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

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

- Consider an abstract specification

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

- Let \( \text{Impl}_f \) be an implementation of \( f \) (e.g., as a recursive function)

- The implementation \( \text{Impl}_f \) satisfies the specification \( \text{Spec}_f \) iff:

\[ \forall \text{In} \in \text{Dom}. \ \forall \text{Out} \in \text{CoDom}. \ (\text{Impl}_f(\text{In}) = \text{Out}) \implies \text{Spec}_f(\text{In}, \text{Out}) \]

- Correctness is always defined with respect to a given specification!
Example: The Append function

- **Type:**
  \[
  \text{Append} : \text{List}[\star] \times \text{List}[\star] \rightarrow \text{List}[\star]
  \]

- **Specification:**
  \[
  \text{Spec}_{\text{Append}}(\ell_1, \ell_2, \ell) = \\
  |\ell| = |\ell_1| + |\ell_2| \land \\
  \forall i \in \text{Nat}. (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i] \land \\
  \forall i \in \text{Nat}. (0 \leq i < |\ell_2|) \Rightarrow \ell[|\ell_1| + i] = \ell_2[i]
  \]

- **Implementation:**
  \[
  [] @ \ell = \ell \\
  (a \cdot \ell_1) @ \ell_2 = a \cdot (\ell_1 @ \ell_2)
  \]

- **Correctness:**
  \[
  \forall \ell_1, \ell_2, \ell. (\ell_1 @ \ell_2 = \ell) \implies \text{Spec}_{\text{Append}}(\ell_1, \ell_2, \ell)
  \]
Correctness proof: Induction

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

$$(|\ell| = 0 + |\ell_2|) \land$$
$$(\forall i. 0 \leq i < 0 \Rightarrow ...) \land$$
$$(\forall i. 0 \leq i < |\ell_2| \Rightarrow \ell[0 + i] = \ell_2[i])$$
Correctness proof: Induction

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

- Induction hypothesis:
  
  \[
  (|\ell'| = |\ell'_1| + |\ell_2|) \land
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell'_1|) \Rightarrow \ell'[i] = \ell'_1[i]) \land
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell'_1| + i] = \ell_2[i])
  \]
Correctness proof: Induction

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

- Induction hypothesis:
  
  $$(|\ell'| = |\ell'_1| + |\ell_2|) \land
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell'_1|) \Rightarrow \ell'[i] = \ell'_1[i]) \land
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell_2|) \Rightarrow \ell'[^{|\ell'_1|} + i] = \ell_2[i])$$

- 1st point: $|\ell| = 1 + |\ell'_1 \circ \ell_2| = 1 + |\ell'_1| + |\ell_2| = |\ell_1| + |\ell_2|$
Correctness proof: Induction

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

- Induction hypothesis:
  
  \[ (|\ell'| = |\ell'_1| + |\ell_2|) \land \\
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell'_1|) \Rightarrow \ell'[i] = \ell'_1[i]) \land \\
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell'_1| + i] = \ell_2[i]) \]

- 1st point: \( |\ell| = 1 + |\ell'_1 \odot \ell_2| = 1 + |\ell'_1| + |\ell_2| = |\ell_1| + |\ell_2| \)

- 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 $\ell_1 = a \cdot \ell'_1$: $\ell = a \cdot (\ell'_1 \odot \ell_2)$. Let $\ell' = \ell'_1 \odot \ell_2$.

- **Induction hypothesis:**
  
  \[
  (|\ell'| = |\ell'_1| + |\ell_2|) \land \\
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell'_1|) \Rightarrow \ell'[i] = \ell'_1[i]) \land \\
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell'_1| + i] = \ell_2[i])
  \]

- **1st point:** $|\ell| = 1 + |\ell'_1 \odot \ell_2| = 1 + |\ell'_1| + |\ell_2| = |\ell_1| + |\ell_2|$

- **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:**
  1. **IH.2** $\Rightarrow \forall i. \ (1 \leq i < |\ell'_1| + 1) \Rightarrow \ell'[i - 1] = \ell'_1[i - 1]$
  2. **(2)** $\Rightarrow \forall i. \ (1 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i]$
  3. **(1)** $\Rightarrow \forall i. \ (0 \leq i < |\ell_1|) \Rightarrow \ell[i] = \ell_1[i]$

3rd point: left as an exercise.
Correctness proof: Induction

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

- Induction hypothesis:
  
  \[
  (|\ell'| = |\ell'_1| + |\ell_2|) \land \\
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell'_1|) \Rightarrow \ell'[i] = \ell'_1[i]) \land \\
  (\forall i \in \text{Nat.} \ (0 \leq i < |\ell_2|) \Rightarrow \ell'[|\ell'_1| + i] = \ell_2[i])
  \]

- 1st point: $|\ell| = 1 + |\ell'_1 \odot \ell_2| = 1 + |\ell'_1| + |\ell_2| = |\ell_1| + |\ell_2|$

- 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.
Reason about the structure of the input list?

\[ \text{Sort}([]) = \]
\[ \text{Sort}(a \cdot \ell) = \]

How to sort \( a \cdot \ell \) if we can sort \( \ell \)?
Sorting function: An Implementation

- Reason about the structure of the input list?

\[
\text{Sort}([\]) = [] \\
\text{Sort}(a \cdot \ell) = \text{Insert}(a, \text{Sort}(\ell))
\]

- We need to insert \(a\) in the sorted list corresponding to \(\ell\).
Reason about the structure of the input list?

\[ \text{Sort}(\emptyset) = \emptyset \]
\[ \text{Sort}(a \cdot \ell) = \text{Insert}(a, \text{Sort}(\ell)) \]

We need to insert \( a \) in the sorted list corresponding to \( \ell \).

What is the formal specification of \textit{Insert}?
Sorting function: An Implementation

- Reason about the structure of the input list?
  
  \[
  \text{Sort}([]) = []
  \]
  \[
  \text{Sort}(a \cdot \ell) = \text{Insert}(a, \text{Sort}(\ell))
  \]

- We need to insert \(a\) in the sorted list corresponding to \(\ell\).

- What is the formal specification of \(\text{Insert}\)?

- Type:
  
  \[
  \text{Insert} : \star \times \text{List}[^\star] \to \text{List}[^\star]
  \]

- Input-Output relation:
  
  \[
  \text{Spec} _{\text{Insert}}(a, \ell, \ell') = \\
  \text{Ordered}(\ell) \Rightarrow (\text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell)))
  \]
Reason about the structure of the output list?

\[ \text{Sort}([]) = \]
\[ \text{Sort}(a \cdot \ell) = \]

If the output is of the form \( e \cdot \ell' \), what is \( e \) ? and how to obtain \( \ell' \) ?
Reason about the structure of the output list?

\[
\text{Sort}([]) = []
\]

\[
\text{Sort}(a \cdot \ell) = \text{let } (m, \ell_m) = \text{Extract min}(a \cdot \ell) \text{ in } m \cdot \text{Sort}(\ell_m)
\]

Extract the minimal element \( m \) of \( \ell \), and sort the rest of the list \( \ell_m \).
Reason about the structure of the output list?

\[
\begin{align*}
    \text{Sort}([],) & = [] \\
    \text{Sort}(a \cdot \ell) & = \text{let } (m, \ell_m) = \text{Extract\_min}(a \cdot \ell) \text{ in } m \cdot \text{Sort}(\ell_m)
\end{align*}
\]

- Extract the minimal element \( m \) of \( \ell \), and sort the rest of the list \( \ell_m \).

- Specification of \text{Extract\_min}:
  - Type: \text{Extract\_min} : List[\ast] \rightarrow \ast \times List[\ast]
  - Input-Output relation:
    \[
    \text{Spec\_Extract\_min}(\ell_1, m, \ell_2) = \\
    \ell_1 \neq [] \Rightarrow \text{Is\_in}(m, \ell_1) \land \\
    \forall a \in \ast. \text{Is\_in}(a, \ell_1) \Rightarrow m \leq a \land \\
    \text{Ms}(\ell_1) = \text{Sg}(m) \mathbin{\uplus} \text{Ms}(\ell_2)
    \]
Sorting function: Yet Another Implementation

- Reason again about the structure of the output list?

\[ \text{Sort}([]) = \]
\[ \text{Sort}(a \cdot \ell) = \]

- Assume that when \( a \) is at its place in the output, it has \( \ell_{\text{left}} \) and \( \ell_{\text{right}} \) to its left and right, respectively. How to compute \( \ell_{\text{left}} \) and \( \ell_{\text{right}} \)?
Sorting function: Yet Another Implementation

- Reason again about the structure of the output list?

\[
Sort([]) = []
\]

\[
Sort(a \cdot \ell) = \text{let } (\ell_1, \ell_2) = \text{split } (a, \ell) \text{ in } Sort(\ell_1) \odot (a \cdot Sort(\ell_2))
\]

- Split \(\ell\) 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?

\[
\begin{align*}
Sort([]) &= [] \\
Sort(a \cdot \ell) &= \text{let } (\ell_1, \ell_2) = \text{split } (a, \ell) \text{ in } Sort(\ell_1) \oplus (a \cdot Sort(\ell_2))
\end{align*}
\]

- Split \( \ell \) into 2 lists containing the elements smaller and greater than \( a \).

- Specification of \( \text{Split} \):
  - Type: \( \text{Split} : \bigstar \times \text{List}[\bigstar] \rightarrow \text{List}[\bigstar] \times \text{List}[\bigstar] \)
  - Input-Output relation:

\[
\begin{align*}
\text{Spec}\_\text{Split}(a, \ell, \ell_1, \ell_2) &= \\
\text{Ms}(\ell) &= \text{Ms}(\ell_1) \uplus \text{Ms}(\ell_2) \land \\
\forall e \in \bigstar. \ ((\text{Is}\_\text{In}(e, \ell_1) \Rightarrow e \leq a) \land (\text{Is}\_\text{In}(e, \ell_2) \Rightarrow a < e))
\end{align*}
\]
Proving correctness of the Recursive Insertion Sort

Consider the implementation:

\[
\text{Ins\_Sort}([], []) = [] \\
\text{Ins\_Sort}(a \cdot \ell) = \text{Insert}(a, \text{Ins\_Sort}(\ell))
\]

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

\[
\forall a \in \star. \forall \ell, \ell' \in \text{List}[\star]. \text{Insert}(a, \ell) = \ell' \implies \text{Spec\_Insert}(a, \ell, \ell')
\]

where

\[
\text{Spec\_Insert}(a, \ell, \ell') =
\text{Ordered}(\ell) \implies (\text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell)))
\]

and prove that:

\[
\forall \ell, \ell' \in \text{List}[\star]. (\text{Ins\_Sort}(\ell) = \ell') \implies \text{Spec\_Sort}(\ell, \ell')
\]

where

\[
\text{Spec\_Sort}(\ell, \ell') =
\forall i, j, \in \text{Nat}. (0 \leq i < j < |\ell'| \implies \ell'[i] \leq \ell'[j]) \land \\
\text{Ms}(\ell) = \text{Ms}(\ell')
\]
Proof

Case $\ell = []$: Trivial.

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

- Let $\ell'_1 = \text{Ins\_Sort}(\ell_1)$.

- Induction hypothesis: $\text{Ordered}(\ell'_1) \land \text{Ms}(\ell_1) = \text{Ms}(\ell'_1)$.

- We assume Insert correct w.r.t. its specification:
  \[
  \text{Spec\_Insert}(a, \ell'_1, \ell') = \\
  \text{Ordered}(\ell'_1) \Rightarrow (\text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell'_1)))
  \]

- Since we have $\text{Ordered}(\ell'_1)$ by Ind. Hyp., then the following holds:
  \[
  \text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell'_1))
  \]

- We have $\text{Ms}(\ell) = \text{Sg}(a) \cup \text{Ms}(\ell_1) = \text{Sg}(a) \cup \text{Ms}(\ell'_1) = \text{Ms}(\ell')$.

- Then, we obtain $\text{Ordered}(\ell') \land \text{Ms}(\ell) = \text{Ms}(\ell')$. 

A. Bouajjani (Univ. Paris Diderot, UP7)
Recursive Insertion

- **Type:**
  \[
  \text{Insert} : \star \times \text{List}[\star] \to \text{List}[\star]
  \]

- **Input-Output specification:**
  \[
  \text{Spec}_{-}\text{Insert}(a, \ell, \ell') =
  \begin{align*}
  \text{Ordered}(\ell) \Rightarrow (\text{Ordered}(\ell') & \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell)))
  \end{align*}
  \]

- **Recursive implementation:**
  \[
  \begin{align*}
  \text{Insert}(a, []) & = a \cdot [] \\
  \text{Insert}(a, b \cdot \ell) & = \text{if } a \leq b \text{ then } a \cdot (b \cdot \ell) \\
  & \quad \text{else } b \cdot (\text{Insert}(a, \ell))
  \end{align*}
  \]
Recursive Insertion: Correctness proof

left as an exercise ...
Correctness of the Quick sort

- Consider the sorting function:

\[
\begin{align*}
qsort([]) &= [] \\
qsort(a \cdot l) &= \text{let } (l_1, l_2) = \text{split}(a, l) \text{ in} \\
&\quad \quad qsort(l_1)@a \cdot qsort(l_2)
\end{align*}
\]

- Prove that:

\[
\forall l, l'. (qsort(l) = l') \implies Spec \_Sort(l, l')
\]
Correctness of the Quick sort

- Consider the sorting function:

  \[ qsort(\[]) = [\] \]
  \[ qsort(a \cdot \ell) = \text{let } (\ell_1, \ell_2) = \text{split } (a, \ell) \text{ in} \]
  \[ qsort(\ell_1) \oplus (a \cdot qsort(\ell_2)) \]

- Prove that:

  \[ \forall \ell, \ell'. (qsort(\ell) = \ell') \implies \text{Spec\_Sort}(\ell, \ell') \]

- We need to assume that the two recursive calls are correct.
Correctness of the Quick sort

- Consider the sorting function:

\[
\begin{align*}
qsort(\[]) &= \[] \\
qsort(a \cdot \ell) &= \text{let } (\ell_1, \ell_2) = \text{split } (a, \ell) \text{ in} \\
& \quad qsort(\ell_1)@(a \cdot qsort(\ell_2))
\end{align*}
\]

- Prove that:

\[\forall \ell, \ell'. (qsort(\ell) = \ell') \implies Spec\_Sort(\ell, \ell')\]

- 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 \(\rho : D \rightarrow E\).
- Let \(\prec_\rho \subseteq D \times D\) be the relation such that:
  \[
  x \prec_\rho y \iff \rho(x) \prec \rho(y)
  \]
- Induction rule:
  \[
  \forall x \in D. ((\forall y. y \prec_\rho x \Rightarrow P(y)) \Rightarrow P(x))
  \]
  \[
  \forall x \in D. P(x)
  \]
Correctness of the Quick sort (cont.)

- Consider the WFS \((\mathbb{N}, <)\) and the function \(\rho : List[\star] \rightarrow \mathbb{N}\) such that
  \[
  \forall \ell \in List[\star]. \rho(\ell) = |\ell|
  \]

- 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.