Proving Non-Termination

- Ashutosh K. Gupta
  - MPI-SWS
- Thomas A. Henzinger
  - EPFL
- Rupak Majumdar
  - UCLA
- Andrey Rybalchenko
  - MPI-SWS
- Ru-Gang Xu
  - UCLA

Non-Termination Errors

Non-Terminating

Proving Non-Termination

- Search for infinite executions
- Needs effective (finitary) representation

Failed Termination Proof vs. Non-Termination

- Successful termination provers:
  - PolyRank, Terminator, ACL2, TerminWeb, AProVE, ...
- Inherently incomplete algorithms
- Failed termination proof ≠ non-termination

TNT: Testing Non-Termination

Tool for proving non-termination

- Input leading to non-termination
- Don’t know

C program ➔ TNT ➔ Proved Non-Terminating

- Terminating
- Non-Terminating
Outline

• Example:
  – Non-termination error in a memory protection system

• TNT algorithm:
  – Lasso search
  – Recurrent set computation

Example: Non-termination Error

• Input to TNT:
  – Mondriaan memory protection system
    (early version courtesy: E. Witchel)
  – Uses recursion for basic operation
  – Termination required

• Output by TNT:
  – Non-termination bug (now fixed) in _mmpt_insert procedure:
    cyclic sequences of calls to _mmpt_insert (.., 0, 3, .., TAB_ROOT, 0, ..)

Non-Termination in _mmpt_insert

```c
void _mmpt_insert (struct* mmpt, base, len, proto, TAB_ROOT, tab, level, ..) {
  if(len == 0) return; // Exit condition
  int id = make_idx(mmpt, base, level);
  if(level < 2 && tab[id] && !uentry_is_data(mmpt, tab[id])) {
    _mmpt_insert(mmpt, base, len, proto, (tab_t*)tab[id], level + 1, ..);
  } else if(level < 2 && tab[id]) {
    _mmpt_insert(mmpt, base, len, proto, mmpt->tab, 0, ..);
  }
}
```
Non-Termination in \_mmpt\_insert
(second call)

```c
void \_mmpt\_insert (struct* mmpt, 0, 3, prot, TAB_MID , 1, ..) {
  if( 3 == 0) return; // Exit condition
  int 0 = make_idx(mmpt, 0, 3);
  if( 1 < 2 as .. as 3 >= 1) {
  } else if( 1 < 2 as tab[ 0 ] && !uentry_is_data(mmpt, tab[ 0 ])) {
    \_mmpt\_insert (mmpt, 0, 3, prot, (tab_t*)tab[ 0 ], 1 + 1, ..);
  } else if( level < 2 && ..) {
  } else {
    for(; len >= subblock_len(mmpt, level) && ..;
     ..;
    ..;
  }
  \_mmpt\_insert (mmpt, base, len, prot, mmpt->tab, 0, ..);
}
```

Non-Termination in \_mmpt\_insert
(third call)

```c
void \_mmpt\_insert (struct* mmpt, 0, 3, prot, TAB_LEAF , 2, ..) {
  if(len == 0) return; // Exit condition
  int idx = make_idx(mmpt, base, level);
  if( level < 2 && .. && len >= tab_len(mmpt, level + 1)) {
  } else if( level < 2 && tab[idx] && !uentry_is_data(mmpt, tab[idx])) {
    \_mmpt\_insert (mmpt, base, len, prot, (tab_t*)tab[idx], level + 1, ..);
  } else if( level < 2 && ..) {
  } else {
    for(; len >= subblock_len(mmpt, level) && ..;
     ..;
    ..;
  }
  \_mmpt\_insert (mmpt, base, len, prot, mmpt->tab, 0, ..);
}
```

Outline:

- Search for lassos
- Prove a lasso is non-terminating through recurrent sets

Path = seq. of statements
Execution = seq. of states

What does TNT do?

- TNT finds a cyclic sequence of calls to \_mmpt\_insert
- The sequence is lasso – shaped
  - Non-termination is proved by analyzing the lasso
    - same valuation of input parameters after the call cycle

Paths and Executions
What do infinite paths look like?

- General paths: e.g. stmtA stmtB stmtC stmtB stmtC ...
- Periodic paths or Lassos: e.g. stmtA stmtB stmtC stmtB stmtC ...
- TNT only considers periodic paths

Lassos

- Pair of paths: (stem, loop)
- Represents infinite periodic path: \(\text{stem}(\text{loop})^\omega\)

Lasso Search

- Find lassos by symbolic execution
- Implementation similar to DART, CUTE, SAGE, ...

TNT Algorithm

- Two-step algorithm:
  1. Search for feasible lassos
     - Uses symbolic execution
     - Quickly find candidates for non-termination
  2. Check each lasso for non-termination
     - Uses SAT and constraint solving
     - Precise reasoning on small program fragments

Outline:

- Lasso search
- Recurrent set computation

Recurrent Sets

- Proves non-termination using inductive argument
- Set of states \(\text{RecSet}\) is recurrent for relation \(p(x, x')\) if:
  - non-empty
  - some successor of each state is in \(\text{RecSet}\)

Theorem:
\(p(x, x')\) is non-terminating iff there exists \(\text{RecSet}\)
Recurrent Sets for Lassos

- \( \text{RecSet} \) is reachable via stem
- \( \text{RecSet} \) is recurrent for loop

Recurrent Sets to Constraints

- Lasso is non-terminating iff there exists \( \text{RecSet} \) such that:
  \[
  \exists x, x' \ ( x' \in \text{RecSet}(x) ) \land ( x \in \text{Stem}(x, x') )
  \]
  Non-empty
  \[
  \forall x \exists x' \ ( x' \in \text{RecSet}(x) ) \land ( x \in \text{Loop}(x, x') )
  \]
  Looping

Computing Recurrent Sets

- Bit-vectors:
  - Implementation level
  - Bit-precise
  - Full C expressions
- Numbers:
  - Algorithmic level (more abstract)
  - Unbounded integers/rationals
  - Linear arithmetic only

Recurrent Sets over Bit-vectors

- Some state has to appear infinitely often:

Non-Termination with Singleton Recurrent Sets

- Non-termination reduces to
  \[
  \exists x_0, x X \ ( x_0 \in \text{RecSet}(x) ) \land ( x_0 \neq x )
  \]

Broken Binary Search

- Joshua Bloch’s blog presents a memory error
- Precondition (signed numbers):
  \( \text{lo} = 1, \text{hi} = \text{MAXINT} \)
- TNT discovers non-termination bug
- Precondition (unsigned numbers):
  \( \text{lo} = 1, \text{hi} = \text{MAXINT}, a[0] < k \)
Computing Recurrent Sets

Bit-vectors:
- Implementation level
- Bit-precise
- Full C expressions

Numbers:
- Algorithmic level (more abstract)
- Unbounded integers/rationals
- Linear arithmetic only

Recurrence Sets over Numbers
- Apply template-based technique from:
  - Invariant generation (Colon et al. 2003)
  - Abstract interpretation
    (Sankaranarayanan et al. 2006, Gulwani et al. 2008)
- RecSet is a conjunction of linear inequalities

Parameters:
- $p_x, p_y, p$
- $q_x, q_y, q$
- Program variables: $x, y$

Possible instantiations:
- $1^x + 0^y <= 2$
- $2^x + 4^y <= 0$
- $6^x + 5^y <= 3$

Template:
- $p_x x + p_y y <= p$
- $q_x x + q_y y <= q$

Non-termination Error in Mondriaan
- Input to TNT:
  - Mondriaan memory protection system (early version courtesy: E. Witchel)
    - Uses recursion for updates to permissions table
- Output by TNT:
  - Non-termination bug (now fixed) in _mmpt_insert procedure:
    - Cyclic sequences of calls to _mmpt_insert

Eventually...
- Detect and prove non-termination at runtime

Conclusion
- 2-step algorithm for proving non-termination
  - Lasso search
  - Recurrent set computation
- TNT
  - Symbolic execution
  - Constraint-based computation of recurrent sets