Implementation vs. Specification

- Assume we want to define
  \[ f : \text{Dom} \rightarrow \text{CoDom} \]
- Consider an abstract specification
  \[ \text{Spec}_f(\text{In}, \text{Out}) \subseteq \text{Dom} \times \text{CoDom} \]
- 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:
  \[ \forall \text{In} \in \text{Dom}. \forall \text{Out} \in \text{CoDom}. (\text{Impl}_f(\text{In}) = \text{Out}) \implies \text{Spec}_f(\text{In}, \text{Out}) \]
  Or equivalently:
  \[ \forall \text{In} \in \text{Dom} \implies \text{Spec}_f(\text{In}, \text{Impl}_f(\text{In})) \]
- Correctness is always defined with respect to a given specification!

Example: The Append function

- Type:
  \[ \text{Append} : \text{List}[\ast] \times \text{List}[\ast] \rightarrow \text{List}[\ast] \]
- Specification:
  \[ \text{Spec}_{\text{Append}}(\ell_1, \ell_2, \ell) = \]
  \[ \vert \ell \vert = \vert \ell_1 \vert + \vert \ell_2 \vert \land \]
  \[ \forall i \in \text{Nat}. (0 \leq i < \vert \ell_1 \vert) \Rightarrow \ell[i] = \ell_1[i] \land \]
  \[ \forall i \in \text{Nat}. (0 \leq i < \vert \ell_2 \vert) \Rightarrow \ell[\vert \ell_1 \vert + i] = \ell_2[i] \]
- Implementation:
  \[ \ell_1 @ \ell_2 = \ell \]
  \[ (a \cdot \ell_1)@\ell_2 = a \cdot (\ell_1@\ell_2) \]
- 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 $\ell_1 = a \cdot \ell'_1$; $\ell = a \cdot (\ell'_1 \cdot \ell_2)$. Let $\ell' = \ell'_1 \cdot \ell_2$.

- Induction hypothesis:
  \begin{align*}
  (|\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'[i+1] = \ell_2[i])
  \end{align*}

- 1st point: $|\ell| = 1 + |\ell'_1 \cdot \ell_2| = 1 + |\ell'_1| + |\ell_2| = |\ell_1| + |\ell_2|$.
- We have (by definition of the At operator):
  \begin{align*}
  & 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_2| \Rightarrow \ell_2[i] = \ell'_2[i]
  \end{align*}

- 2nd point:
  \begin{align*}
  & \text{IH.2} \Rightarrow \forall i. (1 \leq i < |\ell'_1| + 1) \Rightarrow \ell'[i] = \ell'_1[i] = \ell_2[i] \\
  & \text{(2)} \Rightarrow \forall i. (1 \leq i < |\ell_1|) \Rightarrow \ell'[i] = \ell_2[i] \\
  & \text{(1)} \Rightarrow \forall i. (0 \leq i < |\ell_1|) \Rightarrow \ell'[i] = \ell_2[i]
  \end{align*}

- 3rd point: left as an exercise.

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:
    \[ \text{Insert} : \star \times \text{List}[\star] \rightarrow \text{List}[\star] \]
  - Input-Output relation:
    \[ \text{Spec}_{\text{Insert}}(a, \ell, \ell') = \text{Ordered}(\ell) \land (\text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \sqcup \text{Ms}(\ell))) \]
Sorting function: Another Implementation

- Reason about the structure of the output list?
  
  \[
  \text{Sort}([]) = [] \\
  \text{Sort}(a \cdot \ell) = \text{let } (m, \ell_m) = \text{Extract}_{\text{min}}(a, \ell) \text{ in } m \cdot \text{Sort}(\ell_m) \\
  \]

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

---

Sorting function: Yet Another Implementation

- Reason again about the structure of the output list?
  
  \[
  \text{Sort}([]) = [] \\
  \text{Sort}(a \cdot \ell) = \text{let } (\ell_1, \ell_2) = \text{split}(a, \ell) \text{ in } \text{Sort}(\ell_1)@ (a \cdot \text{Sort}(\ell_2)) \\
  \]

- 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 \( \ell_{\text{right}} \)?

---

Specification of \( \text{Extract}_{\text{min}} \):

\[
\text{Type: } \text{Extract}_{\text{min}} : \star \times \text{List}[\star] \to \star \times \text{List}[\star] \\
\text{Input-Output relation: } \\
\text{Spec}_{\text{Extract}_{\text{min}}}(x, \ell_1, m, \ell_2) = \\
\text{Is}_{\text{in}}(m, x \cdot \ell_1) \land \\
\forall a \in \star. \text{Is}_{\text{in}}(a, x \cdot \ell_1) \Rightarrow m \leq a \land \\
\text{Ms}(x \cdot \ell_1) = \text{Sg}(m) \cup \text{Ms}(\ell_2) \\
\]

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

- Split \( \ell \) into 2 lists containing the elements smaller and greater than \( a \).
Reason again about the structure of the output list?

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

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

Specification of \text{Split}:

- Type: \( \text{Split} : \star \times \text{List}[\star] \rightarrow \text{List}[\star] \times \text{List}[\star] \)
- Input-Output relation:
  \[
  \text{Spec}_\text{Split}(a, \ell, \ell_1, \ell_2) =
  \begin{align*}
  &\text{Ms}(\ell) = \text{Ms}(\ell_1) \cup \text{Ms}(\ell_2) \land \\
  &\forall e \in \star. ((\text{Is}_\text{In}(e, \ell_1) \Rightarrow e \leq a) \land (\text{Is}_\text{In}(e, \ell_2) \Rightarrow a < e))
  \end{align*}
  \]

Then, we obtain \( \text{Ms}(\ell) = \text{Ms}(\ell_1) \cup \text{Ms}(\ell_2) \land \forall e \in \star. ((\text{Is}_\text{In}(e, \ell_1) \Rightarrow e \leq a) \land (\text{Is}_\text{In}(e, \ell_2) \Rightarrow a < e)) \).

Proof

Case \( \ell = [] \): Trivial.

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

- Let \( \ell'_1 = \text{Ins}_\text{Sort}(\ell_1) \).
- Induction hypothesis: \( \text{Ordered}(\ell'_1) \land \text{Ms}(\ell_1) = \text{Ms}(\ell'_1) \).
- We assume \text{Insert} correct w.r.t. its specification:
  \[
  \text{Spec}_\text{Insert}(a, \ell_1, \ell') =
  \begin{align*}
  &\text{Ordered}(\ell'_1) \Rightarrow (\text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell'_1)))
  \end{align*}
  \]
- 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 \( \text{Ordered}(\ell') \land \text{Ms}(\ell) = \text{Ms}(\ell') \).

Proving correctness of the Recursive Insertion Sort

- Consider the implementation:
  \[
  \begin{align*}
  &\text{Ins}_\text{Sort}([\quad]) = [] \\
  &\text{Ins}_\text{Sort}(a \cdot \ell) = \text{Insert}(a, \text{Ins}_\text{Sort}(\ell))
  \end{align*}
  \]
- Assume that \text{Insert} is correct w.r.t. its specification:
  \[
  \forall a \in \star. \forall \ell, \ell' \in \text{List}[\star]. \text{Insert}(a, \ell, \ell') \Rightarrow \text{Spec}_\text{Insert}(a, \ell, \ell')
  \]
  where
  \[
  \text{Spec}_\text{Insert}(a, \ell, \ell') =
  \begin{align*}
  &\text{Ordered}(\ell) \Rightarrow (\text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell)))
  \end{align*}
  \]
- and prove that:
  \[
  \forall \ell, \ell' \in \text{List}[\star]. (\text{Ins}_\text{Sort}(\ell) = \ell') \Rightarrow \text{Spec}_\text{Sort}(\ell, \ell')
  \]
  where
  \[
  \text{Spec}_\text{Sort}(\ell, \ell') =
  \begin{align*}
  &\forall i, j \in \text{Nat}. (0 \leq i < j < |\ell'| \Rightarrow \ell'[i] \leq \ell'[j]) \land \\
  &\text{Ms}(\ell) = \text{Ms}(\ell')
  \end{align*}
  \]

Recursive Insertion

- Type:
  \[
  \text{Insert} : \star \times \text{List}[\star] \rightarrow \text{List}[\star]
  \]
- Input-Output specification:
  \[
  \begin{align*}
  &\text{Spec}_\text{Insert}(a, \ell, \ell') =
  \begin{align*}
  &\text{Ordered}(\ell) \Rightarrow (\text{Ordered}(\ell') \land (\text{Ms}(\ell') = \text{Sg}(a) \cup \text{Ms}(\ell)))
  \end{align*}
  \]
- Recursive implementation:
  \[
  \begin{align*}
  &\text{Ins}_\text{Sort}([\quad]) = a \cdot [] \\
  &\text{Ins}_\text{Sort}(a, b \cdot \ell) = \begin{cases} a \cdot (\ell) & \text{if } a \leq b \\
  &\text{else } b \cdot (\text{Insert}(a, \ell))
  \end{cases}
  \end{align*}
  \]
Recursive Insertion: Correctness proof

left as an exercise ... 

Correctness of the Quick sort

Consider the sorting function:

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

Prove that:

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

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

\[
\forall F \subseteq E. \ F \neq \emptyset \Rightarrow (\exists e \in F. \ \forall e' \in F. \ e' \neq e)
\]

Well founded relations: Examples

- \((\mathbb{N}, <)\) is a WFS.
- \((\mathbb{Z}, <)\) is not a WFS.
- \((\mathbb{R}_0, <)\) is not a WFS.
Noetherian Induction

Let \((E, \preceq)\) be a WFS, and let \(\rho : D \to E\).

Let \(\preceq_\rho \subseteq D \times D\) be the relation such that:

\[ x \preceq_\rho y \iff \rho(x) \preceq \rho(y) \]

Induction rule:

\[ \forall x \in D. \left( (\forall y. y \preceq_\rho x \Rightarrow P(y)) \Rightarrow P(x) \right) \]

\[ \forall x \in D. P(x) \]

Correctness of the Quick sort (cont.)

Consider the WFS \((\mathbb{N}, \prec)\) and the function \(\rho : \text{List}\left[\ast\right] \to \mathbb{N}\) such that

\[ \forall \ell \in \text{List}\left[\ast\right]. \rho(\ell) = |\ell| \]

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 framework 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.