Master STl - projet IntAbs

Extension d'un analyseur statique par interprétation abstraite

Sarah Zennou (Airbus Group) et Olivier Bouissou (The Mathworks)
sarah.zennou@airbus.com

1 Présentation générale

`simpleai` est un analyseur statique par interprétation abstraite de programmes `simple`. Les principales caractéristiques de ce langage sont qu'il ne comporte que des variables globales entières et que les fonctions n'ont ni argument, ni valeur de retour.

La classe de programmes `simple` est également une sous-classe du langage `newspeak` [1], spécialement conçu pour aider à l'analyse statique de programmes C. Pour cela, le compilateur open source `c2newspeak` traduit tout programme C en un programme `newspeak` sémantiquement équivalent où les constructions C équivalentes sont normalisées, les éléments dépendant de l'architecture explicités (taille des types entiers, des offsets de pointeurs, etc.), etc.

Combiner ces deux outils `c2newspeak` et `simpleai` permet donc d'obtenir un analyseur statique par interprétation abstraite d'une sous-classe de programmes C. Par exemple, analyser le programme `prog.c` se fait en exécutant la suite de commandes suivantes :

```
$c2newspeak prog.c -o prog.npk
$simpleai prog.npk
```

Le but de ce projet est d'améliorer `simpleai` par un certain nombre de techniques que vous aurez vues en cours : raffinement de fonctions de tranfert existantes, écriture de nouveaux domaines abstraits, combinaison de domaines, amélioration de *widenings*, élaboration de nouvelles tactiques d'itérations, etc.

2 Architecture de `simpleai`

Les sources des deux outils `c2newspeak` et `simpleai` sont regroupées dans un même répertoire `newspeak`. Ce répertoire contient plusieurs sous répertoires parmi lesquels :

— `bin` qui contient les exécutables `c2newspeak` et `simpleai`

— `src` qui contient notamment le sous-répertoire `simpleai` regroupant les sources de `simpleai`

La compilation de `c2newspeak` et `simpleai` se fait par l'unique commande `make`.

Les principales caractéristiques de `simpleai` sont les suivantes :

— le type d'un programme `simple` est `Simple.t`
— un état abstrait correspond à un module de signature `Sigs.State`
— un domaine abstrait correspond à un module de signature `UnrelState.Data`
— la génération d'un état pour un domaine abstrait donné se fait automatiquement par instanciation du foncteur `UnrelState.Make`

La fonction principale de l'analyseur est dans `simpleai.ml` et réalise successivement :

1. l'analyse grammaticale de la ligne de commande (appel à la fonction standard `Arg.parse`)
2. l'extraction du programme `newspeak` du fichier binaire fourni en argument (appel à `Newspeak.read`)
3. la transformation de ce programme est en un programme `simple` (appel à `Filter.process`)
4. l'analyse statique du programme `simple` (appel à `Solver.compute`). La fonction `Solver.compute` réalise le calcul du point fixe de la façon suivante :
   (a) elle crée l'état initial à partir des variables globales (appel à `State.universe` puis à `Solver.add_globals`)
   (b) elle applique les fonctions de transfert du bloc d'initialisation des globales appel à `Solver.compute_blk` avec paramètre `prog.init`
   (c) elle réalise le calul du point fixe à partir de cet état et du graphe de flot de contrôle du corps de la fonction main.

   En parallèle, elle réalise également les vérifications de la validité des appels aux fonctions de transfert

3 Prise en main de l'analyseur

3.1 Représentation interne des programmes

L'analyseur utilise le type OCAML `Simple.t` pour représenter les programmes. Cette représentation est très proche de la représentation par graphe de flot de contrôle décrite en cours, et il est important de la comprendre pour coder l'analyseur. L'analyseur dispose d'une option `--to-dot` :

```
./simpleai --to-dot graph.dot prog.npk
```

dont le but est d'écrire dans le fichier `graph.dot` une représentation au format `DOT` du graphe de flot de contrôle du programme (compilé en Newspeak) `prog.npk`. Cette option appelle la fonction `to_dot` du fichier `simple.ml`. En vous inspirant et aidant de la fonction `to_string`, vous devez donc compléter cette fonction. Pour plus d'information sur le format `DOT`, voir :
`https://en.wikipedia.org/wiki/DOT_(graph_description_language)`

3.2 Un premier domaine à compléter

L'analyseur que nous vous fournissons dispose d'un unique domaine abstrait, le domaine des constantes. Ce domaine n'est pas complet et pour prendre en main l'analyseur nous vous demandons de compléter le domaine ainsi que quelques autres fichiers pour passer les premiers tests. Nous fournissons 4 fichiers de tests pour ce domaine des constantes disponibles dans l'archive `tests_cst.tgz`.

3.2.1 Opérations arithmétiques

Essayez d'analyser le fichier `00.c` de `tests_cst`. Que remarquez-vous ? D'où vient l'erreur ? Complétez les fichiers `cst.ml` et `unrelState.ml` pour passer cet exemple.

3.2.2 Tests

Essayez maintenant le fichier `01.c` qui contient un test. Quel est maintenant le problème ? Corrigez-le en regardant les fonctions `guard` des fichiers `cst.ml` et `unrelState.ml`.

3.2.3 Boucles

Essayez le fichier `02.c` qui contient une boucle. On doit donc calculer un point fixe par algorithme de Kleene pour voir le plus petit invariant de ce programme. Que remarquez-vous ? Trouver l'endroit où ce problème doit être résolu et apportez les changement nécessaires.

3.2.4 Assertions

Finalement, essayer le fichier `03.c` qui utilise les assertions. Ces assertions doivent être gérées par la fonction `implies` des domaines. Compléter la fonction `implies` pour le domaine des constantes.

4 Travaux à effectuer

1. Remplacer le/les domaines abstraits par une analyse par intervalles d'entiers. Pour cela, écrire un nouveau domaine abstrait des intervalles entiers. Ce domaine devra avoir pour signature `UnrelState.Data`.
2. Lever les fausses alarmes des différents cas d'étude. Les cas d'études sont numérotés par difficulté croissante.
3. Ajouter une option `--unroll n` à l'analyseur pour qu'il déplie n fois chaque boucle.
4. Ajouter une option `--delay n` à l'analyseur pour qu'il utilise le widening retardé n fois à la place du widening standard.

5. Analyse disjonctive : étendre l'analyseur pour qu'il combine une analyse de parité et une analyse d'intervalles. En particulier, bien décrire l'extension du foncteur `UnrelState` qui génère un module de signature `Sigs.State`.

Références

[1] Charles Hymans and Olivier Levillain. Newspeak, Doubleplussimple Minilang for Goodthinkful Static Analysis of C. Technical Note 2008-IW-SE-00010-1, EADS IW/SE, 2008.