Theme 1: Abstract Reasoning

Lecture 3: Inductive Correctness Proofs

Ahmed Bouajjani

Paris Diderot University, Paris 7

January 2014

Implementation vs. Specification

- Assume we want to define

$$f : Dom \to CoDom$$

- Consider an abstract specification

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

- Let $Impl\_f$ be an implementation of f (e.g., as a recursive function)
- The implementation $Impl\_f$ satisfies the specification $Spec\_f$ iff:

$$\forall In \in Dom.\ \forall Out \in CoDom.\ (Impl\_f(In) = Out) \Longrightarrow Spec\_f(In, Out)$$

- Correctness is always defined with respect to a given specification!

Example: The Append function

- Type:

$$Append : List[\star] \times List[\star] \rightarrow List[\star]$$

- Specification:

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

- Implementation:

$$\begin{aligned}
[]@\ell &= \ell \\
(a \cdot \ell_1)@\ell_2 &= a \cdot (\ell_1 @ \ell_2)
\end{aligned}$$

- Correctness:

$$\forall \ell_1, \ell_2, \ell.\ (\ell_1 @ \ell_2 = \ell) \Longrightarrow Spec\_Append(\ell_1, \ell_2, \ell)$$

Correctness proof: Induction

Case $\ell_1 = []$: $\ell = []@\ell_2 = \ell_2$.

$$
\begin{array}{l}
(|\ell| = 0 \ + |\ell_2|) \wedge \\
(\forall i.\ 0 \leq i < 0 \Rightarrow ...) \wedge \\
(\forall i.\ 0 \leq i < |\ell_2| \Rightarrow \ell[0+i] = \ell_2[i])
\end{array}
$$

Correctness proof: Induction

Case $\ell_1 = a \cdot \ell_1'$: $\ell = a \cdot (\ell_1' @ \ell_2)$. Let $\ell' = \ell_1' @ \ell_2$.

- Induction hypothesis:

$$
\begin{array}{l}
(|\ell'| = |\ell_1'| \ + |\ell_2|) \wedge \\
(\forall i \in Nat.\ (0 \leq i < |\ell_1'|) \Rightarrow \ell'[i] = \ell_1'[i]) \wedge \\
(\forall i \in Nat.\ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell_1'| + i] = \ell_2[i])
\end{array}
$$

Correctness proof: Induction

Case $\ell_1 = a \cdot \ell_1'$: $\ell = a \cdot (\ell_1' @ \ell_2)$. Let $\ell' = \ell_1' @ \ell_2$.

- Induction hypothesis:

$$
\begin{array}{l}
(|\ell'| = |\ell_1'| + |\ell_2|) \wedge \\
(\forall i \in Nat.\ (0 \leq i < |\ell_1'|) \Rightarrow \ell'[i] = \ell_1'[i]) \wedge \\
(\forall i \in Nat.\ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell_1'| + i] = \ell_2[i])
\end{array}
$$

- 1st point: $|\ell| = 1 + |\ell_1' @ \ell_2| = 1 + |\ell_1'| + |\ell_2| = |\ell_1| + |\ell_2|$

Correctness proof: Induction

Case $\ell_1 = a \cdot \ell_1'$: $\ell = a \cdot (\ell_1' @ \ell_2)$. Let $\ell' = \ell_1' @ \ell_2$.

- Induction hypothesis:

$$
\begin{array}{l}
(|\ell'| = |\ell_1'| \; + |\ell_2|) \wedge \\
(\forall i \in \mathit{Nat}.\ (0 \leq i < |\ell_1'|) \Rightarrow \ell'[i] = \ell_1'[i]) \wedge \\
(\forall i \in \mathit{Nat}.\ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell_1'| + i] = \ell_2[i])
\end{array}
$$

- 1st point: $|\ell| = 1 + |\ell_1' @ \ell_2| = 1 + |\ell_1'| + |\ell_2| = |\ell_1| \; + |\ell_2|$
- We have (by definition of the At operator):
  1. $\ell[0] = a = \ell_1[0]$,
  2. $\forall i.\ 1 \leq i < |\ell_1| \Rightarrow \ell_1[i] = \ell_1'[i-1]$
  3. $\forall i.\ 1 \leq i < |\ell| \Rightarrow \ell[i] = \ell'[i-1]$

Correctness proof: Induction

Case $\ell_1 = a \cdot \ell_1'$: $\ell = a \cdot (\ell_1' @ \ell_2)$. Let $\ell' = \ell_1' @ \ell_2$.

- Induction hypothesis:

$$
\begin{array}{l}
(|\ell'| = |\ell_1'| \ + |\ell_2|) \wedge \\
(\forall i \in Nat.\ (0 \leq i < |\ell_1'|) \Rightarrow \ell'[i] = \ell_1'[i]) \wedge \\
(\forall i \in Nat.\ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell_1'| + i] = \ell_2[i])
\end{array}
$$

- 1st point: $|\ell| = 1 + |\ell_1' @ \ell_2| = 1 + |\ell_1'| + |\ell_2| = |\ell_1| \ + |\ell_2|$
- We have (by definition of the At operator):
  1. $\ell[0] = a = \ell_1[0]$,
  2. $\forall i.\ 1 \leq i < |\ell_1| \Rightarrow \ell_1[i] = \ell_1'[i-1]$
  3. $\forall i.\ 1 \leq i < |\ell| \Rightarrow \ell[i] = \ell'[i-1]$
- 2nd point:
  - IH.2 $\Rightarrow \forall i.\ (1 \leq i < |\ell_1'| + 1) \Rightarrow \ell'[i-1] = \ell_1'[i-1]$
  - (2) $\Rightarrow \forall i.\ (1 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i]$
  - (1) $\Rightarrow \forall i.\ (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i]$

Correctness proof: Induction

Case $\ell_1 = a \cdot \ell'_1$: $\ell = a \cdot (\ell'_1 @ \ell_2)$. Let $\ell' = \ell'_1 @ \ell_2$.

- Induction hypothesis:

$$(|\ell'| = |\ell'_1| + |\ell_2|) \wedge$$
$$(\forall i \in Nat.\ (0 \leq i < |\ell'_1|) \Rightarrow \ell'[i] = \ell'_1[i]) \wedge$$
$$(\forall i \in Nat.\ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell'_1| + i] = \ell_2[i])$$

- 1st point: $|\ell| = 1 + |\ell'_1 @ \ell_2| = 1 + |\ell'_1| + |\ell_2| = |\ell_1| + |\ell_2|$
- We have (by definition of the At operator):
  1. $\ell[0] = a = \ell_1[0]$,
  2. $\forall i.\ 1 \leq i < |\ell_1| \Rightarrow \ell_1[i] = \ell'_1[i-1]$
  3. $\forall i.\ 1 \leq i < |\ell| \Rightarrow \ell[i] = \ell'[i-1]$
- 2nd point:
  - IH.2 $\Rightarrow \forall i.\ (1 \leq i < |\ell'_1| + 1) \Rightarrow \ell'[i-1] = \ell'_1[i-1]$
  - (2) $\Rightarrow \forall i.\ (1 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i]$
  - (1) $\Rightarrow \forall i.\ (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i]$
- 3rd point: left as an exercise.

Sorting function: An Implementation

- Reason about the structure of the input list?

$$\begin{aligned} Sort([]) &= \\ Sort(a \cdot \ell) &= \end{aligned}$$

- How to sort $a \cdot \ell$ if we can sort ℓ ?

Sorting function: An Implementation

- Reason about the structure of the input list?

$$\begin{array}{rcl} Sort([]) & = & [] \\ Sort(a \cdot \ell) & = & Insert(a, Sort(\ell)) \end{array}$$

- We need to insert a in the sorted list corresponding to ℓ.

Sorting function: An Implementation

- Reason about the structure of the input list?

$$
\begin{aligned}
\mathit{Sort}([]) &= [] \\
\mathit{Sort}(a \cdot \ell) &= \mathit{Insert}(a, \mathit{Sort}(\ell))
\end{aligned}
$$

- We need to insert a in the sorted list corresponding to ℓ.
- What is the formal specification of *Insert*?

Sorting function: An Implementation

- Reason about the structure of the input list?

$$\begin{array}{rcl} Sort([]) & = & [] \\ Sort(a \cdot \ell) & = & Insert(a, Sort(\ell)) \end{array}$$

- We need to insert a in the sorted list corresponding to ℓ.
- What is the formal specification of *Insert*?
- Type:

$$Insert : \star \times List[\star] \rightarrow List[\star]$$

- Input-Output relation:

$$\begin{array}{l} Spec\_Insert(a, \ell, \ell') = \\ \quad Ordered(\ell) \Rightarrow \big(Ordered(\ell') \wedge (Ms(\ell') = Sg(a) \uplus Ms(\ell))\big) \end{array}$$

Sorting function: Another Implementation

- Reason about the structure of the output list?

$$
\begin{aligned}
Sort([]) &= \\
Sort(a \cdot \ell) &=
\end{aligned}
$$

- If the output is of the form $e \cdot \ell'$, what is e ? and how to obtain ℓ' ?

Sorting function: Another Implementation

- Reason about the structure of the output list?

$$\begin{array}{rcl} Sort([]) & = & [] \\ Sort(a \cdot \ell) & = & \texttt{let } (m, \ell_m) = Extract\_min(a \cdot \ell) \texttt{ in } m \cdot Sort(\ell_m) \end{array}$$

- Extract the minimal element m of ℓ, and sort the rest of the list ℓ_m.

Sorting function: Another Implementation

- Reason about the structure of the output list?

$$\begin{array}{rcl} Sort([]) & = & [] \\ Sort(a \cdot \ell) & = & \texttt{let } (m, \ell_m) = Extract\_min(a \cdot \ell) \texttt{ in } m \cdot Sort(\ell_m) \end{array}$$

- Extract the minimal element m of ℓ, and sort the rest of the list ℓ_m.
- Specification of *Extract_min*:
  - Type: $Extract\_min : List[\star] \rightarrow \star \times List[\star]$
  - Input-Output relation:

$$\begin{array}{l} Spec\_Extract\_min(\ell_1, m, \ell_2) = \\ \qquad \ell_1 \neq [] \Rightarrow Is\_in(m, \ell_1) \;\wedge \\ \qquad\qquad \forall a \in \star.\ Is\_in(a, \ell_1) \Rightarrow m \leq a \;\wedge \\ \qquad\qquad Ms(\ell_1) = Sg(m) \uplus Ms(\ell_2) \end{array}$$

Sorting function: Yet Another Implementation

- Reason again about the structure of the output list?

$$\begin{aligned} Sort([]) &= \\ Sort(a \cdot \ell) &= \end{aligned}$$

- Assume that when a is at its place in the output, it has ℓ_{left} and ℓ_{right} to its left and right, respectively. How to compute ℓ_{left} and ℓ_{right}?

Sorting function: Yet Another Implementation

- Reason again about the structure of the output list?

$$\begin{array}{rcl} Sort([]) & = & [] \\ Sort(a \cdot \ell) & = & \texttt{let } (\ell_1, \ell_2) = \texttt{split } (a, \ell) \texttt{ in } Sort(\ell_1) @ (a \cdot Sort(\ell_2)) \end{array}$$

- Split ℓ into 2 lists containing the elements smaller and greater than a.

Sorting function: Yet Another Implementation

- Reason again about the structure of the output list?

$$
\begin{array}{rcl}
Sort([]) & = & [] \\
Sort(a \cdot \ell) & = & \texttt{let } (\ell_1, \ell_2) = \texttt{split } (a, \ell) \texttt{ in } Sort(\ell_1) \texttt{@} (a \cdot Sort(\ell_2))
\end{array}
$$

- Split ℓ into 2 lists containing the elements smaller and greater than a.
- Specification of *Split*:
  - Type: $Split : \star \times List[\star] \rightarrow List[\star] \times List[\star]$
  - Input-Output relation:

$$
\begin{array}{l}
Spec\_Split(a, \ell, \ell_1, \ell_2) = \\
\quad Ms(\ell) = Ms(\ell_1) \uplus Ms(\ell_2) \ \wedge \\
\quad \forall e \in \star.\ ((Is\_In(e, \ell_1) \Rightarrow e \leq a) \wedge (Is\_In(e, \ell_2) \Rightarrow a < e))
\end{array}
$$

Proving correctness of the Recursive Insertion Sort

- Consider the implementation:

$$\begin{aligned} Ins\_Sort([]) &= [] \\ Ins\_Sort(a \cdot \ell) &= Insert(a, Ins\_Sort(\ell)) \end{aligned}$$

- Assume that Insert is correct w.r.t. its specification:

$$\forall a \in \star.\ \forall \ell, \ell' \in List[\star]\ .\ Insert(a, \ell) = \ell' \Longrightarrow Spec\_Insert(a, \ell, \ell')$$

where

$$\begin{aligned} & Spec\_Insert(a, \ell, \ell') = \\ & \quad Ordered(\ell) \Rightarrow (Ordered(\ell') \wedge (Ms(\ell') = Sg(a) \uplus Ms(\ell))) \end{aligned}$$

- and prove that:

$$\forall \ell, \ell' \in List[\star]\ .\ (Ins\_Sort(\ell) = \ell') \Longrightarrow Spec\_Sort(\ell, \ell')$$

where

$$\begin{aligned} & Spec\_Sort(\ell, \ell') = \\ & \quad \forall i, j, \in Nat.\ (0 \leq i < j < |\ell'| \Rightarrow \ell'[i] \leq \ell'[j])\ \wedge \\ & \quad Ms(\ell) = Ms(\ell') \end{aligned}$$

Proof

Case $\ell = []$: Trivial.

Case $\ell = a \cdot \ell_1$: We have $\ell' = \mathit{Ins\_Sort}(\ell) = \mathit{Insert}(a, \mathit{Ins\_Sort}(\ell_1))$.

- Let $\ell'_1 = \mathit{Ins\_Sort}(\ell_1)$.
- Induction hypothesis: $\mathit{Ordered}(\ell'_1) \wedge \mathit{Ms}(\ell_1) = \mathit{Ms}(\ell'_1)$.
- We assume Insert correct w.r.t. its specification:

$$\begin{array}{l} \mathit{Spec\_Insert}(a, \ell'_1, \ell') = \\ \quad \mathit{Ordered}(\ell'_1) \Rightarrow (\mathit{Ordered}(\ell') \wedge (\mathit{Ms}(\ell') = \mathit{Sg}(a) \uplus \mathit{Ms}(\ell'_1))) \end{array}$$

- Since we have $\mathit{Ordered}(\ell'_1)$ by Ind. Hyp., then the following holds:

$$\mathit{Ordered}(\ell') \wedge (\mathit{Ms}(\ell') = \mathit{Sg}(a) \uplus \mathit{Ms}(\ell'_1))$$

- We have $\mathit{Ms}(\ell) = \mathit{Sg}(a) \uplus \mathit{Ms}(\ell_1) = \mathit{Sg}(a) \uplus \mathit{Ms}(\ell'_1) = \mathit{Ms}(\ell')$.
- Then, we obtain $\mathit{Ordered}(\ell') \wedge \mathit{Ms}(\ell) = \mathit{Ms}(\ell')$.

Recursive Insertion

- Type:

$$Insert : \star \times List[\star] \rightarrow List[\star]$$

- Input-Output specification:

$$\begin{array}{l} Spec\_Insert(a, \ell, \ell') = \\ \quad Ordered(\ell) \Rightarrow (Ordered(\ell') \wedge (Ms(\ell') = Sg(a) \uplus Ms(\ell))) \end{array}$$

- Recursive implementation:

$$\begin{array}{rcl} Insert(a, []) & = & a \cdot [] \\ Insert(a, b \cdot \ell) & = & \texttt{if}\ a \leq b\ \texttt{then}\ a \cdot (b \cdot \ell) \\ & & \qquad\qquad \texttt{else}\ b \cdot (Insert(a, \ell)) \end{array}$$

Recursive Insertion: Correctness proof

left as an exercise ...

Correctness of the Quick sort

- Consider the sorting function:

$$\begin{array}{rcl} qsort([]) & = & [] \\ qsort(a \cdot \ell) & = & \texttt{let}\ (\ell_1, \ell_2) = \texttt{split}\ (a, \ell)\ \texttt{in} \\ & & \qquad qsort(\ell_1) @ (a \cdot qsort(\ell_2)) \end{array}$$

- Prove that:

$$\forall \ell, \ell'.\ (qsort(\ell) = \ell') \Longrightarrow Spec\_Sort(\ell, \ell')$$

Correctness of the Quick sort

- Consider the sorting function:

$$\begin{array}{rcl} qsort([]) & = & [] \\ qsort(a \cdot \ell) & = & \texttt{let}\ (\ell_1, \ell_2) = \texttt{split}\ (a, \ell)\ \texttt{in} \\ & & \qquad qsort(\ell_1) @ (a \cdot qsort(\ell_2)) \end{array}$$

- Prove that:

$$\forall \ell, \ell'.\ (qsort(\ell) = \ell') \Longrightarrow Spec\_Sort(\ell, \ell')$$

- We need to assume that the two recursive calls are correct.

Correctness of the Quick sort

- Consider the sorting function:

$$\begin{array}{rcl} qsort([]) & = & [] \\ qsort(a \cdot \ell) & = & \texttt{let}\ (\ell_1, \ell_2) = \texttt{split}\ (a, \ell)\ \texttt{in} \\ & & \qquad qsort(\ell_1) @ (a \cdot qsort(\ell_2)) \end{array}$$

- Prove that:

$$\forall \ell, \ell'.\ (qsort(\ell) = \ell') \Longrightarrow Spec\_Sort(\ell, \ell')$$

- We need to assume that the two recursive calls are correct.
- What is the proof principle which allows that ?

Well founded relations

- Let E be a set, and let $\prec \subseteq E \times E$ a binary relation over E.
- The relation $\prec$ is well founded if it has no infinite descending chains, i.e., no sequences of the form

$$e_0 \succ e_1 \succ \cdots \succ e_i \succ \cdots$$

- $(E, \prec)$ is said to be a well founded set (WFS for short).
- Thm: $\prec$ is well founded iff

$$\forall F \subseteq E.\ F \neq \emptyset \Rightarrow (\exists e \in F.\ \forall e' \in F.\ e' \not\prec e)$$

Well founded relations: Examples

- $(\mathbb{N}, <)$ is a WFS.
- $(\mathbb{Z}, <)$ is not a WFS.
- $(\mathbb{R}_{\geq 0}, <)$ is not a WFS.

Noetherian Induction

- Let $(E, \prec)$ be a WFS, and let $\rho : D \to E$.
- Let $\prec_\rho \subseteq D \times D$ be the relation such that:

$$x \prec_\rho y \iff \rho(x) \prec \rho(y)$$

- Induction rule:

$$\frac{\forall x \in D.\ \big((\forall y.\ y \prec_\rho x \Rightarrow P(y)) \Rightarrow P(x)\big)}{\forall x \in D.\ P(x)}$$

Correctness of the Quick sort (cont.)

- Consider the WFS $(\mathbb{N}, <)$ and the function $\rho : \mathit{List}[\star] \rightarrow \mathbb{N}$ such that

$$\forall \ell \in \mathit{List}[\star].\ \rho(\ell) = |\ell|$$

- The rest of the proof is left as an exercise ...

Conclusion

- Specifications are abstract definitions of the effect of functions.
- No implementation details are imposed. Several implementations can be provided and proved correct w.r.t. an abstract specification.
- 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. This allows to separate issues.
  - The implementor must ensure the satisfaction of the specification: He/she must prove that its implementation satisfies the required satisfaction.
  - It is possible to implement a function and prove its correctness w.r.t. to its specification, assuming that the functions it uses (in external modules) are correct w.r.t. their own specifications.