Proof Pearl: The Termination Analysis of TERMINATOR

Joe Hurd

Computing Laboratory
University of Oxford

TPHOLs 2007
Monday 10 September 2007

Talk Plan

1. Introduction
2. Formalizing TERMINATOR
3. Correctness Proof
4. Verifying Optimizations
5. Summary

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

```
already_saved_state := false;
```

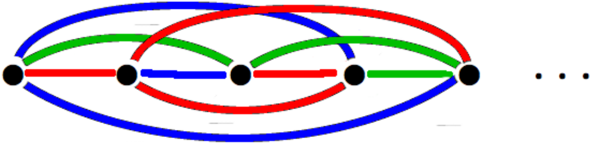
at the start of the program, and the following code just before l:

Code

```
if (already_saved_state) {
  if ¬(R_1 state saved_state ∨ ··· ∨ R_n 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:



- 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.

TERMINATOR Example (I)

Code

```
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)); }
}
```

TERMINATOR Example (II)

Code

```
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 $(1, 0) \to (0, 1)$
Relation Synthesizer Says: $R\ (m', n')\ (m, n)\ \equiv\ m' < m$

TERMINATOR Example (III)

Code

```
unsigned int A (unsigned int m, unsigned int n) {
  /* R0 (m', n') (m, n)  ≡  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 $(1, 1) \to (1, 0)$
Relation Synthesizer Says: $R\ (m', n')\ (m, n) \ \equiv\ n' < n$

TERMINATOR Example (IV)

Code

```
unsigned int A (unsigned int m, unsigned int n) {
  /* R0 (m', n') (m, n)  ≡  m' < m
     R1 (m', n') (m, n)  ≡  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)); }
}
```

SLAM Says: Proved
TERMINATOR Says: Terminating

Programs

Model programs as a state transition system with an explicit program counter.

Type Definition

$$
\begin{aligned}
&(\textit{'state}, \textit{'location})\ \textit{program} \quad \equiv \\
&\quad <|\ \text{states} : \textit{'state} \rightarrow \textit{bool}; \\
&\quad\quad \text{location} : \textit{'state} \rightarrow \textit{'location}; \\
&\quad\quad \text{initial} : \textit{'state} \rightarrow \textit{bool}; \\
&\quad\quad \text{transition} : \textit{'state} \rightarrow \textit{'state} \rightarrow \textit{bool}\ |>
\end{aligned}
$$

Well-Formed Programs

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

Constant Definition

$$
\begin{aligned}
&\text{programs} \equiv \\
&\quad \{\ p \mid \\
&\quad\quad \text{finite (locations } p) \ \land \\
&\quad\quad p.\text{initial} \subseteq p.\text{states} \ \land \\
&\quad\quad \forall s, s'.\ p.\text{transition } s\ s' \implies s \in p.\text{states} \land s' \in p.\text{states}\ \}
\end{aligned}
$$

where locations $p \equiv$ image p.location p.states.

Terminating Programs

Define the set of program traces.

Constant Definition

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

Terminating programs have no infinite traces.

Constant Definition

$$\text{terminates}\ p \ \equiv\ \forall t \in \text{traces}\ p.\ \text{finite}\ t$$

The TERMINATOR Program Analysis (I)

Constant Definition

$$
\begin{array}{l}
\text{terminator\_property\_at\_location } p\ l \ \equiv \\
\quad \exists R, n. \\
\quad\quad (\forall k \in \{0, \ldots, n-1\}.\ \text{well\_founded } (R\ k)) \ \wedge \\
\quad\quad \forall t \in \text{traces } p.\ \forall x_i < x_j \in \text{trace\_at\_location } p\ l\ t. \\
\quad\quad\quad \exists k \in \{0, \ldots, n-1\}.\ R\ k\ x_j\ x_i
\end{array}
$$

where trace_at_location $p\ l\ t \ \equiv$ filter $(\lambda s.\ p.\text{location } s = l)\ t$.

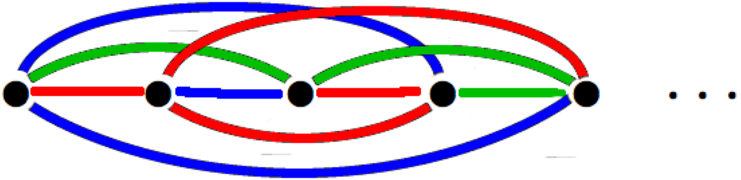
The TERMINATOR Program Analysis (II)

Constant Definition

$$\text{terminator\_property}\ p \ \equiv$$
$$\forall l \in \text{locations}\ p.\ \text{terminator\_property\_at\_location}\ p\ l$$

Deducing Termination

Recall the example with three well-founded relations,
where R_1 is , R_2 is  and R_3 is :



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:

Theorem

$$
\begin{array}{l}
\vdash \quad \forall V, E. \\
\qquad \text{infinite } V \implies \\
\qquad \exists M \subseteq V. \\
\qquad\quad \text{infinite } M \;\wedge \\
\qquad\quad ((\forall i, j \in M.\ i < j \implies E\ i\ j) \;\vee \\
\qquad\quad\ (\forall i, j \in M.\ i < j \implies \neg E\ i\ j))
\end{array}
$$

Ramsey's Theorem (Infinite Version)

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

Theorem

$$\begin{array}{l}\vdash \ \forall V, C, n. \\ \quad \text{infinite } V \ \wedge \\ \quad (\forall i, j \in V.\ \exists k \in \{0, \ldots, n-1\}.\ i < j \implies C\ k\ i\ j) \implies \\ \quad \exists M \subseteq V.\ \exists k \in \{0, \ldots, n-1\}. \\ \qquad \text{infinite } M \ \wedge \ \forall i, j \in M.\ i < j \implies C\ k\ i\ j)\end{array}$$

Proof.

Put on your turquoise spectacles. □

Verifying TERMINATOR (I)

At a program location p, colour the edge $i < j$ with colour k if $R\ k\ x_j\ x_i$.

Theorem

$$
\begin{aligned}
\vdash\quad & \forall p \in \text{programs}.\ \forall l \in \text{locations}\ p. \\
& \quad \text{terminator\_property\_at\_location}\ p\ l \implies \\
& \quad \forall t \in \text{traces}\ p.\ \text{finite}\ (\text{trace\_at\_location}\ p\ l\ t)
\end{aligned}
$$

Proof.

Ramsey's Theorem. □

Verifying TERMINATOR (II)

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

Theorem

$$\vdash \quad \forall p \in \text{programs. terminator\_property } p \implies \text{terminates } p$$

Proof.

By colouring states on the program trace with their location, this result can be seen as a 1-dimensional Ramsey theorem. □

Optimization 1: Single Relation (I)

If there is only one relation TERMINATOR modifies the program to simply compare states with previous states, by inserting

```
already_saved_state := false;
```

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

Code

```
if (already_saved_state ∧ ¬R state saved_state) {
  error("possible non-termination");
}
saved_state := state;
already_saved_state := true;
```

Optimization 1: Single Relation (II)

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

Constant Definition

terminator_property_at_location $p\ l\ \equiv$

$(\exists R.$

well_founded $R\ \wedge$

$\forall t \in$ traces $p.\ \forall x_i, x_{i+1} \in$ trace_at_location $p\ l\ t.\ R\ x_{i+1}\ x_i)\ \vee$

[...old definition of terminator_property_at_location $p\ l$*...]*

Optimization 2: Cut Sets

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

Constant Definition

$$\begin{aligned}&\text{cut\_sets } p \equiv \\ &\quad \{\, L \mid L \subseteq \text{locations } p \;\wedge \\ &\qquad \forall t \in \text{traces } p. \\ &\qquad\quad \text{infinite } t \implies \exists l \in L.\ \text{infinite } (\text{trace\_at\_location } p\ l\ t) \}\end{aligned}$$

- 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.