Slicing zur Fehlersuche in (Constraint-) Logikprogrammen

Stefan Kral, Fred Mesnard, Ulrich Neumerkel

Technische Universität Wien, Université de la Réunion

- transformationsbasierte Slicing-Techniken
- lokalisieren Programmteile, die geändert werden *müssen*
- unterstützen Lesarten
- keine Benutzerinteraktion
- alle üblichen Fehlerarten
  - fehlende Lösungen (insufficiency)
  - unerwartete Lösung (incorrectness)
  - Nichttermination
- Monotonie erforderlich (pures Prolog ohne Negation, CLP(FD) etc.)

Lesarten

- Traditionell: deklarativ und prozedural
- Verfeinerung der Lesarten durch Transformationen:

  Verallgemeinerung: Löschen von Zielen.

  vater(Vater) ←
  * ~~männlich(Vater)~~,
  kind_von(_Kind, Vater).

  Spezialisierung: Hinzufügen von Zielen (v.a. false/0).

  ~~verheiratet_mit(Gatte, Gattin) ← **false**~~,
  ~~gatte_gattin(Gatte, Gattin)~~.
  verheiratet_mit(Gattin, Gatte) ←
  gatte_gattin(Gatte, Gattin).

  \+ ermöglicht informelles Lesen größerer Programme

  \+ Quelltexttreue, einfache Veranschaulichung (Durchstreichen, Zudecken)

  \+ für (unvollständige) Constraints geeignet, kein neuer Formalismus wie Beweisbaum, Trace etc.

Slicing zur Fehlersuche

fehlende Lösung: maximale Verallgemeinerung, die scheitert

erklärt *data inconsistency* und *modelling error*

unerwartete Lösung: maximale Spezialisierung, die erfüllt ist (momentan nur mit false/0)

Nichttermination: maximale Spezialisierung, die nicht terminiert.

Gemeinsame Eigenschaften:

+ Fehler in Fragment bedingt Fehler im ursprünglichen Programm
+ Fragment *muß* verändert werden
+ keine Benutzerinteraktion (daher keine Bedienungsfehler)
? noch *slicing*, oder schon *program modification*?

Beispiel: Fehlende Lösung

← bruder_von(B, P). % Fehler

bruder_von(B, P) ←
    dif(B,P),
    männlich(B),
    kind_von(B,V),
    männlich(V),
    kind_von(P,V),
    kind_von(B,M),
    kind_von(P,M),
    weiblich(V).

männlich(franz_I).
männlich(joseph_II).
männlich(leopold_II).

weiblich(maria_theresia).

kind_von(joseph_II, maria_theresia).
kind_von(joseph_II, franz_I).
kind_von(leopold_II, maria_theresia).
kind_von(leopold_II, franz_I).

1,

Beispiel: Fehlende Lösung in maximaler Verallgemeinerung

← bruder_von(B, P). % Fehler

bruder_von(B, P) ←
* ~~dif(B,P),~~
* ~~männlich(B),~~
* ~~kind_von(B,V),~~
  männlich(V),
* ~~kind_von(P,V),~~
* ~~kind_von(B,M),~~
* ~~kind_von(P,M),~~
  weiblich(V).

männlich(franz_I).
männlich(joseph_II).
männlich(leopold_II).

weiblich(maria_theresia).

~~kind_von(joseph_II, maria_theresia).~~
~~kind_von(joseph_II, franz_I).~~
~~kind_von(leopold_II, maria_theresia).~~
~~kind_von(leopold_II, franz_I).~~

1, 2

Beispiel: Unerwartete Lösung

$\not\leftarrow$ kind_von(K, E), kind_von(E, K). % Fehler

kind_von(joseph_II, maria_theresia).
kind_von(joseph_II, franz_I).
kind_von(leopold_II, maria_theresia).
kind_von(marie_antoinette, maria_theresia).
kind_von(maria_theresia, marie_antoinette).
kind_von(leopold_II, franz_I).

1,

Beispiel: Unerwartete Lösung in maximaler Spezialisierung

$\not\leftarrow$ kind_von(K, E), kind_von(E, K). % Fehler

~~kind_von(joseph_II, maria_theresia).$\Leftarrow$ **false**.~~
~~kind_von(joseph_II, franz_I).$\Leftarrow$ **false**.~~
~~kind_von(leopold_II, maria_theresia).$\Leftarrow$ **false**.~~
kind_von(marie_antoinette, maria_theresia).
kind_von(maria_theresia, marie_antoinette).
~~kind_von(leopold_II, franz_I).$\Leftarrow$ **false**.~~

1, 2

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 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.

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

1,

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).← **false.**~~

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

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

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

~~child_of(Desc,Anc).~~

ancestor_of(Anc,Desc) ←

ancestor_of(Child, Desc), **false,**

~~child_of(Child, Anc).~~

1, 2.

missing termination proof

← phrase(regexp(Expr), Xs0,Xs) terminates_if
    finiteground(Expr), boundlist(Xs0).
regexp([]) ⟶
    [].
regexp([E]) ⟶
    [E].
regexp({_Expr}) ⟶
    [].
regexp({Expr}) ⟶
    regexp(Expr),
    regexp({Expr}).
regexp(A*B) ⟶
    regexp(A),
    regexp(B).

1,

Min.f-slice explains missing termination proof

← phrase(regexp(Expr), Xs0,Xs) terminates_if
finiteground(Expr), boundlist(Xs0).

regexp([]) ⟶
[].

~~regexp([E]) ⟶ {false},~~
~~[E].~~

~~regexp({_Expr}) ⟶ {false},~~
~~[].~~

regexp({Expr}) ⟶
regexp(Expr),
regexp({Expr}), **{false}.**

~~regexp(A*B) ⟶ {false},~~
~~regexp(A),~~
~~regexp(B).~~

1, 2,

?: Min.f-slice explains with subst's missing termination proof

← **Expr = {[]},** phrase(regexp(Expr), Xs0,Xs) terminates_if
~~finiteground(Expr),~~ boundlist(Xs0).
regexp([]) ⟶
[].
~~regexp([E]) ⟶ {false},~~
~~[E].~~
~~regexp({_Expr}) ⟶ {false},~~
~~{}.~~
regexp({Expr}) ⟶ **{Expr = []}, % ... not yet implemented ...**
regexp(Expr),
regexp({Expr}), **{false}.**
~~regexp(A*B) ⟶ {false},~~
~~regexp(A),~~
~~regexp(B).~~

1, 2, 3.

URLs

http://www.univ-reunion.fr/~gcc

http://www.complang.tuwien.ac.at/ulrich