Calling Context Graphs

Pete Manolios
Northeastern

Formal Methods, Lecture 8                                                   October 2008
Termination Analysis

- Quintessential undecidable software verification problem
- Calling Context Graphs and Measures
- Fully automatic termination analysis
- Domain: fully-featured, first-order, applicative functional PLs
- Combination of static analysis and theorem proving
- Implemented in ACL2s
- Experimental results: ACL2 regression suite
  - > 100MB of code (JVM, linear algebra, model checker, ...)
  - > 10,000 function definitions
  - Over a decade of submissions from worldwide user base
  - 98.7% success rate
The Idea

- Non-termination in our domain iff
  \[ \exists \text{ some input that leads to an infinite sequence of function calls} \]
- Goal: Conservative, precise, analyzable abstraction
- First attempt: use call graphs
- Example:

\[
\text{define } f(x) = \\
\quad \text{if } (\neg \text{intp}(x) \text{ or } x \leq 1) \\
\quad \quad \text{then } 0 \\
\quad \text{else if } (x \mod 2 = 1) \\
\quad \quad \text{then } f(x+1) \\
\quad \quad \text{else } 1 + f(x/2)
\]

Not enough
Governors

- The governors of \( e' \subseteq e \) are the branching conditions that need to be true for execution of \( e \) to lead to the execution of \( e' \)

- Example

\[
\text{define } f(x) = \\
\quad \text{if} \ (\text{!intp}(x) \text{ or } x \leq 1) \\
\quad \quad \text{then } 0 \\
\quad \text{else if} \ (x \text{ mod } 2 = 1) \\
\quad \quad \text{then } f(x+1) \\
\quad \text{else } 1 + f(x/2)
\]
**Governors**

- The governors of $e' \subseteq e$ are the branching conditions that need to be true for execution of $e$ to lead to the execution of $e'$

- Example
  
  define $f(x) =$
  
  if (!intp(x) or $x \leq 1$)
    then 0
  
  else if ($x \text{ mod } 2 = 1$)
    then $f(x+1)$
  
  else $1 + f(x/2)$

- Governors for $f(x+1)$: \{intp(x), $x > 1$, $x \text{ mod } 2 = 1$\}
Governors

- The governors of $e' \subseteq e$ are the branching conditions that need to be true for execution of $e$ to lead to the execution of $e'$
- Example
  
  define $f(x) =$
  
  if $(!\text{intp}(x) \text{ or } x \leq 1)$
  
  then 0
  
  else if $(x \mod 2 = 1)$
  
  then $f(x+1)$
  
  else $1 + f(x/2)$

- Governors for $f(x+1)$: \{intp(x), $x > 1$, $x \mod 2 = 1$\}
- Governors for $f(x/2)$: \{intp(x), $x > 1$, $x \mod 2 \neq 1$\}
Precise Calling Contexts

A precise calling context for a call $e$ is a triple containing:
- The name of the function containing $e$
- The governors for $e$ in the function body
- The call, $e$

Example: define $f(x) =$

```
  if (!intp(x) or x \leq 1)
    then 0
  else if (x mod 2 = 1)
    then $f(x+1)$
  else 1 + $f(x/2)$
```

1. \langle f, \{intp(x), x > 1, x \ mod \ 2 = 1\}, f(x+1) \rangle
2. \langle f, \{intp(x), x > 1, x \ mod \ 2 \neq 1\}, f(x/2) \rangle
Example: define \( f(x) = \)
\[
\text{if } (!\text{intp}(x) \text{ or } x \leq 1) \\
\text{then } 0 \\
\text{else if } (x \mod 2 = 1) \\
\text{then } f(x+1) \\
\text{else } 1 + f(x/2)
\]

1. \( \langle f, \{\text{intp}(x), x > 1, x \mod 2 = 1\}, f(x+1) \rangle \)
2. \( \langle f, \{\text{intp}(x), x > 1, x \mod 2 \neq 1\}, f(x/2) \rangle \)

- Vertices are calling contexts
- An edge from \( c_1 \) to \( c_2 \) if it is possible for execution to reach \( c_1 \) and \( c_2 \) in consecutive recursive calls
- Set of all paths is an overapproximation of recursive behavior
Building Calling Context Graphs

- The edge condition:
  - $\exists$ values that satisfy the governors of both contexts
- Governors are arbitrary predicates
- Building a minimal CCG is undecidable
- Theorem prover queries used to eliminate unnecessary edges
- Edge included if theorem prover cannot disprove the edge condition
- We need more
Calling Context Measures

- Map function formals into some well-founded structure
- Each calling context is given a set of CCMs
- Example:

1. \( \langle f, \{\text{intp}(x), x > 1, x \ mod \ 2 = 1\}, f(x+1) \rangle \)
2. \( \langle f, \{\text{intp}(x), x > 1, x \ mod \ 2 \neq 1\}, f(x/2) \rangle \)

![Diagram showing non-increasing and decreasing edges]

- Decreasing edge \( >: \)
- Non-increasing edge \( \geq: \)
- Neither \( \neq: \)
The Termination Condition

- For every infinite path through the CCG,
- There should exist a corresponding sequence of CCMs,
- Such that for some tail of the sequence, each adjacent pair of CCMs is never increasing and infinitely decreasing
- Solved by Size Change back-end algorithm
define \( f(x) = \)

\[
\begin{align*}
&\text{if } (!\text{intp}(x) \text{ or } x \leq 1) \\
&\quad \text{then } 0 \\
&\quad \text{else if } (x \mod 2 = 1) \\
&\quad \quad \text{then } f(x+1) \\
&\quad \quad \text{else } 1 + f(x/2)
\end{align*}
\]

1. \( \langle f, \{\text{intp}(x), x > 1, x \mod 2 = 1\}, f(x+1) \rangle \)

2. \( \langle f, \{\text{intp}(x), x > 1, x \mod 2 \neq 1\}, f(x/2) \rangle \)
define f(x) =
  if (!intp(x) or x ≤ 1)
    then 0
  else if (x mod 2 = 1)
    then f(x+1)
  else 1 + f(x/2)

1. ⟨f, {intp(x), x > 1, x mod 2 = 1, intp(x+1), x+1 > 1, (x+1) mod 2 ≠ 1}, f((x+1)/2)⟩
2. ⟨f, {intp(x), x > 1, x mod 2 ≠ 1}, f(x/2)⟩
Ackermann’s Function

define ack (a, b) =
    if (!intp(a) or a ≤ 0)
        then 1
        else if (!intp(b) or b ≤ 0)
            then if a=1
                then 2
                else a+2
            else ack(ack(a-1, b), b-1)
    else ack(ack(a-1, b), b-1)

1. 〈ack, \{intp(a), 0 < a, intp(b), 0 < b\}, ack(a-1, b)〉

2. 〈ack, \{intp(a), 0 < a, intp(b), 0 < b\}, ack(ack(a-1, b), b-1)〉
We presented a simplified version of our analysis

- Mutual Recursion
- SCC analysis: can have more SCCs than functions
- Hierarchical Analysis
- Merging on a per-node basis: different edges correspond to different numbers of steps
- Multiple CCMs
- CCMs that combine formals (e.g., x - y)
- Different CCMs for different nodes
  - Saturation algorithm for propagating CCMs
JVM Example

(mutual-recursion
;; mma2 :: num, counts, s, ac --> [refs]
(defun mma2 (c1 c2 s ac)
  (declare (xargs :measure (cons (len (cons c1 c2))
                                (natural-sum (cons c1 c2)))))
  (if (zp c1)
    (mv (heap s) ac)
    (mv-let (new-addr new-heap)
      (mma c2 s)
      (mma2 (- c1 1) c2
        (make-state (thread-table s)
                    new-heap
                    (class-table s))
        (cons (list 'REF new-addr) ac)))))

;; mma :: [counts], s --> addr, new-heap
(defun mma (counts s)
  (declare (xargs :measure (cons (+ 1 (len counts))
                                (natural-sum counts)))
  (if (<= (len counts) 1)
    ;; "Base case" Handles initializing the final dimension
    (mv (len (heap s))
      (bind (len (heap s))
        (makearray 'T_REF
                    (car counts)
                    (init-array 'T_REF (car counts))
                    (class-table s))
        (heap s)))
    ;; "Recursive Case"
    (mv-let (heap-prime lst-of-refs)
      (mma2 (car counts)
            (cdr counts)
            s
data)
      (let* ((obj (makearray 'T_REF
                           (car counts)
                           lst-of-refs
                           (class-table s)))
             (new-addr (len heap-prime))
             (new-heap (bind new-addr obj heap-prime))
             (mv new-addr new-heap))))))
(mutual-recursion
  (defun mma2 (c1 c2 s ac)
    (if (zp c1)
      ...  
      (mv-let (new-addr new-heap)
        (mma c2 s)
        (mma2 (- c1 1) c2 e e'))) )

  (defun mma (c s)
    (if (<= (len c) 1)
      ... 
      (mma2 (car c) (cdr c) s nil)
      ... ))))

mma2 measure: (cons (len (cons c1 c2))
  (natural-sum (cons c1 c2)))
mma measure: (cons (+ 1 (len c)) (natural-sum c))
Experimental Results

- Implemented in ACL2s
- Ran our algorithm over the ACL2 Regression Suite
  - Over 100MB, 11,000 function definitions
- Ignored all user hints and other explicit assistance
- Over 98% success rate

Experiment: No Theorems

<table>
<thead>
<tr>
<th>Problems</th>
<th>Total</th>
<th>CCG (%)</th>
<th>ACL2 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Trivial</td>
<td>1762</td>
<td>1408 (80%)</td>
<td>1056 (60%)</td>
</tr>
<tr>
<td>Recursive</td>
<td>4348</td>
<td>3394 (92%)</td>
<td>3642 (84%)</td>
</tr>
</tbody>
</table>

Experiment: Theorems

<table>
<thead>
<tr>
<th>Problems</th>
<th>Total</th>
<th>CCG (%)</th>
<th>ACL2 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Trivial</td>
<td>1762</td>
<td>1544 (87%)</td>
<td>1056 (67%)</td>
</tr>
<tr>
<td>Recursive</td>
<td>4348</td>
<td>3394 (95%)</td>
<td>3642 (87%)</td>
</tr>
</tbody>
</table>