Localisation des erreurs par « slicing » dans les programmes logiques avec contraintes

Ulrich Neumerkel

Technische Universität Wien

- techniques de slicing basées sur des transformations de programmes
- fragments localisés *doivent* être changés
- facilite des lectures sélectives
- sans interaction d'usager
- explique les phénomènes suivants :
  - échec inattendu (insuffisance)
  - solution inattendue (incorrection)
  - non terminaison
- monotonie indispensable (Prolog pur sans négation, avec CLP(FD) etc.)

Lectures des programmes

– lectures traditionnelles : déclaratives et opérationnelles

– lectures sélectives : raffiner par transformations

généralisation : enlever des buts

père(Père) ←
***** ~~masculin(Père)~~,
enfant_de(_E, Père).

spécialisation : ajouter des buts (spé. false/0 : tranche d'échec).

~~conjoint_de(Époux, Épouse) ← **false**~~,
~~époux_épouse(Époux, Épouse)~~.
conjoint_de(Épouse, Époux) ←
époux_épouse(Époux, Épouse).

+ facilitent les lectures des programmes plus volumineux

+ fidélité au code source, présentation simplifiée (rayer, cacher)

+ pas de nouveau formalisme comme les arbres de preuve, les traces

+ convient avec contraintes incomplètes

Localisation automatisée — explications

insuffisance (échec inattendu) : généralisation maximale qui échoue,
explique inconsistance de données (*data inconsistency*)
et erreurs de modélisation (*modelling error*)

incorrection (solution inattendue) : spécialisation maximale qui est vraie
(couramment avec false/0)

non terminaison : spécialisation maximale qui ne se termine pas

Propriétés communes :
+ l'erreur dans le fragment implique l'erreur dans le programme original
+ changement du fragment obligatoire
+ aucune interaction avec l'usager (sauf continue)
+ donc pas d'erreurs introduites pendant le débogage
? notion *slicing*, ou plutôt *program modification*?

Exemple : échec inattendu

← frère_de(B, P). % insuffisance

frère_de(F, P) ←
    dif(F,P),
    masculin(F),
    enfant_de(F,V),
    masculin(V),
    enfant_de(P,V),
    enfant_de(F,M),
    enfant_de(P,M),
    féminin(V).

masculin(franz_I).
masculin(joseph_II).
masculin(leopold_II).

féminin(maria_theresia).

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

Exemple : échec inattendu de la généralisation maximale

← frère_de(B, P). % insuffisance

frère_de(F, P) ←
* ~~dif(F,P),~~
* ~~masculin(F),~~
* ~~enfant_de(F,V),~~
  masculin(V),
* ~~enfant_de(P,V),~~
* ~~enfant_de(F,M),~~
* ~~enfant_de(P,M),~~
  féminin(V).

masculin(franz_I).
masculin(joseph_II).
masculin(leopold_II).

féminin(maria_theresia).

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

1, 2

Recherche des explications d'échec inattendu

– non-terminaison à cause des fragments généralisés
  → analyser la terminaison (cTI)
– complexité : sous-problème déjà NP-hard, pas d'approximation
  (*Monotone Minimum Satisfying Assignment*, Umans 1999)
  → chercher les minima locaux (un par un)
– « labeling » pour fragments généralisés souvent plus coûteux que l'original
  → adapter stratégie de « labeling »

Problème similaire : Explications dans PPC (Jussien)

– généralisation d'un système de contraintes
– beaucoup plus de contraintes que points de programme
  - plus coûteux
  - explication lisible pour usager ?
– utilise une recherche entrelacée avec « labeling »

Exemple : solution inattendue

$\not\leftarrow$ enfant_de(K, E), enfant_de(E, K). % incorrection

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

1,

Exemple : solution inattendue de la spécialisation maximale

$\not\leftarrow$ enfant_de(K, E), enfant_de(E, K). % incorrection

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

1, 2

Exemple : non terminaison

```
← ancêtre_de(Anc, leopold_I).          % non terminaison
enfant_de(karl_VI, leopold_I).
enfant_de(maria_theresia, karl_VI).
enfant_de(joseph_II, maria_theresia).
enfant_de(leopold_II, maria_theresia).
enfant_de(leopold_II, franz_I).
enfant_de(marie_antoinette, maria_theresia).
enfant_de(franz_I, leopold_II).

ancêtre_de(Anc,Desc) ←
    enfant_de(Desc,Anc).
ancêtre_de(Anc,Desc) ←
    enfant_de(Enf, Anc),
    ancêtre_de(Enf, Desc).
```

1,

Exemple : non terminaison de la tranche d'échec minimale

← ancêtre_de(Anc, leopold_I)., **false.** % non terminaison

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

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

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

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

enfant_de(leopold_II, franz_I).

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

enfant_de(franz_I, leopold_II).

~~ancêtre_de(Anc,Desc) ←**false,**~~

~~enfant_de(Desc,Anc).~~

ancêtre_de(Anc,Desc) ←

enfant_de(Enf, Anc),

ancêtre_de(Enf, Desc)., **false.**

1, 2.

Exemple : non terminaison

```
← ancêtre_de(Anc, leopold_I).          % non terminaison
enfant_de(karl_VI, leopold_I).
enfant_de(maria_theresia, karl_VI).
enfant_de(joseph_II, maria_theresia).
enfant_de(leopold_II, maria_theresia).
enfant_de(leopold_II, franz_I).
enfant_de(marie_antoinette, maria_theresia).
enfant_de(franz_I, leopold_II).

ancêtre_de(Anc,Desc) ←
    enfant_de(Desc,Anc).
ancêtre_de(Anc,Desc) ←
    ancêtre_de(Enf, Desc),
    enfant_de(Enf, Anc).
```

1,

Exemple : non terminaison de la tranche d'échec minimale

← ancêtre_de(Anc, leopold_I), **false.** % non terminaison

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

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

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

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

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

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

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

~~ancêtre_de(Anc,Desc) ← **false,**~~

~~enfant_de(Desc,Anc).~~

ancêtre_de(Anc,Desc) ←

ancêtre_de(Enf, Desc), **false,**

~~enfant_de(Enf, Anc).~~

1, 2.

```
←                    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,

← 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