Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

SMT-megoldók programozói felületének vizsgálata

Pünkösd Marcell III. évf, (BSc) mérnökinformatikus szakos hallgató
Konzulens: Tóth Tamás tudományos segédmunkatárs, MIT
Hibatűrő Rendszerek Kutatócsoport
Téma laboratórium 1. összefoglaló
2018/19. I. félév

Manapság szinte mindenhol szoftver fut. Ezeknek a szoftvereknek egy része biztonságkritikus rendszereken fut, ahol bármilyen apró kis szoftverhiba is akár emberi életekbe kerülhet. Ezért nagyon fontos, hogy ezeknek a szoftvereknek a megbízhatóságára garanciát tudjunk adni.

Arra, hogy a szoftvereket ellenőrizni tudjuk, és ezáltal garanciát adni a biztonságára, gyakran formális módszereket használunk. Egyik ilyen formális módszer például a modellellenőrzés. Ez abból áll, hogy a szoftvernek már a tervezési fázisában egy modellt készítünk és ennek a helyességére keresünk bizonyítást. Esetleg később, ezekből a verifikált modellekből generálunk kódot. Ezeknek a modelleknek az ellenőrzésére használhatunk többek között különböző SAT/SMT bizonyítókat. A témalaborom során én egy ilyen SMT megoldóval kísérleteztem.

A SAT (Boolean SATisfiability problem) megoldó lényege, hogy adunk neki egy logikai formulát (pl.: A & !B) És megpróbálja megállapítani róla, hogy kielégíthető-e (SAT; azaz a változókat megfelelően választva igazra értékelődik-e ki a kifejezés). Az SMT (Satisfiability Modulo Theories) ennél okosabb, neki nem csak logikai formulákat tudunk megadni, hanem belevonhatunk az egyenletünkbe más típusokat is, például integereket (pl.: A > 5 & A <10). Fontos tulajdonságuk még, hogy a legtöbb esetben egy példát is szolgáltat a megoldó, ami kielégíti a formulánkat. Én az utóbbinak ezt a tulajdonságát felhasználva egy Sudoku megoldót készítettem.

A konkrét megvalósításhoz a programot Python 3-ban írtam. És az SMT solverem pedig a Microsoft Research által fejlesztett Z3 volt. Illetve felhasználtam egy PySMT nevű absztrakciós library-t, amely egy egységes felületet biztosított a logikai formulák leírására, majd azt fordította a támogatott megoldók saját API-jára (többek között a Z3-ra is).

Ahhoz, hogy a megfelelő formulára le tudjunk írni egy Sudoku feladványt, 5 szabályt kell betartani egy nxn-es táblára. Az első szabály az, hogy minden szám legyen 1 és n között. A következő három, hogy egy sorban, oszlopban, illetve minden $\sqrt{n}$x$\sqrt{n}$-es csoportban egy szám csak egyszer szerepelhet. Az utolsó szabály pedig, hogy ha adott egy szám (a feladat által), akkor azt nem változtathatjuk meg. Utóbbi triviálisnak tűnhet, de nagyon fontos, hiszen ez definiálja magát a feladatot.

A program, amit írtam, ezt a modellt procedurálisan építi fel. És alkalmazza rá a feladat által szolgáltatott megkötéseket. Majd ezt tovább adja a PySMT megoldónak, ami eldönti, hogy megoldható-e. Ha megoldható akkor a példa, amit ad, az a Sudoku feladvány megoldása

```
(temalab) marcsello@luna:~/temalab/z3_sudoku_solver$ time ./simple_exa
Loading puzzle...
Building 9x9 table model...
Applying puzzle to model...
Input:
_       _       3       2       _       _       8       4       _
1       _       9       8       7       _       _       _       5
_       8       2       _       _       1       _       7       3
4       2       _       5       1       _       3       8       _
_       9       _       _       3       7       6       _       _
_       6       _       9       2       _       _       _       _
_       _       _       _       _       5       _       _       4
2       5       _       _       9       _       _       _       6
7       1       4       3       _       2       _       9       _
Looking for solution...
Solution found!
6       7       3       2       5       9       8       4       1
1       4       9       8       7       3       2       6       5
5       8       2       6       4       1       9       7       3
4       2       7       5       1       6       3       8       9
8       9       5       4       3       7       6       1       2
3       6       1       9       2       8       4       5       7
9       3       6       1       8       5       7       2       4
2       5       8       7       9       4       1       3       6
7       1       4       3       6       2       5       9       8

real    0m1.014s
user    0m0.588s
sys     0m0.052s
```

Miután PySMT-vel tökéletesen működött, átírtam a programot úgy, hogy közvetlenül a Z3 API-t használja. Ez a verzió jelentősen gyorsabban futott, valószínűleg azért, mert itt már használhattam a Distinct kulcsszót, illetve a PySMT fordítási overhead-je is kimarad.