Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Ellenőrzési módszerek

Témalaboratórium összefoglaló
2017/18. I. félév

Bajkai Viktória Dorina

III. évf, mérnökinformatikus szakos hallgató
BSc Rendszertervezés specializáció

Konzulens:

Hajdu Ákos doktorandusz
(Méréstechnika és Információs Rendszerek Tanszék)

1. Közös előadások

Az első hetekben közös előadások keretében ismerkedtünk meg az alábbi témakörökkel:

- **Modellellenőrzés**
  Kipróbáltuk az UPPAAL, PetriDotNet és Gamma modellellenőrző eszközöket.
  Az UPPAAL és a PetriDotNet valós idejű rendszerek, illetve petri hálók analízisére és szimulációjára használhatóak, a Gamma pedig állapottérképek kompozícionális modellezésére.

- **Forráskód alapú hibakeresés**
  Tanultunk a statikus és dinamikus tulajdonságok vizsgálatáról. Statikus analízis esetén hibaminták vizsgálata alapján szűrnek ki olyan hibákat, mint pl.: nem használt változó. (Megismert programok: FindBugs, Sonar Qube, Codacy) Dinamikusnál adatfolyam és vezérlésfolyam analízis segítségével lehet felderíteni olyan problémákat, mint pl.: null pointer vagy túlindexelés. (Megismert programok: Infer, Viper)

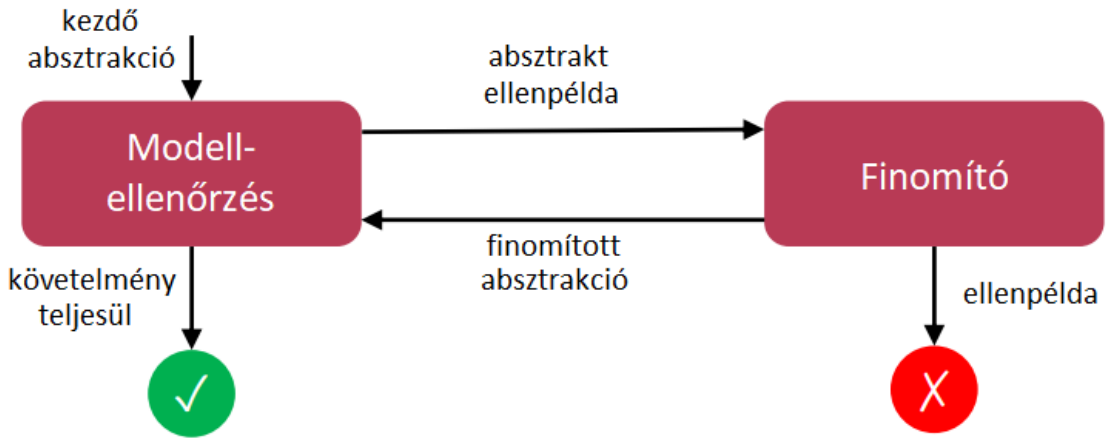
- **Szoftver modellellenőrzés**
  Megismerkedtünk a Theta modellellenőrző keretrendszerrel.

2. Önálló munka

Az önálló munkám során szoftver modellellenőrzéssel foglalkoztam. Először megismerkedtem különböző modellellenőrző algoritmusokkal, predikátumabsztrakcióval és korlátos modellellenőrzéssel. Végül a Thetában implementáltam egy „okosított” korlátos modellellenőrzőt.

- **Predikátumabsztrakció**
  Ennél az algoritmusnál a változók értékeit nem tartjuk nyilván, csak predikátumokat.

  Működése:

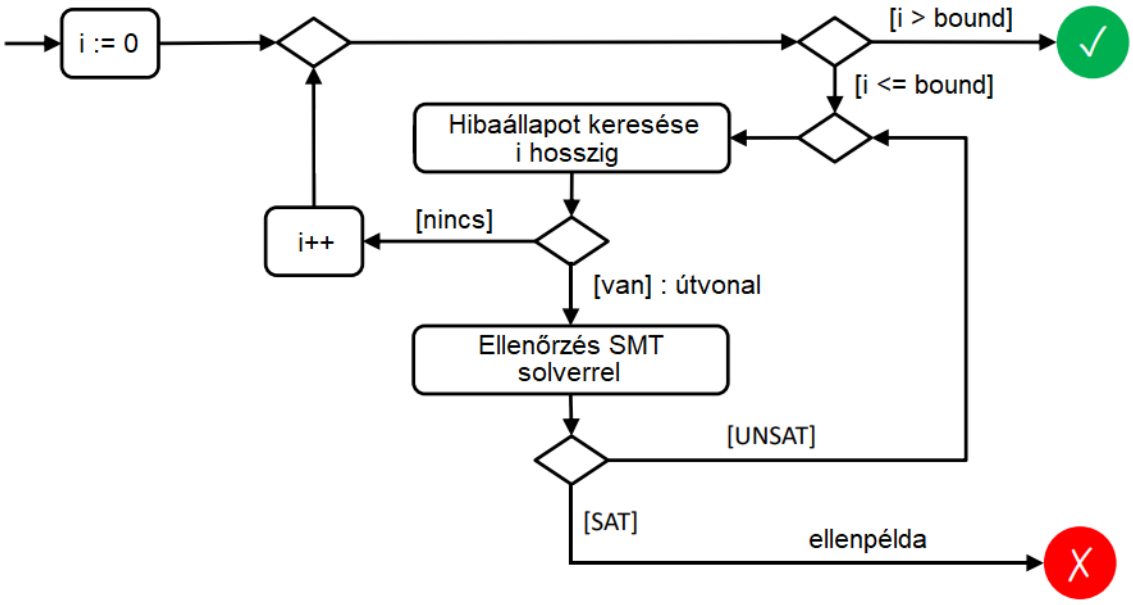


  A finomító először ellenőrzi egy SMT solver segítségével, hogy az ellenpélda kielégíthető-e. Ha nem, akkor új predikátumot ad hozzá a predikátumhalmazhoz, és folytatja az ellenőrzést.

  Mivel a predikátumabsztrakció felülbecsüli az eredeti modellt, ezért ha az absztrakció ellenőrzése során teljesül a követelmény, biztosak lehetünk benne, hogy az eredeti modellre is teljesül.

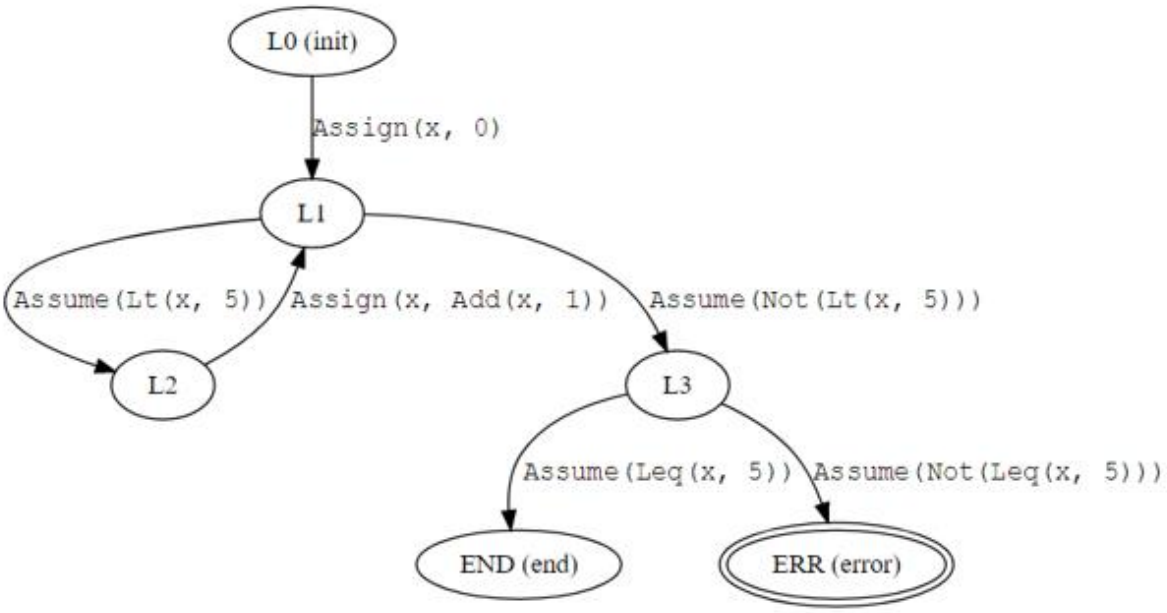
- **Korlátos modellellenőrzés**
  Ennek az algoritmusnak a bemenete egy Control Flow Automata(CFA), ezt kibontva kell bejárni az állapotteret egy adott útvonalhossz korlátig.

  Működése:



  Ez az algoritmus nem alkalmas biztonságos modellek ellenőrzésére, hiszen csak akkor tudjuk megállapítani egy modellről, hogy biztonságos, ha az egész állapotterét bejártuk, és ha bejut egy ciklusba, akkor az állapottere végtelen nagyságú lesz.

- **„Okos" Korlátos Modellellenőrző**
  A Thetában létrehozott modellellenőrzőm annyiban különbözik a simától, hogy nem csak a hibaállapotok útvonalának kielégíthetőségét ellenőrzi le, hanem minden állapotét. Ezzel elkerülhető, hogy olyan részfákat járjon be az algoritmus fölöslegesen, amik amúgy el sem érhetők. Az alábbi ábrán látható biztonságos CFA-n például el tudtuk kerülni a ciklusba való fölösleges belépést, így egyértelműen megállapítható, hogy biztonságos:



A sima és okos algoritmusokat 19 CFA-n futtattam le. A CFA-k a CERN-ből és modellellenőrző versenyekről származnak.

Név	Futási idő átlag (ms)	Talált hibaállapotok száma	Futási idő átlag (ms)	Talált hibaállapotok száma	Mélység
cfa_1.cfa	633,5	2	673,5	1	102
cfa_2.cfa	401	2	617,5	1	102
cfa_3.cfa	10733,75	499	4542,25	1	102
cfa_4.cfa			56976,25	1	140
cfa_5.cfa	31854,25	4205	31003,75	1	134
locks_14_false.c_0.cfa	3245,5	1	744,75	1	6
locks_14_false.c_1.cfa	30,25	1	33,75	1	6
locks_15_false.c_0.cfa	30,75	1	26	1	6
locks_15_false.c_1.cfa	54,75	1	20	1	6
s3_clnt_1_false.c_0.cfa			14806,5	1	139
s3_clnt_3_false.c_0.cfa			18663,25	1	148
s3_clnt_4_false.c_0.cfa			17846,75	1	146
s3_srvr_2_false.c_0.cfa			26553,25	1	168
s3_srvr_10_false.c_0.cfa	113,25	47	191,25	1	56
s3_srvr_1_false.c_0.cfa			25164,75	1	161
s3_srvr_6_false.c_0.cfa	110	2	95,25	1	50
s3_srvr_14_false.c_0.cfa			2230,5	1	103
counter5_unsafe.cfa	27,75	6	31,75	1	12
gcd_concrete_unsafe.cfa	95,75	47	86,25	1	18

A narancssárga oszlopok a sima algoritmushoz tartoznak, a kékek pedig az okoshoz. Pirossal azok az értékek vannak kiemelve, ahol a sima algoritmus futott gyorsabban, míg zölddel azok, ahol az okos. Amelyik sorban nem találhatók értékek, ott az algoritmus futása nem fért bele a 3 perces időkorlátba.

Látható, hogy az okosítás általában meggyorsította az algoritmus futását. A sima azért lehet néhol gyorsabb, mivel a folyamatos útvonal ellenőrzés is elég sok időbe telhet, de a táblázatból kiolvasható értékek alapján sehol sem volt a sima algoritmus számottevően gyorsabb. Ezek alapján levonható az a következtetés, hogy a sima algoritmus helyett érdemesebb az okosat használni modellellenőrzésre.