Examples of Program Construction Using UNITY

Notes on UNITY: 20-90

Edgar Knapp[†]
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
(512) 471-9547
knapp@cs.utexas.edu

10/29/90

1 Introduction

We give two examples of deriving parallel programs from their specifications. The examples share the fact that they both deal with transformational programs, i.e. programs that operate on a certain input, and upon termination yield certain output. They are also alike in that their proofs of correctness depend crucially on a notion of weak fairness, which has complicated existing proofs of these programs considerably (c.f. [?]). Since fairness is "built into" the formal reasoning system of UNITY[?], we are able to avoid these complications. Another aspect that has made existing proofs long and tedious is the fact that in traditional methods concerns of termination detection and the actual computation of the results were lumped together (c.f. [?, ?]). UNITY allows us to concentrate on solving the problem at hand, and treats the problem of termination detection as a separate concern to be addressed by the implementation of a UNITY program. A pleasant surprise to us was that the programs obtained satisfy Reynold's condition [?, ?] in that each statement in them mentions at most one shared variable. Such programs are very easily implemented on synchronous and asynchronous architectures, since the stipulation that each individual statement of the program be executed atomically can be dropped. A final merit of our approach is the fact that much of our programs are derived rather straightforwardly from their specifications, in contrast to the mostly a posteriori verifications found in the literature.

Below we give an overview of how transformational programs are specified in UNITY. In sequential programming, a specification of a program s has the form

$$\{P\} \quad s \quad \{Q\}$$

[†] This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 003658–065 and by the Office of Naval Research Contract N00014-90-J-1640.

with P and Q denoting pre- and postcondition, respectively. In UNITY this program is specified as:

$$[\mathbf{FP}.s \;\Rightarrow\; Q]$$
$$P \;\mapsto\; \mathbf{FP}.s$$

The first line expresses that when the program reaches a fixpoint (which is the notion of termination in UNITY) Q holds. The second line states that if P holds the fixpoint will be reached eventually. Program development proceeds by transforming the initial specification by a series of strengthening steps into one which can be implemented straightforwardly. In the course of these refinements, new program variables may be added to the specification, invariants may be introduced, etc.

The two examples we shall consider differ in that they use different heuristics to obtain the programs. In the first example the progress requirement is refined to obtain the statements of the program, whereas the second example uses a transformation of the fixpoint predicate to arrive first at a correct but inefficient program, which is then refined to a smaller grain of atomicity.

2 Parallel Linear Search

We offer a formal derivation and proof of a solution to the following problem: given a function from integers to booleans, find a point at which the function yields *true*. This note was prompted by a lecture by Krzysztof Apt in which he used program transformations to demonstrate the correctness of a solution to the Parallel Linear Search problem, and by my conviction that in UNITY we should be able to do better.

It turns out that not only become the proofs an order of magnitude shorter, but we are also able to develop program and proof hand in hand. This is not surprising since a posteriori correctness arguments tend to be neither concise nor elegant.

Problem Specification

The problem is the following: Given a function $f : int \rightarrow bool$, find an i such that $f.i$, if such an i exists.

Couched in UNITY notation, we obtain the following specification for program ZS with integer variable i:

$$[\mathbf{FP}.ZS \;\Rightarrow\; f.i] \qquad \text{(ZS0)}$$
$$\langle \exists z :: f.z \rangle \;\mapsto\; \mathbf{FP}.ZS \qquad \text{(ZS1)}$$

We propose to search the positive and negative integers in parallel. To this end we introduce two new variables x and y, and transform the previous specification into a specification of program PZS by replacing $f.i$ by $f.x \vee f.y$. We also add two invariants.

$$[\mathbf{FP}.PZS \;\Rightarrow\; f.x \vee f.y] \qquad \text{(PZS0)}$$
$$\langle \exists z :: f.z \rangle \;\mapsto\; \mathbf{FP}.PZS \qquad \text{(PZS1)}$$
$$\mathbf{invariant}\;\; x \geq 0 \qquad \text{(PZS2)}$$
$$\mathbf{invariant}\;\; y \leq 0 \qquad \text{(PZS3)}$$

The parallel components communicate through a shared boolean variable *found* which indicates successful termination of the search. This idea is captured formally by the following strengthening of the above specification (PZS0 is replaced; PZS1, PZS2 and PZS3 are retained):

$$[\mathbf{FP}\,.\mathit{PZS} \;\Rightarrow\; \mathit{found}] \tag{PZS4}$$

$$\mathbf{invariant}\;\; found \;\Rightarrow\; f.x \vee f.y \tag{PZS5}$$

In a final strenghtening step we eliminate the occurrence of $\mathbf{FP}\,.\mathit{PZS}$ from PZS1 by replacing PZS1 and PZS4 by

$$[\mathbf{FP}\,.\mathit{PZS} \;\equiv\; \mathit{found}] \tag{PZS6}$$

$$\langle \exists z :: f.z \rangle \;\mapsto\; \mathit{found} \tag{PZS7}$$

Clearly, PZS6 implies PZS4, and PZS6 and PZS7 imply PZS1.

Our task is now to design two programs, *POS* and *NEG*, such that $\mathit{POS} \,\square\, \mathit{NEG}$ implements *PZS*. We propose the following design for *POS* with local variable x:

$$[\mathbf{FP}\,.\mathit{POS} \;\equiv\; \mathit{found}] \tag{POS0}$$

$$\langle \exists z : z{\geq}0 : f.z \rangle \;\mapsto\; \mathit{found} \tag{POS1}$$

$$\mathbf{invariant}\;\; \mathit{found} \;\Rightarrow\; f.x \tag{POS2}$$

$$\mathbf{invariant}\;\; x \geq 0 \tag{POS3}$$

Note that all properties are properties in *POS*. Hence we have achieved a complete separartion of the specifications of the two parallel tasks.

The design of *NEG* is completely symmetric and is obtained by replacing $\mathit{POS}, x, \geq$ by $\mathit{NEG}, y, \leq$ in the previous specification.

Correctness of the Design

We now demonstrate the correctness of our design.
Ad PZS2, PZS3: Follow from POS3, symmetry, and the locality of x and y.
Ad PZS5:

$$\begin{array}{ll}
 & \mathbf{invariant}\;\; \neg\mathit{found} \vee f.x \vee f.y \;\;\mathbf{in}\;\; \mathit{POS} \,\square\, \mathit{NEG} \\
\Leftarrow & \quad \{\text{Union Theorem}\} \\
 & (\mathbf{invariant}\;\; \neg\mathit{found} \vee f.x \vee f.y \;\;\mathbf{in}\;\; \mathit{POS}) \;\wedge \\
 & (\mathbf{invariant}\;\; \neg\mathit{found} \vee f.x \vee f.y \;\;\mathbf{in}\;\; \mathit{NEG}) \\
\Leftarrow & \quad \{\text{Consequence Rule}\} \\
 & (\mathbf{invariant}\;\; \neg\mathit{found} \vee f.x \;\;\mathbf{in}\;\; \mathit{POS}) \;\wedge\; (\mathbf{stable}\;\; f.y \;\;\mathbf{in}\;\; \mathit{POS}) \;\wedge \\
 & (\mathbf{invariant}\;\; \neg\mathit{found} \vee f.y \;\;\mathbf{in}\;\; \mathit{NEG}) \;\wedge\; (\mathbf{stable}\;\; f.x \;\;\mathbf{in}\;\; \mathit{NEG}) \\
= & \quad \{\text{POS2; locality; symmetry}\} \\
 & \mathit{true}
\end{array}$$

Ad PZS6: Follows directly from POS0, symmetry, and the Union Theorem.
Ad PZS7:

We use the following Composition Theorem[**?**] for $\mapsto$:

$$\frac{A \;\mapsto\; B \;\;\mathbf{in}\;\; F}{A \;\wedge\; S{=}K \;\mapsto\; B \;\vee\; S{\neq}K \;\;\mathbf{in}\;\; F \,\square\, G}$$

where S is the list of *all* variables shared between F and G, and K is a free variable.

$$
\begin{array}{ll}
 & \langle \exists z : z{\geq}0 : f.z \rangle \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{POS} \\
\Rightarrow & \quad \{\text{Composition Theorem for } \mapsto \text{ with } S, K := \mathit{found}, \mathit{false}\} \\
 & \langle \exists z : z{\geq}0 : f.z \rangle \wedge (\mathit{found}{=}\mathit{false}) \ \mapsto\ \mathit{found} \vee (\mathit{found}{\neq}\mathit{false})\ \ \mathbf{in}\ \ \mathit{POS} \ \Box\ F \\
= & \quad \{\text{predicate calculus}\} \\
 & \langle \exists z : z{\geq}0 : f.z \rangle \wedge \neg\mathit{found} \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{POS} \ \Box\ F \\
\Rightarrow & \quad \{\text{Disjunction with } \langle \exists z : z{\geq}0 : f.z \rangle \wedge \mathit{found} \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{POS} \ \Box\ F\} \\
 & \langle \exists z : z{\geq}0 : f.z \rangle \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{POS} \ \Box\ F
\end{array}
$$

Symmetrically we obtain $\langle \exists z : z{\leq}0 : f.z \rangle \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{NEG} \ \Box\ G$. Combining both we observe:

$$
\begin{array}{ll}
 & (\langle \exists z : z{\geq}0 : f.z \rangle \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{POS} \ \Box\ F) \ \wedge \\
 & (\langle \exists z : z{\leq}0 : f.z \rangle \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{NEG} \ \Box\ G) \\
\Rightarrow & \quad \{\text{instantiation } F, G := \mathit{NEG}, \mathit{POS}\text{; Disjunction}\} \\
 & \langle \exists z :: f.z \rangle \ \mapsto\ \mathit{found}\ \ \mathbf{in}\ \ \mathit{POS} \ \Box\ \mathit{NEG}
\end{array}
$$

Hence the correctness of the design has been established.

Implementation of *POS*

We now develop a program from the specification of *POS*. What do we already know about this program? It has (at least) two variables, x and *found*. Those have to be initialized such that all invariants are met. Since we don't know anything about f the only way to establish POS2 initially is by setting *found* $=$ *false*. There is yet no obvious initial value for x. So we arrive at:

```
program    POS
declare    x : int □ found : bool
initially  x = ? □ found = false
assign     ?
end
```

where the places marked ? remain to be filled in. If we look at POS2 we see that *found* $=$ *true* indicates that our search has been successful. Remember that POS0 implies **stable** *found*, i.e. we had better set *found* only to *true*. Combining these observations we propose as our first statement:

$$\mathit{found} := \mathit{true}\ \ \mathbf{if}\ \ f.x$$

Note that the statement establishes (all properties are in *POS*):

$$f.x\ \ \mathbf{ensures}\ \ \mathit{found} \qquad \text{(POS4)}$$

Furthermore, the fixpoint predicate for this statement is $[f.x \Rightarrow \mathit{found}]$.

Our next goal is to investigate how we can establish POS1. Let us therefore massage this property a bit.

$$\begin{array}{ll}
 & \langle \exists z : z{\geq}0 : f.z \rangle \;\mapsto\; \mathit{found} \\
= & \quad \{\text{trading}\} \\
 & \langle \exists z : z{\geq}0 \wedge f.z : \mathit{true} \;\mapsto\; \mathit{found} \rangle \\
\Leftarrow & \quad \{\text{Disjunction}\} \\
 & \langle \forall z : z{\geq}0 \wedge f.z : \mathit{true} \;\mapsto\; \mathit{found} \rangle \\
\Leftarrow & \quad \{\text{induction and POS5 below}\} \\
 & \langle \forall z : z{\geq}0 \wedge f.z : \langle \forall m : 0{\leq}m \wedge m{<}z : x{=}m \;\mapsto\; \mathit{found} \vee x{>}m \rangle \rangle
\end{array}$$

For the induction to be well-founded we have to require

$$\textbf{invariant } \langle \forall z : z{\geq}0 \wedge f.z : x{\leq}z \rangle \qquad \text{(POS5)}$$

Remark: From POS5 it also follows that POS finds the least $x \geq 0$ such that $f.x$.

Note that our first statement trivially satisfies this invariant. Note also that in order to establish POS5 initially, we had better see to it that initially $x = 0$.

Returning to our previous derivation we observe for any z and m, such that $z{\geq}0 \wedge f.z$ and $0{\leq}m \wedge m{<}z$:

$$\begin{array}{ll}
 & x{=}m \;\mapsto\; \mathit{found} \vee x{>}m \\
\Leftarrow & \quad \{\text{Disjunction}\} \\
 & (x{=}m \wedge \mathit{found} \;\mapsto\; \mathit{found}) \;\wedge\; (x{=}m \wedge f.x \;\mapsto\; \mathit{found}) \;\wedge \\
 & (x{=}m \wedge \neg f.x \wedge \neg \mathit{found} \;\mapsto\; x{>}m)
\end{array}$$

The last step is not as surprising as it may first seem. It just consists of separating what we already know from what we still have to establish. The first conjunct of the last formula holds trivially. The second conjunct is a direct consequence of POS4. Hence we are left with the third conjunct above:

$$\begin{array}{ll}
 & x{=}m \wedge \neg f.x \wedge \neg \mathit{found} \;\mapsto\; x{>}m \\
\Leftarrow & \quad \{\text{property of } \mapsto \} \\
 & x{=}m \wedge \neg f.x \wedge \neg \mathit{found} \;\; \textbf{ensures} \;\; x{>}m
\end{array}$$

We call this last property POS6. We conclude that POS1 can be replaced by POS4, POS5, and POS6.

Now it takes only a moment's reflection to find the statement that establishes POS6, i.e.

$$x := x + k \;\; \textbf{if} \;\; \neg f.x \wedge \neg \mathit{found}$$

for some positive k. Calculating the fixpoint predicate for the new statement we obtain (after some simplification) $[\neg f.x \Rightarrow \mathit{found}]$.

From the conjunction of the fixpoints of the two statements we now get $\mathbf{FP}\,.POS \equiv \mathit{found}]$, as required by POS0. In addition, the first statement of our program does not violate $x{=}m \wedge \neg f.x \wedge \neg \mathit{found}$ **unless** $x{>}m$ since it has no effect when $\neg f.x$. Below we demonstrate the preservation of the stability parts of POS2 and POS5, together with $f.x$ **unless** found, as required by POS4.

Our first proof obligation is:

$$[(\neg \mathit{found} \vee f.x) \wedge \neg f.x \wedge \neg \mathit{found} \;\Rightarrow\; \mathbf{wp}\,.(x := x + k).(\neg \mathit{found} \vee f.x)]$$

which simplifies (by Absorption and the definition of **wp**) to the tautology

$$[\neg f.x \wedge \neg \mathit{found} \;\Rightarrow\; \neg \mathit{found} \vee f.(x+k)]$$

From POS0 and POS2 we conclude as before **stable** $f.x$, from which $f.x$ **unless** *found* follows.

Concerning POS5 we observe

$$\begin{array}{ll}
 & \neg f.x \;\wedge\; \neg \mathit{found} \;\wedge\; \langle \forall z : z{\geq}0 \wedge f.z : x{\leq}z \rangle \\
\Rightarrow & \quad \{\text{predicate calculus; } f.x{\neq}f.z \Rightarrow x{\neq}z\} \\
 & \langle \forall z : z{\geq}0 \wedge f.z : x{<}z \rangle \\
= & \quad \{\text{arithmetic}\} \\
 & \langle \forall z : z{\geq}0 \wedge f.z : x+1{\leq}z \rangle \\
= & \quad \{\text{definition } \mathbf{wp}\} \\
 & \mathbf{wp}\,.(x := x+1).\langle \forall z : z{\geq}0 \wedge f.z : x{\leq}z \rangle
\end{array}$$

which matches our proof obligation just in case $k = 1$.

The Program

We summarize all of the above by the following program:

```
program    $POS$
declare    $x : int$ ▯ $found : bool$
initially  $x{=}0$ ▯ $found = false$
assign     $found := true$ if $f.x$              ▯
           $x := x+1$ if $\neg f.x \wedge \neg found$
end
```

NEG is obtained again completely symmetrically by the substitution $POS, x, + := NEG, y, -$.

Observe that each statement of the program accesses the shared variable *found* at most once. Finally we would like to point out that *POS* (and therefore *NEG*) generalizes easily to an arbitrary number N, $N{>}0$ of parallel programs in the following way. Let i be such that $0{\leq}i \;\wedge\; i{<}N$. Then we get for program i:

```
program    $POS.i$
declare    $x.i : int$ ▯ $found : bool$
initially  $x.i{=}i$ ▯ $found = false$
assign     $found := true$ if $f.(x.i)$              ▯
           $x.i := x.i + N$ if $\neg f.(x.i) \wedge \neg found$
end
```

The only significant difference to the previous program (apart from the increased parallelism) is that for $N{>}1$ we can no longer guarantee to find the least k such that $f.k$.

3 Asynchronous Fixpoint Computation

We develop a parallel algorithm to solve the problem of finding fixpoints of monotonic functions. This problem has many applications, especially in the area of logic programming and deductive databases[**?**].

Our development proceeds in stages. First, a solution with a large grain of atomicity is obtained. In a second step the atomicity concern is addressed separately, leading to a program in which each statement mentions at most one shared variable.

Problem Specification

Let $(L, \prec)$ be a cpo in which all increasing chains are finite. We are given a monotonic function $f : L^N \rightarrow L^N$, $N > 0$. With $x = (x.0, x.1, \ldots, x.(N-1))$ the relation $\prec$ is extended to L^N in the following manner:

$$x \prec y \quad \equiv \quad \langle \forall i :: x.i \preceq y.i \rangle \ \wedge \ \langle \exists i :: x.i \prec y.i \rangle$$

Note that in $(L^N, \prec)$ all increasing chains are finite since $(L, \prec)$ has this property.

The problem is to compute the least fixed point of f, i.e. an $x \in L^N$ such that

0. $x = f.x$ and

1. $\langle \forall y : y \prec x : y \neq f.y \rangle$.

From the mathematical specification above we obtain a UNITY specification of program *fix* with variable x in a straightforward manner:

$$[\mathbf{FP}\,.\mathit{fix} \ \Rightarrow \ x = f.x \ \wedge \ \langle \forall y : y \prec x : y \neq f.y \rangle] \qquad \text{(S0)}$$
$$\mathit{true} \ \mapsto \ \mathbf{FP}\,.\mathit{fix} \qquad \text{(S1)}$$

Using a heuristic from sequential program derivation, we refine S0 to

$$[\mathbf{FP}\,.\mathit{fix} \ \Rightarrow \ x = f.x] \qquad \text{(S2)}$$
$$\mathbf{invariant} \ \langle \forall y : y \prec x : y \neq f.y \rangle \qquad \text{(S3)}$$

Finally, we eliminate **FP** .*fix* from S1 by strengthening the specification even further, replacing S1 and S2 by:

$$[\mathbf{FP}\,.\mathit{fix} \ \equiv \ x = f.x] \qquad \text{(S4)}$$
$$\mathit{true} \ \mapsto \ x = f.x \qquad \text{(S5)}$$

Derivation of a Program

Since x is a vector, say $(x.0, x.1, \ldots, x.(N-1))$, we split f into $(f.0, f.1, \ldots, f.(N-1))$ and propose to update all $x.i$ in parallel. Note that since f is monotonic, so are all the $f.i$.

Now our goal is to design programs *fix.i* with variables $x.i$ such that $\mathit{fix} = \langle \Box\ i :: \mathit{fix}.i \rangle$. Rewriting S4 in light of this we get

$$\mathbf{FP}\,.\langle \Box\ i :: \mathit{fix}.i \rangle = \langle \forall i :: x.i = (f.i).x \rangle$$

which by the Union Theorem suggests to postulate for all i:

$$[\,\mathbf{FP}\,.(\mathit{fix}.i) \quad \equiv \quad x.i{=}(f.i).x]$$

The simplest way to achieve this is to convert this last property directly into a program by reversing the substitution of := by = in the fixpoint predicate (note that from S3 we derive the initial value for $x.i$).

```
program    $\mathit{fix}.i$
declare    $x.i : L$
initially  $x.i = \bot$
assign     $x.i := (f.i).x$
end
```

Note that

$$x{=}m \ \ \textbf{unless} \ \ x{\succ}m \qquad \text{(S6)}$$
$$x{=}m \ \ \textbf{ensures} \ \ x.i{=}(f.i).x \vee x{\succ} \qquad \text{(S7)}$$
$$\textbf{invariant} \ \ \langle \forall i :: x.i{\preceq}(f.i).x \rangle \qquad \text{(S8)}$$

can be proved straightforwardly about the above program. By the Union Theorem, S6 holds in *fix* as well. Hence S7 also holds in *fix*. Similarly, since **stable** $\langle \forall i :: x.i{\preceq}(f.i).x \rangle$ holds in $\mathit{fix}.j$, for all j, by the monotonicity of the $f.i$, S8 holds in *fix*.

It remains to show the validity of S3 and S5.

Verification of S3

Since the initial conditions were chosen so as to satisfy the invariant, all we need to show is **stable** $\langle \forall y : y{\prec}x : y{\neq}f.y \rangle$ **in** *fix*, which by the Union Theorem reduces to

$$\textbf{stable}\ \langle \forall y : y{\prec}x : y{\neq}f.y \rangle \ \ \textbf{in} \ \ \mathit{fix}.i$$

for all i. We will need the following

Lemma 0 *For monotonic* $g\colon A \rightarrow A$*, and any* $a, b \in A$*:*

$$a{\preceq}b \ \wedge \ b{\prec}g.a \ \Rightarrow \ b{\neq}g.b$$

Proof (of Lemma 0): We observe for any a and b

$$\begin{array}{ll} & a{\preceq}b \ \wedge \ b{\prec}g.a \\ \Rightarrow & \quad \{g \text{ monotonic}\} \\ & b{\prec}g.a \ \wedge \ g.a{\preceq}g.b \\ \Rightarrow & \quad \{\text{transitivity}\} \\ & b{\prec}g.b \\ \Rightarrow & \quad \{\prec \text{ irreflexive}\} \\ & b{\neq}g.b \end{array}$$

(*End of Proof*)

Now define $g : L^N \to L^N$ by

$$g.(x.j) = \begin{cases} x.j & \text{if } j{\neq}i \\ (f.i).x & \text{if } j{=}i \end{cases}$$

for all j. Clearly, g is monotonic. Also, $x{\preceq}g.x$ by S8. Then we observe

$$\begin{array}{ll}
 & \mathbf{wp}\,.(x.i := (f.i).x).\langle\forall y : y{\prec}x : y{\neq}f.y\rangle \\
= & \quad \{\text{definition } \mathbf{wp}\text{; definiton } g\} \\
 & \langle\forall y : y{\prec}g.x : y{\neq}f.y\rangle \\
= & \quad \{\text{range splitting; } x{\preceq}g.x\} \\
 & \langle\forall y : y{\prec}x : y{\neq}f.y\rangle \;\wedge\; \langle\forall y : x{\preceq}y \wedge y{\prec}g.x : y{\neq}f.y\rangle \\
= & \quad \{\text{Lemma 0}\} \\
 & \langle\forall y : y{\prec}x : y{\neq}f.y\rangle
\end{array}$$

which had to be proved.

Verification of S5

We observe in *fix*:

$$\begin{array}{ll}
 & \mathit{true} \;\mapsto\; x{=}f.x \\
\Leftarrow & \quad \{\text{Induction; all increasing chains are finite}\} \\
 & x{=}m \;\mapsto\; x{=}f.x \;\vee\; x{\succ}m \\
\Leftarrow & \quad \{\text{RHS rule}\} \\
 & x{=}m \;\mapsto\; (x{=}f.x \wedge x{=}m) \;\vee\; x{\succ}m \\
\Leftarrow & \quad \{\text{Completion Theorem with } p.i, q.i, b := x{=}m, x{=}m \wedge x.i{=}(f.i).x, x{\succ}m\} \\
 & \langle\forall i :: x{=}m \;\mapsto\; (x{=}m \wedge x.i{=}(f.i).x) \;\vee\; x{\succ}m\rangle \;\wedge \\
 & \langle\forall i :: x{=}m \wedge x.i{=}(f.i).x \;\; \mathbf{unless} \;\; x{\succ}m\rangle
\end{array}$$

We demonstrate each of the remaining proof obligations in turn. Concerning the first conjunct we observe in *fix* for any i:

$$\begin{array}{ll}
 & x{=}m \;\mapsto\; (x{=}m \wedge x.i{=}(f.i).x) \;\vee\; x{\succ}m \\
\Leftarrow & \quad \{\text{PSP with S6 and predicate calculus}\} \\
 & x{=}m \;\mapsto\; x.i{=}(f.i).xx{\succ}m \\
\Leftarrow & \quad \{\text{definition } \mapsto\} \\
 & x{=}m \;\; \mathbf{ensures} \;\; x.i{=}(f.i).xx{\succ}m \\
= & \quad \{\text{S7}\} \\
 & \mathit{true}
\end{array}$$

Continuing now with the second conjunct we observe for any i:

$$\begin{array}{ll}
 & x{=}m \wedge x.i{=}(f.i).x \;\; \mathbf{unless} \;\; x{\succ}m \;\; \mathbf{in} \;\; \mathit{fix} \\
= & \quad \{\text{Union Theorem}\} \\
 & \langle\forall j :: x{=}m \wedge x.i{=}(f.i).x \;\; \mathbf{unless} \;\; x{\succ}m \;\; \mathbf{in} \;\; \mathit{fix}.j\rangle
\end{array}$$

whose proof from the program text is straightforward, using S8.

Reducing the Grain of Atomicity of the Solution

If we look at program *fix* we find that each of the N statements contains $N+1$ accesses to shared variables. In light of the monotonicity of the $f.i$ we propose to keep local copies of x which are updated asynchronously. With local vectors $a.i$ we then get the refined programs

```
program    $fix'.i$
declare    $x.i : L \; ; \; a.i : L^N$
initially  $x.i = \perp \;\wedge\; \langle \forall j :: a.i.j = \perp \rangle$
assign      $x.i := (f.i).(a.i)$        $\square$
            $a.i := x$
end
```

The grain of atomicity of the second statement is still too large. We therefore propose to update all $a.i.j$ asynchronously:

```
program    $fix''.i$
declare    $x.i : L \; ; \; a.i : L^N$
initially  $x.i = \perp \;\wedge\; \langle \forall j :: a.i.j = \perp \rangle$
assign      $x.i := (f.i).(a.i)$        $\square$
           $\langle \square\, j :: a.i.j := x.j \rangle$
end
```

Note that S6 still holds of *fix*″. In addition, we have

$$a.i{=}k \;\; \textbf{unless} \;\; a.i{\succ}k \;\; \textbf{in} \;\; \mathit{fix}'' \tag{S9}$$

The following properties are similar to S7, and are again proved straightforwardly from the program text and the UNION Theorem (all properties are in *fix*″):

$$\mathit{true} \;\; \textbf{ensures} \;\; x.i = (f.i).(a.i) \tag{S10}$$

$$\mathit{true} \;\; \textbf{ensures} \;\; a.i.j := x.j \tag{S11}$$

Analogous to S8, we conjecture the following invariant in *fix*″:

$$\textbf{invariant} \;\; \langle \forall i :: a.i{\preceq}x \;\wedge\; x.i{\preceq}(f.i).(a.i) \rangle \tag{S12}$$

Proof (of S12): Since it clearly holds initially, it is sufficient to show stability in all *fix*″.*k*.

$$\begin{array}{ll}
 & \textbf{wp}\,.\text{“}x.k := (f.k).(a.k)\text{”}.\langle \forall i :: a.i{\preceq}x \;\wedge\; x.i{\preceq}(f.i).(a.i) \rangle \\
= & \quad \{\text{definition } \textbf{wp} \text{ and predicate calculus}\} \\
 & \langle \forall i,j : j{\neq}k : a.i.j{\preceq}x.j \rangle \;\wedge\; \langle \forall i :: a.i.k{\preceq}(f.k).(a.k) \rangle \;\wedge \\
 & \langle \forall i : i{\neq}k : x.i{\preceq}(f.i).(a.i) \rangle \;\wedge\; (f.k).(a.k){\preceq}(f.k).(a.k) \\
\Leftarrow & \quad \{\text{transitivity of } \preceq\} \\
 & \langle \forall i,j : j{\neq}k : a.i.j{\preceq}x.j \rangle \;\wedge\; \langle \forall i :: a.i.k{\preceq}x.k \wedge x.k{\preceq}(f.k).(a.k) \rangle \;\wedge \\
 & \langle \forall i : i{\neq}k : x.i{\preceq}(f.i).(a.i) \rangle \\
= & \quad \{\text{predicate calculus}\} \\
 & \langle \forall i :: a.i{\preceq}x \;\wedge\; x.i{\preceq}(f.i).(a.i) \rangle
\end{array}$$

Now define $b.k$ by

$$b.k.j = \begin{cases} a.k.j & \text{if } j{\neq}l \\ x.l & \text{if } j{=}l \end{cases}$$

for all j. Then we observe

$$\begin{array}{ll}
 & \mathbf{wp}\,.\text{“}a.k.l := x.l\text{”}.\langle\forall i :: a.i{\preceq}x \;\wedge\; x.i{\preceq}(f.i).(a.i)\rangle \\
= & \quad \{\text{definition } \mathbf{wp} \text{ and predicate calculus}\} \\
 & \langle\forall i,j : i{\neq}k \wedge j{\neq}l : a.i.j{\preceq}x.j\rangle \;\wedge\; x.l{\preceq}x.l \;\wedge \\
 & \langle\forall i : i{\neq}k : x.i{\preceq}(f.i).(a.i)\rangle \;\wedge\; x.k{\preceq}(f.k).(b.k) \\
\Leftarrow & \quad \{\text{transitivity of } {\preceq} \text{ and predicate calculus}\} \\
 & \langle\forall i,j : i{\neq}k \wedge j{\neq}l : a.i.j{\preceq}x.j\rangle \;\wedge \\
 & \langle\forall i : i{\neq}k : x.i{\preceq}(f.i).(a.i)\rangle \;\wedge\; x.k{\preceq}(f.k).(a.k) \;\wedge\; (f.k).(a.k){\preceq}(f.k).(b.k) \\
\Leftarrow & \quad \{\text{predicate calculus and monotonicity of } f.k\} \\
 & \langle\forall i,j : i{\neq}k \wedge j{\neq}l : a.i.j{\preceq}x.j\rangle \;\wedge\; \langle\forall i :: x.i{\preceq}(f.i).(a.i)\rangle \;\wedge\; a.k{\preceq}b.k \\
\Leftarrow & \quad \{a.k{\preceq}b.k \;\Leftarrow\; a.k.l{\preceq}x.l\} \\
 & \langle\forall i,j : i{\neq}k \wedge j{\neq}l : a.i.j{\preceq}x.j\rangle \;\wedge\; \langle\forall i :: x.i{\preceq}(f.i).(a.i)\rangle \;\wedge\; a.k.l{\preceq}x.l \\
= & \quad \{\text{predicate calculus}\} \\
 & \langle\forall i :: a.i{\preceq}x \;\wedge\; x.i{\preceq}(f.i).(a.i)\rangle
\end{array}$$

(*End of Proof*)

From S12 property S8 now follows. Indeed, we observe for any i:

$$\begin{array}{ll}
 & x.i{\preceq}(f.i).x \\
\Leftarrow & \quad \{\text{transitivity of } {\preceq}\} \\
 & x.i{\preceq}(f.i).(a.i) \;\wedge\; (f.i).(a.i){\preceq}(f.i).x \\
\Leftarrow & \quad \{\text{S12; } f.i \text{ monotonic}\} \\
 & a.i{\preceq}x \\
= & \quad \{\text{S12}\} \\
 & \mathit{true}
\end{array}$$

We now show that S3 and S5 hold in fix'' as well.

Proof of S3

Initially, the invariant is true vacuously. We now show stability.

$$\begin{array}{ll}
 & \mathbf{wp}\,.\text{“}a.i.j := x.j\text{”}.\langle\forall y : y{\prec}x : y{\neq}f.y\rangle \\
= & \quad \{a.i.j \text{ does not occur in } \langle\forall y : y{\prec}x : y{\neq}f.y\rangle\} \\
 & \langle\forall y : y{\prec}x : y{\neq}f.y\rangle
\end{array}$$

Next define d and e as

$$d.j = \begin{cases} x.j & \text{if } j{\neq}i \\ (f.i).(a.i) & \text{if } j{=}i \end{cases}$$

$$e.j = \begin{cases} x.j & \text{if } j{\neq}i \\ (f.i).x & \text{if } j{=}i \end{cases}$$

for all j. Then we observe

$$\begin{array}{ll}
 & \mathbf{wp}\,.\text{“}x.i := (f.i).(a.i)\text{”}.\langle\forall y : y \prec x : y \neq f.y\rangle \\
= & \quad \{\text{definition } \mathbf{wp} \text{ and } D\} \\
 & \langle\forall y : y \prec d : y \neq f.y\rangle \\
\Leftarrow & \quad \{\text{See Lemma 1 below}\} \\
 & \langle\forall y : y \prec e : y \neq f.y\rangle \\
= & \quad \{\text{definition } \mathbf{wp} \text{ and E}\} \\
 & \mathbf{wp}\,.\text{“}x.i := (f.i).x\text{”}.\langle\forall y : y \prec x : y \neq f.y\rangle \\
\Leftarrow & \quad \{\text{proof of first program}\} \\
 & \langle\forall y : y \prec x : y \neq f.y\rangle
\end{array}$$

Lemma 1 $\quad d \preceq e$

Proof (of Lemma 1):

$$\begin{array}{ll}
 & d \preceq e \\
= & \quad \{\text{definition } \preceq\} \\
 & \langle\forall j :: d.j \preceq e.j\rangle \\
= & \quad \{\text{definition } d \text{ and } e\} \\
 & \langle\forall j : j \neq i : x.j \preceq x.j\rangle \;\wedge\; (f.i).(a.i) \preceq (f.i).x \\
\Leftarrow & \quad \{\text{predicate calculus; } f.i \text{ monotonic}\} \\
 & a.i \preceq x \\
= & \quad \{\text{S12}\} \\
 & \mathit{true}
\end{array}$$

(*End of Proof*)

Proof of S5

Looking at the progress proof we realize that $x{=}m \wedge x.i{=}(f.i).x$ **unless** $x \succ m$ still holds in all $\mathit{fix}''.j$ because of S8. So it is sufficient to show for all i, j:

$$x{=}m \;\mapsto\; (x{=}m \wedge x.i{=}(f.i).x) \;\vee\; x \succ m \;\;\mathbf{in}\;\; \mathit{fix}''$$

So we observe in fix'' for any i:

$$\begin{array}{ll}
 & x{=}m \;\mapsto\; (x{=}m \wedge x.i{=}(f.i).x) \;\vee\; x \succ m \\
\Leftarrow & \quad \{\text{strengthening RHS of } \mapsto\} \\
 & x{=}m \;\mapsto\; (x{=}m \wedge x.i{=}(f.i).(a.i) \wedge a.i{=}x) \;\vee\; x \succ m \\
\Leftarrow & \quad \{\text{induction}\} \\
 & x{=}m \wedge a.i{=}k \;\mapsto\; (x{=}m \wedge a.i \succ k) \;\vee\; (x{=}m \wedge x.i{=}(f.i).(a.i) \wedge a.i{=}x) \;\vee\; x \succ m \\
\Leftarrow & \quad \{\text{strengthening RHS of } \mapsto\} \\
 & x{=}m \wedge a.i{=}k \;\mapsto\; (x{=}m \wedge a.i \succ k) \;\vee\; (x{=}m \wedge x.i{=}(f.i).(a.i) \wedge a.i{=}x \wedge a.i{=}k) \;\vee\; x \succ m \\
\Leftarrow & \quad \{\text{Completion Theorem}\} \\
 & (x{=}m \wedge a.i{=}k \;\mapsto\; (x{=}m \wedge a.i \succ k) \;\vee\; (x{=}m \wedge x.i{=}(f.i).(a.i) \wedge a.i{=}k) \;\vee\; x \succ m) \;\wedge \\
 & \langle\forall j :: x{=}m \wedge a.i{=}k \;\mapsto\; (x{=}m \wedge a.i \succ k) \;\vee\; (x{=}m \wedge a.i.j{=}x.j \wedge a.i{=}k) \;\vee\; x \succ m\rangle \;\wedge \\
 & (x{=}m \wedge a.i{=}k \wedge x.i{=}(f.i).(a.i) \;\;\mathbf{unless}\;\; (x{=}m \wedge a.i \succ k) \;\vee\; x \succ m) \;\wedge \\
 & \langle\forall j :: x{=}m \wedge a.i{=}k \wedge a.i.j{=}x.j \;\;\mathbf{unless}\;\; (x{=}m \wedge a.i \succ k) \;\vee\; x \succ m\rangle
\end{array}$$

The third and fourth conjuncts above are proved straightforwardly from the program text. Now we continue to massage the first conjunct:

$$\begin{array}{ll}
 & x{=}m \wedge a.i{=}k \;\mapsto\; (x{=}m \wedge a.i{\succ}k) \;\vee\; (x{=}m \wedge x.i{=}(f.i).(a.i) \wedge a.i{=}k) \;\vee\; x{\succ}m \\
= & \quad \{\text{predicate calculus}\} \\
 & x{=}m \wedge a.i{=}k \;\mapsto\; (((x.i{=}(f.i).(a.i) \wedge a.i{=}k) \;\vee\; a.i{\succ}k) \;\wedge\; x{=}m) \;\vee\; x{\succ}m \\
\Leftarrow & \quad \{\text{PSP with S6}\} \\
 & a.i{=}k \;\mapsto\; (x.i{=}(f.i).(a.I) \wedge a.i{=}k) \;\vee\; a.i{\succ}k \\
\Leftarrow & \quad \{\text{PSP with S9}\} \\
 & \mathit{true} \;\mapsto\; x.i{=}(f.i).(a.i) \\
= & \quad \{\text{S10}\} \\
 & \mathit{true}
\end{array}$$

The second conjunct is proved analogously using S11 instead of S10.

Acknowledgement

Thanks are due to Jay Misra for suggesting a couple of significant simplifications to our treatment of the Parallel Linear Search and for pointing out a problem with one of the proofs . Ken Calvert assisted in formulating S12. We are also grateful to Jacob Kornerup for many stimulating discussions.

References