IZLO – Projekt 2: SMT solvery

Úvod

Hodiny na vašem monitoru poskočí na 23:55. Odevzdání projektu se nezadržitelně blíží a váš kolega stále nedokončil implementaci funkce, na které závisí funkčnost celého zbytku programu. Zuřivě refreshujete github v prohlížeči a konečně se v repozitáři objeví soubor `foo.c` s funkcí `f`. Na první pohled není jasné co funkce dělá a váš kolega s vámi nekomunikuje. Vaše situace je komplikována faktem, že pro optimální funkčnost vašeho programu potřebujete volat funkci `f` s co nejmenšími vstupy tak, aby vrátila `true`. Nezbývá vám nic jiného než poslední minuty do deadlinu využít k analyzování funkce pomocí SMT solveru.

Zadání

Předpokládejme následující kód v jazyce podobném C. Pro tento jazyk předpokládáme, že pracuje s celými čísly o libovolné přesnosti (není tedy potřeba řešit situace jako přetečení).

```
bool f(int A, int B, int C, int D, int E) {
  if (D <= 0 || E <= 0) {
    return false;
  }

  int x, y, z;

  x = A*B*2;

  if (x < E) {
    y = x + 5*B;
  } else {
    y = x - C;
  }

  if (y + 2 < D) {
    z = x*A - y*B;
  } else {
    z = x*B + y*A;
  }

  if (z < E+D) {
    return true;
  } else {
    return false;
  }
}
```

Vaším cílem je vytvořit formuli (ve formátu SMT), která je pro zadané hodnoty parametrů `A`, `B` a `C` (ty reprezentují jednu instanci problému) splnitelná právě tehdy, když:

1. existují hodnoty parametrů `D` a `E` takové, že funkce `f` vrátí `true`,
2. součet `D + E` je nejmenší možný.

Převod kódu na formuli

Při převodu kódu na formuli předpokládáme, že kód je v tzv. single-static assignment formě (SSA) (do žádné proměnné není přiřazeno více než jednou), ve které se náš kód již nachází. Pro kód bez cyklů lze pak takový kód převést postupným zapsáním jednotlivých přiřazení do formule. Například:

```
x = 2;
y = x * 2;
z = y - x + 3;
```

Tento kód lze převést na formuli:

$$x = 2 \ \wedge \ y = x * 2 \ \wedge \ z = y - x + 3$$

Vaším úkolem je, mimojiné, vymyslet jak takový postup rozšířit pro větvení a podmínky.

Řešení

Při řešení byste si měli vystačit s SMT-LIB příkazy `declare-fun` (pro deklaraci proměnných) a `assert` (pro přidání formulí, které mají být splněny). V těchto formulích si vystačíte s booleovskými spojkami (`and`, `or`, `not`, `=>`, ...), kvantifikátory (`forall`, `exists`) a termy v teorii celočíselné aritmetiky. Tato teorie obsahuje klasické funkce nad čísly (`+`, `-`, `*`, `div`, `mod`, `abs`) a predikáty pro porovnání čísel (`=`, `>`, `<`, `<=`, `>=`). Příklady použití lze nalézt níže, případně na stránce předmětu.

Referenčním (a doporučeným) SMT solverem je cvc5[1]. Pro řešení projektu zcela stačí použít jeho online rozhraní. Alternativně lze solver stáhnout jako binárku z githubu, případně je k dispozici na serveru Merlin, kde ji lze použít pomocí příkazu:

```
/usr/local/groups/verifikace/IZLO/cvc5-Linux projekt2.smt2
```

Formát SMT-LIB

Formát SMT-LIB využívá prefixový zápis operátorů. Term `x + y = x * y` je tedy zapsán jako `(= (+ x y) (* x y))`. Při deklaraci proměnných je potřeba uvést její tzv. *sort* (podobně jako typ proměnné například v jazyce C), v tomto projektu budeme pracovat pouze s celočíselnými proměnnými a odpovídajícím sortem `Int`.

```
; Nastavení teorie, ve které má SMT solver pracovat.
; ALL značí všechny teorie podporované solverem.
(set-logic ALL)

; Nastavení parametrů SMT solveru.
(set-option :produce-models true)

; Deklarace celočíselný konstant jako nulárních funkcí)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)

; Deklarace binární funkce c na celých číslech
(declare-fun c (Int Int) Int)

; Deklarace binárního predikátu pred na celých číslech
(declare-fun pred (Int Int) Bool)

; Přidání omezení reprezentujícího formuli (x * y = 0) → (x = 0 v y = 0)
(assert
  (=>
    (= (* x y) 0)
    (or (= x 0) (= y 0))
  )
)

; Přidání dalšího omezení reprezentujícího formuli ∀a ∀b (∃c (a + b = c))
(assert
  (forall ((a Int) (b Int))
    (exists ((c Int))
      (= (+ a b) c)
    )
  )
)

; Ověření splnitelnost konjunkce všech omezení
(check-sat)

; Pokud je status sat, vypíše nalezený model
(get-model)

; Pokud je status sat, vypíše hodnoty termů x, y a x + y v nalezeném modelu
(get-value (x y (+ x y)))
```

Kostra

Kostra projektu je ke stažení zde. Odevzdejte tento soubor, s řádkem "Zde doplnte vase reseni" nahrazeným Vaším řešením. **Pozor! Nemodifikujte nic jiného, jinak Vám automatické testy zbytečně strhnou body.**

Testování

Soubor s kostrou je doplněn několika testovacími vstupy, které fungují následujícím způsobem. Každý test je rozdělen do dvou částí:

- **a)** Ověří zda pro vstupní parametry existuje řešení a vypíše jej. Očekávaný status je `sat`.
- **b)** Ověří zda pro vstupní parametry neexistuje jiné řešení. Očekáváný status je `unsat`.

Skript samotný nekontroluje správnost výstupů. Jednotlivé testy jsou implementované pomocí příkazu `(check-sat-assuming (formulae))`, který ověří splnitelnost globálních omezení (definovaných pomocí `assert`) v konjunci s lokálními omezeními `formulae`. Pro debugování modelů v jednotlivých testech lze využít příkaz `(get-value (terms))`.

Další pokyny a doporučení

- V případě špatného nebo příliš komplikovaného řešení se může stát, že se solver „zasekne“. Při správném řešení by měl solver doběhnout během několika sekund.
- Odevzdáváte pouze soubor `projekt2.smt2` do IS VUT.
- Své řešení vypracujte samostatně. Odevzdané projekty budou kontrolovány proti plagiátorství, za nějž se považuje i sdílení vlastního řešení s jinými lidmi.
- Případné dotazy směřujte do fóra “Diskuze k projektům”.

---

1. https://github.com/cvc5/cvc5↩