Available at the COMP2111 website.

COMP2111 Notes Week 11
Case Study in Forward Simulation

Kai Engelhardt

May 12, 2016

Revision: 1.3

1 Introduction

This note gives examples of how to use note 10 to prove refinement between syntactic data types.

Consult note 10 for most of the relevant definitions of syntactic data type and forward simulation predicates.

Before we discuss a simple example of a data refinement, let us specialise the forward simulation conditions for the common case where r is given as a (total) function $r(a, c, x) = (f(c) = a)$. This satisfies condition (5) in note 10 automatically and allows us to simplify the remaining conditions to the following versions.

$$\mathsf{init}^{\mathsf{C}} \Rightarrow \mathsf{init}^{\mathsf{A}}[{}^{f(c)}/_{a}] \tag{2_f}$$

$$\mathsf{pre}^{\mathsf{A}}_j[{}^{f(c)}/_{a}] \Rightarrow \mathsf{pre}^{\mathsf{C}}_j \quad \text{, for } j \in J \tag{3_f}$$

$$\mathsf{pre}^{\mathsf{A}}_j[{}^{f(c_0),x_0}/_{a,x}] \wedge \mathsf{post}^{\mathsf{C}}_j \Rightarrow \mathsf{post}^{\mathsf{A}}_j[{}^{f(c_0),f(c)}/_{a_0,a}] \quad \text{, for } j \in J \tag{4_f}$$

2 A Stack Data Type

Consider a syntactic stack data type S inspired by the tute 8 solutions. Its representation variable is a stack-valued s. The data type comprises of an initialisation predicate $\mathsf{init}^{\mathsf{S}} = (s = \langle\,\rangle)$ and the following operations.

```
func $isempty^{\mathsf{S}}() : \mathbb{B}\,\cdot$
    var $b \cdot b : [\text{TRUE}, b = (s = \langle\,\rangle)]$; return $b$
proc $pop^{\mathsf{S}}()\,\cdot$
    $s : [s \neq \langle\,\rangle, \exists y\,(s_0 = \langle y, s\rangle)]$
func $top^{\mathsf{S}}() : \mathbb{Z}\,\cdot$
    var $y \cdot y : [s \neq \langle\,\rangle, \exists t\,(s = \langle y, t\rangle)]$; return $y$
proc $push^{\mathsf{S}}($value $x)\,\cdot$
    $s : [\text{TRUE}, s = \langle x, s_0\rangle]$
```

3 A Refinement

We're refining this to a data type A where we replace s with an infinite array a and a counter i that holds the index of the next free cell in the array. This suggests we should confine the state space to mappings from i to $\mathbb{N}$ and from a to functions from $\mathbb{N} \to \mathbb{Z}$. (This is our *data invariant* at this level.) The correspondence between the two state spaces is captured by the inductively defined predicate

$$r = (i = 0 \Leftrightarrow s = \langle\,\rangle) \wedge (i > 0 \Leftrightarrow \exists t\,\big(s = \langle a[i-1], t\rangle \wedge r[{}^{t,i-1}/_{s,i}]\big))$$

which we can translate into a recursive function from concrete to abstract values:

$$f(a,i) = \begin{cases} \langle\,\rangle & \text{if } i = 0 \\ \langle a[i-1], f(a, i-1)\rangle & \text{otherwise} \end{cases}$$

With that in mind we propose the initialisation predicate $\mathsf{init}^{\mathsf{A}} = (i = 0)$ and operations given as follows.

```
func $isempty^{\mathsf{A}}() : \mathbb{B}$ ·
    var $b \cdot b : [\text{TRUE}, b = (i = 0)]$; return $b$
proc $pop^{\mathsf{A}}()$ ·
    $i : [i > 0, i = i_0 - 1]$
func $top^{\mathsf{A}}() : \mathbb{Z}$ ·
    var $y \cdot y : [i > 0, y = a[i-1]]$; return $y$
proc $push^{\mathsf{A}}$(value $x$) ·
    $a, i : [\text{TRUE}, a[i_0] = x \wedge i = i_0 + 1 \wedge \forall k < i_0\, (a[k] = a_0[k])]$
```

or, alternatively,

$$a, i : [\text{TRUE}, a = (a_0 : i_0 \mapsto x) \wedge i = i_0 + 1]$$

That this indeed is a refinement requires checking the relevant proof obligations. We begin with the initialisations. Condition (2_f) becomes

$$\begin{array}{ll} & \mathsf{init}^{\mathsf{A}} \Rightarrow \mathsf{init}^{\mathsf{S}}[{}^{f(a,i)}/_{s}] \\ \Leftrightarrow & \quad \langle \text{def. of } \mathsf{init}^{\mathsf{A}} \text{ and } \mathsf{init}^{\mathsf{S}} \rangle \\ & i = 0 \Rightarrow f(a,i) = \langle\,\rangle \end{array}$$

which is immediate from the definition of f. Next we deal with the conditions concerning the operations. But what are these operations exactly? The syntactic proof technique presumes operation specifications in a single form, which means that we first have to rewrite the ones above into that form. Usually this just means that we have to add freeze predicates to the postconditions to allow the frame to be the same for all operations.

Condition (3_f) is only interesting when the concrete precondition is non-trivial. This is the case only for *pop* and *push*. In both cases condition (3_f) becomes

$$f(a,i) \neq \langle\,\rangle \Rightarrow i > 0$$

which is again immediate from the definition of f. Finally, condition (4_f) needs to be checked for each operation.

Remember that before we embark on the proofs, we need to beef up the specs to have all variables at that level in the frame. This means just adding freeze predicates to the postconditions for the variables we add to the frame. (Otherwise the POs aren't sound!)

For *isempty* we prove

$$\begin{aligned}
&\mathsf{pre}_{isempty^{\mathsf{S}}}[^{f(a_0,i_0)}/_{s}] \wedge \mathsf{post}_{isempty^{\mathsf{A}}}\\
&\Leftrightarrow \qquad \langle \text{def. of } isempty^{\mathsf{A}} \text{ and } isempty^{\mathsf{S}} \rangle\\
&\text{TRUE}[^{f(a_0,i_0)}/_{s}] \wedge b = (i = 0) \wedge a, i = a_0, i_0\\
&\Rightarrow \qquad \langle \text{def. of } f \rangle\\
&b = (f(a,i) = \langle\,\rangle) \wedge f(a,i) = f(a_0,i_0)\\
&\Leftrightarrow \qquad \langle \text{def. of } isempty^{\mathsf{A}} \text{ and } isempty^{\mathsf{S}} \rangle\\
&\mathsf{post}_{isempty^{\mathsf{S}}}[^{f(a_0,i_0),f(a,i)}/_{s_0,s}]
\end{aligned}$$

For *pop* we prove

$$\begin{aligned}
&\mathsf{pre}_{pop^{\mathsf{S}}}[^{f(a_0,i_0)}/_{s}] \wedge \mathsf{post}_{pop^{\mathsf{A}}}\\
&\Leftrightarrow \qquad \langle \text{def. of } pop^{\mathsf{A}} \text{ and } pop^{\mathsf{S}} \rangle\\
&f(a_0,i_0) \neq \langle\,\rangle \wedge a, i = a_0, i_0 - 1\\
&\Rightarrow \qquad \langle \text{def. of } f \rangle\\
&f(a_0,i_0) = \langle a[i_0 - 1], f(a,i) \rangle\\
&\Rightarrow \qquad \langle \text{logic} \rangle\\
&\exists y\, (f(a_0,i_0) = \langle y, f(a,i) \rangle)\\
&\Leftrightarrow \qquad \langle \text{def. of } pop^{\mathsf{A}} \text{ and } pop^{\mathsf{S}} \rangle\\
&\mathsf{post}_{pop^{\mathsf{S}}}[^{f(a_0,i_0),f(a,i)}/_{s_0,s}]
\end{aligned}$$

For *top* we do almost the same.

$$\begin{aligned}
&\mathsf{pre}_{top^{\mathsf{S}}}[^{f(a_0,i_0)}/_{s}] \wedge \mathsf{post}_{top^{\mathsf{A}}}\\
&\Leftrightarrow \qquad \langle \text{def. of } top^{\mathsf{A}} \text{ and } top^{\mathsf{S}} \rangle\\
&f(a_0,i_0) \neq \langle\,\rangle \wedge y = a[i-1] \wedge a, i = a_0, i_0\\
&\Rightarrow \qquad \langle \text{def. of } f \rangle\\
&\exists t\, (f(a,i) = \langle y, t \rangle) \wedge f(a,i) = f(a_0,i_0)\\
&\Leftrightarrow \qquad \langle \text{def. of } top^{\mathsf{A}} \text{ and } top^{\mathsf{S}} \rangle\\
&\mathsf{post}_{top^{\mathsf{S}}}[^{f(a_0,i_0),f(a,i)}/_{s_0,s}]
\end{aligned}$$

An finally for *push*:

$$\begin{aligned}
&\mathsf{pre}_{push^{\mathsf{S}}}[^{f(a_0,i_0)}/_{s}] \wedge \mathsf{post}_{push^{\mathsf{A}}}\\
&\Leftrightarrow \qquad \langle \text{def. of } push^{\mathsf{A}} \text{ and } push^{\mathsf{S}} \rangle\\
&\text{TRUE}[^{f(a_0,i_0)}/_{s}] \wedge a = (a_0 : i_0 \mapsto x) \wedge i = i_0 + 1\\
&\Rightarrow \qquad \langle \text{def. of } f \rangle
\end{aligned}$$

$$
\begin{aligned}
& f(a,i) = \langle x, f(a_0, i_0)\rangle \\
\Leftrightarrow \quad & \langle \text{def. of } push^{\mathsf{A}} \text{ and } push^{\mathsf{S}} \rangle \\
& \mathsf{post}_{push^{\mathsf{S}}}[^{f(a_0,i_0),f(a,i)}/_{s_0,s}]
\end{aligned}
$$

4 A More Realistic Data Refinement

While "infinite" arrays such as a in A are convenient for the specifier, they don't really exist in most programming languages. Java for instance, maps its stacks to vectors which are grown if the need arises. So both are always finite objects yet unbounded. To illustrate the point we're going to refine A to C which uses representation variables c, i, and k, where c is an array of k integers and i is as in A, with the additional data invariant $i \leq k$. We could initialise using the predicate $i = 0 \wedge k > 0$, allowing the implementation to pick a reasonable first size for the stack. The only interesting change is to the *push* operation. Here we need to extend c if the stack is full. Typical implementation would either add a fixed number of cells or double the size.

```
func $isempty^{\mathsf{C}}() : \mathbb{B}\ \cdot$
    var $b \cdot b : [\text{TRUE}, b = (i = 0)]$; return $b$
proc $pop^{\mathsf{C}}()\ \cdot$
    $i : [i > 0, i = i_0 - 1]$
func $top^{\mathsf{C}}() : \mathbb{Z}\ \cdot$
    var $y \cdot y : [i > 0, y = c[i-1]]$; return $y$
proc $push^{\mathsf{C}}(\textbf{value } x)\ \cdot$
    $c, i, k : [\text{TRUE}, c = (c_0 : i_0 \mapsto x) \wedge i = i_0 + 1 \wedge (i \leq k_0 \Rightarrow k = k_0) \wedge (i > k_0 \Rightarrow k > k_0)]$
```

Finding a forward simulation predicate is not too hard. We used the same name for i at both levels because it's used the same at both levels. It still features in the predicate.

$$r'(a,i,c,k) = \forall 0 \leq \ell < i\,(a[\ell] = c[\ell])$$

Unfortunately, it does not represent a function from concrete values to abstract values. The abstract level a is full of garbage above i and even above k. So there's infinitely many abstract states for each concrete state. This means we have to go back to the original proof obligations for forward simulation predicates.[1]

The instances of the forward simulation conditions we need to check are

$$\mathsf{init}^{\mathsf{C}} \Rightarrow \exists a\left(\mathsf{init}^{\mathsf{A}} \wedge r'(a,i,c,k)\right) \tag{1}$$

$$\mathsf{pre}_{j^{\mathsf{A}}} \wedge r'(a,i,c,k) \Rightarrow \mathsf{pre}_{j^{\mathsf{C}}} \quad , \text{for } j \in J \tag{2}$$

$$\mathsf{pre}_{j^{\mathsf{A}}}[^{a_0,i_0}/_{a,i}] \wedge r'(a_0,i_0,c_0,k_0) \wedge \mathsf{post}_{j^{\mathsf{C}}} \Rightarrow \exists a\left(\mathsf{post}_{j^{\mathsf{A}}} \wedge r'(a,i,c,k)\right) \quad , \text{for } j \in J \tag{3}$$

$$\forall c,i,k\,(\exists a\,(r'(a,i,c,k))) \quad , \text{if } \exists j,c,i,k\left(\neg \mathsf{pre}_{j^{\mathsf{C}}}\right) \tag{4}$$

where $J = \{isempty, pop, top, push\}$.

[1] Technically, there is a way around this inconvenience. We could regain a functional correspondence by mapping (c,i,k) to (a,i) where $\forall \ell \in \mathbb{N}\,((\ell < k \Rightarrow a[\ell] = c[\ell]) \wedge (\ell \geq k \Rightarrow a[\ell] = 0))$. We don't do this because it would mandate resetting surrendered stack cells to 0 upon a *pop*.

Let's begin with (1).

$$\begin{aligned}
&\mathsf{init}^{\mathsf{C}}\\
&\Leftrightarrow \quad \langle\text{def. of } \mathsf{init}^{\mathsf{C}}\rangle\\
&i = 0 \wedge k > 0\\
&\Rightarrow \quad \langle\text{logic}\rangle\\
&\exists a\,(i = 0 \wedge \forall 0 \leq \ell < 0\,(a[\ell] = c[\ell]))\\
&\Leftrightarrow \quad \langle\text{def. of } \mathsf{init}^{\mathsf{A}} \text{ and } r'\rangle\\
&\exists a\,\big(\mathsf{init}^{\mathsf{A}} \wedge r'(a, i, c, k)\big)
\end{aligned}$$

Condition (2) is trivially satisfied even for the two operation with notionally non-trivially preconditions because we're using the same precondition on both levels formulated entirely in terms of the one representation variable, i, which we preserve by the refinement. Similarly, condition (3) is rather easy to discharge for the first three operations. The only interesting one is *push* so we focus on that one.

$$\begin{aligned}
&\mathsf{pre}_{push^{\mathsf{A}}}[{}^{a_0,i_0}/_{a,i}] \wedge r'(a_0, i_0, c_0, k_0) \wedge \mathsf{post}_{push^{\mathsf{C}}}\\
&\Leftrightarrow \quad \langle\text{def. of } push^{\mathsf{A}} \text{ and } push^{\mathsf{C}}\rangle\\
&\text{TRUE}[{}^{a_0,i_0}/_{a,i}] \wedge \forall 0 \leq \ell < i_0\,(a_0[\ell] = c_0[\ell]) \wedge\\
&c = (c_0 : i_0 \mapsto x) \wedge i = i_0 + 1 \wedge (i \leq k_0 \Rightarrow k = k_0) \wedge (i > k_0 \Rightarrow k > k_0)\\
&\Leftrightarrow \quad \langle\text{logic: drop true conjunct}\rangle\\
&\forall 0 \leq \ell < i_0\,(a_0[\ell] = c_0[\ell]) \wedge\\
&c = (c_0 : i_0 \mapsto x) \wedge i = i_0 + 1 \wedge (i \leq k_0 \Rightarrow k = k_0) \wedge (i > k_0 \Rightarrow k > k_0)\\
&\Rightarrow \quad \langle\text{logic: extend quantification with } x = x\rangle\\
&\forall 0 \leq \ell < i_0 + 1\,((a_0 : i_0 \mapsto x)[\ell] = (c_0 : i_0 \mapsto x)[\ell]) \wedge\\
&c = (c_0 : i_0 \mapsto x) \wedge i = i_0 + 1 \wedge (i \leq k_0 \Rightarrow k = k_0) \wedge (i > k_0 \Rightarrow k > k_0)\\
&\Rightarrow \quad \langle\text{logic}\rangle\\
&\exists a\,(a = (a_0 : i_0 \mapsto x) \wedge i = i_0 + 1 \wedge \forall 0 \leq \ell < i\,(a[\ell] = c[\ell]))\\
&\Leftrightarrow \quad \langle\text{def. of } push^{\mathsf{A}} \text{ and } r'\rangle\\
&\exists a\,\big(\mathsf{post}_{push^{\mathsf{A}}} \wedge r'(a, i, c, k)\big)
\end{aligned}$$

Since there are operations with non-trivial preconditions in C we need to check (4).

$$\begin{aligned}
&\forall c, i, k\,(\exists a\,(r'(a, i, c, k)))\\
&\Leftrightarrow \quad \langle\text{def. of } r'\rangle\\
&\forall c, i, k\,(\exists a\,(\forall 0 \leq \ell < i\,(a[\ell] = c[\ell])))
\end{aligned}$$

but this is trivially satisfied by choosing a the same as c only extending it past k with, say, 0 or the digits of π.