Верификација хардвера свођењем на CAT

Лука Станковић
mi241055

31. oktobar 2025.

Sažetak

Производња хардвера је дуг и скуп процес и зато га је пожељно верификовати пре производње. Верификација хардвера, у ствари провера еквивалентности комбинаторних кола, може се свести на CAT. Свођење на CAT постиже се изражавањем комбинаторног кола преко исказне формуле, која се преводи у нормалну форму. Нормална форма споја кола се уноси у *Minisat* решавач који доказује да ли је спој два кола CAT или УНСАТ.

1 Увод

Производња новог хардвера је скуп процес, а грешке у хардверу је након испоруке теже отклонити него грешке у софтверу[1]. Због таквих околности и потреба, већ дуже време изучавају се начини формалне провере хардвера. Формална верификација може се добро искористити над моделима који су добро дефинисани, као што су то комбинаторна кола или РТЛ(енг. *Register Transfer Level*).

Процес коришћен у овом раду заснива се на дедуктивној верификацији, приступу формалне верификације. Генеришу се математичке формуле које описују спецификацију система, а онда се над њима примењује CAT решавач.

Пројекат је написан у C++ и GDScript коду. Део у C++ коду описује логику формула, док део у GDScript описује визуелизацију кола. Користи се исказна логика са свођењем апстрактног синтаксног стабла формуле на КНФ. КНФ се прослеђује *Minisat* решавачу. Визуелизација се користи да употпуни слику и олакша ручно налажење спорних места у комбинаторном колу.

Овакав приступ даје лако разумљив и лако проширљив код, што нису занемарљиве особине у контексту аутоматског резоновања. Исказна логика темељ је сложенијих логика, тако да се и сам пројекат може надоградити тако да користи више логике.

Остатак рада је организован у четири поглавља која се баве терминологијом у раду, методом и имплементацијом. Последње поглавље садржи закључке који се могу извести.

2 Основе

Најосновнији термин који се среће у овом раду је формула. Формула се састоји од атома и операција над њима. Операције над атомима које се срећу у овом раду су унарна операција негације и бинарне операције: конјункција, дисјункција, следи, еквиваленција. Формула може садржати 'тачно' и 'нетачно'(убудуће скраћено на 'Т' и 'Н'), мада се у раду не обрађују ти случајеви. Валуација је функција која пресликава атоме у Т или Н. Из валуација се може закључити да ли су две формуле једнаке. Две формуле су једнаке ако су им валуације за сваку комбинацију додељивања вредности атомима једнаке.

Формуле су нам потребне зато што их можемо изразити преко апстрактних синтаксних стабала, а та стабла можемо превести у нормалну форму формуле. Апстактно синтаксно стабло(за наше потребе) је стабло у коме важи:

1. Чворови осим листова су операције
2. Листови су атоми
3. У случају унарне операције, дете операције је подформула
4. У случају бинарне операције, лево дете је лева подформула, а десно десна подформула операције

Нормална форма формуле **ф** је формула евкивалентна формули **ф** која прати нека правила. У нашем домену издвајају се две врсте нормалне форме, негативна нормална форма(скр. ннф) и конјуктивна нормална форма(скр. кнф). Ннф је формула у којој једине негације могу бити негације атома, док су јој операције негација, конјункција и дисјункција. Кнф је нормална форма која се састоји само од конјукција, док се подформуле састоје само од дисјункција. Негације могу постојати у кнф-у, али само над атомима. У кнф-у, литерал је атом или негација атома, док је клаузула једна дисјункција литерала кнф-а. Конјункција клаузула је оно што чини кнф.

Формула **ф** је задовољива ако постоји валуација **в** над њеним атомима која је тачна. Сазнање да ли је нека формула задовољива је један од најосновнијих проблема у информатици и назива је САТ проблем. Решавач САТ проблема је софтвер који проналази да ли је нека формула задовољива или не. *Minisat* је један такав софтвер намењен за почетнике. *Minisat* чита из фајла који је написан у DIMACS нотацији, а такав фајл за наше употребе има тип .cnf, што означава да садржај фајла јесте формула у кнф-у. Треба нагласити да је САТ проблем НП-комплетан, што значи да се не може решити у полиномијалном времену. Преко решавача *Minisat* можемо доказати да су неке две формуле еквивалентне.

3 Опис методе

Алгоритам се састоји из више .hpp фајлова подељених по намени. Те намене су дефинисање формуле, парсирање, дефинисање нормалних форми, прављење DIMACS фајла и прављење графа за визуелизацију.

3.1 Парсирање

Парсер је једноставан и пролази кроз ниску карактер по карактер. Белине се игноришу, заграде и имена атома се у истој функцији обрађују. Рекурзијом се постиже редослед операција, зато што се функције које су позване раније у рекурзији резолвирају после каснијих функција.

Корак	**Карактер**	**Функција**	**Дубина рекурзије**
1	а	parse_string_into_formula	0
2	а	parse_or	1
3	а	parse_and	2
4	а	parse_not	3
5	а	parse_atom	4
6	&	parse_not	3
7	&	parse_and	2
8	(	**parse_not**	3
9	(	parse_atom	4
10	б	**parse_or**	5
11	б	parse_and	6
12	б	parse_not	7
13	б	parse_atom	8
14	|	parse_not	7
15	|	parse_and	6
16	|	parse_or	5
17	в	parse_and	6
18	в	parse_not	7
19	в	parse_atom	8
20	)	parse_not	7
21	)	parse_and	6
22	)	parse_or	5
23	)	parse_and	4
24	)	parse_not	3
25	)	parse_atom	2
26	)	parse_or	1
27	)	parse_string_into_formula	0

Tabela 1: Табела позива функција и карактера који парсер у том тренутку разматра.

Примера ради, рецимо да имамо формулу **а** $\wedge$ **(б** $\vee$ **в)**. Оператор конјункције би се, да нема заграда, примењивао над **а** и **б**, док би се онда оператор дисјункције примењивао над резултатом конјункције и **в**. То не одговара редоследу операција, па није пожељно. Први пролаз кроз рекурзију долази до функције атома и онда се тај атом враћа кроз функције све док не дође до знака $\wedge$. Пошто баш тај знак одговара конјункцији, покреће се рекурзија за десну страну бинарне релације, али од негације, да би се и са те стране одржао редослед оператора. Пошто десна страна формуле **а** $\wedge$ **(б** $\vee$ **в)** креће са отвореном заградом, то значи да се позива рекурзија од дисјункције, пошто је она последња по редоследу. Опет се рекурзијом долази до атома, б, па се рекурзија враћа до дисјункције, пошто наилази на знак $\vee$. Ова функција онда позива рекурзију за десну страну која враћа атом, па се

онда целокупна дисјункција враћа до функције конјункције као десно дете. На тај начин су релације увек у добром редоследу.

3.2 Нормалне форме

Прва нормална форма формуле је негативна нормална форма. Алгоритам се састоји из две функције које праве рекурзију једна са другом. Друга функција се бави негацијама које добија од прве, а прва негацијама од друге. Прва функција прослеђује негацију другој, да би је друга обрадила тако што препознаје о којој је операцији реч и онда ДеМоргановим законима проследи негацију на атоме. У случају дупле негације, то јест ако друга функција препознаје негацију, позива се прва функција над формулом без негација. Импликација и еквиваленција своде се на конјункцију и дисјункцију.

Друга нормална форма формуле, кнф, изводи се из ннф. Процес је сличан ннф, са тиме да се прво врши рекурзија над ннф. а онда новонастал кнф формула мења тако да не постоје дуплирани литерали и дуплиране клаузуле.

3.3 DIMACS

Да би се DIMACS фајл попунио, потребно је извести формулу од две унесене формуле. То се постиже прављењем споја(енг. *miter*) тако да од првобитних формула **ф** и **г** настане формула **(ф & г) | (ф & г)**. Таква формула се преводи у кнф и онда се тај кнф уноси клаузулу по клаузулу у фајл.

3.4 JSON граф

Да би могла да се изврши визуелизација, потребно је превести апстрактно синтаксно стабло у повољнију форму и послати га Godot-у. Стабло се преводи у JSON помоћу библиотеке nlohmann. Стабло се претвара у мрежу, а смер кретања се одржава преко поља 'dist'(скр. од енл. *distance*). Уводи се разлика између унарног и бинарних операција, као и атома, ради правилог исписа. Мрежа такође садржи и истинитосну таблицу сваког атома, као и сваке операције.

4 Имплементација

Имплементација скоро целокупне логике написана је у C++. Једини део ван C++ је визуелизација написана у GDScript-у.

4.1 Формула

Формула је дефинисана преко типски сигурне уније која се зове *variant*. Унија у себи може садржати једну од структура које описују градивне елементе формуле: нетачно, тачно, атом, негација, бинарна операција. Свака структура садржи у себи поља њој намењена. На пример, негација садржи показивач на своју потформулу, док бинарна операција садржи тип и два показивача који показују на леву и десну потформулу.[2]

Показивач на формулу је дељени показивач(енг. *shared pointer*). Сваки овакав показивач дели референцу на исти објекат, тако да није потребно ручно правити конструкторе и деструкторе.[2]

Валуација је представљена са *std::map*, тако да мапира име атома у Т или Н. Скуп атома који код чува ради обраде налази се у *std::set*, уређеном складишту без дупликата.[2]

Функција која прави спој ради то тако што узме обе формуле и направи 5 нових формула од њих. Те формуле се онда слажу једна у другу док се на крају не добије спој формула такав да је нетачан само ако формуле јесу еквивалентне.

4.2 Парсирање

Парсирање је програмирано преко структуре која се зове Parser и која садржи све њој потребне функције. Парсирање сваке операције своди се на функцију која парсира једну операцију или, ако није ниједна операција, заграде и атом.

Правилан упис формуле се подразумева, да би код био разумљивији.

4.3 Нормалне форме

Литерали нормалне форме изражени су преко структуре која у себи садржи ниску која представља име и логичку вредност која означава да ли је литерал негиран или није.[2]

Клаузула је низ литерала, док је нормална форма низ клаузула. Додате су особине норманој форми; конкатенација и укрштање. Конкатенација листи је копирање листи у нову листу, листу по листу. Укрштање користи конкатенацију тако што се сваки елемент прве листе конкатенише са сваким елементом из друге листе.[2]

Ннф и кнф имплементирана су рекурзивно, опирући се на структуру формуле описану у подсекцији о формули[2].

4.4 DIMACS

Конструкција DIMACS-а креће додељивањем бројчаног идентификатора сваком литералу нормалне форме. Ово се постиже кроз *std::map*, тако што се ниска мапира у цео број. Успутно се бележи и укупан број клаузула.

Остаје да се направи ниска која ће се убацити у фајл. Ниска увек креће са 'p cnf', после чега иде број литерала и број клаузула. Напослетку се поново пролази кроз све клаузуле и њене литерале. Редни број литерала је познат зато што су сви литерали мапирани пре другог проласка кроз клаузуле. Додељује се карактер '-' сваком литералу који је у негацији.

4.5 JSON граф

Граф се изражава преко структуре. Памте се чворови и ивице. Ивице су сачуване у вектору парова идентификатора чворова. Чворови садрже идентификатор, опис, дистанцу од краја, тип и листу валуација.

Главна функција у структури је *from_formula*, која рекурзивно пролази кроз формулу и позива функције које се баве њеним деловима. Формула

може бити улаз или операција, тако да главна функција прослеђује одговарајућој помоћној функци коју вредност треба да запише у мапу чворова. Успутно се записују и ивице, тако да крајњи граф буде усмерен од улаза до излаза.

5 Закључак

Исказна логика се показала као добар темељ за проверавање еквивалентности формула које представљају комбинаторна кола. Слично, C++ се показао као добар језик за имплементацију формула и нормалних форми. Методе имплементације формула настале су из вежби предмета[2].

Рад чини једну целину која од унесене ниске прави апстрактно синтаксно стабло, а онда од њега нормалну форму погодну за ефикасан пролаз *Minisat* решавача. Успутно је могуће проверити и изглед самих кола кроз Godot визуелизацију.

Literatura

[1] Vujošević Janičić, *Verifikacija softvera*, 6.XI 2023.

[2] Код вежби из Аутоматског резоновања, 2024/25, линк