

# Решавање неких логичких игара користећи

## СМТ решаваче

Семинарски рад у оквиру курса

Аутоматско резоновање

Математички факултет

Немања Мићовић, Лазар Ранковић

nmicovic@outlook.com, lazar.rankovic@outlook.com

### Абстракт

Логичке игре дуги низ година поседују велику дозу занимљивости и неким представљају чест хоби. Како комплексност логичких игара може бити висока, јавља се проблем генерисања логичких игара за које смо сигурни да постоји решење и да је јединствено. Рад разматра примену СМТ решавача у верификацију решења логичких игара и доказивање јединствености решења. Део 1 илуструје логичке игре, и наводи пример једне од популарнијих. у делу 2 се уводи концепт СМТ решавача који је коришћен у раду за решавање логичке игре, део 3 приказује пример решавања једне од познатих логичких игара користећи СМТ решавач. СМТ решавачи су се показали као одличан алат за решавање поменутих проблема јер природно описују ограничења која постоје у логичким играма.

### Садржај

<b>1</b>	<b>Логичке игре</b>	<b>2</b>
<b>2</b>	<b>СМТ решавачи</b>	<b>2</b>
2.1	Кратак преглед релевантних историјских догађаја . . . . .	2
2.2	Примене СМТ решавача . . . . .	3
2.3	СМТ решавач . . . . .	3
2.3.1	Неке од коришћених теорија . . . . .	3
<b>3</b>	<b>Решавање логичких игара користећи СМТ решаваче</b>	<b>4</b>
3.1	Логичка игра Три суседне . . . . .	4
3.1.1	Ограничења . . . . .	4
3.1.2	Питање јединствености решења . . . . .	6
3.2	Yices program . . . . .	6
<b>4</b>	<b>Закључак</b>	<b>10</b>
	<b>Literatura</b>	<b>11</b>

# 1 Логичке игре

Логичке игре за собом имају дубоку традицију и историју, а упркос страховито брзом развоју технологије у последњих неколико година, и даље поседују велику популарност. Разлог за њихову популарност јесте често једноставност правила и неочекивана комплексност која уме да заинтригира људе. У многим часописима се редовно штампају игре као што је судоку (приказан на слици 1).

Логичке игре односно загонетке су погодне за математички опис користећи логику и представљају чест пример примене САТ и СМТ решавача за генерисање инстанци проблема, решавање и валидацију да ли је решење јединствено.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Слика 1: Судоку

## 2 СМТ решавачи

### 2.1 Кратак преглед релеватних историјских догађаја

Готфрид Вилхелм Лајбниц<sup>1</sup> је међу првима изнео идеју о механизацији процеса људског резоновања.

Године 1928. Давид Хилберт<sup>2</sup> је поставио проблем одлучивања - *Entscheidungsproblem*. Проблем тражи алгоритам који узима као улаз формулу логики првог реда и одговара ДА или НЕ да ли је формула универзално тачна или не.

Алонзо Черч<sup>3</sup> и Алан Тјуринг<sup>4</sup> у засебним радовима 1936. године показују да опште решење за Хилбертов проблем одлучивања не постоји. Черч доказ изводи свођењем на ламбда рачун, а Алан Тјуринг редукцијом на халтинг проблем.

Но уколико се изврши рестрикција на одређену теорију, ипак можемо добити многе одлучиве фрагменте теорије логики првог реда.

Током деведесетих велика популарност проблема задовољности исказних формула (проблем САТ). Иако је проблем НП комплетан, САТ решавачи могу решавати и формуле које имају чак и милионе променљивих. Наравно, постоје формуле које ће изазвати експоненцијално понашање алгоритма који имплементира решавач.

<sup>1</sup>Gottfried Wilhelm Leibniz (1646-1716)

<sup>2</sup>David Hilbert (1862-1943)

<sup>3</sup>Alonzo Church (1903-1995)

<sup>4</sup>Alan Turing (1912-1954)

Популарност САТ проблема природно води у идеју да се САТ комбинује са резонувањем у одређеним теоријама. Појављује се велики број радова и новонастала област се назива СМТ (eng. Satisfiability Modulo Theories).

## 2.2 Примене СМТ решавача

СМТ решавачи имају широку примену у области верификације софтвера и хардвера [1, 4, 5, 6], но корисни су и у другим ситуацијама у којима је формула логике првог реда редукована на неку теорију адекватна за опис проблема.

Једна од примена СМТ решавача јесу и логичке игре где се СМТ решавач може користити за генерисање инстанци игре, генерисање решења игре, провере да ли је решење тачно као и доказ да је решење игре јединствено. У делу 3 биће приказана примена СМТ решавача над једном логичком игром.

## 2.3 СМТ решавач

СМТ решавачи су засновани на САТ решавачима но нуде виши степен апстракције чиме задржавају и семантику која се налази у формули. То омогућава и примену специјализованих алгоритама који могу узети додатну семантику у обзир.

СМТ решавачи имају модуларну архитектуру која се најчешће састоји из два централна дела, CDCL (eng. Conflict Driven Clause Learning) заснован САТ решавач и СМТ језгро које имплементира неку од теорија (приказане у делу 2.3.1). При промени теорије над којом се резонује, модуларни приступ омогућава да се задржи део система који се бави решавањем проблема САТ.

### 2.3.1 Неке од коришћених теорија

**ЕУФ теорија** ЕУФ (eng. Equality with Uninterpreted Functions) теорија је теорија која садржи један предикатски симбол, симбол једнакости  $=$  и произвољан број сорти и функцијских симбола који се могу попутно слободно интерпретирати. СМТ проблем за ову теорију је неодлучив, но базни фрагмент теорије је одлучив. Задовољивост коњункције базних литерала је одлучив у полиномијалном времену користећи Нелсон-Орен процедуру [7].

**Реална аритметика** Теорија која описује реалну аритметику поседује сорту *Real* и симболе  $0, 1, +, \cdot, -, /, \leq$ . Модел теорије је структура реалних бројева  $R$ . Теорија је одлучива, а проблем испитивања задовољивости коњункције линеарних базних литерала је одлучив у полиномијалном времену. Користе се Фурије-Моцкинова процедура [7] и Симплекс процедура [2].

**Целобројна аритметика** Теорија описује целобројну аритметику, поседује сорту *Int* и симболе  $0, 1, +, \cdot, -, \leq$ . Модел теорије је структура целих бројева  $Z$ . У општем случају теорија је неодлучива, но њен линеарни фрагмент (Презбургерова аритметика [7]) је одлучив. Проблема испитивања задовољивости коњункције линеарних базних литерала је одлучив и НП комплетан.

**Теорија низова** Теорија низова се користи за апстракцију концепта низа. Поседује сорте *Array*, *Index* и *Value* који представљају, редом, низ, индекс низа и вредност коју елементи низа могу узимати. Символи *select* и *store* апстрахују индексирање низа и додељивање вредности елементу низа. Приметимо да се неће мењати вредности низа при операцији *store* већ ће операција враћати нови низ који настане када се у оригиналном низу промени вредност. Теорија је неодлучива у општем случају, но базни фрагмент теорије је одлучив. Проблем испитивања задовољности коњункције базних литерала је НП комплетан.

**Теорија битвектора** Теорија битвектора се користи за опис бинарних бројева и има значајну примену у области верификације хардвера. Теорија је одлучива, а проблем испитивања задовољности коњункције базних литерала је НП комплетан.

### 3 Решавање логичких игара користећи СМТ решаваче

У овом делу ће бити изложен пример решавања логичке игре коришћењем YICES СМТ решавача [3].

#### 3.1 Логичка игра Три суседне

Логичка игра *Три суседне* се састоји из матрице поља димензије  $n \times n$ . На почетку игре, поља су обојена у плаво, бело и сиво. Поља која су обојена у сиво, играч може да промени у плава или бела, док оригинална плава и бела поља не може да мења. Ово ћемо звати *почетно стање игре*.

Циљ игре је да играч сва сива поља обоји у плаво или бело при чему морају да важе следећа ограничења:

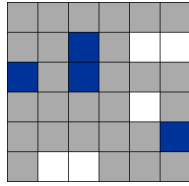
- Не постоје три суседна поља у истој врсти која су обојена истом бојом (услов 1)
- Не постоје три суседна поља у истој колони која су обојена истом бојом (услов 2)
- У свим врстама мора бити једнак број плавих и белих поља (услов 3)
- У свим колонама мора бити једнак број плавих и белих поља (услов 4)

Стање у којем се налази табла (матрица) игре када је игра решена назваћемо *задовољено стање игре*.

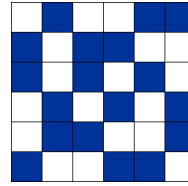
На слици 2 је приказано почетно стање игре, а на слици 3 решена инстанца игре *Три суседне*.

##### 3.1.1 Ограничења

Како би СМТ решавачем генерисали решења биће нам потребно кодирање претходно наведених услова. Таблу игре ћемо кодирати користећи  $n \times n$  променљивих које ћемо означавати са  $x_{i,j}$ .



Слика 2: Почетно стање игре



Слика 3: Задовољено стање игре

$$x_{i,j} = \begin{cases} -1, & \text{за бела поља} \\ 0, & \text{за сива поља} \\ 1, & \text{за плава поља} \end{cases}$$

При чему важи:

$$I = \{0, 1, \dots, n-1\}$$

$$J = \{0, 1, \dots, n-1\}$$

$$T = \{-2, -1, 0, 1, 2\}$$

$$i \in I, j \in J$$

**Услов 1** Кодирање услова 1 изводимо захтевом да збир три суседна поља у врсти мора припадати скупу  $T$ . Како имамо  $n$  врсти, при чему по свакој врсти имамо  $n-2$  ограничења, добијамо  $n(n-2)$  ограничења.

$$x_{i,j} + x_{i,j+1} + x_{i,j+2} \in T$$

$$i \in \{0, 1, \dots, n-1\}$$

$$j \in \{0, 1, \dots, n-3\}$$

**Услов 2** Услов 2 је симетричан услову 1 тако да се добија још  $n(n-2)$  додатних ограничења облика:

$$x_{i,j} + x_{i+1,j} + x_{i+2,j} \in T$$

$$i \in \{0, 1, \dots, n-3\}$$

$$j \in \{0, 1, \dots, n-1\}$$

**Услов 3** Намећемо ограничење да збир у свакој врсти мора бити 0. Како променљиве  $x_{i,j}$  имају вредност 1 или -1, ограничење имплицира једнак број плавих и белих поља у врсти. Ако је број врсти  $n$  тиме добијамо још  $n$  ограничења облика:

$$x_{0,0} + x_{0,1} + \dots + x_{0,n-1} = 0$$

$$x_{1,0} + x_{1,1} + \dots + x_{1,n-1} = 0$$

...

$$x_{n-1,0} + x_{n-1,1} + \dots + x_{n-1,n-1} = 0$$

**Услов 4** Услов 4 је симетричан услову 3 те добијамо додатних  $n$  ограничења облика:

$$x_{0,0} + x_{0,1} + \dots + x_{0,n-1} = 0$$

$$x_{0,1} + x_{1,1} + \dots + x_{n-1,1} = 0$$

...

$$x_{0,n-1} + x_{1,n-1} + \dots + x_{n-1,n-1} = 0$$

**Услови домена** Решавачу је потребно наметнути дозвољене вредности за сваку од променљивих. Како решавање логичке игре започињемо од унапред задатог стања (нека поља су већ обојена и не могу се мењати), имамо две врсте ограничења.

Оригинално задата плава и бела поља, решавачу намећемо вредности променљивих 1 или -1. Ако је на пример поље (0, 3) обојено у плаво, а поље (3, 2) у бело мора важити:

$$x_{0,3} = 1$$

$$x_{3,2} = -1$$

За свако сиво обојено поље  $x_{i,j}$ , намећемо ограничења да вредности могу бити или -1 или 1.

$$x_{i,j} \neq 0 \wedge -1 \geq x_{i,j} \leq 1$$

Како имамо  $n \times n$  променљивих, добијамо још  $n \times n$  услова.

### 3.1.2 Питање јединствености решења

Како би се показала и доказала јединственост решења, може се користити решавач да се добију вредности променљивих, а да се након тога додају ограничења да променљиве не могу узети вредности које представљају решења. Овде треба бити опрезан и приметити да само оригинална сива поља треба ограничити, а плава и бела (из почетног стања игре) не.

## 3.2 Yices program

Написан је с++ програм који за дато почетно стање игре генерише претходно наведена ограничења у синтакси погодној за yices CMT решавач. Следи генерисани код за једну од инстанци игре.

```
(set-logic QF_LIA)
(declare-fun x0_0 () Int)
(declare-fun x0_1 () Int)
(declare-fun x0_2 () Int)
(declare-fun x0_3 () Int)
(declare-fun x0_4 () Int)
(declare-fun x0_5 () Int)
(declare-fun x1_0 () Int)
(declare-fun x1_1 () Int)
(declare-fun x1_2 () Int)
(declare-fun x1_3 () Int)
(declare-fun x1_4 () Int)
(declare-fun x1_5 () Int)
(declare-fun x2_0 () Int)
(declare-fun x2_1 () Int)
(declare-fun x2_2 () Int)
(declare-fun x2_3 () Int)
(declare-fun x2_4 () Int)
(declare-fun x2_5 () Int)
(declare-fun x3_0 () Int)
(declare-fun x3_1 () Int)
(declare-fun x3_2 () Int)
(declare-fun x3_3 () Int)
```

```

(declare-fun x3_4 () Int)
(declare-fun x3_5 () Int)
(declare-fun x4_0 () Int)
(declare-fun x4_1 () Int)
(declare-fun x4_2 () Int)
(declare-fun x4_3 () Int)
(declare-fun x4_4 () Int)
(declare-fun x4_5 () Int)
(declare-fun x5_0 () Int)
(declare-fun x5_1 () Int)
(declare-fun x5_2 () Int)
(declare-fun x5_3 () Int)
(declare-fun x5_4 () Int)
(declare-fun x5_5 () Int)
(assert
  (and
    ;; Ограничења домена
    (and (<= (- 1) x0_0) (>= 1 x0_0) (distinct 0 x0_0))
    (and (<= (- 1) x0_1) (>= 1 x0_1) (distinct 0 x0_1))
    (= x0_2 1)
    (and (<= (- 1) x0_3) (>= 1 x0_3) (distinct 0 x0_3))
    (and (<= (- 1) x0_4) (>= 1 x0_4) (distinct 0 x0_4))
    (and (<= (- 1) x0_5) (>= 1 x0_5) (distinct 0 x0_5))
    (and (<= (- 1) x1_0) (>= 1 x1_0) (distinct 0 x1_0))
    (= x1_1 1)
    (and (<= (- 1) x1_2) (>= 1 x1_2) (distinct 0 x1_2))
    (and (<= (- 1) x1_3) (>= 1 x1_3) (distinct 0 x1_3))
    (= x1_4 (- 1))
    (and (<= (- 1) x1_5) (>= 1 x1_5) (distinct 0 x1_5))
    (and (<= (- 1) x2_0) (>= 1 x2_0) (distinct 0 x2_0))
    (= x2_1 (- 1))
    (and (<= (- 1) x2_2) (>= 1 x2_2) (distinct 0 x2_2))
    (and (<= (- 1) x2_3) (>= 1 x2_3) (distinct 0 x2_3))
    (and (<= (- 1) x2_4) (>= 1 x2_4) (distinct 0 x2_4))
    (and (<= (- 1) x2_5) (>= 1 x2_5) (distinct 0 x2_5))
    (= x3_0 (- 1))
    (and (<= (- 1) x3_1) (>= 1 x3_1) (distinct 0 x3_1))
    (= x3_2 1)
    (and (<= (- 1) x3_3) (>= 1 x3_3) (distinct 0 x3_3))
    (= x3_4 (- 1))
    (and (<= (- 1) x3_5) (>= 1 x3_5) (distinct 0 x3_5))
    (and (<= (- 1) x4_0) (>= 1 x4_0) (distinct 0 x4_0))
    (and (<= (- 1) x4_1) (>= 1 x4_1) (distinct 0 x4_1))
    (and (<= (- 1) x4_2) (>= 1 x4_2) (distinct 0 x4_2))
    (and (<= (- 1) x4_3) (>= 1 x4_3) (distinct 0 x4_3))
    (and (<= (- 1) x4_4) (>= 1 x4_4) (distinct 0 x4_4))
    (= x4_5 (- 1))
    (and (<= (- 1) x5_0) (>= 1 x5_0) (distinct 0 x5_0))
    (= x5_1 1)
    (= x5_2 1)
    (and (<= (- 1) x5_3) (>= 1 x5_3) (distinct 0 x5_3))
    (= x5_4 1)
    (and (<= (- 1) x5_5) (>= 1 x5_5) (distinct 0 x5_5))
  )
)

```

```
; Y istoj vrsti tri susjedna poља ne smeју biti ste boје
(and (> (+ x0_0 x0_1 x0_2) (- 3))(< (+ x0_0 x0_1 x0_2) 3))
(and (> (+ x0_0 x0_1 x0_2) (- 3))(< (+ x0_0 x0_1 x0_2) 3))
(and (> (+ x0_1 x0_2 x0_3) (- 3))(< (+ x0_1 x0_2 x0_3) 3))
(and (> (+ x0_1 x0_2 x0_3) (- 3))(< (+ x0_1 x0_2 x0_3) 3))
(and (> (+ x0_2 x0_3 x0_4) (- 3))(< (+ x0_2 x0_3 x0_4) 3))
(and (> (+ x0_2 x0_3 x0_4) (- 3))(< (+ x0_2 x0_3 x0_4) 3))
(and (> (+ x0_3 x0_4 x0_5) (- 3))(< (+ x0_3 x0_4 x0_5) 3))
(and (> (+ x0_3 x0_4 x0_5) (- 3))(< (+ x0_3 x0_4 x0_5) 3))
(and (> (+ x1_0 x1_1 x1_2) (- 3))(< (+ x1_0 x1_1 x1_2) 3))
(and (> (+ x1_0 x1_1 x1_2) (- 3))(< (+ x1_0 x1_1 x1_2) 3))
(and (> (+ x1_1 x1_2 x1_3) (- 3))(< (+ x1_1 x1_2 x1_3) 3))
(and (> (+ x1_1 x1_2 x1_3) (- 3))(< (+ x1_1 x1_2 x1_3) 3))
(and (> (+ x1_2 x1_3 x1_4) (- 3))(< (+ x1_2 x1_3 x1_4) 3))
(and (> (+ x1_2 x1_3 x1_4) (- 3))(< (+ x1_2 x1_3 x1_4) 3))
(and (> (+ x1_3 x1_4 x1_5) (- 3))(< (+ x1_3 x1_4 x1_5) 3))
(and (> (+ x1_3 x1_4 x1_5) (- 3))(< (+ x1_3 x1_4 x1_5) 3))
(and (> (+ x2_0 x2_1 x2_2) (- 3))(< (+ x2_0 x2_1 x2_2) 3))
(and (> (+ x2_0 x2_1 x2_2) (- 3))(< (+ x2_0 x2_1 x2_2) 3))
(and (> (+ x2_1 x2_2 x2_3) (- 3))(< (+ x2_1 x2_2 x2_3) 3))
(and (> (+ x2_1 x2_2 x2_3) (- 3))(< (+ x2_1 x2_2 x2_3) 3))
(and (> (+ x2_2 x2_3 x2_4) (- 3))(< (+ x2_2 x2_3 x2_4) 3))
(and (> (+ x2_2 x2_3 x2_4) (- 3))(< (+ x2_2 x2_3 x2_4) 3))
(and (> (+ x2_3 x2_4 x2_5) (- 3))(< (+ x2_3 x2_4 x2_5) 3))
(and (> (+ x2_3 x2_4 x2_5) (- 3))(< (+ x2_3 x2_4 x2_5) 3))
(and (> (+ x3_0 x3_1 x3_2) (- 3))(< (+ x3_0 x3_1 x3_2) 3))
(and (> (+ x3_0 x3_1 x3_2) (- 3))(< (+ x3_0 x3_1 x3_2) 3))
(and (> (+ x3_1 x3_2 x3_3) (- 3))(< (+ x3_1 x3_2 x3_3) 3))
(and (> (+ x3_1 x3_2 x3_3) (- 3))(< (+ x3_1 x3_2 x3_3) 3))
(and (> (+ x3_2 x3_3 x3_4) (- 3))(< (+ x3_2 x3_3 x3_4) 3))
(and (> (+ x3_2 x3_3 x3_4) (- 3))(< (+ x3_2 x3_3 x3_4) 3))
(and (> (+ x3_3 x3_4 x3_5) (- 3))(< (+ x3_3 x3_4 x3_5) 3))
(and (> (+ x3_3 x3_4 x3_5) (- 3))(< (+ x3_3 x3_4 x3_5) 3))
(and (> (+ x4_0 x4_1 x4_2) (- 3))(< (+ x4_0 x4_1 x4_2) 3))
(and (> (+ x4_0 x4_1 x4_2) (- 3))(< (+ x4_0 x4_1 x4_2) 3))
(and (> (+ x4_1 x4_2 x4_3) (- 3))(< (+ x4_1 x4_2 x4_3) 3))
(and (> (+ x4_1 x4_2 x4_3) (- 3))(< (+ x4_1 x4_2 x4_3) 3))
(and (> (+ x4_2 x4_3 x4_4) (- 3))(< (+ x4_2 x4_3 x4_4) 3))
(and (> (+ x4_2 x4_3 x4_4) (- 3))(< (+ x4_2 x4_3 x4_4) 3))
(and (> (+ x4_3 x4_4 x4_5) (- 3))(< (+ x4_3 x4_4 x4_5) 3))
(and (> (+ x4_3 x4_4 x4_5) (- 3))(< (+ x4_3 x4_4 x4_5) 3))
(and (> (+ x5_0 x5_1 x5_2) (- 3))(< (+ x5_0 x5_1 x5_2) 3))
(and (> (+ x5_0 x5_1 x5_2) (- 3))(< (+ x5_0 x5_1 x5_2) 3))
(and (> (+ x5_1 x5_2 x5_3) (- 3))(< (+ x5_1 x5_2 x5_3) 3))
(and (> (+ x5_1 x5_2 x5_3) (- 3))(< (+ x5_1 x5_2 x5_3) 3))
(and (> (+ x5_2 x5_3 x5_4) (- 3))(< (+ x5_2 x5_3 x5_4) 3))
(and (> (+ x5_2 x5_3 x5_4) (- 3))(< (+ x5_2 x5_3 x5_4) 3))
(and (> (+ x5_3 x5_4 x5_5) (- 3))(< (+ x5_3 x5_4 x5_5) 3))
(and (> (+ x5_3 x5_4 x5_5) (- 3))(< (+ x5_3 x5_4 x5_5) 3))
```



```

(and (> (+ x3_0 x4_0 x5_0) (- 3))(< (+ x3_0 x4_0 x5_0) 3))
(and (> (+ x0_1 x1_1 x2_1) (- 3))(< (+ x0_1 x1_1 x2_1) 3))
(and (> (+ x1_1 x2_1 x3_1) (- 3))(< (+ x1_1 x2_1 x3_1) 3))
(and (> (+ x2_1 x3_1 x4_1) (- 3))(< (+ x2_1 x3_1 x4_1) 3))
(and (> (+ x3_1 x4_1 x5_1) (- 3))(< (+ x3_1 x4_1 x5_1) 3))
(and (> (+ x0_2 x1_2 x2_2) (- 3))(< (+ x0_2 x1_2 x2_2) 3))
(and (> (+ x1_2 x2_2 x3_2) (- 3))(< (+ x1_2 x2_2 x3_2) 3))
(and (> (+ x2_2 x3_2 x4_2) (- 3))(< (+ x2_2 x3_2 x4_2) 3))
(and (> (+ x3_2 x4_2 x5_2) (- 3))(< (+ x3_2 x4_2 x5_2) 3))
(and (> (+ x0_3 x1_3 x2_3) (- 3))(< (+ x0_3 x1_3 x2_3) 3))
(and (> (+ x1_3 x2_3 x3_3) (- 3))(< (+ x1_3 x2_3 x3_3) 3))
(and (> (+ x2_3 x3_3 x4_3) (- 3))(< (+ x2_3 x3_3 x4_3) 3))
(and (> (+ x3_3 x4_3 x5_3) (- 3))(< (+ x3_3 x4_3 x5_3) 3))
(and (> (+ x0_4 x1_4 x2_4) (- 3))(< (+ x0_4 x1_4 x2_4) 3))
(and (> (+ x1_4 x2_4 x3_4) (- 3))(< (+ x1_4 x2_4 x3_4) 3))
(and (> (+ x2_4 x3_4 x4_4) (- 3))(< (+ x2_4 x3_4 x4_4) 3))
(and (> (+ x3_4 x4_4 x5_4) (- 3))(< (+ x3_4 x4_4 x5_4) 3))
(and (> (+ x0_5 x1_5 x2_5) (- 3))(< (+ x0_5 x1_5 x2_5) 3))
(and (> (+ x1_5 x2_5 x3_5) (- 3))(< (+ x1_5 x2_5 x3_5) 3))
(and (> (+ x2_5 x3_5 x4_5) (- 3))(< (+ x2_5 x3_5 x4_5) 3))
(and (> (+ x3_5 x4_5 x5_5) (- 3))(< (+ x3_5 x4_5 x5_5) 3))

;; За сваку врсту мора постојати једнак број плавих и белих поља.
(= 0 (+ x0_0 x0_1 x0_2 x0_3 x0_4 x0_5))
(= 0 (+ x1_0 x1_1 x1_2 x1_3 x1_4 x1_5))
(= 0 (+ x2_0 x2_1 x2_2 x2_3 x2_4 x2_5))
(= 0 (+ x3_0 x3_1 x3_2 x3_3 x3_4 x3_5))
(= 0 (+ x4_0 x4_1 x4_2 x4_3 x4_4 x4_5))
(= 0 (+ x5_0 x5_1 x5_2 x5_3 x5_4 x5_5))

;; За сваку колону мора постојати једнак број плавих и белих поља.
(= 0 (+ x0_0 x1_0 x2_0 x3_0 x4_0 x5_0))
(= 0 (+ x0_1 x1_1 x2_1 x3_1 x4_1 x5_1))
(= 0 (+ x0_2 x1_2 x2_2 x3_2 x4_2 x5_2))
(= 0 (+ x0_3 x1_3 x2_3 x3_3 x4_3 x5_3))
(= 0 (+ x0_4 x1_4 x2_4 x3_4 x4_4 x5_4))
(= 0 (+ x0_5 x1_5 x2_5 x3_5 x4_5 x5_5))
))
(check-sat)
(get-value (
  x0_0 x0_1 x0_2 x0_3 x0_4 x0_5
  x1_0 x1_1 x1_2 x1_3 x1_4 x1_5
  x2_0 x2_1 x2_2 x2_3 x2_4 x2_5
  x3_0 x3_1 x3_2 x3_3 x3_4 x3_5
  x4_0 x4_1 x4_2 x4_3 x4_4 x4_5
  x5_0 x5_1 x5_2 x5_3 x5_4 x5_5
))
)
(exit)

Покретање речавање даје да је проблем задовољив и даје нам вред-
ности променљивих  $x_{i,j}$  које представљају распоред поља.

sat
((x0_0 1)

```

```

(x0_1 (- 1))
(x0_2 1)
(x0_3 (- 1))
(x0_4 (- 1))
(x0_5 1)
(x1_0 (- 1))
(x1_1 1)
(x1_2 (- 1))
(x1_3 1)
(x1_4 (- 1))
(x1_5 1)
(x2_0 1)
(x2_1 (- 1))
(x2_2 (- 1))
(x2_3 1)
(x2_4 1)
(x2_5 (- 1))
(x3_0 (- 1))
(x3_1 1)
(x3_2 1)
(x3_3 (- 1))
(x3_4 (- 1))
(x3_5 1)
(x4_0 1)
(x4_1 (- 1))
(x4_2 (- 1))
(x4_3 1)
(x4_4 1)
(x4_5 (- 1))
(x5_0 (- 1))
(x5_1 1)
(x5_2 1)
(x5_3 (- 1))
(x5_4 1)
(x5_5 (- 1)))

```

## 4 Закључак

У делу [3](#) приказан је пример решавања логичке игре користећи СМТ решавач. Блискост логичких игара области математичке логике природно доводи до елегантног кодирања ограничења логичке игре. Такође, ефикасност при решавању и могућност добијања решења без потребе за развојем комплексног алгоритма чине да СМТ решавачи (и САТ решавачи) буду алат број један при раду са логичким играма.

## Литература

- [1] Leonardo Alt, Sepideh Asadi, Hana Chockler, Karine Even Mendoza, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina. Hifrog: Smt-based function summarization for software verification. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Uppsala, 2017. Springer, Springer.
- [2] George B. Dantzig and Mukund N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.
- [3] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.
- [4] Rajdeep Mukherjee, Daniel Kroening, and Tom Melham. Hardware verification using software analyzers. In *2015 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2015, Montpellier, France, July 8-10, 2015*, pages 7–12. IEEE Computer Society, 2015.
- [5] Rajdeep Mukherjee, Mitra Purandare, Raphael Polig, and Daniel Kroening. Formal techniques for effective co-verification of hardware/software co-designs. In *Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, June 18-22, 2017*, pages 35:1–35:6, 2017.
- [6] Rajdeep Mukherjee, Peter Schrammel, Leopold Haller, Daniel Kroening, and Tom Melham. Lifting CDCL to template-based abstract domains for program verification. *CoRR*, abs/1707.02011, 2017.
- [7] Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 2001.