Andrew Ireland
November 21, 2002

Automatic Discovery for RTC Proofs

1 Introduction

At the recent SPARK User Group meeting[1], Phil Thornley (BAE Systems, Warton) presented a user's view of exception freedom proofs. The presentation was based upon the experience of his team in tackling the post-hoc verification of a non-trivial application (approx 150 KLOC). While a significant percentage of RTC VCs generated for the application were discharged automatically by the Simplifier, proving the remaining VCs involved non-trivial user interaction. I focus here on the example that Phil used to illustrate the nature of the non-trivial user interaction. While it is dangerous to generalize, I believe that the example illustrates features that will be common to many RTC VCs. Based on this hypothesis, I explore within this note how we could **automatically** prove Phil's example within NuSPADE.

2 Phil's example

The fragment of SPARK source upon which the example is based follows:

```
subtype Index_Type is Integer range 0 .. 9;
type Data_Set_Type is array (Index_Type) of Integer;
...
Result:= 0;
for I in Index_Type loop
   --# assert true;
   if Data_Set(I) >= 0 and Data_Set(I) <= 100 then
      Result:= Result+Data_Set(I);
   end if;
end loop;
```

The set of RTC VCs generated for this code includes the following:

```
H1:     for_all (i___1: integer, ((i___1 >= index_type__first) and (
           i___1 <= index_type__last)) -> ((element(
           data_set, [i___1]) >= integer__first) and (element(
           data_set, [i___1]) <= integer__last))) .
H2:    loop__1__i >= index_type__first .
H3:    loop__1__i <= index_type__last .
H4:    element(data_set, [loop__1__i]) >= 0 .
H5:    element(data_set, [loop__1__i]) <= 100 .
        ->
C1:    result + element(data_set, [loop__1__i]) >= integer__first .
C2:    result + element(data_set, [loop__1__i]) <= integer__last .
```

This VC is not provable by the Simplifier and relates to the following assignment statement within the code:

```
Result:= Result+Data_Set(I);
```

Proving the VC corresponds to guaranteeing that the assignment will never cause `Result` to exceed the integer bounds `integer__first` and `integer__last`. The Simplifier fails to prove the VC because there is not enough proof context to establish `C2`. Overcoming this failure requires a *eureka step* – an appropriate loop invariant. In Phil's presentation a loop invariant was provided manually of the form:

```
--# assert Result >= 0 and Result <= I * 100;
```

Note that it is the construction of the right conjunct, *i.e.* `Result <= I * 100`, that represents the eureka step.

*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).

[1] October 16^{th} 2002.

3 Identifying the need for generalization

As is often the case, an analysis of a failed proof attempt provides useful pointers as to how best to progress the search for a proof. Below I focus upon the key structure of the failed VC presented above[2]:

$$\begin{array}{lrcl}
\text{Givens:} & \ldots & & \ldots \\
& ele(data_set, i) & \leq & 100 \\
& \ldots & & \ldots \\
\text{Goal:} & r + ele(data_set, i) & \leq & integer_last
\end{array}$$

Note that there exists similarities between the goal and the given hypothesis. These similarities can be highlighted using the annotations of the ripple method, *i.e.*

$$\begin{array}{lrcl}
\text{Givens:} & \ldots & & \ldots \\
& ele(data_set, i) & \leq & 100 \\
& \ldots & & \ldots \\
\text{Goal:} & \boxed{r + \underline{ele(data_set, i)}}^{\uparrow} & \leq & \boxed{integer_last}^{\uparrow}
\end{array}$$

The wave-front annotations have been applied in order to hide all the mismatching term structure. We could now search for wave-rules (lemmas) that will reduce the differences and achieve a match between goal and given.

Alternatively, we could assume that parts of the mismatching term structure corresponds to a hypothesis that is missing. Such term structure would correspond to an additional skeleton. In this particular case we could consider r as part of a missing hypothesis. Multiple skeletons requires *coloured rippling* [6]. Using red and blue for the alternative skeletons, our annotated goal now becomes[3]:

$$\begin{array}{lrcl}
\text{Givens:} & \ldots & & \ldots \\
& r & \leq & \ldots \\
& ele(data_set, i) & \leq & 100 \\
& \ldots & & \ldots \\
\text{Goal:} & \boxed{\underline{r} + \underline{ele(data_set, i)}}^{\uparrow} & \leq & \boxed{integer_last}^{\uparrow}
\end{array}$$

Note the use of the purple to indicate where skeletons overlap. The wave-front on the right-hand side of the goal contains no wave-holes. According to the definition of rippling, hole-less wave-fronts can only occur within *sinks*, *i.e.* positions that correspond to universally quantified variables within a given hypothesis. Patching this problem corresponds to introducing additional skeleton term structure into the right-hand side goal. Adopting a middle-out style of reasoning, higher-order meta-variables are required to represent unknown term structure:

$$\begin{array}{lrcl}
\text{Givens:} & \ldots & & \ldots \\
& r & \leq & F(i, r) \\
& ele(data_set, i) & \leq & 100 \\
& \ldots & & \ldots \\
\text{Goal:} & \boxed{\underline{r} + \underline{ele(data_set, i)}}^{\uparrow} & \leq & F(\boxed{\underline{i} + 1}^{\uparrow}, \boxed{\underline{r} + ele(data_set, i)}^{\uparrow})
\end{array}$$

Note that F is a second-order meta-variable. The choice of i and r as the context of F is determined by the fact that they are the only variables within the loop that are modified. Following the middle-out reasoning approach, the patching process involves progressing the proof in both directions, *i.e.* rippling forward to the point of fertilization with the given hypotheses and rippling back to the origins of the differences. The idea being that F will be instantiated as a side-effect of the proof planning.

[2]Note that r is an abbreviation for `result` and ele is an abbreviation for `element`.

[3]For the full effect colour printing is recommended.

4 Bridging the gaps by proof

Instead of tackling the general case via middle-out reasoning I focus on the partially defined condition, *i.e.* $r \leq \ldots$. In particular I attempt to construct specific proofs for the condition over the first n iterations of the loop. The idea is that the structure of the n proofs will suggest a suitable generalization. The overall strategy that is being proposed comprises of the following 3 basic steps:

1. generate proofs for the first n iterations, *i.e.* $P(0), P(1), \ldots, P(n-1)$.
2. identify a repeated pattern across the n proofs.
3. use the pattern to suggest a candidate generalization (invariant).

Because I am concerned with RTC VCs that arise within the context of program loops, the construction of the n proofs corresponds to an unrolling of the first n iterations of the loop. The value of n will be decided heuristically, but I believe it will be relatively small in practise, *i.e.* 3 or 4. The key component of the strategy is the mechanism that suggests candidate invariants. This is essentially the problem of inductive generalization, and is well known to be undecidable in general. However, the interesting question is can we develop a generalization technique that will have significant success in generating the kinds of invariants that are typically required within industrial applications.

Within the context of the specific example, I will take n to equal 4. First time through the loop we know the following:

$$\begin{array}{rcll} i_0 & = & 0 & \wedge \\ ele(data\_set, i_0) & \geq & 0 & \wedge \\ ele(data\_set, i_0) & \leq & 100 & \wedge \\ r_0 & = & 0 & \end{array}$$

Note that I use historical subscripts for variables. Note also that $r_0 = 0$ is induced directly from the code. This bottom style of analysis will be common place and provision will need to be made for it. By a process of forward inference, an upper bound on r_0 can be inferred:

$$\begin{array}{rcll} i_0 & = & 0 & \wedge \\ ele(data\_set, i_0) & \geq & 0 & \wedge \\ ele(data\_set, i_0) & \leq & 100 & \wedge \\ r_0 & = & 0 & \wedge \\ r_0 & \leq & 0 & \end{array}$$

The above corresponds to $P(0)$, that is a proof of the upper bound on r on entry into the loop.

A proof of $P(1)$ is constructed from the proof of $P(0)$ using the *strongest postcondition* semantics for assignment together forward inference. The result is as follows:

$$\begin{array}{rcll} i_0 & = & 0 & \wedge \\ ele(data\_set, i_0) & \geq & 0 & \wedge \\ ele(data\_set, i_0) & \leq & 100 & \wedge \\ r_0 & = & 0 & \wedge \\ r_0 & \leq & 0 & \wedge \\ i_1 & = & 1 & \wedge \\ ele(data\_set, i_1) & \geq & 0 & \wedge \\ ele(data\_set, i_1) & \leq & 100 & \wedge \\ r_1 & = & r_0 + ele(data\_set, i_0) & \wedge \\ r_1 & \leq & 0 + 100 & \end{array}$$

The key inference here involves deducing

$$r_1 \leq 0 + 100$$

from the fact that r_1 is equal to

$$r_0 + ele(data\_set, i_0).$$

In a similar way we can construct a proof of $P(2)$ and $P(3)$:

$$
\begin{array}{rcll}
i_0 & = & 0 & \wedge \\
ele(data\_set, i_0) & \geq & 0 & \wedge \\
ele(data\_set, i_0) & \leq & 100 & \wedge \\
r_0 & = & 0 & \wedge \\
r_0 & \leq & 0 & \wedge \\
i_1 & = & 1 & \wedge \\
ele(data\_set, i_1) & \geq & 0 & \wedge \\
ele(data\_set, i_1) & \leq & 100 & \wedge \\
r_1 & = & r_0 + ele(data\_set, i_0) & \wedge \\
r_1 & \leq & 0 + 100 & \wedge \\
i_2 & = & 2 & \wedge \\
ele(data\_set, i_2) & \geq & 0 & \wedge \\
ele(data\_set, i_2) & \leq & 100 & \wedge \\
r_2 & = & r_1 + ele(data\_set, i_1) & \wedge \\
r_2 & \leq & (0 + 100) + 100 & \wedge \\
i_3 & = & 3 & \wedge \\
ele(data\_set, i_3) & \geq & 0 & \wedge \\
ele(data\_set, i_3) & \leq & 100 & \wedge \\
r_3 & = & r_2 + ele(data\_set, i_2) & \wedge \\
r_3 & \leq & ((0 + 100) + 100) + 100 &
\end{array}
$$

Given the specific proofs of $P(0)$ through to $P(3)$, the next step involves identifying repeated patterns across the proofs. Or more precisely I am looking for a repeated pattern within those parts of the proofs that differ on each iteration. On inspection, the value of i and the upper bound of r can be seen as the relevant parts of the proofs:

$$
\begin{array}{lcll}
i_0 & = & 0 & \wedge \\
r_0 & \leq & 0 & \wedge \\
\ldots & & \ldots & \\
i_1 & = & 1 & \wedge \\
r_1 & \leq & 0 \underbrace{+100}_{\alpha} & \wedge \\
\ldots & & \ldots & \\
i_2 & = & 2 & \wedge \\
r_2 & \leq & \underbrace{(0 \underbrace{+100}_{\alpha}) + 100}_{2*\alpha} & \wedge \\
\ldots & & \ldots & \\
i_3 & = & 3 & \wedge \\
r_3 & \leq & \underbrace{\underbrace{((0 \underbrace{+100}_{\alpha}) + 100)}_{2*\alpha} + 100}_{3*\alpha} & \wedge
\end{array}
$$

As highlighted above, the repeated pattern corresponds to the accumulation of the "$\ldots + 100$" terms. The aim is to express the repeated pattern as a constant (invariant) term. Here this involves factoring-out the repeated pattern by expressing it in terms of i, giving:

$$i * 100$$

This suggests that the partial condition for r identified by the ripple analysis in §3 should take the form

$$r \leq i * 100$$

This discovery gives rise to the key part of the missing loop invariant:

```
subtype Index_Type is Integer range 0 .. 9;
type Data_Set_Type is array (Index_Type) of Integer;
...
Result:= 0;
for I in Index_Type loop
--# assert Result >= 0 and Result <= I * 100;
   if Data_Set(I) >= 0 and Data_Set(I) <= 100 then
      Result:= Result+Data_Set(I);
   end if;
end loop;
```

Note that within this note I do not consider the generation of the lower bound on r as it is relatively trivial.

5 Proof planning the revised VC

With the revised proof annotation given above, the revised VC takes the form:

```
H1:    result >= 0 .
H2:    result <= loop__1__i * 100 .
H3:     for_all (i___1: integer, ((i___1 >= index_type__first) and (
           i___1 <= index_type__last)) -> ((element(
           data_set, [i___1]) >= integer__first) and (element(
           data_set, [i___1]) <= integer__last))) .
H4:    loop__1__i >= index_type__first .
H5:    loop__1__i <= index_type__last .
H6:    element(data_set, [loop__1__i]) >= 0 .
H7:    element(data_set, [loop__1__i]) <= 100 .
H8:    result + element(data_set, [loop__1__i]) >= integer__first .
H9:    result + element(data_set, [loop__1__i]) <= integer__last .
H10:   not (loop__1__i = index_type__last) .
        ->
C1:    result + element(data_set, [loop__1__i]) >= 0 .
C2:    result + element(data_set, [loop__1__i]) <= (loop__1__i + 1) * 100 .
```

A proof of C2 now follows by coloured rippling. The required wave-rules are:

$$\boxed{(X+Y)}^{\uparrow} * Z \;\Rightarrow\; \boxed{(X*Z)+(Y*Z)}^{\uparrow} \tag{1}$$

$$\boxed{1*X}^{\uparrow} \;\Rightarrow\; X \tag{2}$$

$$\boxed{(W+X)}^{\uparrow} \leq \boxed{(Y+Z)}^{\uparrow} \;\Rightarrow\; \boxed{W \leq Y \wedge X \leq Z}^{\uparrow} \tag{3}$$

Using these wave-rules a ripple proof can be constructed as follows:

$$\begin{array}{llcl}
\text{Givens:} & r & \leq & i * 100 \\
 & ele(data_set, i) & \leq & 100 \\
\text{Goal:} & \boxed{r + ele(data_set, i)}^{\uparrow} & \leq & \boxed{(i+1)}^{\uparrow} * 100 \\
 & & & \qquad\qquad \text{by } (1) \\
 & \boxed{r + ele(data_set, i)}^{\uparrow} & \leq & \boxed{(i*100) + (1*100)}^{\uparrow} \\
 & & & \qquad\qquad \text{by } (2) \\
 & \boxed{r + ele(data_set, i)}^{\uparrow} & \leq & \boxed{(i*100) + 100}^{\uparrow} \\
 & & & \qquad\qquad \text{by } (3) \\
 & \multicolumn{3}{c}{\boxed{r \leq (i*100) \wedge ele(data_set, i) \leq 100}^{\uparrow}}
\end{array}$$

Fertilization against both the given hypotheses completes the proof.

6 What next?

There are a number of tasks that need to be achieved if these ideas are to be realized within NuSPADE:

- More example VCs are required in order to validate and develop the ideas presented here.
- A mechanism for inducing meta-data directly from SPARK code is required.
- Coloured rippling needs to be ported for use within NuSPADE.
- Proof generation based upon strongest postcondition predicate transformation is required.
- A method of generalization is required based upon identifying repeated patterns within a series of proofs.
- A new generalization critic is required that implements the style of patching presented above.

The method of generalization will be of crucial importance. Work on inductive learning techniques should therefore be investigated. In relation to proof in particular, work on generalization within the context of the constructive omega rule looks to be very relevant [2, 1].

7 Conclusion

I have shown how the partial application of coloured rippling can be used to predict the need for a generalization. Moreover, coloured rippling can be used to partially specify the structure of the generalization. This is new, previous work on proof patching [3, 4, 5] has not considered coloured rippling. Another departure from previous work is the approach of using specific proofs in completing the generalization. From an implementation perspective, I see this approach as more practical than the traditional middle-out reasoning approach. Finally, the key to taking these ideas forward lies in getting access to more RTC VCs.

References

[1] S. Baker. *Aspects of the Constructive Omega Rule within Automated Deduction*. PhD thesis, Edinburgh, 1993.

[2] S. Baker, A. Ireland, and A. Smaill. On the use of the constructive omega rule within automated deduction. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning — LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 214–225. Springer-Verlag, 1992.

[3] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996. Also available as DAI Research Paper No 716, Dept. of Artificial Intelligence, Edinburgh.

[4] A. Ireland and A. Bundy. Automatic Verification of Functions with Accumulating Parameters. *Journal of Functional Programming: Special Issue on Theorem Proving & Functional Programming*, 9(2):225–245, March 1999. A longer version is available from Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/97/11.

[5] A. Ireland and J. Stark. Proof planning for strategy development. *Annals of Mathematics and Artificial Intelligence*, 29(1-4):65–97, February 2001. An earlier version is available as Research Memo RM/00/3, Dept. of Computing and Electrical Engineering, Heriot-Watt University.

[6] Tetsuya Yoshida, Alan Bundy, Ian Green, Toby Walsh, and David Basin. Coloured rippling: An extension of a theorem proving heuristic. In A. G. Cohn, editor, *Proceedings of ECAI-94*, pages 85–89. John Wiley, 1994.