Proof Pearl: The Termination Method of TERMINATOR

Joe Hurd
Computing Laboratory
University of Oxford

Galois Connections
Tuesday 27 February 2007
<table>
<thead>
<tr>
<th></th>
<th>Talk Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Formalizing TERMINATOR</td>
</tr>
<tr>
<td>3</td>
<td>Correctness Proof</td>
</tr>
<tr>
<td>4</td>
<td>Verifying Optimizations</td>
</tr>
<tr>
<td>5</td>
<td>Summary</td>
</tr>
</tbody>
</table>
“[...] Vista is the most secure operating system we’ve ever done, and if it’s administered properly, absolutely, it can be used to run a hospital or any kind of mission critical thing.” Bill Gates, 1 Feb 2007
TERMINATOR

- If a Windows device driver goes into an infinite loop, the whole computer can hang.
- TERMINATOR is a static analysis tool developed by Microsoft Research to prove termination of device drivers, typically thousands of lines of C code.
- It works by modifying the program to transform the termination problem into a safety property, which is then proved by the SLAM tool.
Transforming Termination to a Safety Property

Given a program location $l$ and well-founded relations $R_1, \ldots, R_n$ between program states at location $l$, insert

$$\text{already_saved_state} := \text{false};$$

at the start of the program, and the following code just before $l$:

```c
if (already_saved_state) {
    if $\neg (R_1 \text{ state saved_state} \lor \cdots \lor R_n \text{ state saved_state})$ {
        error("possible non-termination");
    }
}
else if (*) {
    saved_state := state;
    already_saved_state := true;
}
```
Proving the Safety Property

- SLAM is called to verify that the error statement is never executed.
- This guarantees that between the $i$th and $j$th time that program location $l$ is reached, the state goes down in at least one of $R_1, \ldots, R_n$.
- E.g., suppose $R_1$ is red, $R_2$ is blue and $R_3$ is green:

  ![Diagram of program locations with red, blue, and green paths]

- If this is true at all program locations it is possible to conclude that the program must always terminate. This Proof Pearl!
Constructing Well-Founded Relations

- The choice of well-founded relations is irrelevant for the correctness proof.
- TERMINATOR first calls SLAM with no relations.
  - This proof will succeed if the program location is executed at most once.
- If the proof fails, SLAM will provide a counterexample program trace.
- An external tool heuristically synthesizes a well-founded relation that would eliminate the counterexample trace.
- This is added to the set of relations, and SLAM is called again.
unsigned int A (unsigned int m, unsigned int n) {
    /* Ackermann’s function
     [Zum Hilbertschen Aufbau der reellen Zahlen, 1928] */
    if (m == 0) { return n + 1; }
    else if (n == 0) { return A (m - 1, 1); }
    else { return A (m - 1, A (m, n - 1)); }
}
unsigned int A (unsigned int m, unsigned int n) {
    /* No relations */
    if (m == 0) {
        return n + 1;
    }
    else if (n == 0) {
        return A (m - 1, 1);
    }
    else {
        return A (m - 1, A (m, n - 1));
    }
}

SLAM Says: Counterexample trace (1, 0) → (0, 1)
Relation Synthesizer Says: $R(m', n')(m, n) \equiv m' < m$
TERMINATOR Example (III)

Code

```c
unsigned int A (unsigned int m, unsigned int n) {
   /* R_0 (m', n') (m, n) \equiv m' < m */
   if (m == 0) {
      return n + 1;
   } else if (n == 0) {
      return A (m - 1, 1);
   } else {
      return A (m - 1, A (m, n - 1));
   }
}
```

SLAM Says: Counterexample trace \((1, 1) \rightarrow (1, 0)\)
Relation Synthesizer Says: \(R (m', n') (m, n) \equiv n' < n\)
unsigned int A (unsigned int m, unsigned int n) {
    /* \( R_0 (m', n')(m, n) \equiv m' < m \) 
    \( R_1 (m', n')(m, n) \equiv n' < n \) */
    if (m == 0) { return n + 1; }
    else if (n == 0) { return A (m - 1, 1); }
    else { return A (m - 1, A (m, n - 1)); }
}
Model programs as a state transition system with an explicit program counter.

**Type Definition**

\[
('\text{state, location}) \text{ program} \equiv \left< \begin{array}{l}
\text{states} : '\text{state} \rightarrow \text{bool}; \\
\text{location} : '\text{state} \rightarrow '\text{location}; \\
\text{initial} : '\text{state} \rightarrow \text{bool}; \\
\text{transition} : '\text{state} \rightarrow '\text{state} \rightarrow \text{bool} \end{array} \right>
\]
Well-Formed Programs

Well-formed programs have a finite text and stay within their state space.

Constant Definition

\[
\text{programs} \equiv \\
\{ \ p \mid \\
\text{finite (locations } p) \land \\
p.\text{initial} \subseteq p.\text{states} \land \\
\forall s, s'. \ p.\text{transition } s \ s' \implies s \in p.\text{states} \land s' \in p.\text{states} \} \\
\]

where locations \( p \equiv \text{image } p.\text{location} \ p.\text{states} \).
Terminating Programs

Define the set of program traces.

\[
\text{traces } p \equiv \{ t \mid t_0 \in p.\text{initial} \land \forall i. \ p.\text{transition} \ t_i \ t_{i+1} \}
\]

Terminating programs have no infinite traces.

\[
\text{terminates } p \equiv \forall t \in \text{traces } p. \text{ finite } t
\]
The TERMINATOR Program Analysis (I)

Constant Definition

\[
\text{terminator\_property\_at\_location } p / l \equiv \\
\exists R, n. \\
(\forall k \in \{0, \ldots, n-1\}. \text{well\_founded } (R k)) \land \\
\forall t \in \text{traces } p. \forall x_i < x_j \in \text{trace\_at\_location } p / t. \\
\exists k \in \{0, \ldots, n-1\}. R k x_j x_i
\]

where trace\_at\_location } p / t \equiv \text{filter } (\lambda s. p.\text{location } s = l) t.
The TERMINATOR Program Analysis (II)

Constant Definition

\[
\text{terminator\_property } p \equiv \\
\forall l \in \text{locations } p. \text{ terminator\_property\_at\_location } p \ l
\]
Recall the example with three well-founded relations, where $R_1$ is red, $R_2$ is blue and $R_3$ is green:

Why should such a trace necessarily be finite?

**Answer:** Find a subtrace where all states are connected by a single well-founded relation.
Ramsey Theory To The Rescue

- Named for Frank Plumpton Ramsey (1903–1930).
  - A Cambridge mathematician who worked in logic, economics and probability.
  - He was Wittgenstein’s Ph.D. supervisor!
- Ramsey theory is about “finding order in chaos”.
  - Ramsey created his theorem to prove a result in logic.
    [On a problem of formal logic, 1930]
  - It has been extended to many applications, e.g., high dimensional noughts and crosses.
  - Paul Erdős used Ramsey Theory to tempt promising young mathematicians into studying combinatorics.
Ramsey’s Theorem (Infinite Graph Version)

Every infinite graph has an infinite subgraph that is either complete or empty:

\[ \vdash \forall V, E. \]

\[ \{ \text{infinite } V \Rightarrow \exists M \subseteq V. \} \]

\[ \{ \text{infinite } M \land (\forall i, j \in M. i < j \Rightarrow E i j) \lor (\forall i, j \in M. i < j \Rightarrow \neg E i j) \} \]
Ramsey’s Theorem (Infinite Version)

Every complete infinite graph coloured with finitely many colours has an infinite monochromatic subgraph:

\[ \forall V, C, n. \]
\[ \text{infinite } V \land \\
(\forall i, j \in V. \exists k \in \{0, ..., n - 1\}. i < j \implies C_{k i j}) \implies \\
\exists M \subseteq V. \exists k \in \{0, ..., n - 1\}. \\
\text{infinite } M \land \forall i, j \in M. i < j \implies C_{k i j} \]

Proof.

Put on your turquoise spectacles.
At a program location $p$, colour the edge $i < j$ with colour $k$ if $R k x_j x_i$.

Theorem

\[ \vdash \forall p \in \text{programs}. \ \forall l \in \text{locations} \ p. \ \ \ \ \ \text{terminator\_property\_at\_location} \ p \ l \ \ \Rightarrow \ \ \ \ \ \forall t \in \text{traces} \ p. \ \text{finite} \ (\text{trace\_at\_location} \ p \ l \ t) \]

Proof.

Ramsey’s Theorem.
Any infinite program trace will visit some program location infinitely often, so deduce the correctness of TERMINATOR.

**Theorem**

\[ \forall p \in \text{programs}. \ terminating\_property\ p \implies \ terminates\ p \]

**Proof.**

By colouring states on the program trace with their location, this result can be seen as a 1-dimensional Ramsey theorem.
If there is only one relation TERMINATOR modifies the program to simply compare states with previous states, by inserting

\[
\text{already\_saved\_state} := \text{false};
\]

at the start of the program, and the following code just before `/`:

```
if (already\_saved\_state \land \neg R \text{ state saved\_state}) {
    error("possible non-termination");
}

\text{saved\_state} := \text{state};
\text{already\_saved\_state} := \text{true};
```
Optimization 1: Single Relation (II)

To account for this optimization, the result of the TERMINATOR program analysis must be weakened to:

**Constant Definition**

\[
\text{terminator\_property\_at\_location } pl \equiv \\
(\exists R. \\
\quad \text{well\_founded } R \land \\
\quad \forall t \in \text{traces } p. \forall x_i, x_{i+1} \in \text{trace\_at\_location } pl t. R x_{i+1} x_i) \lor \\
[\ldots \text{old definition of } \text{terminator\_property\_at\_location } pl \ldots]
\]
Optimization 2: Cut Sets

TERMINATOR finds well-founded relations only at a cut set of program locations.

Constant Definition

\[
\text{cut\_sets } p \equiv \\
\{ L \mid L \subseteq \text{locations } p \land \\
\forall t \in \text{traces } p. \\
\text{infinite } t \implies \exists l \in L. \text{infinite (trace\_at\_location } p \ l \ t)\}
\]

- Being a cut set is a semantic property, and as hard to prove as termination.
- In practice, choose a set containing locations at the start of all loops and functions that are called (mutually) recursively.
Optimized TERMINATOR Program Analysis

The optimized TERMINATOR program analysis guarantees:

**Constant Definition**

\[
\text{terminator\_property } p \equiv \\
\exists C \in \text{cut\_sets } p. \forall l \in C. \text{terminator\_property\_at\_location } p l
\]

But the same correctness theorem is still true.

**Theorem**

\[
\vdash \forall p \in \text{programs}. \text{terminator\_property } p \implies \text{terminates } p
\]
Summary

- This talk has presented a formal verification of the termination argument relied on by TERMINATOR.
- The model of programs used is the simplest one that can verify the termination argument.
- The next step would be to add some program structure:
  - the initial program transformation could be represented;
  - cut sets could be defined syntactically; and
  - more TERMINATOR optimizations could be verified.