CS-Note* 78

Andrew J. Cook
May 14, 2004

Array Sieve

1 The problem

During the proof of certain problems which access(*ele*) and update(*upd*) arrays there may arise situations where an update may block a ripple proof when it isn't to be applied to any element of the array that is relevant. To unblock the proof in this case we may need to show that such an update will have no effect on the value of the term we are looking at and so can safely discard it.

This problem is discussed in CS-Notes 30 and 62. These CS-Notes describe the *Polish Flag Problem* (CS-Note 10) At one point in the proof of this problem we have the goal:

$$(\forall p : int.((0 \leq p) \wedge (p < i - 2)) \rightarrow P(table, j, p) \leq P(table, j, p + 1)) \tag{1}$$

Where P(A,B,C) is:

$$ele(upd(upd(A, [B - 1], ele(A, [B])), [B], ele(A, [B - 1])), [C]) \tag{2}$$

Expanding one of the Ps in (1) with (2) yeilds the following on the right hand side of the implication:

$$ele(\boxed{upd(upd(\boxed{table}, [j - 1], ele(table, [j])), [j], ele(table, [j - 1]))}^{\uparrow}, [p]) \leq P(table, j, p + 1) \tag{3}$$

Rippling is blocked because of the two *upd*s in the *ele* access statement. However, the *ele* accesses an element at p while the *upd* change the values at j and $j - 1$. If we can show that the j and $j - 1$ cannot lie within the range of p we may get rid of the two *upd* statements on the grounds that it will not affect the result of the *ele* access statement.

On removing the updates and unfolding the right hand side of the inequality (3) becomes:

$$ele(table, [p]) \leq ele(\boxed{upd(upd(\boxed{table}, [j - 1], ele(table, [j])), [j], ele(table, [j - 1]))}^{\uparrow}, [p + 1]) \tag{4}$$

By also showing that j and $j - 1$ cannot lie within the range of $p + 1$, we remove the updates on the right hand side of the inequality. So equation (1) is reduced to:

$$(\forall p : int.((0 \leq p) \wedge (p < i - 2)) \rightarrow ele(table, [p]) \leq ele(table, [p + 1])) \tag{5}$$

With the wave-fronts gone rippling is now unblocked.

2 Straining the Variables

To show that j and $j - 1$ cannot lie within the range of p we must first gather what information we can about the values of j and p. Information about j can be found in the hypothesese while information about p is on the left hand side of the $\rightarrow$ within the goal.

From the hypothesese we have:

$$1 \leq i \tag{6}$$
$$i \leq last \tag{7}$$
$$i \leq j \tag{8}$$
$$j \leq last \tag{9}$$
$$j = i \tag{10}$$
$$i \neq last \tag{11}$$

And from within the goal we have:

$$0 < p \tag{12}$$
$$p < i - 2 \tag{13}$$

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

This is enough to allow us to deduce that j and $j-1$ cannot lie within the range of p. So when(along with the rest of the information) we assert that j and $j-1$ equal p the prolog constraint solver fails(here $last$ takes the value of 9):

```
| ?- 1 #=< I,
     I #=< 9,
     I #=< J,
     J #=< 9,
     J #= I,
     I #\= 9,
     0 #=< P,
     P #< (I-2),
     J #= P.
no
| ?- 1 #=< I,
     I #=< 9,
     I #=< J,
     J #=< 9,
     J #= I,
     I #\= 9,
     0 #=< P,
     P #< (I-2),
     (J-1) #= P.
no
```

But the hypothesese also contain the following which involve i or j:

$$ordered(table, 0, i-2) \tag{14}$$

$$partitioned(table, 0, i-2, last) \tag{15}$$

$$minimum(table, j, last) \tag{16}$$

$$ele(table, [j-1]) \quad > \quad ele(table, [j]) \tag{17}$$

In these the i or j is an argument to a function and not part of an inequality. These particular functions do not add any additional constraints to the problem we are attempting to solve but can we say that for other functions they will not? If the problem is not adequately constrained by the obvious constraint information we have access to we may be forced to unfold the definitions of functions to discover extra information.

3 Sieve Method

3.1 Preconditions

The preconditions to the sieve method would be quite specialised for the case of array index sieveing. The sieve method requires the existence of an update within an element accessor otherwise we risk throwing away updates that we shouldn't.

$$ele(upd(array, j, val), p)$$

In order to allow the removal of the upd we must show that the value at p which is being accessed isn't within the range of j where an update may happen.

3.2 Information Collection

A sieve method requires the collection of all the information pertaining to the problem that we are interested in. This information can come from the hypothesese or from within the goal on the left side of an implication. It seems like collecting all the information necessary may be quite tricky in some situations.

3.3 Invoking the constraint solver

This is the easy part though we use the constraint solver's failure to indicate success.

3.4 Effects

The effect of a successful application of the method to the goal should be the removal of the *upd* from the goal.

4 Collecting The Information

If we look at (3) we can extract the variables that are important for our constraint problem:

$$ele(\boxed{upd(\ldots,[j],\ldots)}^{\uparrow},[p]) \leq \ldots$$

We have p and j as the elements being accessed and updated. If we search for information regarding these variables we come up with (8), (9), (10), (12) and (13):

$$\begin{array}{lccc} 8: & i & \leq & j \\ 9: & j & \leq & last \\ 10: & j & = & i \\ 12: & 0 & < & p \\ 13: & p & < & i-2 \end{array}$$

However, this may not be sufficient(though it is in this case due to (10)) because both p and j have constraints involving i so we need to take into account the hypotheses involving i as well.

We could examine each variable in turn and include any variables it is related to in our search or we could just look for every relation we can find and extract all the variables from each.

5 Translating to Spade

Getting SPADE to explicitly perform the constraint solving step could be quite tricky or it could be very straight forward.

In the spade rule library we have the rule:

```
array(3):  element(update(A,J,X),K) & element(A,K) are_interchangeable if [ J<>K ].
```

This expresses exactly the rule we wish to apply to our goal.

The possible problem comes when SPADE tries to establish that $j \neq k$(`[ J<>K ]`). SPADE has some fairly sophisticated code for satisfying conditional expressions using hypothesese but the tamed checker we have been using for implementing plans has bypassed much of this to allow the construction of implication rewrites. Also the use of information from within the goal may prove to be a problem and stripping quantification and/or forward chaining to introduce this information into the hypothesese may be necessary for SPADE to successfully use the above rule.

The possible necessity of moving constraints to the hypotheses is a toublesome one. If we are to use the sieve method to unblock rippling, as in the example, then typically we may not be able to perform the form of simplification at the SPADE level that is required to make the step go through.

6 Conclusion

A sieve method for *Clam* looks possible but translating this to SPADE could be tricky.