Specifying functional programs

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

DI/UM (2014)

First program

What does the following (recursive) program do?

$$
\begin{array}{l}
x \div y = \\
\quad \textbf{if } x < y \textbf{ then } 0 \\
\quad \textbf{else } 1 + (x - y) \div y
\end{array}
$$

It *seems* to be counting the number of times y fits in x, e.g.

$$
\begin{array}{rcl}
7 \div 2 & = & 1 + 5 \div 2 \\
& = & 1 + (1 + 3 \div 2) \\
& = & 1 + (1 + (1 + 1 \div 2)) \\
& = & 3 + 0 \\
& = & 3
\end{array}
$$

Is this the *same* as **whole division**?

Program testing

What about this other version of the *same* algorithm?

$$
\begin{array}{l}
x \div y = \\
\quad \textbf{if } x \equiv y \textbf{ then } 1 \\
\quad \textbf{else } 1 + (x - y) \div y
\end{array}
$$

Is it **correct**? $7 \div 7 = 1$, ok. What about $6 \div 7$?

Uups — not a correct implementation **whole division***!*

Questions:

- What is the **specification** of whole division?
- Why is programming (so...) *tricky*?

Program testing

What about this other version of the *same* algorithm?

$$
\begin{array}{l}
x \div y = \\
\quad \textbf{if } x \equiv y \textbf{ then } 1 \\
\quad \textbf{else } 1 + (x - y) \div y
\end{array}
$$

Is it **correct**? $7 \div 7 = 1$, ok. What about $6 \div 7$?

Uups — not a correct implementation **whole division***!*

Questions:

- What is the **specification** of whole division?
- Why is programming (so...) *tricky*?

Problems = Easy + Hard

Requirements:

$x \div y$ should yield the largest number which, multiplied by x, is at most y.

Superlatives in problem requirements, e.g.

"... the **largest** *such number"*
"... the **longest** *such list"*
"... the **best** *approximation"*

Two layers in **specifications**:

- **easy** — **broad** class of solutions
- **hard** — **optimal** solution required.

Back to the primary school desk

The **whole division** algorithm

$$\begin{array}{r|l} 7 & 2 \\ \hline 1 & 3 \end{array} \qquad 2 \times 3 + 1 = 7 \quad \text{, ie.} \qquad 3 = 7 \div 2$$

However

$$\begin{array}{r|l} 7 & 2 \\ \hline 3 & 2 \end{array} \qquad 2 \times 2 + 3 = 7 \quad \wedge \quad 2 \neq 7 \div 2$$

$$\begin{array}{r|l} 7 & 2 \\ \hline 5 & 1 \end{array} \qquad 2 \times 1 + 5 = 7 \quad \wedge \quad 1 \neq 7 \div 2$$

That is: for some r,

$$\begin{array}{r|l} n & d \\ \hline r & q \end{array} \qquad q = n \div d \;\equiv\; d \times q + r = n$$

provided q is the **largest** such q (r **smallest**)

Back to the primary school desk

The **whole division** algorithm

$$\begin{array}{r|l} 7 & 2 \\ \hline 1 & 3 \end{array} \qquad 2 \times 3 + 1 = 7 \quad \text{, ie.} \qquad 3 = 7 \div 2$$

However

$$\begin{array}{r|l} 7 & 2 \\ \hline 3 & 2 \end{array} \qquad 2 \times 2 + 3 = 7 \quad \wedge \quad 2 \neq 7 \div 2$$

$$\begin{array}{r|l} 7 & 2 \\ \hline 5 & 1 \end{array} \qquad 2 \times 1 + 5 = 7 \quad \wedge \quad 1 \neq 7 \div 2$$

That is: for some r,

$$\begin{array}{r|l} n & d \\ \hline r & q \end{array} \qquad q = n \div d \;\equiv\; d \times q + r = n$$

provided q is the **largest** such q (r **smallest**)

Example — specifying $x \div y$

First version (literal):

$$x \div y = \langle \bigvee z :: z \times y \leqslant x \rangle \tag{35}$$

Second version (involved):

$$z = x \div y \equiv \langle \exists r : 0 \leqslant r < y : x = z \times y + r \rangle \tag{36}$$

Third version (clever!):

$$z \times y \leqslant x \equiv z \leqslant x \div y \qquad (y > 0) \tag{37}$$

Why is (37) the **best** of all three specifications?

Why (37) is better than (35,36)

Equivalence (37),

$$z \times y \leqslant x \;\equiv\; z \leqslant x \div y \qquad (y > 0)$$

captures the **requirements** in an elegant way:

- It is a solution: $x \div y$ multiplied by y approximates x

$$(x \div y) \times y \leqslant x$$

  — let $z := x \div y$ in (37) and simplify.

- Is it the **best** solution? Yes, because it provides the **largest** such number:

$$z \times y \leqslant x \;\Rightarrow\; z \leqslant x \div y \qquad (y > 0)$$

  — the $\Rightarrow$ part of the $\equiv$ of (37).

Why (37) is better than (35,36)

Equivalence (37),

$$z \times y \leqslant x \;\equiv\; z \leqslant x \div y \qquad (y > 0)$$

captures the **requirements** in an elegant way:

- It is a solution: $x \div y$ multiplied by y approximates x

$$(x \div y) \times y \leqslant x$$

  — let $z := x \div y$ in (37) and simplify.

- Is it the **best** solution? Yes, because it provides the **largest** such number:

$$z \times y \leqslant x \;\Rightarrow\; z \leqslant x \div y \qquad (y > 0)$$

  — the $\Rightarrow$ part of the $\equiv$ of (37).

Reasoning

Equivalence (37)

$$z \times y \leqslant x \;\equiv\; z \leqslant x \div y \qquad\qquad (y > 0)$$

is not only **simple** to write but **effective** to reason about.

EXAMPLE: we want to derive the property

$$(n \div m) \div d = n \div (d \times m)$$

from the specification. What about

- using (35)? $\bigvee$ difficult!
- using (36)? $\exists$ difficult!
- using (37)? **easy** — see the next slide.

Reasoning

Equivalence (37)

$$z \times y \leqslant x \;\equiv\; z \leqslant x \div y \qquad\qquad (y > 0)$$

is not only **simple** to write but **effective** to reason about.

EXAMPLE: we want to derive the property

$$(n \div m) \div d = n \div (d \times m)$$

from the specification. What about

- using (35)? $\bigvee$ difficult!
- using (36)? $\exists$ difficult!
- using (37)? **easy** — see the next slide.

Proving $(n \div m) \div d = n \div (d \times m)$

$$
\begin{array}{ll}
 & z \leqslant (n \div m) \div d \\
\equiv & \quad \{ \ (37) \ \} \\
 & z \times d \leqslant n \div m \\
\equiv & \quad \{ \ (37) \ \} \\
 & (z \times d) \times m \leqslant n \\
\equiv & \quad \{ \ \times \text{ is associative} \ \} \\
 & z \times (d \times m) \leqslant n \\
\equiv & \quad \{ \ (37) \ \} \\
 & z \leqslant n \div (d \times m) \\
:: & \quad \{ \text{ indirection — see (38) below } \} \\
 & (n \div m) \div d = n \div (d \times m)
\end{array}
$$

Indirect equality

IMPORTANT: Note the use of the (generic) **indirect equality** rule

$$\langle \forall\ z\ ::\ z \leqslant x \equiv z \leqslant y \rangle\ \equiv\ (x = y) \tag{38}$$

valid for **any** partial order $\leqslant$.

Exercise 16: Derive from (37) the two **cancellation** laws

$$\begin{array}{rcl} q & \leqslant & (q \times d) \div d \\ (n \div d) \times d & \leqslant & n \end{array}$$

and **reflexion** law:

$$n \div d \geqslant 1\ \equiv\ d \leqslant n \tag{39}$$

□

Specifying functions on lists

Consider the following **requirements** about the take function in Haskell:

---

take *n xs* *should yield the* **longest** *possible* **prefix** *of xs not exceeding n in* **length**.

Warming up (with examples):

take $2\ [10, 20, 30] = [10, 20]$
take $20\ [10, 20, 30] = [10, 20, 30]$
etc

How do we write a formal **specification** for these requirements?

Specifying functions on lists

We observe that:

- take $n\ xs$ is a **prefix** of xs — specify this as e.g.

$$\text{take}\ n\ xs \preceq xs$$

  where $\preceq$ denotes the ordering (leave this for later).
- the length of take $n\ xs$ cannot exceed n — easy to specify:

$$\text{length}\ (\text{take}\ n\ xs) \leqslant n$$

Altogether:

$$\text{length}\ (\text{take}\ n\ xs) \leqslant n\ \wedge\ \text{take}\ n\ xs \preceq xs \tag{40}$$

But this is not **enough** — (silly) implementation take $n\ xs = [\,]$ meets (40)!

Superlatives...

We have not yet thought of how to formally specify the **superlative** in

---

*...*take *n xs* *should yield the* **longest possible** *prefix...*

This is the **hard** part but there is a standard method to follow:

- think of an arbitrary list *ys* also satisfying (40)

$$\text{length } ys \leqslant n \ \wedge \ ys \preceq xs$$

- Then (following the **requirements**) *ys* should be a prefix of take *n xs*:

$$\text{length } ys \leqslant n \wedge ys \preceq xs \Rightarrow ys \preceq \text{take } n \ xs \tag{41}$$

Final touch

So we need the two clauses,

the **easy** *one (40)*

and

the **hard** *one (41).*

Interestingly, (40) can be derived from the converse of (41)

$$\text{length } ys \leqslant n \wedge ys \preceq xs \Leftarrow ys \preceq \text{take } n\ xs$$

by letting $ys := \text{take } n\ xs$ and simplifying.

So a single line is enough to **formally specify** *take*:

$$length\ ys \leqslant n \ \wedge \ ys \preceq xs \ \equiv \ ys \preceq take\ n\ xs \qquad (42)$$

Exercises

Exercise 17: Follow the **specification method** of the previous example to formally specify the requirement

---

The function takeWhile p xs should yield the longest prefix of xs such that all x in such a prefix satisfy predicate p.

---

and

The function filter p xs should yield the longest sublist of xs such that all x in such a sublist satisfy predicate p.

NB: assume the existence of the sublist ordering $ys \sqsubseteq xs$ such that e.g. `"ab"` $\sqsubseteq$ `"acb"` holds but `"ab"` $\sqsubseteq$ `"bca"` **does not** hold.

□

Exercises

Exercise 18: (a) Check that specification

$$X \subseteq A \wedge X \subseteq B \;\equiv\; X \subseteq A \cap B \tag{43}$$

meets the (abstract) requirements:

> *The intersection $A \cap B$ is the largest common subset of A and B.*

(b) Then write a similar specification for the union $A \cup B$ of two sets.

(c) Finally guess the (textual) requirements which lead to the following specification of the least (minimum) of two numbers:

$$x \leqslant m \wedge x \leqslant n \;\equiv\; x \leqslant min\ m\ n \tag{44}$$

□

Reasoning about specifications

One of the advantages of **formal specification** is that one may **quest** the specification (aka **model**) to derive useful properties of the design **before the implementation phase**.

The principle of **indirect equality** (38) is the main tool in this process, as we have already seen with $x \div y$.

Exercise 19: Solely relying on specification (42) use indirect equality to prove that

$$\text{take } (\text{length } xs)\ xs = xs \tag{45}$$
$$\text{take } 0\ xs = [\,] \tag{46}$$
$$\text{take } n\ [\,] = [\,] \tag{47}$$

hold.

□

Reasoning about specifications

A more interesting calculation is that of simplifying

$$\text{take } m \text{ (take } n \text{ } xs\text{)}$$

to an expression involving a single take.

As before, we head for an **indirect equality** derivation by starting with term $ys \preceq \text{take } m \text{ (take } n \text{ } xs\text{)}$ (complete the claims):

$$\begin{array}{ll} & ys \preceq \text{take } m \text{ (take } n \text{ } xs\text{)} \\ \equiv & \quad \{ \ \ldots \ \} \\ & ys \preceq (\text{take } n \text{ } xs) \wedge \text{length } ys \leqslant m \\ \equiv & \quad \{ \ \ldots \ \} \\ & (ys \preceq xs \wedge \text{length } ys \leqslant n) \wedge \text{length } ys \leqslant m \end{array}$$

Reasoning about specifications

$$
\begin{array}{ll}
 & ys \preceq xs \wedge (\text{length } ys \leqslant n \wedge \text{length } ys \leqslant m) \\
\equiv & \quad \{ \ \ldots \ \} \\
 & ys \preceq xs \wedge (\text{length } ys \leqslant min\ n\ m) \\
\equiv & \quad \{ \ \ldots \ \} \\
 & ys \preceq \text{take } (min\ n\ m)\ xs
\end{array}
$$

$\square$

Thus:

$$\text{take } m\ (\text{take } n\ xs) \quad = \quad \text{take } (min\ n\ m)\ xs$$

is a property of take.

Exercises

Exercise 20: Derive from (44) the following properties of the *min* function (on $\mathbb{N}_0$)

$$min\ m\ (min\ n\ p) = min\ (min\ m\ n)\ p \quad (48)$$

$$min\ m\ m = m \quad (49)$$

$$min\ 0\ m = 0 \quad (50)$$

and infer (by formal analogy) the corresponding properties of intersection (43).

□

Deriving programs from specifications

Back to the starting example, can we derive **program**

$$
\begin{array}{l}
x \div y = \\
\quad \textbf{if } x < y \textbf{ then } 0 \\
\quad \textbf{else } 1 + (x - y) \div y
\end{array}
$$

from the **specification**

$$z \times y \leqslant x \;\equiv\; z \leqslant x \div y$$

(for $y > 0$) thereby ensuring **correctness**?

Deriving programs from specifications

A simple example first: deriving the algorithm of *min* from its specification

$$x \leqslant m \wedge x \leqslant n \quad \equiv \quad x \leqslant min\ m\ n$$

Note that either $m \leqslant n$ or $n \leqslant m$ holds. Case $m \leqslant n$:

$$\begin{array}{ll} & x \leqslant min\ m\ n \\ \equiv & \quad \{\ \text{specification}\ \} \\ & x \leqslant m \wedge x \leqslant n \\ \equiv & \quad \{\ \text{because } m \leqslant n \text{ (transitivity)}\ \} \\ & x \leqslant m \\ :: & \quad \{\ \text{indirect equality}\ \} \\ & min\ m\ n = m \end{array}$$

$\square$

Deriving programs from specifications

Case $n \leqslant m$ is similar:

$$
\begin{array}{ll}
 & x \leqslant min\ m\ n \\
\equiv & \quad \{\ \text{specification}\ \} \\
 & x \leqslant m \wedge x \leqslant n \\
\equiv & \quad \{\ \text{because } n \leqslant m \text{ (transitivity)}\ \} \\
 & x \leqslant n \\
:: & \quad \{\ \text{indirect equality}\ \} \\
 & min\ m\ n = n \\
\Box &
\end{array}
$$

Thus we obtain the **program**

$$min\ m\ n = \textbf{if}\ n \leqslant m\ \textbf{then}\ n\ \textbf{else}\ m$$

Integer division now

Specification (in $\mathbb{N}_0$, for $y > 0$):

$$z \times y \leqslant x \;\equiv\; z \leqslant x \div y$$

Derivation:

$$\begin{array}{ll} & z \leqslant x \div y \\ \equiv & \quad \{ \text{ specification } \} \\ & z \times y \leqslant x \\ \equiv & \quad \{ \text{ assume } x \geqslant y \text{ and subtract } \} \\ & z \times y - y \leqslant x - y \\ \equiv & \quad \{ \text{ arithmetics } \} \\ & (z-1) \times y \leqslant x - y \end{array}$$

($\longrightarrow$ next slide)

Deriving programs from specifications

(← previous slide)

$$
\begin{array}{ll}
\equiv & \{ \text{ specification } \} \\
& z - 1 \leqslant (x - y) \div y \\
\equiv & \{ \text{ trivial } \} \\
& z \leqslant 1 + (x - y) \div y
\end{array}
$$

Altogether (for $x \geqslant y$):

$$z \leqslant x \div y \quad \equiv \quad z \leqslant 1 + (x - y) \div y$$

By indirect equality:

$$x \div y = \textbf{if } x \geqslant y \textbf{ then } 1 + (x - y) \div y \textbf{ else} \ldots$$

Deriving programs from specifications

Case $x < y$ is simpler, as expected:

$$
\begin{array}{ll}
 & z \leqslant x \div y \\
\equiv & \quad \{ \text{ specification } \} \\
 & z \times y \leqslant x \\
\equiv & \quad \{ \text{ since } x < y \} \\
 & z \times y \leqslant x \wedge z \times y < y \\
\equiv & \quad \{ \ z = 0 \text{ iff } z \times y < y \text{ in } \mathbb{N}_0 \ \} \\
 & z \leqslant 0 \\
\Box &
\end{array}
$$

By indirect equality (altogether):

$$x \div y = \textbf{if } x < y \textbf{ then } 0 \textbf{ else } 1 + (x - y) \div y$$

Summary

One-line specifications of functions

Superlatives in requirements are the **hard** bit

Derivation of (functional) programs from specifications

Formal methods for **functional** programming easier than for **imperative** programming.

Exercise

Exercise 21: Derive fact

$$x \div x = 1 \tag{51}$$

from (37). So the base case of program

$$\begin{array}{l} x \div y = \\ \quad \textbf{if } x \equiv y \textbf{ then } 1 \\ \quad \textbf{else } 1 + (x - y) \div y \end{array}$$

is correct. What is wrong about the algorithm, then?

□