Slicing nichtterminierender Programme

(Localizing and explaining reasons for nonterminating logic programs with failure slices)

Ulrich Neumerkel

Institut für Computersprachen
Technische Universität Wien
ulrich@mips.complang.tuwien.ac.at

- slicing approach to nontermination
- constraint based analysis
- combination of static and dynamic techniques

Nontermination of logic programs

Problems in understanding:

- two different control flows + coroutining + constraint propagation
- nondeterminism
- unusual dataflow (partially known data structures)
- universal vs. existential termination (termination condition)

  existential termination: easy to observe, difficult to reason about

  universal termination: difficult to observe, but more robust
- most guesses for culprits are wrong (often too late)

Nontermination of logic programs (cont.)

Current solutions:

- termination analysis
  - limited power
  - no explanation
- debuggers, proof trees
  - produce irrelevant detail
  - complete display difficult
  - suggest imperative *step-by-step* understanding (most steps irrelevant)

Our solution based on slicing.

Example: Nonterminating program

```
← ancestor_of(Anc, leopold_I).          % Does not terminate
child_of(karl_VI, leopold_I).
child_of(maria_theresia, karl_VI).
child_of(joseph_II, maria_theresia).
child_of(leopold_II, maria_theresia).
child_of(leopold_II, franz_I).
child_of(marie_antoinette, maria_theresia).
child_of(franz_I, leopold_II).

ancestor_of(Anc,Desc) ←
    child_of(Desc,Anc).
ancestor_of(Anc,Desc) ←
    child_of(Child, Anc),
    ancestor_of(Child, Desc).
```

1,

Example: Nonterminating program and minimal slice

← ancestor_of(Anc, leopold_I). % Does not terminate

~~child_of(karl_VI, leopold_I).~~

~~child_of(maria_theresia, karl_VI).~~

~~child_of(joseph_II, maria_theresia).~~

~~child_of(leopold_II, maria_theresia).~~

child_of(leopold_II, franz_I).

~~child_of(marie_antoinette, maria_theresia).~~

child_of(franz_I, leopold_II).

~~ancestor_of(Anc,Desc) ←~~
    ~~child_of(Desc,Anc).~~

ancestor_of(Anc,Desc) ←
    child_of(Child, Anc),
    ancestor_of(Child, Desc).

1, 2,

Example: Nonterminating program and minimal failure slice

← ancestor_of(Anc, leopold_I).**, false.** % Does not terminate

~~child_of(karl_VI, leopold_I).← **false.**~~

~~child_of(maria_theresia, karl_VI).← **false.**~~

~~child_of(joseph_II, maria_theresia).← **false.**~~

~~child_of(leopold_II, maria_theresia).← **false.**~~

child_of(leopold_II, franz_I).

~~child_of(marie_antoinette, maria_theresia).← **false.**~~

child_of(franz_I, leopold_II).

~~ancestor_of(Anc,Desc) ←**false,**~~
    ~~child_of(Desc,Anc).~~

ancestor_of(Anc,Desc) ←
    child_of(Child, Anc),
    ancestor_of(Child, Desc).**, false.**

1, 2, 3.

Failure slice of logic program

- insert goal **false** at some program points
- nontermination of failure slice ⇒ universal nontermination of program
- accessible to analysis *and* execution
- 2^n possible failure slices
  Criteria for interesting failure slices:
  - eliminate terminating slices
  - eliminate redundant slices
    a ← **false,** b, c, d.
    a ← **false,** b, c, **false,** d.
    a ← **false,** b, c, d.
    % canonical representant:
    a ← **false,** b, **false,** c, **false,** d, **false.**

Implementation: Static analysis

- very large search space: $2^n, n =$ number of program points
- avoid abstract interpretation techniques: cost per slice $\gg n$, total $\gg n2^n$
- constraint based control flow analysis
  - represent program points as 0/1 variables (0 = false inserted)
  - establish relations between program points with CLP(FD) constraints
  - approximate termination/nontermination
  - propagate failure (left to right, and back when terminating)

	% actual encoding
perm(Xs, [X|Ys]) $\leftarrow$ % P2	$\neg$P2 $\Rightarrow$ $\neg$P3, $\neg$P3 $\Rightarrow$ $\neg$P4, % left to right
del(X, Xs, Zs), % P3	$\neg$P4 $\wedge$ AllwTermPerm $\Rightarrow$ $\neg$P3, % right to
perm(Zs, Ys). % P4	$\neg$P3 $\wedge$ AllwTermDel $\Rightarrow$ $\neg$P2. % —”—

- weighting to find minimal failure slices first.
- analysis only approximates set of nonterminating slices!

Implementation: Dynamic execution

Generic failure slice simulates all possible failure slices.

$$
\begin{array}{ll}
p(...) \leftarrow & \text{genericslice}p(...,\text{FVect}) \leftarrow \\
 & \quad \text{arg}(\text{n1},\text{FVect},1), \\
\quad q(...), & \quad \text{genericslice}q(...,\text{FVect}), \\
 & \quad \text{arg}(\text{n2},\text{FVect},1), \\
\quad ..., & \quad ..., \\
 & \quad \text{arg}(\text{ni},\text{FVect},1), \\
\quad r(...). & \quad \text{genericslice}r(...,\text{FVect}), \\
 & \quad \text{arg}(\text{ni}+1,\text{FVect},1).
\end{array}
$$

Implementation: Combining static analysis and dynamic execution

- failure vector of program points shared

  $\leftarrow$ fvectPQ_weights(FVect, Weights), % "analysis"
  FVect =.. [_|Fs],
  labeling([], Weights),
  labeling([], Fs),
  time_out(genericslicePQ(...,FVect), t, time_out). % execution

- analysis performed for all slices "at once"

Extensions for further language constructs

DCGs: insert **{false}** into rules

~~list([]) ⟶~~
    ~~**{false}**~~.
list([E|Es]) ⟶
    [E],
    list(Es).

CLP: unification should terminate. Useful to separate loops from labeling.

$\leftarrow$ constraints(Zs), **false**, ~~labeling([], Zs)~~.

Side effect free BIPs: is/2 etc. remain unchanged

Cut: similar to existential termination. Best results for shallow cuts.

Side effect BIPs: must not be executed, always insert **false** before them (Incomplete slicing).

Conclusion

- complement to termination analysis
- combines static and dynamic techniques
- constraint based analysis permits reduction of search space

Further work

- integrating better termination proofs (Mesnard et. al)
- better criteria for interesting failure slices
- argument slicing
- proof of nontermination (loop checking)