Op, Pred)\) be a domain of interpretation.
- Let \(Var\) be a set of variables.
- Terms:
  \[ t ::= v \in Var \mid op(t, \ldots, t) \]
  where \(v \in Var\) and \(op \in Op\).
- Examples: \(x, 2, x + 2, x + y + 3\), and \(2x\) as an abbreviation of \(x + x\).
- Terms are interpreted as elements of the domain \(D\):
  - Let \(\nu : Var \rightarrow D\) be a valuation of the variables.
  - Then, \(\langle t \rangle_\nu\) is the value in \(D\) obtained by the evaluation of \(t\), using \(\nu\) as valuation of the variables.
  - Example: Given \(\nu = \{(x, 2), (y, 1), (z, 4)\}\), we have
    \[ \langle x \rangle_\nu = 2 \quad \langle x + 2y \rangle_\nu = 4 \quad \langle (x \ast z) + (y + 1) \rangle_\nu = 10 \]
First Order Logic: Syntax of formulas

- Formulas:
  \[ \phi ::= p(t_1, \ldots, t_n) \mid \neg \phi \mid \phi \lor \phi \mid \exists v. \phi \]
  where \( p \in \text{Pred} \) and \( v \in \text{Var} \).

- Examples: \( 2x + y \leq z, x = y \) as an abbreviation of \( x \leq y \land y \leq x, x < y \) as an abbrev. of \( x \leq y \land \neg (x = y) \).

- An occurrence of a variable \( x \) is bound in a formula \( \phi \) if it is under a quantifier \( \exists x \). We assume that all occurrences of a variable are either bound or unbound in a formula. A variable is free in \( \phi \) if its occurrences in \( \phi \) are unbound. A formula is closed if it has not free variables.

- Examples:
  - \( \phi_1 = \forall x, y. \ x \leq y \Rightarrow \exists z. \ (x \leq z \land z < y) \) is a closed formula.
  - \( \phi_2 = \exists x. \ \forall y. \ x \leq y \) is a closed formula.
  - \( \phi_3 = \forall y. \ x \leq y \), is an open formula. It has \( x \) as free variable.
  - \( \phi_4 = x \leq y \land \exists z. \ y \leq z \land z \leq 5 \) is an open formula. Its free variables are \( x \) and \( y \).
First Order Logic: Semantics of formulas

- Given a valuation $\nu : \text{Var} \rightarrow D$ of the variables, $\nu$ satisfies $\phi$ if and only if $\phi[\nu(x)/x]$ is true, i.e., when interpreting the formula using $\nu$, the formula is true.

- Formulas are interpreted as relations over $D$, i.e., the sets of valuations of the variables that satisfy the formula.

- Let $[\phi]$ be the set of valuations $\nu$ which satisfy $\phi$.

- A formula is valid if it is satisfied by all valuations. A formula is satisfiable if there exists a valuation that satisfies it.

- Remark:

  Closed formulas are either true (valid) or false: Their value does depend on the variable valuation. Either all variable valuations satisfy them, or none of the valuations can satisfy them.

- Question: what can we say about the formulas in the previous slides?
Example: The head and tail functions

- head function:

  \[\text{head} : \text{List}[\ast] \rightarrow \ast\]

  \[\text{Spec}_{\text{head}}(\ell, a) = \exists \ell' \in \text{List}[\ast]. \ell = a \cdot \ell'\]

- tail function:

  \[\text{tail} : \text{List}[\ast] \rightarrow \text{List}[\ast]\]

  \[\text{Spec}_{\text{tail}}(\ell, \ell') = \exists a \in \ast. \ell = a \cdot \ell'\]
Multi-sorted Logics

- In general we need to reason about several data domains simultaneously.
- We will consider domains of interpretation of the form

\[(D_1, \ldots, D_n, Op, Rel)\]

where the operations and relations are defined over one or several of the data domains \(D_1, \ldots, D_n\).
- Example: \((\text{List}[\ast], \text{Nat}, \{[], \cdot, @, \text{Lgth}, \text{At}, 0, s, +\}, \{=, \leq\})\)
Specifying a sorting function

Define an Input-Output relation $\text{Spec\_Sort}(\ell, \ell')$ ?
Specifying a sorting function

Define an Input-Output relation $\text{Spec}_{\text{Sort}}(\ell, \ell')$?

- The output list is ordered:

$$\text{Ordered}(\ell) = \forall i, j, \in \text{Nat.} \ (0 \leq i < j < |\ell| \Rightarrow \ell[i] \leq \ell[j])$$
Specify a sorting function

Define an Input-Output relation \( \text{Spec}_\text{Sort}(\ell, \ell') \) ?

- The output list is ordered:
  
  \[
  \text{Ordered}(\ell) = \forall i, j, \in \text{Nat}. \ (0 \leq i < j < |\ell| \Rightarrow \ell[i] \leq \ell[j])
  \]

- Is it complete ?
Specifying a sorting function (cont.)

- The output list is a permutation of the input list.
The output list is a permutation of the input list.

Can we express this property in

\[ \text{FO}(\text{List}[\ast], \text{Nat}, \{[], \cdot, @, \text{Lgth}, \text{At}, 0, s, +\}, \{=, \leq\})? \]
Specifying a sorting function (cont.)

- The output list is a permutation of the input list.

- Can we express this property in
  \[ \text{FO}(List[\star], \text{Nat}, \{ [], \cdot, \oplus, Lgth, At, 0, s, + \}, \{ =, \leq \})? \]

- Every element in the input appears in the output, and vice-versa:
  \[
  \forall i \in \text{Nat}. \ 0 \leq i < |\ell_1| \Rightarrow \exists j \in \text{Nat}. \ (0 \leq j < |\ell_2| \land \ell_1[i] = \ell_2[j])
  \]
  \[
  \forall i \in \text{Nat}. \ 0 \leq i < |\ell_2| \Rightarrow \exists j \in \text{Nat}. \ (0 \leq j < |\ell_1| \land \ell_1[i] = \ell_2[j])
  \]
Specifying a sorting function (cont.)

- The output list is a permutation of the input list.
- Can we express this property in \( \text{FO}(\text{List}[\star], \text{Nat}, \{[], \cdot, @, \text{Lgth}, \text{At}, 0, s, +\}, \{=, \leq\}) \)?
- Every element in the input appears in the output, and vice-versa:
  \[ \forall i \in \text{Nat}. \ 0 \leq i < |\ell_1| \Rightarrow \exists j \in \text{Nat}. \ (0 \leq j < |\ell_2| \land \ell_1[i] = \ell_2[j]) \]
  \[ \forall i \in \text{Nat}. \ 0 \leq i < |\ell_2| \Rightarrow \exists j \in \text{Nat}. \ (0 \leq j < |\ell_1| \land \ell_1[i] = \ell_2[j]) \]
- Still not sufficient: \( \ell_1 = [2, 5, 2] \) and \( \ell_2 = [2, 5] \)
The output list is a permutation of the input list.

Can we express this property in $\text{FO}(List[\ast], Nat, \{[], \cdot, \@, \text{Lgth}, \text{At}, 0, s, +\}, \{=, \leq\})$?

Every element in the input appears in the output, and vice-versa:

$$\forall i \in Nat. 0 \leq i < |\ell_1| \Rightarrow \exists j \in Nat. (0 \leq j < |\ell_2| \land \ell_1[i] = \ell_2[j])$$

$$\forall i \in Nat. 0 \leq i < |\ell_2| \Rightarrow \exists j \in Nat. (0 \leq j < |\ell_1| \land \ell_1[i] = \ell_2[j])$$

Still not sufficient: $\ell_1 = [2, 5, 2]$ and $\ell_2 = [2, 5]$

The input and output lists have the same length: $|\ell_1| = |\ell_2|$
The output list is a permutation of the input list.

Can we express this property in $\text{FO}(\text{List}[\ast], \text{Nat}, \{[], \cdot, @, \text{Lgth}, \text{At}, 0, s, +\}, \{=, \leq\})$?

Every element in the input appears in the output, and vice-versa:

\[
\forall i \in \text{Nat}. \ 0 \leq i < |\ell_1| \Rightarrow \exists j \in \text{Nat}. (0 \leq j < |\ell_2| \land \ell_1[i] = \ell_2[j])
\]

\[
\forall i \in \text{Nat}. \ 0 \leq i < |\ell_2| \Rightarrow \exists j \in \text{Nat}. (0 \leq j < |\ell_1| \land \ell_1[i] = \ell_2[j])
\]

Still not sufficient: $\ell_1 = [2, 5, 2]$ and $\ell_2 = [2, 5]$

The input and output lists have the same length: $|\ell_1| = |\ell_2|$

Counter-example: $\ell_1 = [2, 5, 2]$ and $\ell_2 = [5, 2, 5]$
Specifying a sorting function (cont.)

- The output list is a permutation of the input list.
- Can we express this property in
  \[\text{FO}(\text{List}[\ast], \text{Nat}, \{[], \cdot, \@, \text{Lgth}, \text{At}, 0, s, +\}, \{=, \leq\})?\]
- Every element in the input appears in the output, and vice-versa:
  \[
  \forall i \in \text{Nat}. \, 0 \leq i < |\ell_1| \Rightarrow \exists j \in \text{Nat}. \, (0 \leq j < |\ell_2| \land \ell_1[i] = \ell_2[j])
  \]
  \[
  \forall i \in \text{Nat}. \, 0 \leq i < |\ell_2| \Rightarrow \exists j \in \text{Nat}. \, (0 \leq j < |\ell_1| \land \ell_1[i] = \ell_2[j])
  \]
- Still not sufficient: \(\ell_1 = [2, 5, 2]\) and \(\ell_2 = [2, 5]\)
- The input and output lists have the same length: \(|\ell_1| = |\ell_2|\)
- Counter-example: \(\ell_1 = [2, 5, 2]\) and \(\ell_2 = [5, 2, 5]\)
- We must to count the number of occurrences of each element!
Multisets

- The domain of multisets: $Multiset[⋆] \equiv ⋆ \rightarrow Nat$
Multisets

- The domain of multisets: $\text{Multiset}[\star] \equiv \star \rightarrow \text{Nat}$

- Operations on multisets:
  - $\emptyset : \text{Multiset}[\star]$
  - $\text{Sg} : \star \rightarrow \text{Multiset}[\star]$
  - $\uplus : \text{Multiset}[\star] \times \text{Multiset}[\star] \rightarrow \text{Multiset}[\star]$
Multisets

- The domain of multisets: $\textit{Multiset}[\ast] \equiv \ast \rightarrow \textit{Nat}$

- Operations on multisets:
  - $\emptyset : \textit{Multiset}[^\ast]$
  - $Sg : \ast \rightarrow \textit{Multiset}[\ast]$
  - $\uplus : \textit{Multiset}[\ast] \times \textit{Multiset}[\ast] \rightarrow \textit{Multiset}[\ast]$

- Definitions:
  - $\emptyset = \lambda x \in \ast. \ 0$
  - $Sg(a) = \lambda x \in \ast. \ if \ x = a \ then \ 1 \ else \ 0$
  - $M_1 \uplus M_2 = \lambda x \in \ast. \ M_1(x) + M_2(x)$
Multisets

- The domain of multisets: $\text{Multiset}[\star] \equiv \star \rightarrow \text{Nat}$

- Operations on multisets:
  - $\emptyset : \text{Multiset}[\star]$
  - $\text{Sg} : \star \rightarrow \text{Multiset}[\star]$
  - $\uplus : \text{Multiset}[\star] \times \text{Multiset}[\star] \rightarrow \text{Multiset}[\star]$

- Definitions:
  - $\emptyset = \lambda x \in \star. 0$
  - $\text{Sg}(a) = \lambda x \in \star. \text{if } x = a \text{ then } 1 \text{ else } 0$
  - $M_1 \uplus M_2 = \lambda x \in \star. M_1(x) + M_2(x)$

- Example:
  
  $\text{Sg}(0) \uplus (\text{Sg}(5) \uplus \text{Sg}(0)) = \lambda x \in \text{Nat}. \text{if } x = 0 \text{ then } 2 \text{ else } (\text{if } x = 5 \text{ then } 1 \text{ else } 0)$
Multisets: Properties

- Neutral element: $\emptyset \uplus M = M \uplus \emptyset = M$
- Commutativity: $M_1 \uplus M_2 = M_2 \uplus M_1$
- Associativity: $M_1 \uplus (M_2 \uplus M_3) = (M_1 \uplus M_2) \uplus M_3$
Multisets: Properties

- Neutral element: $\emptyset \uplus M = M \uplus \emptyset = M$
- Commutativity: $M_1 \uplus M_2 = M_2 \uplus M_1$
- Associativity: $M_1 \uplus (M_2 \uplus M_3) = (M_1 \uplus M_2) \uplus M_3$

Proofs: Use properties of natural numbers.
Abstracting order in a list:

\[ Ms : List[\star] \rightarrow Multiset[\star] \]

Definition:

\[
\begin{align*}
Ms([]) &= \emptyset \\
Ms(a \cdot \ell) &= Sg(a) \uplus Ms(\ell)
\end{align*}
\]
From Lists to Multisets

- Abstracting order in a list:

\[ Ms : List[\star] \rightarrow Multiset[\star] \]

- Definition:

\[
Ms([]) = \emptyset \\
Ms(a \cdot \ell) = Sg(a) \sqcup Ms(\ell)
\]

- Example: \( Ms(b \cdot a \cdot b \cdot []) = \lambda x \in \{a, b\}. \text{if } x = a \text{ then } 1 \text{ else } 2 \)
From Lists to Multisets (cont.): Properties

- $Ms(\ell_1 @ \ell_2) = Ms(\ell_2 @ \ell_1) = Ms(\ell_1) \uplus Ms(\ell_2)$
- $Ms(Rev(\ell)) = Ms(\ell)$
From Lists to Multisets (cont.): Properties

- \( Ms(\ell_1 \circ \ell_2) = Ms(\ell_2 \circ \ell_1) = Ms(\ell_1) \uplus Ms(\ell_2) \)
- \( Ms(\text{Rev}(\ell)) = Ms(\ell) \)

Proofs: Induction the structure of lists.
From Lists to Multisets (cont.): Checking membership

- **Type:**
  \[ Is\_in : \star \times List[\star] \rightarrow Bool \]

- **Definition:**
  \[ Is\_in(a, \ell) = Ms(\ell)(a) > 0 \]
Specifying a sorting function (cont.)

\[ \text{Spec}_{\text{Sort}}(\ell, \ell') = \]

\[ \forall i, j, \in \text{Nat. } (0 \leq i < j < |\ell| \Rightarrow \ell'[i] \leq \ell'[j]) \]

\[ \land \]

\[ Ms(\ell) = Ms(\ell') \]
Conclusion

- Specifications are abstract definitions of the effect of functions
- No implementation details are imposed.
- Logic is a natural for abstract description of input-output relations
- Abstraction allows modular design:
  - The user of a function needs only to know its specification.
  - The implementor must ensure the satisfaction of the specification.