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

$$f : Dom \rightarrow CoDom$$

- How to describe in an abstract way its behavior ?
- Abstraction: No implementation details.
- Specification: A relation $Spec\_f$ between inputs and outputs of f

$$Spec\_f(In, Out) \subseteq Dom \times CoDom$$

- What is a suitable (natural) formalism for describing such a relation?

Logic-based Specification Language

- Example: Specification of the Append function:

$$
\begin{aligned}
&\mathit{Spec\_Append}(\ell_1, \ell_2, \ell) = \\
&\qquad |\ell| = |\ell_1| + |\ell_2| \;\wedge \\
&\qquad \forall i \in \mathit{Nat}.\ (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i] \;\wedge \\
&\qquad \forall i \in \mathit{Nat}.\ (0 \leq i < |\ell_2|) \Rightarrow \ell[|\ell_1| + i] = \ell_2[i]
\end{aligned}
$$

where:

$$
\begin{aligned}
&\forall \ell \in \mathit{List}[\star].\ \forall i \in \mathit{Nat}.\ \forall e \in \star.\ \ \ell[i] = e \iff \\
&\qquad (i < |\ell|) \wedge \\
&\qquad \exists \ell'.\ \big(\ell = a \cdot \ell' \wedge \\
&\qquad\qquad ((i = 0 \wedge e = a) \vee (i > 0 \wedge e = \ell'[i-1]))\big)
\end{aligned}
$$

Logic-based Specification Language

- Example: Specification of the Append function:

$$
\begin{aligned}
&Spec\_Append(\ell_1, \ell_2, \ell) = \\
&\qquad |\ell| = |\ell_1| + |\ell_2| \;\wedge \\
&\qquad \forall i \in Nat.\ (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i] \;\wedge \\
&\qquad \forall i \in Nat.\ (0 \leq i < |\ell_2|) \Rightarrow \ell[|\ell_1| + i] = \ell_2[i]
\end{aligned}
$$

where:

$$
\begin{aligned}
&\forall \ell \in List[\star].\ \forall i \in Nat.\ \forall e \in \star.\ \ \ell[i] = e \iff \\
&\qquad (i < |\ell|) \wedge \\
&\qquad \exists \ell'.\ \big(\ell = a \cdot \ell' \wedge \\
&\qquad\qquad ((i = 0 \wedge e = a) \vee (i > 0 \wedge e = \ell'[i-1])\big)\big)
\end{aligned}
$$

- $\Rightarrow$ 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:
  - $(Bool, \{\mathtt{tt}, \mathtt{ff}, \mathtt{not}, \mathtt{or}, \mathtt{and}\}, \{=\})$
  - $(Nat, \{0, s, +\}, \{\leq\})$
  - $(List[\star], \{[], \cdot, \mathtt{@}\}, \{=\})$

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:

$$t ::= v \in Var \mid op(t, \dots, t)$$

  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:

$$t ::= v \in Var \mid op(t, \dots, t)$$

  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 $\nu : Var \to D$ be a valuation of the variables.
  - Then, $\langle t \rangle_\nu$ is the value in D obtained by the evaluation of t, using ν 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 $\nu : Var \to D$ be a valuation of the variables.
  - Then, $\langle t \rangle_\nu$ is the value in D obtained by the evaluation of t, using ν as valuation of the variables.
  - Example: Given $\nu = \{(x, 2), (y, 1), (z, 4)\}$, we have

$$\langle x \rangle_\nu = 2 \quad \langle x + 2y \rangle_\nu = 4 \quad \langle (x * z) + (y + 1) \rangle_\nu = 10$$

First Order Logic: Syntax of formulas

- Formulas:

$$\phi ::= p(t_1, \ldots, t_n) \mid \neg\phi \mid \phi \vee \phi \mid \exists v.\ \phi$$

  where $p \in Pred$ and $v \in Var$.

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

- An occurrence of a variable x is bound in a formula ϕ 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 ϕ if its occurrences in ϕ are unbound. A formula is closed if it has not free variables.

- Examples:
  - $\phi_1 = \forall x, y.\ x \leq y \Rightarrow \exists z.\ (x \leq z \wedge z < y)$ 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.
  - $\phi_4 = x \leq y \wedge \exists z.\ y \leq z \wedge z \leq 5$ is an open formula. Its free variables are x and y.

First Order Logic: Semantics of formulas

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

$$head \ : \ List[\star] \rightarrow \star$$

$$Spec\_head(\ell, a) \ = \ \exists \ell' \in List[\star].\ \ell = a \cdot \ell'$$

- tail function:

$$tail \ : \ List[\star] \rightarrow List[\star]$$

$$Spec\_tail(\ell, \ell') \ = \ \exists a \in \star.\ \ell = a \cdot \ell'$$

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: $(List[\star], Nat, \{[], \cdot, @, Lgth, At, 0, s, +\}, \{=, \leq\})$

Specifying a sorting function

Define an Input-Output relation $Spec\_Sort(\ell, \ell')$ *?*

Specifying a sorting function

Define an Input-Output relation $Spec\_Sort(\ell, \ell')$ *?*

- The output list is ordered:

$$Ordered(\ell) = \forall i, j, \in Nat.\ (0 \leq i < j < |\ell| \Rightarrow \ell[i] \leq \ell[j])$$

Specifying a sorting function

Define an Input-Output relation $Spec\_Sort(\ell, \ell')$ *?*

- The output list is ordered:

$$Ordered(\ell) = \forall i, j, \in Nat.\ (0 \leq i < j < |\ell| \Rightarrow \ell[i] \leq \ell[j])$$

- Is it complete ?

Specifying a sorting function (cont.)

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

Specifying a sorting function (cont.)

- The output list is a permutation of the input list.
- Can we express this property in $\mathrm{FO}(List[\star], Nat, \{[], \cdot, @, Lgth, At, 0, s, +\}, \{=, \leq\})$?

Specifying a sorting function (cont.)

- The output list is a permutation of the input list.
- Can we express this property in $\mathrm{FO}(List[\star], Nat, \{[], \cdot, @, Lgth, At, 0, s, +\}, \{=, \leq\})$?
- Every element in the input appears in the output, and vice-versa:
  $\forall i \in Nat.\ 0 \leq i < |\ell_1| \Rightarrow \exists j \in Nat.\ (0 \leq j < |\ell_2| \wedge \ell_1[i] = \ell_2[j])$
  $\forall i \in Nat.\ 0 \leq i < |\ell_2| \Rightarrow \exists j \in Nat.\ (0 \leq j < |\ell_1| \wedge \ell_1[i] = \ell_2[j])$

Specifying a sorting function (cont.)

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

Specifying a sorting function (cont.)

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

Specifying a sorting function (cont.)

- The output list is a permutation of the input list.
- Can we express this property in $FO(\mathit{List}[\star], \mathit{Nat}, \{[], \cdot, @, \mathit{Lgth}, \mathit{At}, 0, s, +\}, \{=, \leq\})$?
- Every element in the input appears in the output, and vice-versa:
  $\forall i \in \mathit{Nat}.\ 0 \leq i < |\ell_1| \Rightarrow \exists j \in \mathit{Nat}.\ (0 \leq j < |\ell_2| \wedge \ell_1[i] = \ell_2[j])$
  $\forall i \in \mathit{Nat}.\ 0 \leq i < |\ell_2| \Rightarrow \exists j \in \mathit{Nat}.\ (0 \leq j < |\ell_1| \wedge \ell_1[i] = \ell_2[j])$
- 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 $\mathrm{FO}(List[\star], Nat, \{[], \cdot, @, Lgth, At, 0, s, +\}, \{=, \leq\})$?
- Every element in the input appears in the output, and vice-versa:
  $\forall i \in Nat.\ 0 \leq i < |\ell_1| \Rightarrow \exists j \in Nat.\ (0 \leq j < |\ell_2| \wedge \ell_1[i] = \ell_2[j])$
  $\forall i \in Nat.\ 0 \leq i < |\ell_2| \Rightarrow \exists j \in Nat.\ (0 \leq j < |\ell_1| \wedge \ell_1[i] = \ell_2[j])$
- 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: $\mathit{Multiset}[\star] \;\equiv\; \star \rightarrow \mathit{Nat}$

Multisets

- The domain of multisets: $\mathit{Multiset}[\star] \equiv \star \to \mathit{Nat}$
- Operations on multisets:
  - $\emptyset : \mathit{Multiset}[\star]$
  - $\mathit{Sg} : \star \to \mathit{Multiset}[\star]$
  - $\uplus : \mathit{Multiset}[\star] \times \mathit{Multiset}[\star] \to \mathit{Multiset}[\star]$

Multisets

- The domain of multisets: $\mathit{Multiset}[\star] \equiv \star \to \mathit{Nat}$
- Operations on multisets:
  - $\emptyset : \mathit{Multiset}[\star]$
  - $Sg : \star \to \mathit{Multiset}[\star]$
  - $\uplus : \mathit{Multiset}[\star] \times \mathit{Multiset}[\star] \to \mathit{Multiset}[\star]$
- Definitions:
  - $\emptyset = \lambda x \in \star.\ 0$
  - $Sg(a) = \lambda x \in \star.\ \texttt{if}\ x = a\ \texttt{then}\ 1\ \texttt{else}\ 0$
  - $M_1 \uplus M_2 = \lambda x \in \star.\ M_1(x) + M_2(x)$

Multisets

- The domain of multisets: $\mathit{Multiset}[\star] \;\equiv\; \star \rightarrow \mathit{Nat}$
- Operations on multisets:
  - $\emptyset : \mathit{Multiset}[\star]$
  - $\mathit{Sg} : \star \rightarrow \mathit{Multiset}[\star]$
  - $\uplus : \mathit{Multiset}[\star] \times \mathit{Multiset}[\star] \rightarrow \mathit{Multiset}[\star]$
- Definitions:
  - $\emptyset = \lambda x \in \star.\ 0$
  - $\mathit{Sg}(a) = \lambda x \in \star.\ \texttt{if}\ x = a\ \texttt{then}\ 1\ \texttt{else}\ 0$
  - $M_1 \uplus M_2 = \lambda x \in \star.\ M_1(x) + M_2(x)$
- Example:
  $\mathit{Sg}(0) \uplus (\mathit{Sg}(5) \uplus \mathit{Sg}(0)) =$
  $\lambda x \in \mathit{Nat}.\ \texttt{if}\ x = 0\ \texttt{then}\ 2\ \texttt{else}\ (\texttt{if}\ x = 5\ \texttt{then}\ 1\ \texttt{else}\ 0)$

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.

From Lists to Multisets

- Abstracting order in a list:

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

- Definition:

$$\begin{array}{rcl} Ms([]) & = & \emptyset \\ Ms(a \cdot \ell) & = & Sg(a) \uplus Ms(\ell) \end{array}$$

From Lists to Multisets

- Abstracting order in a list:

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

- Definition:

$$\begin{array}{rcl} Ms([]) & = & \emptyset \\ Ms(a \cdot \ell) & = & Sg(a) \uplus Ms(\ell) \end{array}$$

- Example: $Ms(b \cdot a \cdot b \cdot []) = \lambda x \in \{a, b\}.\ \texttt{if}\ x = a\ \texttt{then}\ 1\ \texttt{else}\ 2$

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

- $Ms(\ell_1 @ \ell_2) = Ms(\ell_2 @ \ell_1) = Ms(\ell_1) \uplus Ms(\ell_2)$
- $Ms(Rev(\ell)) = Ms(\ell)$

- 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.)

$Spec\_Sort(\ell, \ell') =$

$$\forall i, j, \in Nat.\ (0 \leq i < j < |\ell| \Rightarrow \ell'[i] \leq \ell'[j])$$
$$\wedge$$
$$Ms(\ell) = Ms(\ell')$$

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.