Predrag Janičić

# SIMBOLIČKO IZRAČUNAVANJE

*PROBLEM SAT I REŠAVANJE PROBLEMA SVOĐENJEM NA SAT*

**Beograd**
**2023.**

Edmund Clarke, dobitnik Tjuringove nagrade 2007:
„SAT solving is a key technology for 21st century computer science.“

Donald Knuth, dobitnik Tjuringove nagrade 1974:
„SAT is evidently a killer app, because it is key to the solution of so many other problems.“

Stephen Cook, dobitnik Tjuringove nagrade 1982:
„The SAT problem is at the core of arguably the most fundamental question in computer science: What makes a problem hard?“

# Sadržaj


**Sadržaj** **3**

**1 Resursi** **7**

**2 Iskaza logika i problem SAT** **9**
2.1 Sintaksa i semantika iskazne logike . . . 9
2.2 Istinitosne tablice i odlučivost problema valjanosti i zadovoljivosti . . . 10
2.3 Logičke posledice i logički ekvivalentne formule . . . 11
2.4 Klauze . . . 12
2.5 Konjunktivna normalna forma i disjunktivna normalna forma . . . 12

**3 Prevođenje iskazne formule u instancu SAT problema** **15**
3.1 KNF procedura . . . 15
3.2 Cejtinova transformacija . . . 16

**4 Rešavanje instanci problema SAT** **19**
4.1 DPLL procedura . . . 19
4.2 Šening algoritam . . . 21
4.3 Savremeni SAT rešavači . . . 21
4.4 Portfolio SAT rešavači . . . 22
4.5 DIMACS-CNF format . . . 23
4.6 SAT takmičenja . . . 23

**5 Problem SAT i složenost izračunavanja** **25**
5.1 Klase složenosti . . . 25
5.2 NP-kompletnost i problem SAT . . . 26
5.3 Primeri svođenja . . . 28
5.4 Složenost problema srodnih problemu SAT . . . 31

**6 Problem SAT i promena faze** **33**
6.1 Promena faze u fizičkim sistemima . . . 33
6.2 Promena faze u problemu random $k$-SAT . . . 35

**7 Rešavanje problema svođenjem na SAT** **39**
7.1 SAT kao CSP . . . 39
7.2 Faze rešavanja svođenjem na SAT . . . 39

**8 Sistem URSA** **47**
8.1 Sistem URSA i imperativno-deklarativna programska paradigma . . . 47
8.2 Uvodni primer: Bal . . . 47
8.3 Uvodni primer: sabiranje . . . 48
8.4 Uvodni primer: bojenje kuća . . . 48
8.5 Sintaksa jezika URSA . . . 49
8.6 Određivanje maksimuma . . . 51
8.7 Generisanje permutacija . . . 53

8.8 Problem $n$ dama . . . 54
8.9 Problem SEND MORE MONEY . . . 56
8.10 Vežbanje 1: IMO 1960/1 . . . 57
8.11 Vežbanje 2: IMO 1981/3 . . . 58
8.12 Vežbanje 3: IMO 1977/5 . . . 59
8.13 Problem skakačev put . . . 60
8.14 Mali primerak verifikacije softvera 1 . . . 62
8.15 Mali primerak verifikacije softvera 2 . . . 63
8.16 Množenje polinoma . . . 64
8.17 Vežbanje 4: IMO 1963/6 . . . 67
8.18 Vežbanje 5: šahovska KK završnica . . . 68
8.19 Vežbanje 6: šahovska KRK završnica . . . 69
8.20 Problem generisanje pseudoslučajnih brojeva . . . 70
8.21 Analiza blok algoritama za kriptovanje . . . 71

**Bibliografija** **75**

# Predgovor

Ovo su beleške za predavanja za predmet „Simboličko izračunavanje“ na smeru Informatika na Matematičkom fakultetu Univerziteta u Beogradu. Predmet „Simboličko izračunavanje“ čine četiri dela od kojih je jedan „Problem SAT i rešavanje problema svođenjem na SAT“.

Beograd, februar 2023.

Predrag Janičić

*Glava 1*

# Resursi

- CaDiCaL SAT rešavač: https://github.com/arminbiere/cadical
- URSA, rešavač ograničenja zasnovan na problemu SAT: https://github.com/janicicpredrag/URSA

*Glava 2*

# Iskaza logika i problem SAT

Da bi moglo da se govori o ispitivanju zadovoljivosti formula iskazne logike i rešavanju problema SAT, potrebno je najpre definisati sintaksu, semantiku i osnovne pojmove iskazne logike.

## 2.1 Sintaksa i semantika iskazne logike

Skup iskaznih formula obično se definiše za fiksiran, prebrojiv skup *iskaznih promenljivih* (*iskaznih varijabli*) ili *iskaznih slova* $P$:

**Definicija 2.1** (Skup iskaznih formula)**.**

- *Iskazne promenljive (elementi skupa $P$) i logičke konstante ($\top$ (te) i $\bot$ (nete) su iskazne formule;*
- *Ako su $A$ i $B$ iskazne formule, onda su iskazne formule i objekti dobijeni kombinovanjem ovih formula logičkim veznicima unarnog* — negacija *i binarnih* — konjunkcija, disjunkcija, implikacija, ekvivalencija*;*
- *Iskazne formule mogu biti dobijene samo na navedene načine.*

Elemente skupa $P$ i logičke konstante zovemo *atomičkim iskaznim formulama. Literal* je iskazna formula koja je ili atomička iskazna formula ili negacija atomičke iskazne formule. Ako su dve iskazne formule $A$ i $B$ identične (tj. ako su, zapisane u konkretnoj sintaksi, jednake kao nizovi simbola), onda to zapisujemo $A = B$, a inače pišemo $A \neq B$.

Uobičajeno je da se negacija zapisuje kao $\neg$, konjunkcija kao $\wedge$, disjunkcija kao $\vee$, implikacija kao $\Rightarrow$ i ekvivalencija kao $\Leftrightarrow$. U zapisu u vidu nizova simbola, ako su $A$ i $B$ iskazne formule, onda su iskazne formule i $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ i $(A \Leftrightarrow B)$.

Istinitosna vrednost iskazne formule zavisi od istinitosnih vrednosti promenljivih koje se u njoj pojavljuju. Funkcije koje pridružuju istinitosnu vrednost promenljivim (tj. funkcije $v$ iz $P$ u $\{0, 1\}$) zovemo *valuacijama* (eng. *valuation, assignment*). Svaka valuacija $v$ proširuje se, u skladu sa narednom definicijom, do jedne funkcije $I_v$ koju zovemo *interpretacijom za valuaciju* $v$ i koja pridružuje (jedinstvene) istinitosne vrednosti formulama (tj. preslikava skup iskaznih formula u skup $\{0, 1\}$) (ili { *netačno* ili *tačno* }).

**Definicija 2.2** (Interpretacija)**.** Interpretacija $I_v$ *za valuaciju $v$ definiše se na sledeći način:*

- $I_v(\top) = 1$ *i* $I_v(\bot) = 0$*;*
- $I_v(p) = v(p)$*, za svaki element $p$ skupa $P$;*
- $I_v(\neg A) = \begin{cases} 1, & \textit{ako je } I_v(A) = 0 \\ 0, & \textit{inače} \end{cases}$
- $I_v(A \wedge B) = \begin{cases} 1, & \textit{ako je } I_v(A) = 1 \textit{ i } I_v(B) = 1 \\ 0, & \textit{inače} \end{cases}$
- $I_v(A \vee B) = \begin{cases} 0, & \textit{ako je } I_v(A) = 0 \textit{ i } I_v(B) = 0 \\ 1, & \textit{inače} \end{cases}$

- $I_v(A \Rightarrow B) = \begin{cases} 0, & \textit{ako je } I_v(A) = 1 \textit{ i } I_v(B) = 0 \\ 1, & \textit{inače} \end{cases}$
- $I_v(A \Leftrightarrow B) = \begin{cases} 1, & \textit{ako je } I_v(A) = I_v(B) \\ 0, & \textit{inače} \end{cases}$

**Definicija 2.3** (Zadovoljivost, valjanost, kontradiktornost, porecivost)**.** *Iskazna formula $A$ je:*

- zadovoljiva *(eng.* satisfiable*), ako postoji valuacija $v$ u kojoj je $A$ tačna (i tada se kaže da je $v$* model *za $A$);*
- valjana *(eng.* valid*) ili* tautologija *(eng.* tautology*), ako je svaka valuacija $v$ model za $A$ i to zapisujemo $\models A$;*
- nezadovoljiva *(eng.* unsatisfiable*) ili* kontradikcija *(eng.* contradictory*), ako ne postoji valuacija u kojoj je tačna;*
- poreciva *(eng.* falsifiable*), ako postoji valuacija u kojoj nije tačna.*

**Definicija 2.4** (Zadovoljivost i kontradiktornost skupa formula)**.** *Skup iskaznih formula $\Gamma$ je*

- zadovoljiv*, ako postoji valuacija $v$ u kojoj je svaka formula iz $\Gamma$ tačna. Za takvu valuaciju $v$ kaže se da je* model *za $\Gamma$.*
- nezadovoljiv *ili* kontradiktoran*, ako ne postoji valuacija u kojoj je svaka formula iz $\Gamma$ tačna.*

Naredna teorema govori da uslov koji rešenje nekog problema mora da zadovolji može da se razmatra ne samo kao skup svih pojedinačnih poduslova, već i kao konjunkcija formula koje odgovaraju tim poduslovima.

**Teorema 2.1.** *Valuacija $v$ je model skupa formula $\{A_1, \ldots, A_n\}$ ako i samo ako je $v$ model formule $A_1 \wedge \ldots \wedge A_n$.*

## 2.2 Istinitosne tablice i odlučivost problema valjanosti i zadovoljivosti

U praktičnim primenama iskazne logike, centralni problem je ispitivanje da li je neka iskazna formula tautologija, da li je zadovoljiva i slično. Početno pitanje je da li su ti problemi uopšte odlučivi, tj. da li postoje algoritmi koji *uvek* mogu da daju odgovor na njih. Odgovor je potvrdan i istinosne tablice daju jedan jednostavan algoritam za ispitivanje zadovoljivosti iskaznih formula, ali i za ispitivanje tautologičnosti, porecivosti i nezaodovoljivosti iskaznih formula. Naime, ovi problemi su blisko povezani i bilo koja tri mogu se lako svesti na četvrti. To znači da je dovoljno da imamo efikasan algoritam za jedan od ova četiri problema. U nastavku ćemo se fokusirati na rešavanje problema zadovoljivosti. Algoritam za ispitivanje zadovoljivosti onda možemo iskoristiti i za preostala tri problema: formula $A$ je tautologija akko $\neg A$ nije zadovoljiva, $A$ je poreciva akko je $\neg A$ zadovoljiva, a $A$ je kontradikcija akko $A$ nije zadovoljiva.

Na osnovu definicije interpretacije, može se konstruisati istinitosna tablica za proizvoljnu iskaznu formulu. U istinitosnoj tablici za neku formulu, svakoj vrsti odgovara jedna valuacija iskaznih slova koja se pojavljuju u toj formuli. Svakoj koloni odgovara jedna potformula te formule. Ukoliko iskazna formula $A$ sadrži iskazne promenljive $p_1, p_2, \ldots, p_n$, onda istinitosna tablica treba da sadrži sve moguće valuacije za ovaj skup promenljivih (valuacije za druge promenljive nisu relevantne). Takvih valuacija ima $2^n$. U zavisnosti od vrednosti iskaznih promenljivih, izračunavaju se vrednosti potformula razmatrane iskazne formule, sve do sâme te formule. Ako su u koloni koja odgovara sâmoj iskaznoj formuli sve vrednosti jednake 1, onda je formula tautologija. Ako je bar jedna vrednost jednaka 1, formula je zadovoljiva. Ako je bar jedna vrednost jednaka 0, formula je poreciva. Ako su sve vrednosti jednake 0, formula je kontradikcija. Opisani algoritam, zasnovan na istinitosnim tablicama, naivan je i potpuno neupotrebljiv za realne probleme. Za rešavanje teških realnih problema potrebno je razviti znatno efikasnije algoritme.

**Primer 2.1.** *Date su tri kutije, zna se da je tačno u jednoj od njih zlato. Na njima piše:*
*na kutiji 1: „zlato nije ovde“*

*na kutiji 2: „zlato nije ovde“*
*na kutiji 3: „zlato je u kutiji 2“*
*Ako se zna da je tačno jedna tvrdnja tačna, treba pogoditi gde je zlato.*
*Neka iskazne promenljive $b_1$, $b_2$ i $b_3$ odgovaraju tvrdnjama „zlato je u kutiji 1“, „zlato je u kutiji 2“ i „zlato je u kutiji 3“.*
*Uslov da je tačno u jednoj od njih zlato može se opisati sledećom formulom:*
$A = (b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge \neg b_2 \wedge b_3)$
*Uslov da je tačno jedna ispisana tvrdnja tačna, može se opisati sledećom formulom:*
$B = (\neg b_1 \wedge \neg\neg b_2 \wedge \neg b_2) \vee (\neg\neg b_1 \wedge \neg b_2 \wedge \neg b_2) \vee (\neg\neg b_1 \wedge \neg\neg b_2 \wedge b_2)$
*Potrebno je pronaći valuaciju u kojoj su obe formule $A$ i $B$ tačne:*

| $b_1$ | $b_2$ | $b_3$ | $A$ | $B$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |

*Postoji samo jedna valuacija $v$ u kojoj su obe formule tačne. U njoj važi $I_v(b_1) = 1$, te je zlato u prvoj kutiji.*

## 2.3 Logičke posledice i logički ekvivalentne formule

Često je veoma važno pitanje da li je neki iskaz posledica nekih drugih iskaza. Ovo pitanje može se opisati u terminima pojma *logičke posledice*.

**Definicija 2.5** (Logička posledica i logička ekvivalencija)**.** *Ako je svaki model za skup iskaznih formula $\Gamma$ istovremeno i model za iskaznu formulu $A$, onda se kaže da je $A$* logička posledica *(eng.* logical consequence*) skupa $\Gamma$ i piše se $\Gamma \models A$.*

*Ako je svaki model iskazne formule $A$ model i iskazne formule $B$ i obratno (tj. ako važi $\{A\} \models B$ i $\{B\} \models A$), onda se kaže se da su formule $A$ i $B$* logički ekvivalentne *(eng.* logically equivalent*) i piše se $A \equiv B$.*

**Primer 2.2.** *Neke od logičkih ekvivalencija (ili, preciznije, neke od shema logičkih ekvivalencija) su:*

| | | | |
|---|---|---|---|
| $\neg\neg A$ | $\equiv$ | $A$ | *zakon dvojne negacije* |
| $A \vee \neg A$ | $\equiv$ | $\top$ | *zakon isključenja trećeg* |
| $A \wedge A$ | $\equiv$ | $A$ | *zakon idempotencije za $\wedge$* |
| $A \vee A$ | $\equiv$ | $A$ | *zakon idempotencije za $\vee$* |
| $A \wedge B$ | $\equiv$ | $B \wedge A$ | *zakon komutativnosti za $\wedge$* |
| $A \vee B$ | $\equiv$ | $B \vee A$ | *zakon komutativnosti za $\vee$* |
| $A \wedge (B \wedge C)$ | $\equiv$ | $(A \wedge B) \wedge C$ | *zakon asocijativnosti za $\wedge$* |
| $A \vee (B \vee C)$ | $\equiv$ | $(A \vee B) \vee C$ | *zakon asocijativnosti za $\vee$* |
| $A \wedge (A \vee B)$ | $\equiv$ | $A$ | *zakon apsorpcije* |
| $A \vee (A \wedge B)$ | $\equiv$ | $A$ | *zakon apsorpcije* |
| $A \vee (B \wedge C)$ | $\equiv$ | $(A \vee B) \wedge (A \vee C)$ | *zakon distributivnosti $\vee$ u odnosu na $\wedge$* |
| $(B \wedge C) \vee A$ | $\equiv$ | $(B \vee A) \wedge (C \vee A)$ | *zakon distributivnosti $\vee$ u odnosu na $\wedge$* |
| $\neg(A \wedge B)$ | $\equiv$ | $\neg A \vee \neg B$ | *De Morganov zakon* |
| $\neg(A \vee B)$ | $\equiv$ | $\neg A \wedge \neg B$ | *De Morganov zakon* |
| $A \wedge \top$ | $\equiv$ | $A$ | *zakon konjunkcije sa tautologijom* |
| $A \vee \top$ | $\equiv$ | $\top$ | *zakon disjunkcije sa tautologijom* |
| $A \wedge \bot$ | $\equiv$ | $\bot$ | *zakon konjunkcije sa kontradikcijom* |
| $A \vee \bot$ | $\equiv$ | $A$ | *zakon disjunkcije sa kontradikcijom* |

**Primer 2.3.** *Tvrđenje „Boban živi u Kruševcu“ je logička posledica skupa tvrđenja { „Boban živi u Beogradu ili Boban živi u Kruševcu“, „Boban ne živi u Beogradu” }. Zaista, ako je valuacija $v$ proizvoljan model formula „Boban živi u Beogradu ili Boban živi u Kruševcu“ i „Boban ne živi u Beogradu”, na osnovu definicije semantike, u interpretaciji $I_v$ mora da je tačan iskaz „Boban živi u Beogradu“ ili iskaz “Boban živi u Kruševcu“. Prva mogućnost otpada zbog uslova „Boban ne živi u Beogradu”, pa mora da je tačan iskaz „Boban živi u Kruševcu“.*

Ako ne važi $\Gamma \models A$, onda to zapisujemo $\Gamma \not\models A$.

Ako važi $\{\} \models A$, onda je svaka valuacija model za formulu $A$, tj. $A$ je valjana formula. Važi i obratno — ako je $A$ valjana formula, onda važi $\{\} \models A$. Zato umesto $\{\} \models A$, pišemo kraće $\models A$, kao što zapisujemo valjane formule.

Ako je skup $\Gamma$ kontradiktoran, onda je proizvoljna formula njegova logička posledica. Specijalno, svaka formula je logička posledica skupa $\{\bot\}$.

Ako je skup $\Gamma$ konačan, onda umesto $\{A_1, \ldots, A_n\} \models B$ pišemo kraće $A_1, \ldots, A_n \models B$.

U razmatranju skupova uslova, bilo koji uslov očigledno može biti zamenjen nekim njemu logički ekvivalentnim uslovom jer time neće biti promenjen skup modela celokupnog uslova.

**Teorema 2.2.** *Važi $A_1, \ldots, A_n \models B$ ako i samo ako je formula $A_1 \wedge \ldots \wedge A_n \Rightarrow B$ tautologija.*

**Primer 2.4.** *Ako važi „kiša pada“ i ”ako kiša pada, onda će meč biti otkazan“, onda važi i „meč će biti otkazan“. Naime, iskaz „meč će biti otkazan“ je logička posledica skupa iskaza { „kiša pada“, „ako kiša pada, onda će meč biti otkazan“ } ako i samo ako je formula*

*„kiša pada“ $\wedge$ „ako kiša pada, onda će meč biti otkazan” $\Rightarrow$ „meč će biti otkazan“*

*tautologija (što jeste ispunjeno).*

Već je rečeno da u razmatranju skupova uslova, svaki uslov može biti zamenjen nekim njemu logički ekvivalentnim uslovom (i time neće biti promenjen skup modela celokupnog uslova). Naredna teorema tvrdi i sledeće: ako se u formuli $A$ neka njena potformula zameni njoj logički ekvivalentnom formulom, biće dobijena formula koja je logički ekvivalentna formuli $A$. Ova teorema opravdava *transformisanje* iskazne formule u neki oblik pogodan za, na primer, ispitivanje zadovoljivosti.

**Teorema 2.3** (Teorema o zameni)**.** *Ako je $C \equiv D$, onda je $A[C \mapsto D] \equiv A$.*

## 2.4 Klauze

Logičke ekvivalencije navedene u primeru 2.2, pokazuju, između ostalog, da su konjunkcija i disjunkcija komutativni i asocijativni veznici. Zato možemo smatrati da konjunkcija (i disjunkcija) mogu da povezuju više od dve formule, pri čemu ne moramo da vodimo računa o njihovom poretku. Svaki član uopštene konjunkcije zovemo *konjunkt*, a svaki član uopštene disjunkcije zovemo *disjunkt*. Disjunkciju više literala (pri čemu njihov poredak nije bitan) zovemo *klauza* (eng. *clause*). Klauza je *jedinična* ako sadrži samo jedan literal.

## 2.5 Konjunktivna normalna forma i disjunktivna normalna forma

Napredne procedure za ispitivanje valjanosti ili zadovoljivosti se, zbog jednostavnosti i veće efikasnosti, obično definišu samo za neke specifične vrste iskaznih formula, za formule koje su u nekoj specifičnoj formi.

**Definicija 2.6** (Konjunktivna normalna forma)**.** *Iskazna formula je u* konjunktivnoj normalnoj formi *(*KNF*) ako je oblika*

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

*pri čemu je svaka od formula $A_i$ $(1 \leq i \leq n)$ klauza (tj. disjunkcija literala).*

**Definicija 2.7** (Disjunktivna normalna forma)**.** *Iskazna formula je u* disjunktivnoj normalnoj formi *(*DNF*) ako je oblika*

$$A_1 \vee A_2 \vee \ldots \vee A_n$$

*pri čemu je svaka od formula $A_i$ $(1 \leq i \leq n)$ konjunkcija literala.*

Ako je iskazna formula $A$ logički ekvivalentna iskaznoj formuli $B$ i iskazna formula $B$ je u konjunktivnoj (disjunktivnoj) normalnoj formi, onda se kaže da je formula $B$ konjunktivna (disjunktivna) normalna forma formule $A$. Jedna iskazna formula može da ima više različitih konjunktivnih (disjunktivnih) normalnih formi (na primer, i formula $(p \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge (q \vee s)$ i formula $(s \vee q) \wedge (p \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge (p \vee \neg p)$ su konjunktivne normalne forme formule $(p \wedge q) \vee (r \wedge s)$). Slično, jedna formula koja je u konjunktivnoj normalnoj formi može biti konjunktivna normalna forma za više iskaznih formula.

### 2.5.1 Problem SAT i srodni problemi

Za svaku iskaznu formulu postoji njena konjunktivna normalna forma i većina primena iskazne logike svodi se na ispitivanje zadovoljivosti neke formule koja je u tom, KNF obliku.

**Definicija 2.8** (Problem SAT)**.** *Problem* SAT *(od engleskog* satisfiability problem *— problem zadovoljivosti) je problem ispitivanja zadovoljivosti date iskazne formule u* KNF *obliku.*

Programi koji rešavaju instance SAT problema zovu se SAT rešavači.

**Definicija 2.9** (Problem UNSAT)**.** *Problem* UNSAT *je problem ispitivanja nezadovoljivosti date iskazne formule u* KNF *obliku.*

Postoji razlika između problema SAT i UNSAT: kada rešavač za SAT vraća rezultat „da“, onda rešavač za UNSAT vraća rezultat „ne“, ali oni mogu da rade na potpuno različite načine. Rešavač za SAT će prekinuti rad i vratiti „da“ čim otkrije da za ulaznu formulu postoji model, ali može da radi dugo dok ne pokaže da formula nije zadovoljiva. Nasuprot tome, rešavač za UNSAT vraća „ne“ ako ustanovi da za ulaznu formulu postoji model, a vraća „da“ ako uspe da utvrdi da formula nije zadovoljiva.

**Definicija 2.10** (Problem $k$-SAT)**.** *Problem $k$-*SAT *je podvrsta problema* SAT *u kojoj sve klauze imaju tačno po $k$ literala.*

**Definicija 2.11** (Problem #SAT)**.** *Problem #*SAT *je problem prebrojavanja skupa modela date* SAT *formule.*

Problemi SAT i $k$-SAT su problemi odlučivanja (traži se odgovor „da“ ili „ne“), dok je problem #SAT problem prebrojavanja.

**Definicija 2.12** (Problem MAX-SAT)**.** *Problem* MAX-SAT *je problem određivanja jedne valuacije u kojoj je najveći mogući broj klauza ulazne* SAT *formule tačan.*

Ukoliko je ulazna SAT formula zadovoljiva, onda rešavač za problem MAX-SAT vraća neku njenu valuaciju. Ukoliko ukoliko je ulazna SAT formula nezadovoljiva, onda rešavač za problem MAX-SAT vraća broj tačnih klauza u nekoj valuaciji, pri čemu je taj broj strogo manji od broja ulaznih klauza.

**Primer 2.5.** *Problem pravljenja rasporeda časova može se rešavati svođenjem na* SAT*. Međutim, često nije moguće zadovoljiti sve zadate uslove (tj. želje učenika i nastavnika). Zato, ukoliko je skup uslova nezadovoljiv, još uvek može biti korisno dobiti model u kojem je najveći mogući skup uslova zadovoljen.*

**Definicija 2.13** (Problem TEŽINSKI MAX-SAT)**.** *Problem* TEŽINSKI MAX-SAT *(engl. weighted MAX-SAT)je problem određivanja jedne valuacije u kojoj je najveći moguć zbir težina klauza ulazne* SAT *formule tačan (svim klauzama pridružene su težine).*

**Primer 2.6.** *Kao što je rečeno u primeru 2.5, problem pravljenja rasporeda časova može se rešavati svođenjem na* MAX-SAT*. Dodatno, može se smatrati da su neki uslovi važniji od drugih (na primer, učenici*

*ne treba da imaju pauze između časova), pa je onda još pogodnije rešavati ga svođenjem na* TEŽINSKI MAX-SAT*, pri čemu su svakom uslovu pridružene adekvatne težine.*

*Glava 3*

# Prevođenje iskazne formule u instancu SAT problema

Pogodno je da formula čija se zadovoljivost ispituje ima neku specifičnu sintaksičku formu, kako algoritam za ispitivanje zadovoljivosti ne bi morao da ispituje mnogo različitih slučajeva. Jedna takva forma je konjuktivna normalna forma i tu formu imaju instance SAT problema.

## 3.1 KNF procedura

Korišćenjem pogodnih ekvivalencija, svaka iskazna formula može biti transformisana u svoju konjunktivnu (disjunktivnu) normalnu formu. Transformisanje iskazne formule u konjunktivnu normalnu formu može biti opisano algoritmom prikazanim na slici 3.1. Kada se govori o „primeni neke logičke ekvivalencije“ misli se na korišćenje logičke ekvivalencije na osnovu teoreme o zameni (teorema 2.3).

**Algoritam:** KNF

**Ulaz:** Iskazna formula $F$

**Izlaz:** Konjunktivna normalna forma formule $F$

1: **dok god** je to moguće **radi**
2: primeni logičku ekvivalenciju (eliminiši veznik $\Leftrightarrow$):
$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$;

3: **dok god** je to moguće **radi**
4: primeni logičku ekvivalenciju (eliminiši veznik $\Rightarrow$):
$A \Rightarrow B \equiv \neg A \vee B$;

5: **dok god** je to moguće **radi**
6: primeni neku od logičkih ekvivalencija:
$\neg(A \wedge B) \equiv \neg A \vee \neg B$,
$\neg(A \vee B) \equiv \neg A \wedge \neg B$;

7: **dok god** je to moguće **radi**
8: primeni logičku ekvivalenciju (eliminiši višestruke veznike $\neg$):
$\neg\neg A \equiv A$;

9: **dok god** je to moguće **radi**
10: primeni neku od logičkih ekvivalencija:
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$,
$(B \wedge C) \vee A \equiv (B \vee A) \wedge (C \vee A)$.

Slika 3.1: Algoritam KNF.

Zaustavljanje algoritma KNF može se dokazati korišćenjem pogodno odabrane *mere zaustavljanja*. Za neke pojedinačne korake, može se dokazati da se zaustavljaju korišćenjem jednostavnih mera — na primer, za prvi korak algoritma, kao mera se može koristiti broj veznika $\Leftrightarrow$ u formuli. Izlazna formula logički je ekvivalentna

| vrsta formule | rezultujuće klauze |
|---|---|
| $p_A \Leftrightarrow (\neg p_B)$ | $(p_A \vee p_B) \wedge (\neg p_A \vee \neg p_B)$ |
| $p_A \Leftrightarrow (p_B \wedge p_C)$ | $(p_A \vee \neg p_B \vee \neg p_C) \wedge (\neg p_A \vee p_B) \wedge (\neg p_A \vee p_C)$ |
| $p_A \Leftrightarrow (p_B \vee p_C)$ | $(\neg p_A \vee p_B \vee p_C) \wedge (p_A \vee \neg p_B) \wedge (p_A \vee \neg p_C)$ |
| $p_A \Leftrightarrow (p_B \Rightarrow p_C)$ | $(\neg p_A \vee \neg p_B \vee p_C) \wedge (p_A \vee p_B) \wedge (p_A \vee \neg p_C)$ |
| $p_A \Leftrightarrow (p_B \Leftrightarrow p_C)$ | $(\neg p_A \vee \neg p_B \vee p_C) \wedge (\neg p_A \vee p_B \vee \neg p_C) \wedge$ $(p_A \vee p_B \vee p_C) \wedge (p_A \vee \neg p_B \vee \neg p_C)$ |

Tabela 3.1: Pravila za Cejtinovu transformaciju.

ulaznoj formuli na osnovu teoreme o zameni (teorema 2.3) i činjenice da se u algoritmu koriste samo logičke ekvivalencije.

**Teorema 3.1** (Korektnost algoritma KNF)**.** *Algoritam KNF se zaustavlja i ima sledeće svojstvo: ako je $F$ ulazna formula, onda je izlazna formula $F'$ u konjunktivnoj normalnoj formi i logički je ekvivalentna sa $F$.*

Transformisanje formule u disjunktivnu normalnu formu opisuje se algoritmom analognim algoritmu KNF.

Transformisanje formule u njenu konjunktivnu normalnu formu može da dâ formulu čija je složenost eksponencijalna u odnosu na složenost polazne formule. Na primer, transformisanjem formule

$$(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_n \wedge B_n)$$

(koja ima $n$ disjunkata) u njenu konjunktivnu normalnu formu, dobija se formula koja ima $2^n$ konjunkta.

## 3.2 Cejtinova transformacija

Zbog potencijalno ogromne izlazne formule, umesto algoritma KNF, u praksi se najčešće koristi Cejtinovo kodiranje – koje je linearno i u smislu vremena i u smislu prostora u odnosu na ulaznu formulu, ali uvodi dodatne promenljive, te zato rezultujuća formula nije logički ekvivalentna polaznoj već samo *slabo ekvivalentna*: početna formula je zadovoljiva ako i samo ako je zadovoljiva rezultujuća formula. To je za primene obično dovoljno dobro i, štaviše, iz modela za rezultujuću formulu (ukoliko oni postoje) mogu se rekonstruisati modeli za polaznu formulu. Smatraćemo da su iz polazne formule, korišćenjem adekvatnih logičkih ekvivalencija (poput $A \vee \top \equiv \top$), eliminisane sve konstante $\top$ i $\bot$. Cejtinova transformacija može se opisati na sledeći način: Neka $Sub(F)$ označava skup svih potformula formule $F$. Za svaku formulu $A$ iz $Sub(F)$ koja nije iskazna promenljiva, uvodi se nova iskazna promenljiva (*definiciona promenljiva*) $p_A$. Ako je $A$ iskazna promenljiva, onda $p_A$ označava samu formulu $A$ (i tada se $p_A$ naziva *osnovna promenljiva*). Formula $F$ se najpre transformiše u sledeću formulu (gde $\star$ označava binarni iskazni veznik iz skupa binarnih veznika koji se pojavljuju u $F$):

$$p_F \ \wedge \bigwedge_{\substack{A \in Sub(F) \\ A = B \star C}} (p_A \Leftrightarrow (p_B \star p_C)) \ \wedge \bigwedge_{\substack{A \in Sub(F) \\ A = \neg B}} (p_A \Leftrightarrow \neg p_B)$$

Lako se može dokazati da je navedena formula slabo ekvivalentna sa formulom $F$. Na kraju, navedena formula se trivijalno transformiše u KNF oblik primenom pravila iz tabele 3.1. Svaki konjunkt se transformiše u KNF formulu sa najviše četiri klauze, od kojih svaka ima najviše tri literala.

Cejtinova transformacija daje formulu čija veličina je linearna u odnosu na veličinu polazne formule. Preciznije, ako polazna formula sadrži $n$ logičkih veznika, onda će izlazna formula sadržati najviše $4n$ klauza, od kojih svaka ima najviše tri literala. To se može pokazati jednostavnim induktivnim argumentom.

**Primer 3.1.** *Data je iskazna formula $(p \wedge (q \wedge r)) \vee ((q \wedge r) \wedge \neg p)$. Definicione promenljive $p_4, p_5, p_6, p_7, p_8$ uvode se na sledeći način:*

$$\underbrace{\overbrace{(p \wedge \underbrace{(q \wedge r)}_{p_4})}^{p_6} \vee \overbrace{(\underbrace{(q \wedge r)}_{p_4} \wedge \underbrace{\neg p}_{p_5})}^{p_7}}_{p_8}$$

*Međuoblik za Cejtinovu formu je onda:*

$$p_8 \wedge (p_8 \Leftrightarrow (p_6 \vee p_7)) \wedge (p_6 \Leftrightarrow (p \wedge p_4)) \wedge$$

$$(p_7 \Leftrightarrow (p_4 \wedge p_5)) \wedge (p_4 \Leftrightarrow (q \wedge r)) \wedge (p_5 \Leftrightarrow \neg p))$$

*Konačno, izlazna* KNF *formula je:*

$$\begin{aligned}
&p_8 \wedge \\
&(\neg p_8 \vee p_6 \vee p_7) \wedge (p_8 \vee \neg p_6) \wedge (p_8 \vee \neg p_7) \wedge \\
&(p_6 \vee \neg p \vee \neg p_4) \wedge (\neg p_6 \vee p) \wedge (\neg p_6 \vee p_4) \wedge \\
&(p_7 \vee \neg p_4 \vee \neg p_5) \wedge (\neg p_7 \vee p_4) \wedge (\neg p_7 \vee p_5) \wedge \\
&(p_4 \vee \neg q \vee \neg r) \wedge (\neg p_4 \vee q) \wedge (\neg p_4 \vee r) \wedge \\
&(p_5 \vee p) \wedge (\neg p_5 \vee \neg p)
\end{aligned}$$

Problem sa Cejtinovom transformacijom je u tome što ona uvodi mnogo novih promenljivih. Postoje raznovrsne tehnike za smanjivanje broja promenljivih i broja klauza.

*Glava 4*

# Rešavanje instanci problema SAT

Rešavači za SAT razvijaju se još od pionirskih dana računarstva. U nastavku će najpre biti opisana dva značajna, stara algoritma koji su često u osnovi i mnogih savremenih SAT rešavača.

## 4.1 DPLL procedura

Dejvis–Patnam–Logman–Lavlendova ili DPLL procedura[1] je procedura za ispitivanje zadovoljivosti iskaznih formula u KNF obliku, to jest, procedura za rešavanje instanci SAT problema. Ulazna formula je konjunkcija klauza. Pri tome (kako su konjunkcija i disjunkcija komutativne i asocijativne) nije bitan poredak tih klauza niti je u bilo kojoj od tih klauza bitan poredak literala, te se ulazna formula može smatrati skupom (ili, preciznije, multiskupom[2]) klauza, od kojih se svaka može smatrati skupom (ili, preciznije, multiskupom) literala. Ipak, radi određenosti rada algoritma, smatraćemo da je skup (odnosno multiskup) klauza uređen.

U proceduri se podrazumevaju sledeće konvencije:

- prazan skup klauza je zadovoljiv;
- klauza koja ne sadrži nijedan literal (zvaćemo je *prazna klauza*) je nezadovoljiva i formula koja sadrži praznu klauzu je nezadovoljiva.

DPLL procedura prikazana je na slici 4.1, a njena svojstva daje teorema 4.1.

**Teorema 4.1** (Korektnost DPLL procedure)**.** *Za svaku iskaznu formulu DPLL procedura se zaustavlja i vraća odgovor* DA *ako i samo ako je polazna formula zadovoljiva.*

Procedura DPLL je u najgorem slučaju eksponencijalne vremenske složenosti po broju iskaznih promenljivih u formuli, usled rekurzivne primene pravila *split*:

**Teorema 4.2.** *Složenost (u najgorem slučaju) DPLL procedure za 3-sat problem je* $O(2^{0.762n})$ *(gde je* $n$ *broj iskaznih promenljivih koje se pojavljuju).* („personal communication“)

Eksponencijalne složenosti su i svi drugi do sada poznati algoritmi za ispitivanje zadovoljivosti. Ipak, svi ti algoritmi znatno su efikasniji od metode istinitosnih tablica.

Procedura DPLL može se smatrati algoritmom pretrage potpunog stabla valuacija promenljivih koje učestvuju u formuli. „Koraci zaključivanja“ algoritma (*tautology*, *unit propagation* i *pure literal*) omogućavaju da se ne pretražuje nužno čitavo stablo. Na efikasnost pretrage utiču i heuristike koje je usmeravaju biranjem načina na koji se primenjuje pravilo *split*: treba izabrati jednu promenljivu i treba odlučiti da li se prvo izvršava grana $\mathrm{DPLL}(D[p \mapsto \top])$ ili grana $\mathrm{DPLL}(D[p \mapsto \bot])$ (u algoritmu prikazanom na slici 4.1 poredak je fiksiran zbog jednostavnosti). Pošto se ispituje da li postoji valuacija u kojoj su sve klauze formule tačne, pohlepni algoritam bi mogao da za *split* promenljivu bira onu koja čini najveći broj klauza tačnim u tekućoj parcijalnoj valuaciji. Druge varijante ovog pravila su da se bira iskazna promenljiva sa najviše pojavljivanja u tekućoj formuli, da se bira neko od iskaznih slova iz najkraće klauze itd. Ove heuristike, kao ni druge slične, ne garantuju optimalnost procesa pretrage.

[1] Prva verzija procedure čiji su autori Dejvis (Martin Davis) i Patnam (Hilary Putnam), unapređena je dve godine kasnije u radu Dejvisa, Logmana (Georg Logemann) i Lavlenda (Donald Loveland), pa otuda naziv DPLL.

[2] Neformalno, multiskup je skup u kojem se elementi mogu pojavljivati više puta.

**Algoritam:** DPLL

**Ulaz:** Multiskup klauza $D$ ($D = \{C_1, C_2, \ldots, C_n\}$)

**Izlaz:** *DA*, ako je multiskup $D$ zadovoljiv, *NE*, inače;

**ako** $D$ je prazan **onda**
    vrati *DA*;
zameni sve literale $\neg\bot$ sa $\top$ i zameni sve literale $\neg\top$ sa $\bot$;
obriši sve literale jednake $\bot$;
**ako** $D$ sadrži praznu klauzu **onda**
    vrati *NE*;

{Korak *tautology*:}
**ako** neka klauza $C_i$ sadrži $\top$ ili sadrži neki literal i njegovu negaciju **onda**
    vrati vrednost koju vraća DPLL($D \setminus C_i$);

{Korak *unit propagation*:}
**ako** neka klauza je jedinična i jednaka nekom iskaznom slovu $p$ **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \top]$);
**ako** neka klauza je jedinična i jednaka $\neg p$, za neku iskaznu promenljivu $p$ **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \bot]$);

{Korak *pure literal*:}
**ako** $D$ sadrži literal $p$ (gde je $p$ neka iskazna promenljiva), ali ne i $\neg p$ **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \top]$) ;
**ako** $D$ sadrži literal $\neg p$ (gde je $p$ neka iskazna promenljiva), ali ne i $p$ **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \bot]$);

{Korak *split*:}
**ako** DPLL($D[p \mapsto \top]$) (gde je $p$ jedno od iskaznih slova koja se javljaju u $D$) vraća *DA* **onda**
    vrati *DA*;
**inače**
    vrati vrednost koju vraća DPLL($D[p \mapsto \bot]$).

Slika 4.1: DPLL procedura.

**Primer 4.1.** *Neka je potrebno ispitati zadovoljivost formule date klauzama:*

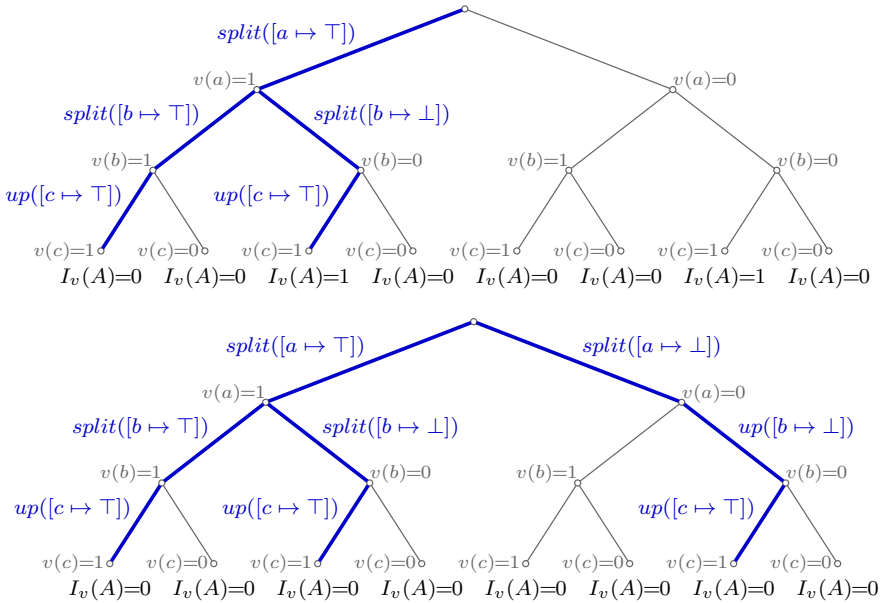
$C_1: \quad \neg a, \neg b, c$
$C_2: \quad a, \neg b$
$C_3: \quad b, c$
$C_4: \quad \neg b, \neg c$

*Formula ima dve zadovoljavajuće valuacije. Proverom zadovoljivosti procedurom DPLL, pronalazi se jedna od te dve valuacije. Prvo stablo na slici 4.2 prikazuje proces pretrage u slučaju datog skupa klauza. Kako obe zadovoljavajuće valuacije pridružuju promenljivoj b vrednost 0, a promenljivoj c vrednost 1, nakon dodavanja klauze*

$C_5: \; b, \neg c$

*prethodni skup klauza postaje nezadovoljiv. Proces pretrage procedurom DPLL u ovom slučaju, prikazan je na drugom stablu na istoj slici. U ovom primeru upečatljivo je da DPLL procedura ispituje svega tri od osam listova zahvaljujući tome što osim koraka pretrage oličenih u pravilu* split, *postoje i koraci zaključivanja koje se vrši primenom pravila* unit propagation *(kada se ne ispituju alternative zamena učinjenih tim pravilom).*

DPLL procedura proverava da li je formula zadovoljiva, ali ona se, kao što je već rečeno, može koristiti i za ispitivanje da li je neka formula valjana, poreciva ili kontradikcija. Na primer, formula $A$ je valjana ako i samo ako je formula $\neg A$ nezadovoljiva, što se može proveriti DPLL procedurom (pri čemu je, naravno, formulu $\neg A$




Slika 4.2: Proces provere zadovoljivosti procedurom DPLL prikazan u vidu pretrage u potpunom stablu valuacija za dva skupa klauza. Pretraga se vrši obilaskom stabla u dubinu sleva nadesno. U prvom slučaju, formula $A$ data je skupom klauza $\{\neg a \vee \neg b \vee c, a \vee \neg b, b \vee c, \neg b \vee \neg c\}$ koji je zadovoljiv. U drugom slučaju, skupu klauza dodata je i klauza $b \vee \neg c$ i dobijeni skup je nezadovoljiv.

potrebno najpre transformisati u konjunktivnu normalnu formu).

## 4.2 Šening algoritam

Šening (Schöning) je 1999. kreirao jednostavni randomizovani algoritam za rešavanje instanci 3-SAT problema, ali se može jednostavno uopštiti i na druge $k$-SAT probleme, pa i šire. Za razliku od procedure DPLL, Šeningov algoritam nema svojstvo potpunosti: ako zadata formula ima model, algoritam će ga možda pronaći a možda neće, i ako zadata formula nema model, algoritam to neće moći da zaključi.

**Algoritam:** Schöningov algoritam

**Ulaz:** $A$ – instanca 3-SAT problema

**Izlaz:** *DA*, ako je zadata instanca zadovoljiva i pronađen je neki njen model

```
Izaberi proizvoljnu valuaciju v za promenljive iz A
za i = 1, ..., n radi
    ako sve klauze su tačne u tekućoj valuaciji onda
        vrati odgovor DA (i tekuću valuaciju kao model)
    inače
        Neka je C neka klauza koja nije tačna u tekućoj valuaciji.
        Izaberi slučajnu promenljivu iz C i promeni njenu vrednost u tekućoj valuaciji.
```

**Teorema 4.3.** *Očekivano vreme izvršavanja Šening algoritma za zadovoljive instance 3-SAT problema pripada klasi $O((4/3)^n)$ (gde je $n$ broj iskaznih promenljivih koje se pojavljuju).*

## 4.3 Savremeni SAT rešavači

Od devedesetih godina dvadesetog veka razvijene su desetine SAT rešavača koji koriste različite pristupe i njihove varijacije. Većina SAT rešavača i svih pratećih resursa raspoloživa je javno i slobodno, u vidu izvornog koda.

Savremeni SAT rešavači uglavnom spadaju u jednu od sledećih grupa:

**CDCL rešavači** CDCL rešavači (eng. *conflict-driven clause-learning solvers*) su proširenja DPLL procedure (4.1) i koriste više dodatnih tehnika od kojih su neke:

- ućenje novih klauza dobijenih iz analize koflikata tokom dotadašnje pretrage.
- izbor promenljive za pravilo *split* ne sme biti previše zahtevan i mora da koristi znanje dobijeno dotadašnjom pretragom.
- nehronološkim vraćanjem unazad („backjumping“).
- povremeno restartovanje pretrage

Neki od znamenitih CDCL SAT rešavača su MiniSat (2005. godina), MiniSAT, PicoSAT i zChaff i, ovdašnji – ArgoSAT (Filip Marić).

**LookAhead rešavači** Rešavači ovog tipa zasnovani su na DPLL proceduri (poglavlje 4.1) i glavna novina je tehnika izbora promenljive na koju će biti primenjeno pravilo *split*. Korišćenjem heuristike koja meri važnost promenljiva i vrši se izbor promenljive koja će biti instancirana. Važnost promenljive meri se količinom pojednostavljivanja koje se može postići, umesto da se vrši jeftina procena na osnovu sintaksičkih svojstava formule. U okviru toga procenjuje se zasebno i kvalitet obe moguće dodele izabranoj promenljivoj.

**Stohastički rešavači** Stohastički SAT rešavači (eng. stochastic local search (SLS) solvers) koriste efikasne algoritme lokalne pretrage. Oni pokušavaju da pronađu zadovoljavajuću valuaciju tj. model za ulaznu formulu, ali ne mogu da zaključe da je SAT instanca nezadovoljiva, te nisu potpuni (za razliku od algoritama kao što je DPLL). SLS rešavači kreću od slučajne valuacije i pokušavaju da pronađu model zadate formule menjajući vrednost jedne po jedne izabrane promenljive (kao i Schöningov algoritam, poglavlje 4.2). Ako je nakon neke iteracije zadovoljavajuća valuacija pronađena, ona se vraća kao rešenje, a inače se bira sledeća promenljiva u skladu sa nekom heurističkom ocenom. Najčešće se sledeća promenljiva bira iz neke klauze koja nije tačna u tekućoj valuaciji. Primeri poznatih SLS rešavača su GSAT i WalkSAT.

**Paralelizovani rešavači** Za razliku od sekvencijalnih SAT rešavača, većina paralelnih rešavača ne garantuje ponovljivo ponašanje zbog maksimiziranja performansi. tj. imaju nestabilno i nedeterminističko ponašanje. Ključni izazovi su efikasna razmena klauza i tačna procena vremena izvršenja intervala razmene klauzula između rešavača.

## 4.4 Portfolio SAT rešavači

Različiti SAT rešavači pogodni su za različite instance SAT problema. Na primer, neki se ističu u dokazivanju nezadovoljivosti, a drugi u pronalaženju modela kod zadovoljivih instanci. Na primer, LookAhead rešavači, na primer, često su efikasniji od CDCL rešavača na zadovoljivim instancama, dok je obratno za nezadovoljive instance. Štaviše, različite vrednosti parametara jednog rešavača ponašaju se bitno različito i pogodne su za različite ulazne instance. Polazeći od ovih ideja, razvijaju se „portfolio“ rešavači koji kombinuju više rešavača ili više razlićitih podešavanja parametara jednog rešavača. Tokom prethodnih godina razvijeno je mnoštvo portfolio sistema za SAT, uglavnom koristeći tehnike mašinskog učenja. Ovdašnji sistem ArgoSmArT k-NN (Mladen Nikolić, Filip Marić, Predrag Janičić) je portfolio za SAT koji je izuzetno jednostavan, ali u isto vreme veoma efikasan i nadmašuje sistem SATzila. Da bi se rešila nova SAT instanca, ArgoSmArT k-NN nalazi njegovih $k$-najbližih suseda iz trening skupa za obuku i poziva rešavač (iz skupa od 13 analiziranih rešavača) koji radi najbolje za te instance. Glavna karakteristika sistema ArgoSmArT je lokalnost izbora — izbor SAT rešavača zasniva se na svega nekoliko instanci sličnih ulaznoj. Najbolji rezultati dobijeni su za $k = 9$. Za rastojanja se koristi nekoliko pogodnih mera. U izboru se koriste sledećih 29 svojstava:

**Svojstva veličine instance:**

- *broj klauza* $L$, *broj promenljivih* $N$;
- *količnik* $N/L$.

**Svojstva grafa promenljive-klauze:**

- *Statistike stepena čvorovova promenljivih*: prosek, varijansa, minimum, maksimum, entropija.
- *Statistike stepena čvorovova klauza*: prosek, varijansa, minimum, maksimum, entropija.

**Svojstva balansa:**

- *Količnik pozitivnih i negativnih literala u svakoj klauzi*: prosek, varijansa, minimum, maksimum, entropija.

- *Količnik pozitivnih i negativnih pojavljivanja svake promenljive*: prosek, varijansa, minimum, maksimum, entropija.
- *Udeo klauza dužine 2 i 3.*

**Svojstva Hornovih klauza**

- *Udeo Hornovih klauza*
- *Broj pojavljivanja svih promenljivih u Hornovim klauzama*: prosek, varijansa, minimum, maksimum, entropija.

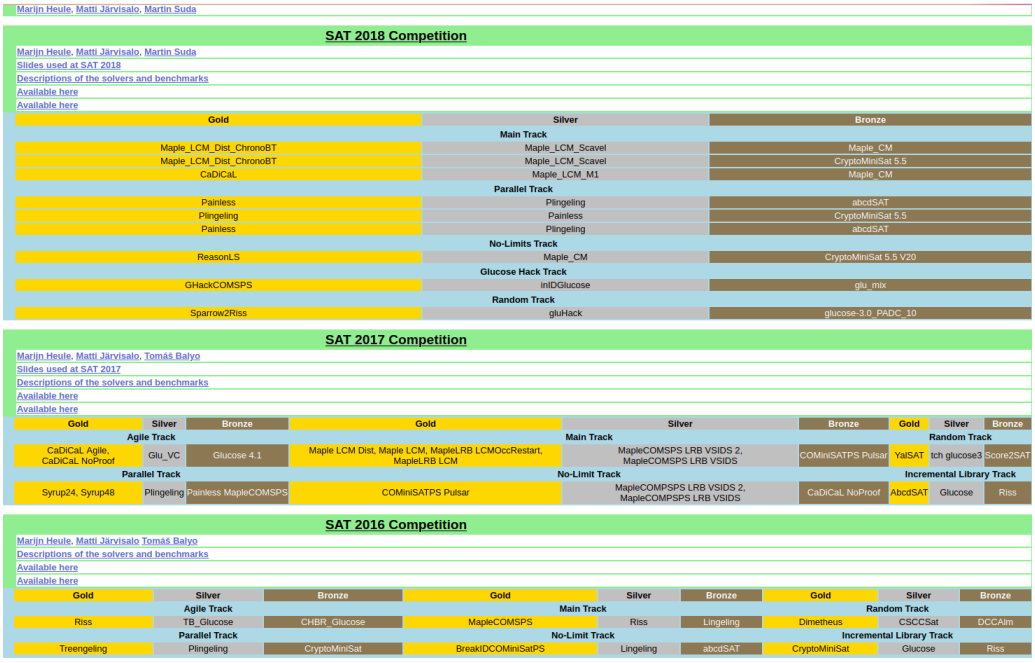
## 4.5 dimacs-cnf format

SAT rešavači obično očekuju ulaz u DIMACS-CNF formatu. U ovom formatu, prvi red sadrži informaciju o broju iskaznih promenljivih i broju klauza, a naredni redovi sadrže zapis po jedne klauze. Promenljive su označene rednim brojevima, negirane promenljive odgovarajućim negativnim brojevima i svaki red završava se brojem 0. Na primer, sadržaj

```
p cnf 3 2
1 -3 0
-1 2 3 0
```

odgovara formuli (sa tri promenljive i dve klauze): $(p_1 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$.

## 4.6 SAT takmičenja



Marijn Heule, Matti Järvisalo, Martin Suda

**SAT 2018 Competition**

Marijn Heule, Matti Järvisalo, Martin Suda
Slides used at SAT 2018
Descriptions of the solvers and benchmarks
Available here
Available here

| Gold | Silver | Bronze |
|---|---|---|
| **Main Track** | | |
| Maple_LCM_Dist_ChronoBT | Maple_LCM_Scavel | Maple_CM |
| Maple_LCM_Dist_ChronoBT | Maple_LCM_Scavel | CryptoMiniSat 5.5 |
| CaDiCaL | Maple_LCM_M1 | Maple_CM |
| **Parallel Track** | | |
| Painless | Plingeling | abcdSAT |
| Plingeling | Painless | CryptoMiniSat 5.5 |
| Painless | Plingeling | abcdSAT |
| **No-Limits Track** | | |
| ReasonLS | Maple_CM | CryptoMiniSat 5.5 V20 |
| **Glucose Hack Track** | | |
| GHackCOMSPS | initDGlucose | glu_mix |
| **Random Track** | | |
| Sparrow2Riss | gluHack | glucose-3.0_PADC_10 |

**SAT 2017 Competition**

Marijn Heule, Matti Järvisalo, Tomáš Balyo
Slides used at SAT 2017
Descriptions of the solvers and benchmarks
Available here
Available here

| Gold | Silver | Bronze | Gold | Silver | Bronze | Gold | Silver | Bronze |
|---|---|---|---|---|---|---|---|---|
| **Agile Track** | | | **Main Track** | | | **Random Track** | | |
| CaDiCaL Agile, CaDiCaL NoProof | Glu_VC | Glucose 4.1 | Maple LCM Dist, Maple LCM, MapleLRB LCMOccRestart, MapleLRB LCM | MapleCOMSPS LRB VSIDS 2, MapleCOMSPS LRB VSIDS | COMiniSATPS Pulsar | YalSAT | tch glucose3 | Score2SAT |
| **Parallel Track** | | | **No-Limit Track** | | | **Incremental Library Track** | | |
| Syrup24, Syrup48 | Plingeling | Painless MapleCOMSPS | COMiniSATPS Pulsar | MapleCOMPSPS LRB VSIDS 2, MapleCOMPSPS LRB VSIDS | CaDiCaL NoProof | AbcdSAT | Glucose | Riss |

**SAT 2016 Competition**

Marijn Heule, Matti Järvisalo Tomáš Balyo
Descriptions of the solvers and benchmarks
Available here
Available here

| Gold | Silver | Bronze | Gold | Silver | Bronze | Gold | Silver | Bronze |
|---|---|---|---|---|---|---|---|---|
| **Agile Track** | | | **Main Track** | | | **Random Track** | | |
| Riss | TB_Glucose | CHBR_Glucose | MapleCOMSPS | Riss | Lingeling | Dimetheus | CSCCSat | DCCAlm |
| **Parallel Track** | | | **No-Limit Track** | | | **Incremental Library Track** | | |
| Treengeling | Plingeling | CryptoMiniSat | BreakIDCOMiniSatPS | Lingeling | abcdSAT | CryptoMiniSat | Glucose | Riss |

Oblast SAT rešavanja doživela je ogroman napredak tokom poslednjih godina. Mnogi problemi (npr. u verifikaciji hardvera i softvera) koji su pre deset godina izgledali potpuno van domašaja, sada se mogu rutinski rešavati. Pored novih algoritama i bolje heuristike, ispostavilo se da su prefinjene tehnike implementacije bile od vitalnog značaja za ovaj uspeh.

Da bi se zadržao trend poboljšavanja SAT rešavača, SAT takmičenja motivišu istraživače da predstave svoje rešavače široj publici i da ih uporede sa drugim rešavačima.

Glavno SAT takmičenje (organizuje se od 2002. godine, http://www.satcompetition.org/) sastoji se od sledećih tokova/disciplina:

- glavni tok (ponekad sa poddisciplinama: zadovoljive, nezadovoljive, pseudoslučajne instance);
- primene u kriptografiji
- tok inkrementalnih biblioteka

- tok za paralelno rešavanje
- tok za rešavanje u klaudu

Glava 5

# Problem SAT i složenost izračunavanja

U strogoj analizi problema, razvrstavanje problema na odlučive i neodlučive je polazna osnova, ali je za razne primene i previše gruba. Za odlučive probleme važno je pitanje složenosti pojedinih procedura odlučivanja. Pojam složenosti obično se vezuje za Tjuringovu mašinu (ili neki drugi ekvivalentan formalizam) i opisuje resurse potrebne da bi neki problem bio rešen. Mere resursa izračunavanja koje oslikavaju složenost algoritma su *vreme* (odnosno broj koraka koje izvršava algoritam) i *prostor* (memorijski prostor koji algoritam koristi). Pitanje složenosti izračunavanja posebno je dobilo na značaju početkom sedamdesetih godina dvadesetog veka rezultatima Stivena Kuka.

## 5.1 Klase složenosti

Za dati, specifični odlučiv problem, gornja granica potrebnog vremena i prostora određuje se razmatranjem konkretnog algoritma koji rešava dati problem. Dodatno, da bi se dokazala optimalnost algoritma, potrebno je imati i odgovarajuće donje granice složenosti problema.

Često se algoritam ne izvršava isto za sve ulaze istih veličina, pa je potrebno naći način za opisivanje i poređenje efikasnosti različitih algoritama. *Analiza najgoreg slučaja* zasniva procenu složenosti algoritma u najgorem slučaju (u slučaju za koji se algoritam najduže izvršava ili zahteva najviše memorijskog prostora). Ta procena može da bude varljiva, ali ne postoji bolji opšti način za poređenje efikasnosti algoritama. Čak i kada bi uvek bilo moguće izračunati prosečno vreme izvršavanja algoritma (odnosno prosečne memorijske zahteve) i takva procena bi često mogla da bude varljiva.

U analizi složenosti algoritma, obično nas najviše interesuje asimptotsko ponašanje i u tome se koristi tzv. *o-notacija*.

**Definicija 5.1.** *Ako postoje pozitivna konstanta $c$ i prirodan broj $n_0$ takvi da za funkcije $f$ i $g$ nad prirodnim brojevima važi*

$$f(n) \leq c \cdot g(n) \quad \text{za sve vrednosti } n \text{ veće od } n_0$$

*onda pišemo*

$$f = O(g)$$

*i čitamo „$f$ je veliko 'o' od $g$“.*

Naglasimo da $O$ nije funkcija — $O$ označava klasu funkcija. Aditivne i multiplikativne konstante ne utiču na klasu kojoj funkcija pripada.

**Definicija 5.2.** *Ako je $T(n)$ vreme izvršavanja algoritma $A$ (čiju veličinu ulaza karakteriše prirodan broj $n$), ako važi $T = O(g)$, onda kažemo da je algoritam $A$* vremenske složenosti *(ili* reda*) $g$ i da pripada klasi $O(g)$.*

Analogno se definiše *prostorna složenost* algoritma.

Formalno se pojmovi vremenske i prostorne složenosti izračunavanja često definišu u terminima Tjuringove mašine (ali analogno mogu da se definišu za druge formalizme izračunavanja). Ako je prilikom izračunavanja promenjeno $t$ konfiguracija mašine, onda je $t$ *vreme* tog izračunavanja. *Prostor* izračunavanja je broj polja mašine kojima se pristupa tokom izračunavanja. Tjuringova mašina *prihvata problem u vremenu (prostoru)* $F(n)$, ako za

svaku ulaznu vrednost (čija veličina može biti opisana prirodnim brojem $n$) koju prihvata (tj. za koju je odgovor na problem potvrdan) postoji izračunavanje koje je prihvata u vremenu (prostoru) koje ne prelazi $F(n)$.

**Primer 5.1.** *Algoritam za izračunavanje vrednosti faktorijela prirodnog broja $n$ je vremenske složenosti $O(n)$, algoritam bubble-sort za sortiranje $n$ elemenata je vremenske složenosti $O(n^2)$, algoritam merge-sort za sortiranje $n$ elemenata je vremenske složenosti $O(n \log n)$, algoritam za ispitivanje zadovoljivosti iskazne formule nad $n$ slova, zasnovan na istinitosnim tablicama je vremenske složenosti $O(2^n)$.*

U teorijskim analizama složenosti, pod veličinom ulaza obično se podrazumeva broj bitova potrebnih za zapisivanje tog ulaza. Potrebno je uvek eksplicitno navesti u odnosu na koju veličinu se razmatra rad algoritma.

## 5.2 NP-kompletnost i problem SAT

Problem odlučivanja je problem za koji su mogući odgovori samo „da“ i „ne“.

Problem se smatra *efikasno rešivim* ako postoji algoritam koji rešava problem za sve njegove instance u broju koraka koji je polinomski ograničen veličinom ulazne instance, pri čemu se podrazumeva „tradicionalni“ model izračunavanja tj. sekvencijalni, deterministički model (kao što je deterministička Tjuringova mašina (DTM) ili UR-mašina). Smatramo da problemi koji nisu u ovoj klasi nisu efikasno rešivi.

**Definicija 5.3.** *Za problem odlučivanja sa ulaznom vrednošću $n$ kažemo da je polinomske vremenske složenosti ako je njegovo vreme izvršavanja $O(P(n))$ gde je $P(n)$ polinom po $n$. Klasu polinomskih algoritama označavamo sa* P.

Nedeterministički algoritam (opisan, na primer, u formalizmu nedeterminističke Tjuringove mašine (NTM)) može u svojim koracima da nedeterministički bira između dva moguća, različita puta za nastavak rada koristeći tzv. *nd-izbore*. Kažemo da nedeterministički polinomski algoritam rešava problem odlučivanja ako važi: odgovor za ulaznu vrednost $x$ je potvrdan ako i samo ako postoji niz nd-izbora takav da vodi prihvatanju vrednosti $x$ u polinomskom vremenu (po veličini ulaza). Kažemo da takav niz nd-izbora čini *sertifikat*. Primetimo da *sertifikat* proveriv u polinomskom vremenu mora da postoji samo za instance za koje je odgovor „da“ (ne nužno i za instance za koje je odgovor „ne“).

**Definicija 5.4.** *Klasu svih problema odlučivanja za koje postoje nedeterministički polinomski algoritmi zovemo klasa* NP.

Na primer, problem SAT pripada klasi NP. Zaista, nedeterministički polinomski algoritam može da koristi nd-izbore da određuju vrednosti promenljivih koje se javljaju u formuli i, kada su sve vrednosti poznate, može se u polinomskom vremenu proveriti da li je formula tačna u tako dobijenoj valuaciji.

Klasa NP može se opisati i kao skup problema odlučivanja takvih da svaka instanca za koju je odgovor „da“ ima *rešenje* proverivo u polinomskom vremenu (npr. na determinističkoj Tjuringovoj mašini). U slučaju problema SAT, rešenje može da bude upravo zadovoljavajuća valuacija.

Vreme rešavanja instanci za koje je odgovor na problem negativan nije relevantan za klasu NP. Nije relevantno čak ni to da li se rešavanje zaustavlja za te instance. Međutim, kako se rešavanje problema zaustavlja za sve instance za koje je odgovor potvrdan i to u vremenu ograničnom vrednošću konkretnog polinoma, to znači da se sistematskim proveravanjem može utvrditi da li neka instanca pripada ili ne pripada skupu instanci za koje je odgovor potvrdan. Odatle sledi da svaki problem koji pripada klasi NP mora da bude odlučiv.

Očigledno, važi $\mathrm{P} \subseteq \mathrm{NP}$, ali se još uvek ne zna da li važi $\mathrm{P} = \mathrm{NP}$. Ako bi se pokazalo da neki problem iz klase NP nema polinomsko rešenje, onda bi to značilo da ne važi $\mathrm{P} = \mathrm{NP}$. Ako neki problem iz klase NP ima polinomsko rešenje, onda to još ne znači da $\mathrm{P} = \mathrm{NP}$.

**Primer 5.2.** *Problem* SAT *pripada klasi* NP*, a ne zna se da li pripada klasi* P.

Za probleme iz posebne potklase klase NP (to je klasa NP-kompletnih problema) važi da ako neki od njih ima polinomsko rešenje, onda važi $\mathrm{P} = \mathrm{NP}$. NP-kompletni problemi su „najteži“ problemi u klasi NP. Okvir za definisanje ovog pojma je pojam *svodljivosti*. Kažemo da se skup $A$ (odnosno problem za koji je odgovor potvrdan za elemente skupa $A$ i samo za njih) polinomski svodljiv na skup $B$ (odnosno odgovarajući problem) i pišemo $A \leq_p B$ ako postoji funkcija $f$ takva da može da bude izračunata u polinomskom vremenu i ima svojstvo $w \in A$ ako i samo ako je $f(w) \in B$ (ovo je takozvano Karp-ovo svođenje, a postoji i pojam Kukovog svođenja).

Intuitivno, algoritam za problem $B$ može biti iskorišćen za rešavanje problema $A$ sa dodatnim polinomskim vremenom potrebnim za svođenje problema (analogno se definiše svodljivost u odnosu na prostor).

**Definicija 5.5.** *Za problem $X$ kažemo da je* NP-težak problem *ako je svaki* NP *problem u polinomskom vremenu svodljiv na $X$.*

**Definicija 5.6.** *Za problem $X$ kažemo da je* NP*-kompletan problem ako pripada klasi* NP *i ako je* NP*-težak.*

Navedena definicija može se i uopštiti: ako problem $A$ pripada klasi $\mathcal{C}$ i ako je svaki problem iz klase $\mathcal{C}$ svodljiv u polinomskom vremenu na $A$, onda za problem $A$ kažemo da je $\mathcal{C}$-kompletan.

**Teorema 5.1.** *Ako bilo koji* NP*-težak problem pripada klasi* P*, onda važi* P=NP.

***Dokaz:*** Neka je problem $X$ NP-težak. To znači da se proizvoljan problem $Y$ iz klase NP može svesti na njega u polinomskom vremenu (tj. $Y \leq_p X$). Ako bi problem $X$ pripadao klasi P, to bi značilo da za njega postoji polinomsko rešenje, a odatle bi sledilo da postoji polinomsko rešenje i za problem $Y$, tj. i problem $Y$ bi pripadao klasi P. Problem $Y$ je proizvoljan problem $Y$ iz klase NP, pa bi iz $X \in$ P sledilo P=NP. □

Na osnovu navedene teoreme, da bi se dokazalo da je P=NP, dovoljno je dokazati za jedan NP-kompletan problem da pripada klasi P. Dodatno, očigledno, da bi se dokazalo da je P $\neq$ NP, dovoljno je dokazati za jedan NP-kompletan problem da ne pripada klasi P. Potrebno je, dakle, razmatrati neki pogodan NP-kompletan problem. Veoma je teško direktno sledeći definiciju dokazati za neki problem da je NP-kompletan. Naredna teorema omogućava jednostavnije dokazivanje da je neki problem NP-kompletan.

**Teorema 5.2.** *Problem $X$ je* NP*-kompletan ako:*

- *$X$ pripada klasi* NP*;*
- *postoji* NP*-kompletan problem $Y$ koji je polinomski svodljiv na $X$ (tj. $Y \leq_p X$).*

***Dokaz:*** Kako $X$ pripada klasi NP, dovoljno je dokazati da je on NP-težak.

Ako je $Y$ NP-kompletan problem, to znači da se proizvoljan problem $Z$ iz klase NP može svesti na problem $Y$ u polinomskom vremenu (tj. $Z \leq_p Y$). To dalje znači da se proizvoljan problem $Z$ iz klase NP može u polinomskom vremenu svesti i na problem $X$ (tj. $Z \leq_p X$), jer je $Y$ polinomski svodljiv na $X$ (tj. $Y \leq_p X$), a kompozicija dva polinomska svođenja je ponovo polinomsko svođenje. Dakle, problem $X$ je NP-težak, što je i trebalo dokazati. □

Navedena teorema omogućava utvrđivanje da je neki problem NP-kompletan, ako je za neki drugi već poznato da je NP-kompletan. Međutim, postavlja se pitanje da li uopšte postoji ijedan NP-kompletan problem. Stiven Kuk prvi je, 1971. godine, dokazao (neposredno, koristeći formalizam Tjuringove mašine) da postoje NP-kompletni problemi[1] i to njegovo tvrđenje jedan je od najznačajnijih rezultata teorijskog računarstva.

**Teorema 5.3** (Kukova teorema)**.** *Problem* SAT *je* NP*-kompletan.*

Koristeći Kukovu teoremu i tehnike svođenja, u godinama koje su sledile za mnoge probleme je dokazano da su NP-kompletni.

**Primer 5.3.** *Zahvaljujući teoremama 5.2 i 5.3 može se, na primer, dokazati da su sledeći problemi* NP*-kompletni:*

- 3-SAT *problem (*SAT *problem u kojem su sve klauze dužine 3) – videti poglavlje 5.3.1;*
- *3-obojivost (ispitivanje da li postoji pridruživanje tri različite boje čvorovima grafa, takvo da je svakom čvoru pridružena neka boja, a da su susednim čvorovima pridružene različite boje);*

[1] Kuk je prvi dokazao i da postoje P-kompletni problemi (jedan od njih je CVP — problem vrednosti kola).

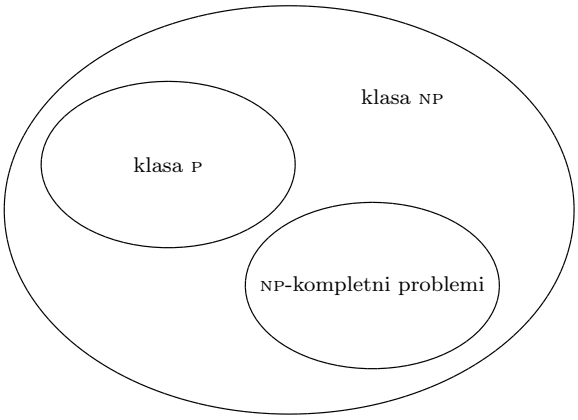
- *pokrivač grana (ispitivanje da li postoji podskup $D$ skupa čvorova grafa takav da $D$ ima manje od $k$ elemenata i da svaka grana grafa sadrži bar jedan čvor iz $D$);*
- *dominirajući skup (ispitivanje da li postoji podskup $D$ skupa čvorova grafa takav da $D$ ima manje od $k$ elemenata i da je svaki čvor grafa ili u $D$ ili je susedan nekom čvoru iz $D$);*
- *problem klika (ispitivanje da li postoji potpun podgraf $G$ grafa takav da $G$ ima bar $k$ čvorova);*
- *Hamiltonov ciklus (ispitivanje da li graf sadrži prost ciklus koji sadrži svaki čvor grafa tačno jednom);*
- *Hamiltonov put (ispitivanje da li graf sadrži prost put koji sadrži svaki čvor grafa tačno jednom);*
- *problem trgovačkog putnika (ispitivanje da li težinski potpun graf sadrži Hamiltonov ciklus sa zbirom težina grana manjim od zadate vrednosti).*

Postojala je nada da je za neki pogodan NP-kompletni problem moguće dokazati da nema polinomskog rešenja, što bi bio dokaz za tvrđenje P $\neq$ NP. S druge strane, ukoliko bi se za neki NP-kompletan problem dokazalo da ima polinomsko rešenje, to bi značilo da važi P=NP. Ni za jedan NP-kompletan problem, međutim, još uvek nije dokazano niti da ima niti da nema polinomsko rešenje. Kako je SAT NP-kompletan, onda važi naredna teorema:

**Teorema 5.4.** P=NP *ako i samo ako* SAT $\in$ P.

Navedena teorema govori da je za odgovor na pitanje P=NP dovoljno razrešiti status problema SAT (što, uostalom, važi i za bilo koji drugi NP-kompletan problem).

Rasprostranjeno je uverenje da važi P $\neq$ NP (te je rasprostranjeno uverenje da ne postoji algoritam polinomske složenosti za rešavanje SAT problema). Samo malobrojni istraživači veruju da važi P = NP. Postoje i mišljenja da je tvrđenje P = NP neodlučivo u matematičkim teorijama koje se koriste za njegovo ispitivanje. Ukoliko bi se pokazalo da važi P = NP, onda bi region klase P pokrio čitav region klase NP. Ne zna se da li važi P = NP, ali je poznato da postoje problemi koji su izvan klase NP (i, time, naravno i izvan klase P). Postoje i problemi koji su u klasi NP, ali nisu NP-kompletni. Trenutno dominantno uverenje o odnosu klasa P i NP ilustrovano je na slici 5.1.




Slika 5.1: Klase P i NP

## 5.3 Primeri svođenja

### 5.3.1 Problem 3-SAT je NP-kompletan

Trivijalno je dokazati da problem 3-SAT pripada klasi NP.

Na osnovu teoreme 5.2, kako je SAT NP-kompletan problem, dovoljno je još dokazati da je problem SAT polinomski svodljiv na 3-SAT, tj. SAT $\leq_p$ 3-SAT

Razmotrimo proizvoljnu SAT instancu i pokažimo da ona može biti transformisana u odgovarajuću 3-SAT instancu u polinomskom vremenu. Neka je $A$ proizvoljna instanca SAT problema:

$$A = C_1 \wedge C_2 \wedge ... \wedge C_n$$

gde su $C_i$ klauze. Pretpostavimo da neka klauza, na primer $C_r$ nema 3 literala (ako sve imaju 3 literala, onda je instanca već u 3-SAT obliku).

- Pretpostavimo da $C_r$ ima tačno jedan literal:

$$C_r = l$$

  Tada $C_r$ može biti zamenjena konjunkcijom koja uključuje dve nove promenljive $z_1$ i $z_2$:

$$(l \vee z_1 \vee z_2) \wedge (l \vee \neg z_1 \vee z_2) \wedge (l \vee z_1 \vee \neg z_2) \wedge (l \vee \neg z_1 \vee \neg z_2)$$

- Pretpostavimo da $C_r$ ima tačno dva literala:

$$C_r = l_1 \vee l_2$$

  Tada $C_r$ može biti zamenjena konjunkcijom koja uključuje novu promenljivu $z_1$:

$$(l_1 \vee l_2 \vee z_1) \wedge (l_1 \vee l_2 \vee \neg z_1)$$

- Pretpostavimo da $C_r$ ima tačno $k$ literala:

$$C_r = l_1 \vee l_2 \vee \ldots \vee l_k$$

  gde je $k > 3$. Tada klauza $C_r$ može biti zamenjena konjunkcijom $C'$ koja uključuje $k-3$ novih promenljivih $z_1$, $z_2$, ..., $z_{k-3}$:

$$(l_1 \vee l_2 \vee z_1) \wedge (l_3 \vee \neg z_1 \vee z_2) \wedge (l_4 \vee \neg z_2 \vee z_3) \wedge \ldots \wedge (l_{k-2} \vee \neg z_{k-4} \vee z_{k-3}) \wedge (l_{k-1} \vee l_k \vee \neg z_{k-3})$$

U prva dva slučaja, tako dobijena klauza $C'$ i početna klauza $C_r$ su logički ekvivalentne. U trećem slučaju tako dobijena klauza $C'$ i početna klauza $C_r$ nisu logički ekvivalentne ali su ekvizadovoljive: ako je zadovoljiva jedna, zadovoljiva je i druga i obratno. Štaviše, svaki model za $C_r$ može se proširiti do modela za $C'$ i svaki model za $C'$ može se suziti do modela za $C_r$.

- Pretpostavimo da je klauza $C_r$ zadovoljiva tj. da je ona tačna u nekoj valuaciji $v$. Ako je u toj valuaciji tačan literal $l_1$ ili $l_2$, onda se $v$ može proširiti tako da je vrednost svake promenljive $z_i$ jednaka 0. U tako dobijenoj valuaciji tačna je formula $C'$.

  Ako je u toj valuaciji tačan literal $l_k$ ili $l_{k-1}$, onda se $v$ može proširiti tako da je vrednost svake promenljive $z_i$ jednaka 1. U tako dobijenoj valuaciji tačna je formula $C'$.

  Ako je u toj valuaciji tačan literal $l_j$, onda se $v$ može proširiti tako da je vrednost svake promenljive $z_i$ jednaka 1 za $i = 1, 2, \ldots, j-2$ i jednaka 0 za $i = j-1, \ldots, k-1, k$. U tako dobijenoj valuaciji tačna je formula $C'$.

- Pretpostavimo da klauza $C_r$ nije zadovoljiva. Neka je $v$ proizvoljna valuacija. U toj valuaciji svaki od literala $l_i$ je netačan. Ako bi $C'$ bila tačna u nekom proširenju te valuacije, morala bi promenljiva $z_1$ da bude tačna. Odatle sledi da bi morala i promenljiva $z_2$ da bude tačna, zatim $z_3$, itd. Na kraju sledi da poslednja klauza u $C'$ ne može biti tačna, što je suprotno pretpostavci. Dakle, i $C'$ je nezadovoljiva.

Opisanom transformacijom, sve klauze polazne SAT instance mogu biti zamenjene skupom klauza dužine 3, te se svaka instanca SAT problema može zameniti instancom 3-SAT problema (takvom da je za polazna zadovoljiva ako i samo ako je zadovoljiva dobijena formula). Opisana transformacija je polinomske složenosti, pa je 3-SAT NP-kompletan problem.

### 5.3.2 Problem KLIKA je NP-kompletan

Problem KLIKA je problem ispitivanja da li u datom grafu $G$ postoji klika (potpuni podgraf) date veličine $k$. Trivijalno je dokazati da problem KLIKA pripada klasi NP.

Na osnovu teoreme 5.2, kako je SAT NP-kompletan problem, dovoljno je još dokazati da je problem SAT polinomski svodljiv na problem KLIKA, tj. SAT $\leq_p$ KLIKA.

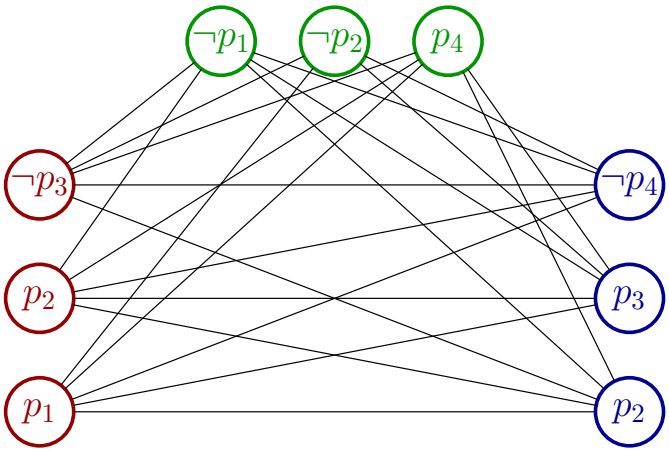
Razmotrimo proizvoljnu SAT instancu $A$ i pokažimo da u polinomskom vremenu može biti konstruisan graf $G$ koji ima kliku veličine $k$ ako i samo ako je polazna SAT instanca zadovoljiva. Konstruišimo graf $G$ sa $k$ grupa čvorova, gde je $k$ broj klauza u formuli $A$. Svaka grupa odgovaraće jednog klauzi iz $A$. Svaki čvor u grupi je

označen kao jedan literal u klauzi. U grafu postoji grana između svaka dva čvora iz dve grupe osim za parove oblika $\{p, \neg p\}$. Ne postoji nijedna grana između čvorova u istoj grupi.

Kao primer, razmotrimo narednu SAT formulu:

$$(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_4) \wedge (p_2 \vee p_3 \vee \neg p_4)$$

Njoj odgovara naredni graf:




Opisana konstrukcija grafa može biti izvršena u polinomskom vremenu.

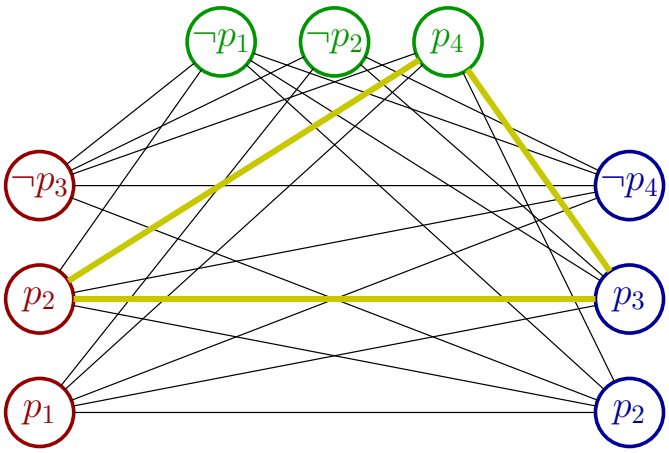
Primetimo sledeće: ako su dva čvora grafa povezana, onda odgovarajućim literalima može biti dodeljena vrednost 1 (jer ne postoji grana između dva komplementarna literala). Dodatno, ako dva literala koja nisu iz iste klauze mogu dobiti vrednost 1 istovremeno, čvorovi koji odgovaraju tim literalima u grafu su povezani.

Konstruisani graf $G$ ima kliku veličine $k$ ako i samo ako je polazna formula $A$ zadovoljiva. Dokažimo to:

- Ako graf $G$ ima kliku veličine $k$: onda klika ima tačno jedan čvor iz svake grupe. Naime, ne postoje dva čvora iz iste grupe koja su povezana, pa ne mogu biti deo iste klike. Svi čvorovi u kliki su povezani, pa se svim odgovarajučim literalima može dodeliti istinitosna vrednost 1. Svaki od tih literala pripada jednoj od klauza, pa je polazna formula $A$ zadovoljiva.

- Ako je $A$ zadovoljiva, neka je $v$ zadovoljavajuća valuacija. Iz svake klauze može se izabrati po jedan literal koji je tačan u $v$. Svi odgovarajući čvorovi u grafu su povezani (nikoja dva ne pripadaju istoj grupi) i oni formiraju kliku veličine $k$. Dakle, graf $G$ ima kliku veličine $k$:

Naredna klika




daje model za formulu:

$$(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_4) \wedge (p_2 \vee p_3 \vee \neg p_4)$$

## 5.4 Složenost problema srodnih problemu SAT

Problem odlučivanja $\mathcal{X}$ je komplementan problemu odlučivanja $\mathcal{Y}$ ako za svaku instancu za koju problem $\mathcal{Y}$ daje odgovor „da“, problem $\mathcal{X}$ daje odgovor „ne“. Klasa problema co-NP je klasa svih problema odlučivanja $\mathcal{X}$ takvih da klasi NP pripada problem koji je komplementan problemu $\mathcal{X}$. Na primer, problem SAT, tj. problem zadovoljivosti za iskazne formule u KNF obliku pripada klasi NP, pa problem UNSAT, tj. problem nezadovoljivosti za iskazne formule u KNF obliku pripada klasi co-NP. Može se dokazati i da je problem UNSAT co-NP-kompletan. Nije poznato da li su klase NP i co-NP jednake (dominantno mišljenje je da nisu).

**Teorema 5.5.** *Problem* UNSAT *je co-*NP*-kompletan.*

Problem ispitivanja zadovoljivosti formula u DNF obliku (tj. problem DNF-SAT) suštinski je drugačiji od problema SAT, tj. od ispitivanja zadovoljivosti formula u KNF obliku. Problem DNF-SAT je trivijalan i polinomski rešiv (podrazumeva se složenost u terminima broja bitova potrebnih za zapis ulazne formule) – dovoljno je razmatrati zadovoljivost disjunkata pojedinačno, a to se svodi na provere da u ne disjunktu postoji logička konstanta $\bot$ niti literal i njegova negacija. Ipak, svođenje problema SAT na problem ispitivanja zadovoljivosti DNF formule nije, u opštem slučaju, razuman put za rešavanje problema SAT, zbog kompleksnosti same transformacije iz KNF oblika u DNF oblik (nije poznat algoritam kojim bi formula u KNF obliku mogla da se transformiše u DNF oblik u polinomskom vremenu).

**Teorema 5.6.** *Problem* DNF-SAT *pripada klasi* P.

Dualno gore rečenom, problem ispitivanja tautologičnosti formule u KNF obliku (tj. problem CNF-TAUT) pripada klasi P (dovoljno je razmatrati tautologičnost konjunkata pojedinačno, a to se svodi na provere da li u konjunktu postoji logička konstanta $\top$ ili literal i njegova negacija).

**Teorema 5.7.** *Problem* CNF-TAUT *pripada klasi* P.

Već je rečeno da je 3-SAT problem NP-kompletan. Isto važi i za $k$-SAT, za svako $k \geq 3$, ali ne i za $k = 2$.

**Teorema 5.8.** *2*-SAT $\in$ P.

Dokaz: Razmotrimo pravilo rezolucije za iskaznu logiku:

$$\frac{C' \vee l \quad C'' \vee \bar{l}}{C' \vee C''}$$

Ako se primenjuje na klauze dužine dva, onda ima specijalan oblik

$$\frac{l_1 \vee l \quad l_2 \vee \bar{l}}{l_1 \vee l_2}$$

gde je $\bar{l}$ suprotan literal literalu $l$. Primetimo da je za klauze dužine najviše 2 dužina rezolvente takođe dužine najviše 2. Broj klauza dužine najviše 2 jednak je $1 + 2n + 4n^2$, pa je moguće izvesni polinomski mnogo novih rezolventi (u polinomskom vremenu). Ako među njima nema prazne klauze, onda je ulazna instanca zadovoljiva, a ako među njima ima prazne klauze, onda je ulazna instanca nezadovoljiva.

Nije poznato nijedno polinomsko svođenje problema SAT ili 3-SAT na 2-SAT. Ako bi postojalo, onda bi problemi SAT i 3-SAT pripadali klasi NP.

Glava 6

# Problem SAT i promena faze

Problem iskazne zadovoljivosti (SAT) jedan je od tipičnih NP-kompletnih problema i duboko razumevanje prirode SAT problema od suštinske je važnosti za razumevanje prirode klase NP. Za sada se ne zna da li za NP-kompletne probleme postoje polinomska rešenja (tj. ne zna se da li važi P=NP), ali nije teško naslutiti da ne zahteva svaka instanca (primerak) NP-kompletnog problema jednako vreme izračunavanjam, tj. nisu sve instance NP-kompletnih problema podjednako teške i važno je utvrditi koje su najteže i šta ih to čini najtežim.

Na primer, lako se može konstruisati iskazna formula nad skupom od $N$ iskaznih slova koja je u konjunktivnoj normalnoj formi i čiju je zadovoljivost trivijalno ispitati (npr. takva je formula $p_1 \wedge p_2 \wedge \ldots \wedge p_N$). Opštije, formule u konjuktivnoj normalnoj formi sa malo klauza (u odnosu na broj promenljivih) su slabo ograničene i relativno jednostavne za svaku proceduru odlučivanja za iskaznu logiku (jer postoji mnogo zadovoljavajućih valuacija); formule sa mnogo klauza (u odnosu na broj promenljivih) su jako ograničene, pa je relativno lako pokazati da su nezadovoljive. Da bi se razumelo šta NP kompletne probleme čini teškim, potrebno je znati koje su instance tog problema najteže. Dodatno, prilikom konstrukcije algoritma za rešavanje nekog NP kompletnog problema potrebno je algoritam ispitivati (i porediti sa drugim algoritmima) upravo na najtežim instancama problema. Dakle, u vezi sa zadovoljivošću iskaznih formula postavljaju se sledeća važna pitanja na koja sâmi metodi za ispitivanje zadovoljivosti ne mogu da odgovore:

- Kakva je distribucija (raspodela) zadovoljivih iskaznih formula i da li za određene klase iskaznih formula možemo da procenimo udeo zadovoljivih formula samo na osnovu njihovih sintaksičkih karakteristika?
- Da li je ispitivanje zadovoljivosti jednako teško za sve iskazne formule? Ako nije, za koje je formule najteže ispitati zadovoljivost?
- Da li su za različite metode jednaki ili ne skupovi iskaznih formula za koje je ispitivanje zadovoljivosti najzahtevnije (u smislu potrebnog vremena)?

Sa motivacijom u navedenim pitanjima, od poslednje decenije dvadesetog veka takozvana *promena faze* za mnoge NP-kompletne probleme postala je predmet mnogih i teorijskih i eksperimentalnih istraživanja. Uopšteno govoreći, promena faze promena je ponašanje neke karakteristike skupova instanci problema, karakteristike koja prelazi iz oblasti u kojoj dominira jedna vrednost u oblast u kojoj dominira neka druga vrednost.

## 6.1 Promena faze u fizičkim sistemima

*Promena faze* javlja se u mnogim fizičkim sistemima. Promena faze u fizičkim sistemima opisuje se na sledeće načine.

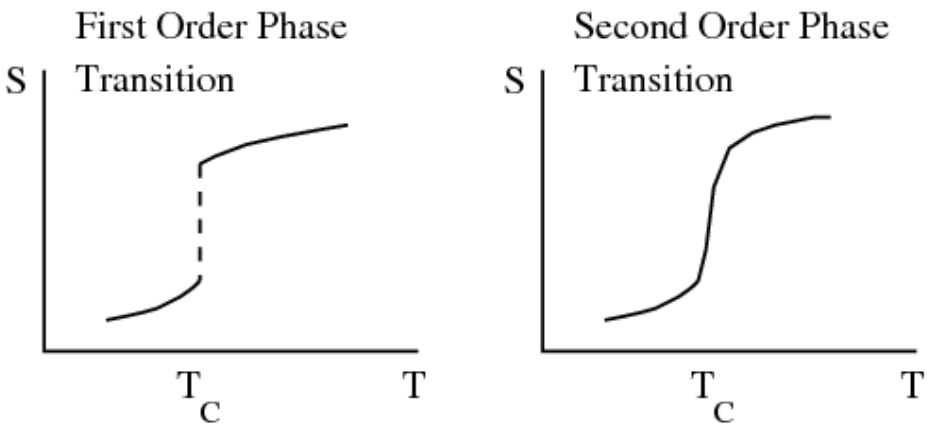
**Definicija 6.1.** Promena faze *je prelazak iz čvrstog, tečnog ili gasovitog (ili plazme) u neko drugo agregatno stanje.*

**Definicija 6.2.** Promena faze *je promena svojstva fizičkog sistema, koja često uključuje apsorpciju ili emitovanje energije i koja rezultuje prelaskom iz jednog u drugo stanje sistema.*

**Definicija 6.3.** Promena faze *je fizički proces prelaska iz jednog stanja medijuma, opisanog nekim vrednostima parametrima, u drugog stanje, sa drugim vrednostima istih tih parametara.*

**Primer 6.1.** *Promena faze često je povezana sa uređenjem molekula i entropijom. Na primer, molekuli vode su neuređeni, dok su molekuli u ledu uređeni. Prelazak vode iz tečnog u čvrsto stanje je primer promene faze.*

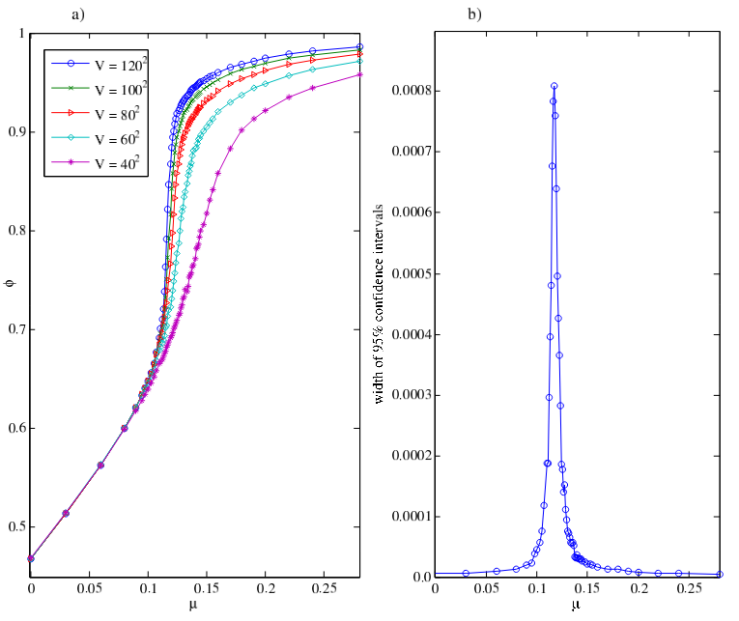
Postoje dva osnovna tipa promene faze: promena faze prvog reda i promena faze drugog reda. Promena faze je prvog reda ako se, na primer, javlja prekid vrednosti entropije $S$ kada se menja vrednost nekog drugog parametra. Entropija vode u tečnom stanju veća je od entropije u čvrstom stanju i između ta dva postoji skok $\Delta S$. U promeni faze drugog reda, parametar koji se prati menja se naglo ali neprekidno.




**Primer 6.2.** *Prelazak vode iz tečnog u čvrsto stanje primer je promene faze prvog reda: u tečnom stanju molekuli vode su neuređeni, ali tokom promene agregatnog stanja, oni naglo postaju uređeni (tj. beleži se nagli skok vrednosti entropije).*

**Primer 6.3.** *Ovo je deo rada „First Order Phase Transition of a Long Polymer Chain“ autora Aristoff i Radin-a (2010):*

> *In polymer physics self-avoiding walks have been used for many years to model long chain molecules [1,2]. One of the oldest such models, due to Flory [3], consists of a single self-avoiding random polygon occupying all the sites of a square lattice, with the randomness controlled by the total energy associated with 90∘-bends in the polygon. Thinking of the polygon as made of many flexibly connected monomers, the model is a canonical ensemble with the temperature behavior analyzed only at the optimally high particle density of 1. At high temperature the Flory model behaves like a disordered fluid, while at low temperature the model displays long-range nematic order. (Flory was modeling the “melting” of a nematically ordered polymer [3].) Although it is generally accepted that there is a true phase transition between these regimes in the model, there has been a dispute over the character of the transition [4], which Flory had originally predicted to be first order. ...*




*Graf prosečne gustine u odnosu na $\mu$ (hemijski potencijal – energija koja može biti apsorbovana ili oslobođena u hemijskoj reakciji) za $\beta = 1.5$ ($\beta = 1/T$) za zapreminu sistema od $V = 40^2$ do $V = 120^2$ (levo). Širina intervala poverenja za 95% za prosečnu gustinu, za $V = 120^2$.*

## 6.2 Promena faze u problemu random $k$-SAT

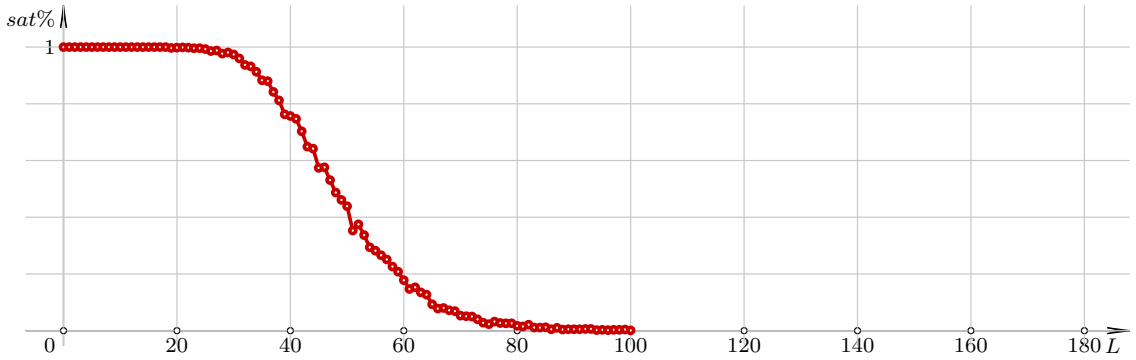
Inspirisano promenom faze u fizičkim sistema, eksperimentalno je pokazano (nad skupovima instanci random SAT problema sa različitim sintaksičkim ograničenjima) da u SAT problemima postoji promena faze između zadovoljivosti i nezadovoljivosti za pogodne parametre.

Sa $M(N, L)$ označavaćemo skup instanci problema SAT koje se sastoje od $L$ klauza nad skupom od $N$ iskaznih slova i zadovoljavaju neka sintaksička ograničenja (na primer, sve klauze imaju dužinu $k$). Sa *sat* označavamo *funkciju zadovoljivosti* koja odgovara procentu zadovoljivih formula: $sat(M(N, L))$ je procenat zadovoljivih formula u skupu $M(N, L)$.

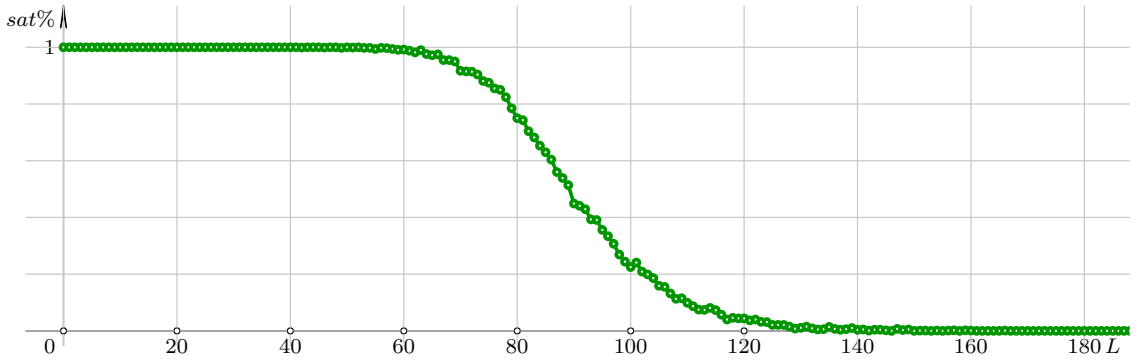
Za mnoge skupove instanci SAT problema $M(N, L)$ može se lako dokazati da funkcija $sat(M(N, L))$ strogo opada kada vrednost $L$ raste, kao i da, za fiskirano $N$, važi $\lim_{L\to\infty} sat(M(N, L)) = 0\%$.

**Definicija 6.4.** *Random $k$-SAT problem je problem za čiju svaku instancu važi: instanca je u konjunktivnoj normalnoj formi, sve njene klauze imaju dužinu $k$, sve promenljive imaju istu verovatnoću pojavljivanja u svakom literalu i svaki literal sadržo veznik negacije sa verovatnoćom 50%.*

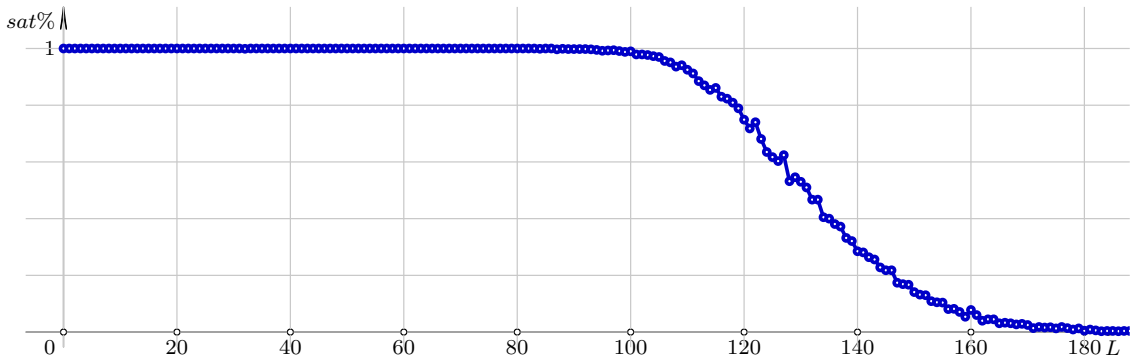
Funkcija zadovoljivost eksperimentalno se može aproksimirati, na primer za random 3-SAT, tako što se za neko $N$ i za vrednosti $L$ iz nekog opsega, slučajno generiše veliki broj instanci random 3-SAT problema i računa udeo zadovoljivih formula.



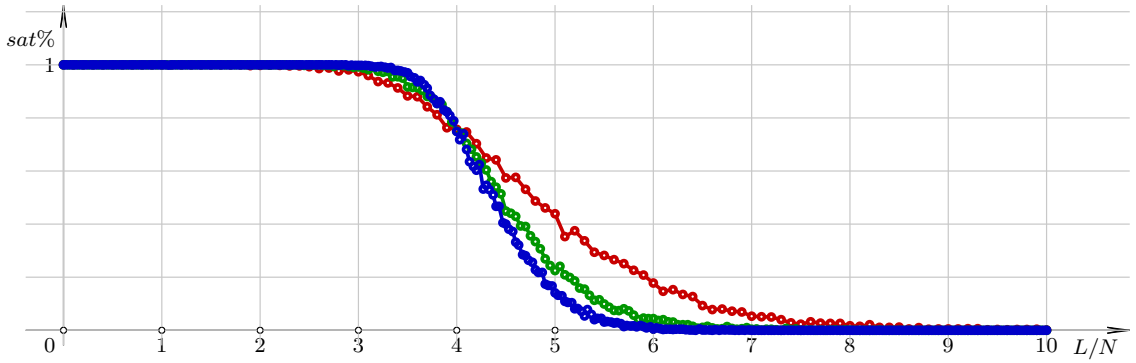

Procenat zadovoljivih 3-SAT formula sa $N = 10$ **promenljivih**, za $L = 0, \ldots, 100$, prosečna vrednost za 1000 formula.



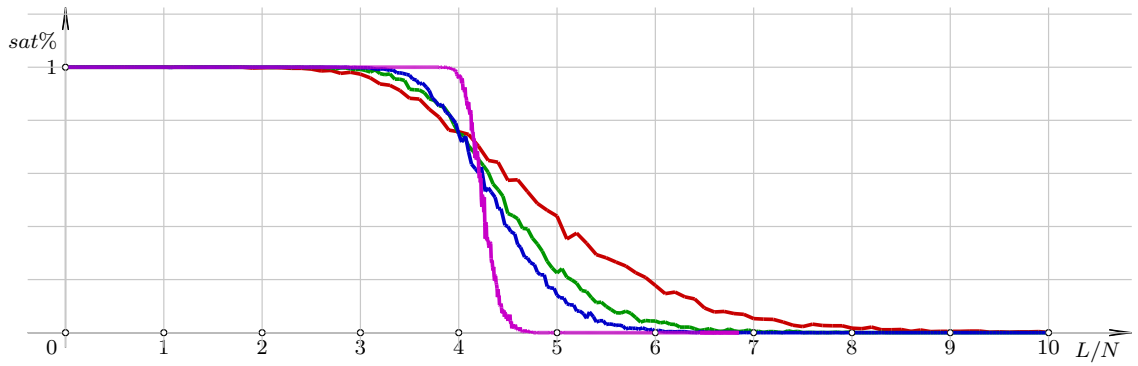

Procenat zadovoljivih 3-SAT formula sa $N = 20$ **promenljivih**, za $L = 0, \ldots, 200$, prosečna vrednost za 1000 formula.



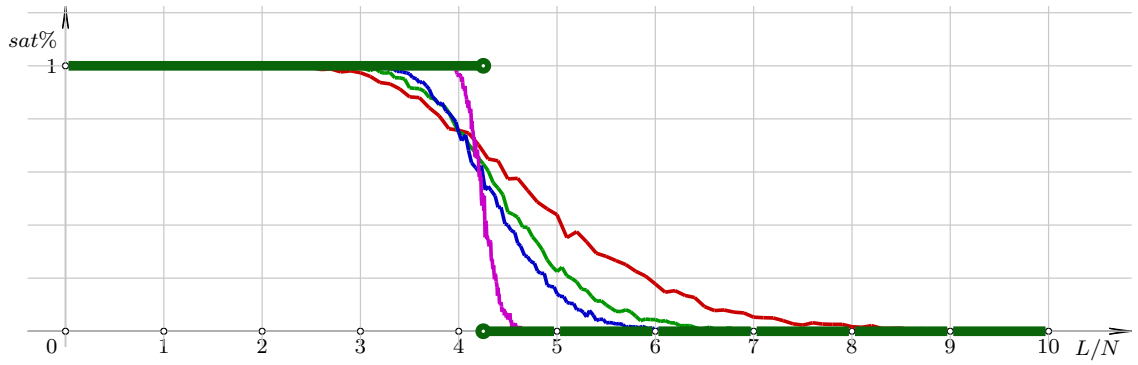

Procenat zadovoljivih 3-SAT formula sa $N = 30$ **promenljivih**, za $L = 0, \ldots, 300$, prosečna vrednost za 1000 formula.



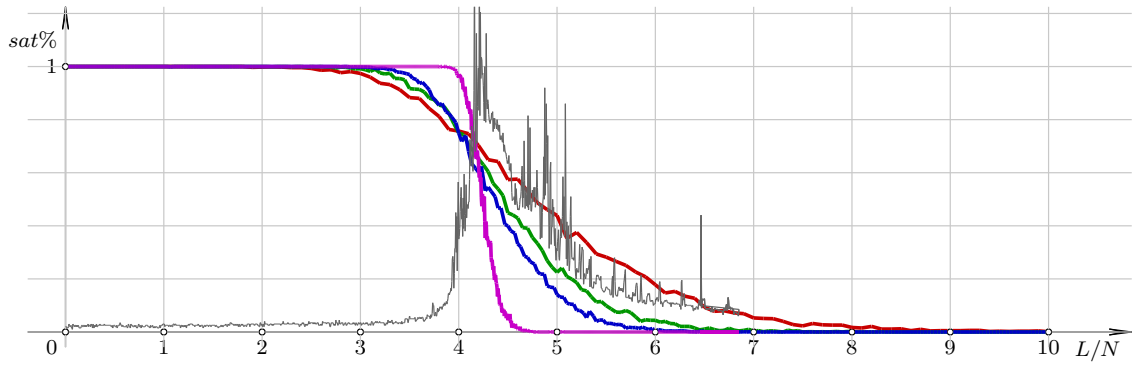

Procenat zadovoljivih 3-SAT formula sa $N = 10, 20, 30$ promenljivih, za $L/N = 0, \ldots, 10$, prosečna vrednost za 1000 formula.




Procenat zadovoljivih 3-SAT formula sa $N = 10, 20, 30, 200$ promenljivih, za $L/N = 0, \ldots, 10$, prosečna vrednost za 1000 formula.




Procenat zadovoljivih 3-SAT formula sa $N = 10, 20, 30, 200$ promenljivih, za $L/N = 0, \ldots, 10$, prosečna vrednost za 1000 formula i vrednosti kojoj konvergira kada $N \to \infty$.




Procenat zadovoljivih 3-SAT formula sa $N = 10, 20, 30, 200$ promenljivih, za $L/N = 0, \ldots, 10$, prosečna vrednost za 1000 formula i vreme izvršavanja za $N = 200$.

Za različite tipove skupova $M(N, L)$, veruje se da postoji kritična vrednost $c_0$ za odnos $L/N$, koju zovemo *prelomna tačka* ili *tačka promene faze* takva da važi:

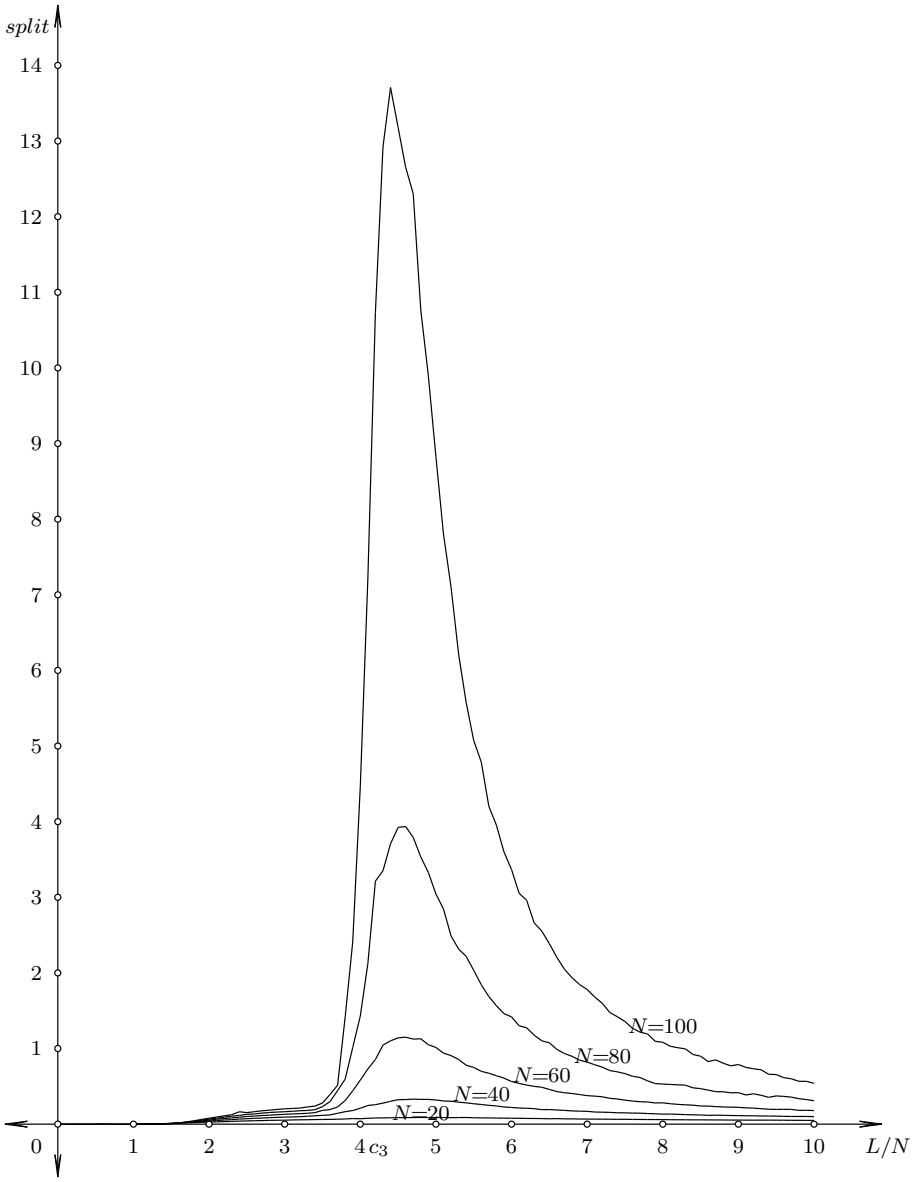
$$\lim_{N\to\infty} sat(M(N, [cN])) = \begin{cases} 100\%, & \text{za } c < c_0 \\ 0\%, & \text{za } c > c_0 \end{cases}$$

Dodatno, ako prelomna tačka postoji, ona je jedinstvena (za jedan tip SAT problema). Do sada ni za jedan tip SAT problema nije teorijski određena prelomna tačka (sa jedinim izuzetkom koji čini problem 2-SAT).

Problem 2-SAT problem je polinomski rešiv, ali i za njega promena faze kao za $k$-SAT probleme za $k > 2$. Dokazano je (teorijski) da je prelomna tačka za 2-SAT problem jednaka 1.

Za random 3-SAT problem, prelomna tačka eksperimentalno je locirana kao vrednost $L/N \approx 4.25$. Za random 4-SAT problem, prelomna tačka je aproksimirana kao $L/N \approx 9.76$. Za random $k$-SAT dokazano je da postoji region promene faze koji se sužava kada raste broj iskaznih varijabli i da je problem prelomna tačka jednaka $2^k \ln 2 - O(k)$.

Kao što je rečeno, u mnogim klasama SAT problema postoji promena faze između zadovoljivih i nezadovoljivih formula, ali interesantno je analizirati i vreme potrebno za rešavanje instanci. Eksperimenti pokazuju i da u svim




Slika 6.1: Eksperimentalna aproksimacija složenosti izračunavanja za 3-SAT problem za $N = 20$, $N = 40$, $N = 60$, $N = 80$ i $N = 100$ za DPLL proceduru (kao mera složenosti uzet je najveći broj primena *split* pravila u jednoj grani prostora pretrage)

random SAT problemima postoji tipičan obrazac *jednostavno-teško-jednostavno* kada se vrednost $L/N$ povećava. Zaista, za male vrednosti $L/N$, problemi su veoma slabo ograničeni i relativno jednostavni za svaku proceduru odlučivanja za iskaznu logiku (jer postoji mnogo zadovoljavajućih valuacija); za velike vrednosti $L/N$, problemi su veoma jako ograničeni, pa je relativno lako pokazati da su oni nezadovoljivi. Interesantno je da su najteže instance SAT problema za sve procedure odlučivanja upravo instance koje se nalaze u regionu fazne promene (videti sliku 6.1).

Promena faze postoji i u drugim NP-kompletnim problemima (za pogodno izabrane parametre).

Glava 7

# Rešavanje problema svođenjem na SAT

## 7.1 SAT kao CSP

*Problem zadovoljenja ograničenja* je trojka $(V, D, C)$, gde je $V$ konačan skup promenljivih $v_1$, $v_2$, ..., $v_n$, $D$ je skup domena $d_1$, $d_2$, ..., $d_n$ za ove promenljive, i $C$ je skup ograničenja $c_1$, $c_2$, ..., $c_k$. U slučaju konačnog CSP, svi skupovi $D$ su konačni. Ograničenja $C$ određuju kombinacije vrednosti promenljivim koje su *dozvoljene* ili koji su *zabranjene*. Instanca problema je *zadovoljiva* ako postoji dodela vrednosti promenljivim tako da su sva ograničenja zadovoljena. Takva dodela naziva se *rešenje*. Problem optimizacije ograničenja je problem u kojem je cilj da se pronađe rešenje koje maksimizira (ili minimizira) datu *ciljnu funkciju* u odnosu na sve dozvoljene vrednosti promenljivih.

**Problem SAT** Problem SAT, očigledno, je specijalan slučaj CSP, sa svim promenljivama koje se imaju vrednost iz skupa $\{0, 1\}$ i sa datim ograničenjima u vidu klauza.

## 7.2 Faze rešavanja svođenjem na SAT

Mnogi praktični problemi mogu se rešiti korišćenjem iskazne logike. Obično je postupak rešavanja ovakav:

- elementarni iskazi (tvrdnje) koji figurišu u opisu problema, predstavljaju se iskaznim promenljivim (u duhu nekog *kodiranja*);
- uslovi problema se predstavljaju iskaznim formulama nad tim iskaznim promenljivim;
- konjunkcija tih iskaznih formula transformiše se u konjunktivnu normalnu formu;
- zadovoljivost formule u konjunktivnoj normalnoj formi ispituje se SAT rešavačem;
- ukoliko je formula zadovoljiva, svaki njen model daje jedno rešenje polaznog problema.

Svođenjem na SAT mogu se pogodno opisati mnogi problemi nad konačnim domenima. Naredni primer pokazuje kako sabiranje prirodnih brojeva, a i rešavanje jednačina koje uključuju takvo sabiranje mogu biti svedeni na SAT.

**Primer 7.1.** *Šef protokola na jednom dvoru treba da organizuje bal za predstavnike ambasada. Kralj traži da na bal bude pozvan Peru ili da ne bude pozvan Katar (Qatar). Kraljica zahteva da budu pozvani Katar ili Rumunija (ili i Katar i Rumunija). Princ zahteva da ne bude pozvan Peru ili da ne bude pozvana Rumunija (ili da ne budu pozvani ni Peru ni Rumunija). Da li je moguće organizovati bal tako da su zadovoljeni zahtevi svih članova kraljevske porodice?*

*Navedeni problem potrebno je najpre modelovati na neki precizan način. Iskaz, tvrdnju „na bal će doći ambasador Perua“ označićemo sa $p$, iskaz „na bal će doći ambasador Katara“ označićemo sa $q$, a iskaz „na bal će doći ambasador Rumunije“ sa $r$. Uslov koji postavlja kralj, onda glasi „važi $p$ ili ne važi $q$“ ili kraće zapisano „$p$ ili ne $q$“. Uslov koji postavlja kraljica glasi „$q$ ili $r$“. Uslov koji postavlja princ glasi „ne $p$ ili ne $r$“.*

*Sva navedena ograničenja, svi ovi iskazi, zajedno čine novi, komplikovaniji iskaz koji možemo da zapišemo na sledeći način:*

*„($p$ ili ne $q$) i ($q$ ili $r$) i (ne $p$ ili ne $r$)“*

*Ovaj složeni iskaz predstavlja precizan zapis problema. Potrebno je proveriti da li polazni iskazi p, q i r mogu da imaju konkretne vrednosti* tačno *ili* netačno *takve da složeni iskaz ima vrednost* tačno. *Da bi se taj problem rešio potrebno je precizno definisati i na koji način se složenim iskazima pridružuje vrednost* tačno *ili* netačno *ukoliko je poznato koje vrednosti su pridružene polaznim iskazima.*

*Problem možemo zapisati u DIMACS obliku (promenljivog p odgovara oznaka* 1*, promenljivoj q oznaka* 2*, promenljivoj r oznaka* 3*):*

```
c
p cnf 3 3
1 -2 0
2 3 0
-1 -3 0
```

*Ako se pozove rešavač CaDiCaL:*

```
\$ ./cadical bal.p
```

*on daje naredni izlaz:*

```
c --- [ banner ] ------------------------------------------------------------
c
c CaDiCaL Radically Simplified CDCL SAT Solver
c Copyright (c) 2016-2021 A. Biere, M. Fleury, N. Froleyks
c
c Version 1.5.2 ca9bff05c11bde5eae4912f9932871d1527e61d8
c g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0 -Wall -Wextra -O3 -DNDEBUG
c Mon Feb 28 11:03:00 CET 2022 Linux lifebook 5.13.0-30-generic x86_64
c
c --- [ parsing input ] -------------------------------------------------
c
c reading DIMACS file from 'bal.p'
c opening file to read 'bal.p'
c found 'p cnf 3 3' header
c parsed 3 clauses in 0.00 seconds process time
c closing file 'bal.p'
c after reading 46 bytes 0.0 MB
c
c --- [ options ] -------------------------------------------------------
c
c all options are set to their default value
c
c --- [ solving ] -------------------------------------------------------
c
c time measured in process time since initialization
c
c  seconds  reductions  redundant irredundant
c         MB    restarts       trail    variables
c          level   conflicts       glue     remaining
c
c *  0.00  2  0 0   0    0    0  0% 0    3   3 100%
c 1  0.00  2  0 0   0    0    0  0% 0    3   3 100%
c 1  0.00  2  0 0   0    0    0  0% 0    3   3 100%
c
c --- [ result ] --------------------------------------------------------
c
s SATISFIABLE
v 1 2 -3 0
c
c --- [ run-time profiling ] --------------------------------------------
c ...
```

*Ključni deo je deo*

```
s SATISFIABLE
v 1 2 -3 0
```

*koji govori da je ulazna instanca zadovoljiva i da je jedan model valuacija u kojoj promenljiva* 1 *(tj.* $p$*) ima vrednost* 1*, promenljiva* 2 *(tj.* $q$*) ima vrednost* 1 *a promenljiva* 3 *(tj.* $r$*) ima vrednost* 0*.*

*Dakle, svi zadati uslovi će biti tačni ako je*

- *„na bal će doći ambasador Perua“ tačno*
- *„na bal će doći ambasador Katara“ tačno*
- *„na bal će doći ambasador Rumunije“ netačno*

*Ovaj model može se kratko zapisati i u obliku:* $p \wedge q \wedge \neg r$*. Sada može da se postavi pitanje da li postoji još neko rešenje polaznog problema, tj. da li navedena formula ima još neki model. To možemo da proverimo ako navedenoj formuli dodamo uslov koji onemogućava,* blokira *upravo dobijeni model. Taj uslov jednak je* $\neg(p \wedge q \wedge \neg r)$ *tj.* $\neg p \vee \neg q \vee r$*. Ovaj uslov (tj.* `-1 -2 3`*) dodaćemo kao dodatnu klauzu (i zvaćemo je* blokirajuća klauza*):*

```
c
p cnf 3 4
1 -2 0
2 3 0
-1 -3 0
-1 -2 3 0
```

*I nova formula je zadovoljiva i dobijamo model:*

```
c --- [ result ] ---------------------------------------------------------------
SATISFIABLE
v -1 -2 3 0
```

*U ovoj valuaciji* $p$ *ima vrednost* 0*,* $q$ *ima vrednost* 0*, a* $r$ *ima vrednost* 1*. Ovaj model može se kratko zapisati i u obliku:* $\neg p \wedge \neg q \wedge r$*. Analogno kao prošli put, možemo blokirati i ovo rešenje dodajući blokirajuću klauzu* $\neg(\neg p \wedge \neg q \wedge r) = p \vee q \vee \neg r$*:*

```
c
p cnf 3 5
1 -2 0
2 3 0
-1 -3 0
-1 -2 3 0
1 2 -3 0
```

*Ova formula je nezadovoljiva:*

```
c --- [ result ] ---------------------------------------------------------------
s UNSATISFIABLE
```

*te zaključujemo da polazna formula ima samo dva modela, tj. zadati problem ima samo dva rešenja.*

**Primer 7.2.** *Date su tri kutije, zna se da je tačno u jednoj od njih zlato. Na njima piše:*

*na kutiji 1: „zlato nije ovde“*
*na kutiji 2: „zlato nije ovde“*
*na kutiji 3: „zlato je u kutiji 2“*

*Ako se zna da je tačno jedna tvrdnja tačna, treba pogoditi gde je zlato.*

*Neka iskazne promenljive* $b_1$*,* $b_2$ *i* $b_3$ *odgovaraju tvrdnjama „zlato je u kutiji 1“, „zlato je u kutiji 2“ i „zlato je u kutiji 3“.*

*Uslov da je tačno u jednoj od njih zlato može se opisati sledećom formulom:*

$A = (b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge \neg b_2 \wedge b_3)$

*Konjunktivna normalna forma navedene formule je, nakon pojednostavljivanja (koristeći poznate logičke ekvivalencije):*

$A' = (\neg b_1 \vee \neg b_2) \wedge (\neg b_1 \vee \neg b_3) \wedge (\neg b_2 \vee \neg b_3) \wedge (b_1 \vee b_2 \vee b_3) \wedge (b_1 \vee \neg b_2 \vee \neg b_3) \wedge (\neg b_1 \vee b_2 \vee \neg b_3) \wedge (\neg b_1 \vee \neg b_2 \vee b_3) \wedge (\neg b_1 \vee \neg b_2 \vee \neg b_3)$

*Uslov da je tačno jedna ispisana tvrdnja tačna, može se opisati sledećom formulom:*

$B = (\neg b_1 \wedge \neg\neg b_2 \wedge \neg b_2) \vee (\neg\neg b_1 \wedge \neg b_2 \wedge \neg b_2) \vee (\neg\neg b_1 \wedge \neg\neg b_2 \wedge b_2)$

*tj. nakon pojednostavljivanja:*

$B' = (b_1 \wedge \neg b_2) \vee (b_1 \wedge b_2)$

*Konjunktivna normalna forma navedene formule je, nakon pojednostavljivanja (koristeći poznate logičke ekvivalencije):*

$B'' = b_1 \wedge (b_1 \vee b_2) \wedge (b_1 \wedge \neg b_2)$

*Potrebno je pronaći valuaciju u kojoj su obe formule $A'$ i $B''$ tačne. Odgovarajuća DIMACS datoteka može da izgleda ovako:*

```
c
p cnf 3 11
-1 -2 0
-1 -3 0
-2 -3 0
1 2 3 0
1 -2 -3 0
-1 2 -3 0
-1 -2 3 0
-1 -2 -3 0
1 0
1 2 0
1 -2 0
```

*Postoji samo jedna valuacija $v$ u kojoj su svi uslovi tačni.*

```
------------------------------------------------------------
s SATISFIABLE
v 1 -2 -3 0
```

*U ovoj valuaciji važi $I_v(b_1) = 1$ (i $I_v(b_2) = 0$, $I_v(b_3) = 0$), te je zlato u prvoj kutiji. Navedena formula nema drugih modela.*

**Primer 7.3.** *Data je iskazna formula $(p \wedge (q \wedge r)) \vee ((q \wedge r) \wedge \neg p)$. Definicione promenljive $p_4, p_5, p_6, p_7, p_8$ uvode se na sledeći način:*

$$\underbrace{\overbrace{(p \wedge \underbrace{(q \wedge r)}_{p_4})}^{p_6} \vee \overbrace{(\underbrace{(q \wedge r)}_{p_4} \wedge \underbrace{\neg p}_{p_5})}^{p_7}}_{p_8}$$

*Međuoblik za Cejtinovu formu je onda:*

$$p_8 \wedge (p_8 \Leftrightarrow (p_6 \vee p_7)) \wedge (p_6 \Leftrightarrow (p \wedge p_4)) \wedge$$

$$(p_7 \Leftrightarrow (p_4 \wedge p_5)) \wedge (p_4 \Leftrightarrow (q \wedge r)) \wedge (p_5 \Leftrightarrow \neg p))$$

*Konačno, izlazna* KNF *formula je:*

$p_8 \wedge$
$(\neg p_8 \vee p_6 \vee p_7) \wedge (p_8 \vee \neg p_6) \wedge (p_8 \vee \neg p_7) \wedge$
$(p_6 \vee \neg p \vee \neg p_4) \wedge (\neg p_6 \vee p) \wedge (\neg p_6 \vee p_4) \wedge$
$(p_7 \vee \neg p_4 \vee \neg p_5) \wedge (\neg p_7 \vee p_4) \wedge (\neg p_7 \vee p_5) \wedge$
$(p_4 \vee \neg q \vee \neg r) \wedge (\neg p_4 \vee q) \wedge (\neg p_4 \vee r) \wedge$
$(p_5 \vee p) \wedge (\neg p_5 \vee \neg p)$

*Odgovarajuća DIMACS datoteka može da izgleda ovako (promenljive $p$, $q$ i $r$ su označene sa 1, 2 i 3):*

```
c
p cnf 8 15
8 0
-8 6 7 0
8 -6 0
8 -7 0
6 -1 -4 0
-6 1 0
-6 4 0
7 -4 -5 0
-7 4 0
-7 5 0
4 -2 -3 0
-4 2 0
-4 3 0
5 1 0
-5 -1 0
```

*Formula je zadovoljiva i ovo je jedan od postojećih modela:*

```
c --- [ result ] -------------------------------------------
s SATISFIABLE
v 1 2 3 4 -5 6 -7 8 0
```

### 7.2.1 Primeri kodiranja

**Retka kodiranja.** U praktičnim problemima koji se rešavaju svođenjem na SAT, ne figurišu samo iskazne promenljive, već često i celobrojne promenljive koje mogu imati vrednosti iz nekog ograničenog skupa. U takvim situacijima često se koristi *retko kodiranje* (eng. *sparse encoding*) u kojem se uvode iskazne promenljive $p_{a,i}$ koje su tačne ako i samo ako promenljiva $a$ ima vrednost $i$. Time se uslov da promenljiva $a$ ima jednu vrednost iz zadatog domena $I$ zadaje uslovom (uslov „barem jedna“):

$$\bigvee_{i \in I} p_{a,i}$$

Promenljiva $a$ ne može imati dve vrednosti istovremeno, što se opisuje formulom (uslov „najviše jedna“):

$$\bigwedge_{i,j \in I, i \neq j} \neg p_{a,i} \vee \neg p_{a,j}$$

Pored uslova koji su potrebni kako bi se iskazalo da promenljiva ima (tačno jednu) vrednost iz nekog konačnog skupa, potrebno je kodirati i razna druga ograničenja. Za neka ograničenja koja se često koriste, postoje ustaljeni načini kodiranja koji daju specifične varijante retkog kodiranja.

U *direktnom kodiranju* (eng. *direct encoding*), za svaku nedozvoljenu kombinaciju vrednosti $f(a)$ promenljivih $a$ iz nekog skupa $S$ uvodi se takozvana „klauza konflikta”:

$$\neg \bigwedge_{a \in S} p_{a,f(a)}$$

to jest

$$\bigvee_{a \in S} \neg p_{a,f(a)}$$

Ovakve klauze mogu se koristiti i kada je potrebno pronaći više rešenja ili sva rešenja nekog problema. Kada se pronađe jedno rešenje, onda polaznoj formuli treba dodati klauzu koja zabranjuje tu kombinaciju vrednosti (zovemo je i „blokirajuća klauza“) i rešiti tako dobijeni problem. Postupak se analogno nastavlja i kada je pronađeno nekoliko rešenja.

U *potpornom kodiranju* (eng. *support encoding*) ograničenja oblika „ako $a$ ima vrednost $i$, onda $b$ mora imati neku od vrednosti iz skupa $I$“ opisuju se formulama sledećeg oblika:

$$\neg p_{a,i} \vee \bigvee_{j \in I} p_{b,j}.$$

**Logaritamsko kodiranje.** U *logaritamskom kodiranju* (eng. *log encoding*) svakom bitu vrednosti numeričkih promenljivih (zapisanih u binarnoj reprezentaciji) pridružuje se jedna iskazna promenljiva. U ovoj reprezentaciji ne postoji potreba za uslovima „barem jedna“ i „najviše jedna“, jer svaka valuacija uvedenih iskaznih promenljivih daje tačno jednu vrednost odgovarajuće promenljive. Naravno, kada je broj mogućih vrednosti numeričke promenljive manji od broja mogućih vrednosti iskaznih promenljivih koji se koriste za njeno kodiranje, neke valuacije iskaznih promenljivih potrebno je zabraniti dodatnim klauzama (na primer, ako promenljiva $n$ može da ima vrednosti od 0 do 6, za njeno kodiranje se koriste tri iskazne promenljive, ali se zabranjuje valuacija koja daje vrednost 7).

I u logaritamskom kodiranju moguće je izraziti uslove koje u slučaju retkih kodiranja izražavaju direktno i potporno kodiranje, ali zbog prirode logaritamskog kodiranja, te uslove potrebno je zadati nad binarnim kombinacijama koje predstavljaju vrednosti numeričkih promenljivih. Na primer, neka promenljive $a$ i $b$ uzimaju celobrojne vrednosti od 0 do 7 i neka su kodirane iskaznim promenljivim $p_{a,0}$, $p_{a,1}$, $p_{a,2}$, $p_{b,0}$, $p_{b,1}$ i $p_{b,2}$, pri čemu viši indeksi označavaju bitove veće težine. Ukoliko se vrednost 3 promenljive $a$ uzajamno isključuje sa vrednošću 6 promenljive $b$, taj uslov može se kodirati u terminima bitova, klauzom

$$p_{a,2} \lor \neg p_{a,1} \lor \neg p_{a,0} \lor \neg p_{b,2} \lor \neg p_{b,1} \lor p_{b,0}.$$

**Primer 7.4.** *Zadatak je obojiti dve kuće (neka su označene sa a i b) po jednom od tri raspoložive boje (neka su označene brojevima 1, 2, 3), ali tako da budu obojene različito. Neka iskazna promenljiva $p_{a,1}$ predstavlja tvrdnju da je kuća a obojena bojom 1, neka $p_{a,2}$ predstavlja tvrdnju da je kuća a obojena bojom 2, ... i neka $p_{b,3}$ predstavlja tvrdnju da je kuća b obojena bojom 3.*

*U retkim kodiranjima problema biće potrebni uslovi „barem jedna“:*

$p_{a,1} \lor p_{a,2} \lor p_{a,3}$
$p_{b,1} \lor p_{b,2} \lor p_{b,3}$

*i uslovi „najviše jedna“:*

$\neg p_{a,1} \lor \neg p_{a,2}$
$\neg p_{a,1} \lor \neg p_{a,3}$
$\neg p_{a,2} \lor \neg p_{a,3}$
$\neg p_{b,1} \lor \neg p_{b,2}$
$\neg p_{b,1} \lor \neg p_{b,3}$
$\neg p_{b,2} \lor \neg p_{b,3}$

*Dodatno, u direktnom kodiranju biće opisan i uslov da nisu obe kuće obojene istom bojom (klauze konflikta):*

$\neg p_{a,1} \lor \neg p_{b,1}$
$\neg p_{a,2} \lor \neg p_{b,2}$
$\neg p_{a,3} \lor \neg p_{b,3}$

*Neka promenljivama $p_{a,1}$, $p_{a,2}$, $p_{a,3}$, $p_{b,1}$, $p_{b,2}$. $p_{b,3}$ odgovaraju oznake 1, 2, 3, 4, 5, 6. Navedeni uslovu mogu se onda zapisati u DIMACS formatu na sledeći način:*

```
c
p cnf 6 11
1 2 3 0
4 5 6 0
-1 -2 0
-1 -3 0
-2 -3 0
-4 -5 0
-4 -6 0
-5 -6 0
-1 -4 0
-2 -5 0
-3 -6 0
```

*CaDiCaL rešavač daje sledeći model:*

```
c --- [ result ] ------------------------------------------------------------
s SATISFIABLE
v 1 -2 -3 -4 5 -6 0
```

*Dakle, prvu kuću treba obojiti prvom bojom, a drugu kuću drugom bojom (postoje, naravno, i druga rešenja).*

*Alternativno, navedene klauze konflikta bi mogle, korišćenjem potpornog kodiranja, da se zadaju na sledeći način:*

$\neg p_{a,1} \vee (p_{b,2} \vee p_{b,3})$
$\neg p_{a,2} \vee (p_{b,1} \vee p_{b,3})$
$\neg p_{a,3} \vee (p_{b,1} \vee p_{b,2})$
$\neg p_{b,1} \vee (p_{a,2} \vee p_{a,3})$
$\neg p_{b,2} \vee (p_{a,1} \vee p_{a,3})$
$\neg p_{b,3} \vee (p_{a,1} \vee p_{a,2})$

*Pažljivom analizom može se pokazati da poslednje tri navedene klauze nisu potrebne.*

*Problem može biti opisan i korišćenjem logaritamskog kodiranja: neka iskazne promenljive* $p_{a,0}$, $p_{a,1}$ *odgovaraju ciframa binarnog zapisa boje koja odgovara kući* $a$, *a* $p_{b,0}$, $p_{b,1}$ *odgovaraju ciframa binarnog zapisa boje koja odgovara kući* $b$, *pri čemu viši indeksi označavaju bitove veće težine. Ako su bojama pridružene vrednosti* 0, 1 *i* 2, *treba zabraniti vrednost* 3, *pa zato postoje uslovi:*

$\neg p_{a,1} \vee \neg p_{a,0}$
$\neg p_{b,1} \vee \neg p_{b,0}$

*Klauze konflikta su sledeće klauze:*

$p_{a,1} \vee p_{a,0} \vee p_{b,1} \vee p_{b,0}$ *(nisu obe boje* 0*)*
$p_{a,1} \vee \neg p_{a,0} \vee p_{b,1} \vee \neg p_{b,0}$ *(nisu obe boje* 1*)*
$\neg p_{a,1} \vee p_{a,0} \vee \neg p_{b,1} \vee p_{b,0}$ *(nisu obe boje* 2*)*

Glava 8

# Sistem ursa

U ovom delu biće predstavljen rešavač ograničenja URSA zasnovan na svođenju na SAT, kao i nekoliko raznorodnih problema koji se mogu rešiti korišćenjem ovog sistema. Neki od tih problema mogu se elegantno rešiti i korišćenjem drugih rešavača ograničenja, te se ovaj deo materijala može smatrati i demonstracijom upotrebe rešavanja ograničenja generalno.

## 8.1 Sistem URSA i imperativno-deklarativna programska paradigma

URSA (od Uniform Reduction do SAt) je sistem koji olakšava rešavanje problema ograničenja svođenjem na SAT. Ovaj sistem može se razmatrati i kao rešavač za programiranje ograničenja opšte namene (za konačne domene)). Sistem URSA koristi namenski jezik za modelovanje, tj. za specifikovanje problema i prateći prevodilac. Nasuprot drugim jezicima za modelovanje, jezik URSA kombinuje karakteristike deklarativne i imperativne programske paradigme. Ono što jezik čini deklarativnim nije način na koji su ograničenja izražena, već činjenica da postupak za pronalaženje rešenja nije potrebno dati eksplicitno. S druge strane, sistem ima karakteristike imperativnih jezika – podržava jednostavan proces modelovanja koji odgovara pisanju, na imperativni način, testa da li su neki objekti zaista rešenje date instance problema. Stoga se predloženi sistem može razmatrati kao proširenje paradigme imperativnog programiranja (sa ugrađenim rešavanjem problema ograničenja), slično kao što su sistemi programiranja ograničenja, na primer, proširenja paradigme logičkog programiranja. Za razliku od drugih jezika za modelovanje, u sistemu URSA petlje se zadaju u imperativnom stilu (a ne preko rekurzija) i dozvoljen je destruktivni update. Ove karakteristike često čine proces modeliranja lakši, a specifikacije čitljivijim i lakšim za održavanje (naročito korisnicima koji su navikli na imperativno programiranje). Generalno, za deklarativni program je često lakše nego za odgovarajući imperativni program ispitati da li zadovoljava datu specifikaciju (i stoga deklarativni jezici mogu dati veći stepen poverenja). Međutim, ovo još uvek ne vodi do većeg poverenja ako sam sistem za rešavanje ograničenja ne može biti verifikovan.

Broj bitova u brojevnim promenljivama zadaje se opcijom `-l`, na primer, `-l10` (default vrednost je 8). U nastavku, ako nije drugačije rečeno, podrazumeva se da je URSA pokrenuta sa default vrednošću ovog parametra. Sve brojevne vrednosti su neoznačene.

Sistem URSA posebno je pogodan za rešavanje problema u kojima je jednostavno proveriti da li je nešto (na primer, niz vrednosti) rešenje (tj. u kojima je jednostavno proveriti *sertifikat*.

## 8.2 Uvodni primer: Bal

**Primer 8.1.** *Razmotrimo ponovo problem organizovanja bala iz primera 7.1. Uslov koji postavlja kralj, onda glasi „važi p ili ne važi q“ ili kraće zapisano „p ili ne q“. Uslov koji postavlja kraljica glasi „q ili r“. Uslov koji postavlja princ glasi „ne p ili ne r“. Ovi uslovi u sistemu URSA mogu se zadati na sledeći način:*

```
bBal = (bPeru || !bRumunija) && (bRumunija || bKatar) && (!bPeru || !bKatar);
assert_all(bBal);
```

*Navedeni kôd formulisan je tako da proverava da li su zaista ispunjeni dati uslovi. Ukoliko vrednosti iskaza nisu zadate, onda će biti tražen model u kojem su tačni. Pronalaženje modela inicira se naredbom* `assert_all(bBal);`*. Sistem URSA poziva se na sledeći način:*

```
\$ ./ursa < bal.urs
```

*i daje se sledeći izlaz:*

```
 ************************************************
****  URSA Interpreter v4.00 (c) 2010-2020  ****
*** Predrag Janicic,  University of Belgrade ***
************************************************

--> Solution 1
bKatar=true;
bPeru=false;
bRumunija=false;

--> Solution 2
bKatar=false;
bPeru=true;
bRumunija=true;

[Formula generation: 0.000164s; conversion to CNF: 0.000203s; total: 0.000367s]
[Solving time: 5.8e-05s]
[Formula size: 3 variables, 3 clauses]
```

## 8.3 Uvodni primer: sabiranje

**Primer 8.2.** *Neka je zadat problem određivanja vrednosti u, ako je poznato da je $v = 2$ i $v = u + 1$ (po modulu 4). Problem se može rešiti korišćenjem sistema URSA na sledeći način:*

```
nv = nu + 1;
assert(nv == 2);
```

*Sistem URSA poziva se na sledeći način:*

```
\$ ./ursa < sabiranje.urs
```

*i daje se sledeći izlaz:*

```
nu=1;

[Formula generation: 0.00044s; conversion to CNF: 0.000754s; total: 0.001194s]
[Solving time: 0.000128s]
[Formula size: 21 variables, 69 clauses]
```

## 8.4 Uvodni primer: bojenje kuća

**Primer 8.3.** *Razmotrimo ponovo problem bojenja kuća opisan u primeru 7.4. Ovaj problem u sistemu URSA može se zadati na sledeći način:*

```
assert_all (nV<=2 && nW<=2 && nV!=nW);
```

*Sistem URSA poziva se na sledeći način (dovoljno je koristiti dužinu 2 za brojevne promenljive):*

```
\$ ./ursa < boje.urs -l2
```

*i daje sledeći izlaz:*

```
 --> Solution 1
nV=2;
nW=1;

--> Solution 2
nV=2;
nW=0;

--> Solution 3
nV=1;
nW=0;

--> Solution 4
nV=0;
nW=1;

--> Solution 5
nV=1;
nW=2;

--> Solution 6
nV=0;
nW=2;

[Formula generation: 0.000281s; conversion to CNF: 0.000661s; total: 0.000942s]
[Solving time: 0.000234s]
[Formula size: 8 variables, 17 clauses]
```

## 8.5 Sintaksa jezika URSA

U jeziku sistema URSA postoje dve vrste promenljivih — brojevne, čiji identifikatori počinju sa sa `n` (na primer, `nKs`) i bulovske, čiji identifikatori počinju sa `b` (na primer, `bKs`). Ista konvencija važi za identifikatore nizova. Promenljive se ne deklarišu, već se uvode dinamički. Postoje funkcije (`bool2num` i `num2bool`) za konverziju bulovskih vrednosti u brojevne vrednosti i obrnuto, kao i funkcija `sgn` koja odgovara funkciju *signum*. Promenljive mogu imati konkretne (bazne) ili simboličke vrednosti (u tom slučaju su interno predstavljene vektorima iskaznih formula).

Podržani su aritmetički, bitni, relacioni i operatori, kao i složeni operatori dodele, u stilu jezika C. Na primer, bitovska konjunkcija nad brojevnim promenljivama `n1` i `n2` zapisuje se `n1 & n2`, bitovsko pomeranje ulevo od `n1` za `n2` zapisuje se `n1 << n2`, a naredba `n1 += n2` ekvivalentan je naredbi `n1 = n1+n2`. Logički operatori primenjuju se nad bulovskim izrazima i zapisuju se u C-stilu, pri čemu postoji i dodatni operator `^^` za logičku isključivu disjunkciju, u duhu drugih logičkih operatora jezika C. Postoje i složeni operatori dodele za logičke operatore, kao što je `&&=` (dodato zbog simetrije i pogodnosti, mada ne postoje u jeziku C). Ne postoji naredba `if-else`, ali postoji izraz `ite` (koji odgovara uslovnom operatoru `?:` u jeziku C): `ite(b,n1,n2)` je jednako `n1` ako je `b` tačno, a jednako `n2` u suprotnom.

Nisu podržane korisnički definisane funkcije, već samo korisnički definisane procedure. Petlje `for` moraju imati konkretne (bazne) granice u uslovima izlaska. Elementi nizova moraju imati konkretne (bazne) indekse. Elementi nizova ne mogu biti argumenti procedura.

URSA program je niz naredbi (i definicija procedura).

URSA specifikacija se izvršava simbolički i konstruišu se iskazne formula koje odgovara promenljivama.

Uloga naredbe `assert` je da potvrdi da je neko ograničenje (dato kao argument) zadovoljeno (kao u C-u). Tokom interpretiranja programa URSA, ova naredba transformiše svoj argument u u KNF oblik, prosleđuje ga jednom od ugrađenih SAT rešavača radi pronalaženja modela. Naredba `assert_all` je analogna, ali traži sve modele. Jedan program može imati nekoliko naredbi ove vrste, ali svaka od njih radi lokalno (kao u C-u), tj. poziva mehanizam rešavanja samo za sopstveni argument. Naredbe `minimize` (i `makimize`) govore da treba da bude pronađena minimalna (ili maksimalna) vrednost unutar datog opsega za datu numeričku promenljivu. Naredbe `assert` i `assert_all` uzimaju u obzir samo poslednju naredbu `minimize`/`makimize`.

Opis sintakse URSA jezika dat je u narednoj tabeli u EBNF notaciji (⟨num var⟩ označava sintaksičku klasu brojevnih promenljvih, ⟨num ekpr⟩ označava sintaksičku klasu numeričkih izraza, ⟨bool ekpr⟩ označava sintaksičku klasu Bulovih izraza, itd).

Postoje dodatne pomoćne naredbe, uglavnom namenjene za korišćenje u interaktivnom režimu (za navođenje vrednosti i statusa promenljivih — `listvars`, za brisanje vrednosti svih postojećih promenljivih — `clear`, kao i zaustavljanje interpretatora — `halt`).

```
⟨program⟩          ::=  ⟨procedure def⟩* ⟨statement⟩*
⟨procedure def⟩    ::=  "procedure"⟨procedure name⟩ ( "("
                          "("(⟨num var id⟩ | ⟨bool var id⟩) ("," (⟨num var id⟩ | ⟨bool var id⟩))* ")")
                          "{"⟨statement⟩* "}"
⟨procedure name⟩   ::=  ⟨letter⟩(⟨letter⟩ | ⟨digit⟩)*
⟨statement⟩        ::=  "{"⟨statement⟩* "}"
                        | ⟨num var⟩ ⟨assign num op⟩ ⟨num expr⟩ ";"
                        | ⟨num var⟩ ⟨num op postfix⟩ ";"
                        | ⟨bool var⟩ ⟨assign bool op⟩ ⟨bool expr⟩ ";"
                        | "while("⟨bool expr⟩ ")"⟨statement⟩
                        | "for("⟨statement⟩ ";"⟨bool expr⟩ ";"⟨statement⟩ ")"⟨statement⟩
                        | čall"⟨procedure name⟩ ( "("
                           "("(⟨num expr⟩ | ⟨bool expr⟩) ("," (⟨num expr⟩ | ⟨bool expr⟩))* ")") ";"
                        | "minimize("⟨num var⟩ "," ⟨num const⟩ "," ⟨num const⟩ ");"
                        | "maximize("⟨num var⟩ "," ⟨num const⟩ "," ⟨num const⟩ ");"
                        | "assert("⟨bool expr⟩ (";"⟨bool expr⟩)* ");"
                        | "assert_all("⟨bool expr⟩ (";"⟨bool expr⟩)* ");"
                        | "print"(⟨num expr⟩ | ⟨bool expr⟩) ";"
                        | "listvars;"
                        | člear;"
                        | "halt;"
⟨num expr⟩         ::=  ⟨num const⟩
                        | ⟨num var⟩
                        | ⟨un num op⟩ ⟨num expr⟩
                        | ⟨num expr⟩ ⟨num op⟩ ⟨num expr⟩
                        | ite "("⟨bool expr⟩ "," ⟨num expr⟩ "," ⟨num expr⟩")"
                        | šgn("⟨num expr⟩ ")"
                        | "bool2num("⟨bool expr⟩ ")"
                        | "("⟨num expr⟩ ")"
⟨num var⟩          ::=  ⟨num var id⟩
                        | ⟨num var id⟩ "["⟨num expr⟩ "]"
                        | ⟨num var id⟩ "["⟨num expr⟩ "]["⟨num expr⟩ "]"
⟨num const⟩        ::=  (⟨digit⟩)+
⟨num var id⟩       ::=  "n"(⟨letter⟩ | ⟨digit⟩)*
⟨assign num op⟩    ::=  "="| "+="| "-="| "*="| "&="| "|="| "^="| "<<="| ">>="
⟨num op⟩           ::=  "+"| "-"| "*"| "&"| "|"| "^"| "<<"| ">>"
⟨un num op⟩        ::=  "−"| "~"
⟨num op postfix⟩   ::=  "++"| "--"
⟨num rel⟩          ::=  "<"| ">"| "<="| ">="| "=="| "!="
⟨bool expr⟩        ::=  ⟨bool const⟩
                        | ⟨bool var⟩
                        | ⟨bool expr⟩ ⟨bool op⟩ ⟨bool expr⟩
                        | ⟨un bool op⟩ ⟨bool expr⟩
                        | ⟨num expr⟩ ⟨num rel⟩ ⟨num expr⟩
                        | ite "("⟨bool expr⟩ "," ⟨bool expr⟩ "," ⟨bool expr⟩")"
                        | "num2bool("⟨num expr⟩ ")"
                        | "("⟨bool expr⟩ ")"
⟨bool const⟩       ::=  ( "true"| "false")
⟨bool var⟩         ::=  ⟨bool var id⟩
                        | ⟨bool var id⟩ "["⟨num expr⟩ "]"
                        | ⟨bool var id⟩ "["⟨num expr⟩ "]["⟨num expr⟩ "]"
⟨bool var id⟩      ::=  "b"(⟨letter⟩ | ⟨digit⟩)*
⟨assign bool op⟩   ::=  "="| "&&="| "||="| "^^="
⟨bool op⟩          ::=  "&&"| "||"| "^^"
⟨un bool op⟩       ::=  "!"
```

## 8.6 Određivanje maksimuma

**Primer 8.4.** *Razmotrimo problem određivanja maksimuma dva broja.*

*U sistemu URSA, maksimum se može opisati slično kao u jeziku C (operator* `ite` *odgovara operatoru* `?:` *jezika C). Ako se izvrši naredni kôd:*

```
nA = 3;
nB = 2;
nMax = ite(nA>nB, nA, nB);
print nMax;
```

*dobićemo izlaz:*

```
 --> ground number
 --> value: 3

no
[Number of solutions: 0]
```

*tj. promenljiva* `nMax` *ima vrednost 3 (a nema ograničenja koja treba rešiti). Ovakvo ponašanje je očekivano i odgovara ponašanju u imperativnom jeziku kao što je C. Međutim, naredni kod možemo iskoristiti i drugačije:*

```
nMax = ite(nA>nB, nA, nB);
assert_all(nMax == 3);
```

*Na ovaj način tražimo sve parove prirodnih brojeva* `nA` *i* `nB` *takvih da je njihov maksimum jednak 3. Izvršavanjem ove specifikacije dobijamo izlaz:*

```
--> Solution 1
nA=2;
nB=3;

--> Solution 2
nA=0;
nB=3;

--> Solution 3
nA=3;
nB=3;

--> Solution 4
nA=1;
nB=3;

--> Solution 5
nA=3;
nB=1;

--> Solution 6
nA=3;
nB=0;

--> Solution 7
nA=3;
nB=2;

[Formula generation: 0.000184s; conversion to CNF: 0.000281s; total: 0.000465s]
[Solving time: 0.000292s]
[Formula size: 62 variables, 162 clauses]
```

*Ukoliko izvršimo sledeći URSA kôd:*

```
nMax = ite(nA>nB, nA, nB);
print nMax;
```

*dobićemo naredni izlaz, koji prikazuje internu reprezentaciju promenljive* `nMax` *(tj. svih njenih 8 bitova):*

```
 --> abstract number - dependent
 --> value:
1.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &

2.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &

3.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &

4.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &

5.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &

6.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &

7.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &

8.  ( ( ~( ( v1  &  ( ~ v9 ) )  |  ( ( ( v2  &  ( ~ v10 ) )  |  ( ( ( v3  &  ( ~ v11 ) )  |  ( ( ( v4  &  ( ~ v12 ) )  |  ( ( ( v5  &  ( ~ v13 ) )  |  ( ( ( v6  &
```

*Default vrednost za broj bitova za brojevne promenljive je 8. Ako je sistem URSA pozvan na sledeći način (tj. za dužinu 1):*

```
./ursa < maksimum.urs -l1
```

*dobili bismo sledeći izlaz:*

```
 --> abstract number - dependent
 --> value:
1.  ( ( ~( v1  &  ( ~ v2 ) ))  |  v1 )  &  ( ( v1  &  ( ~ v2 ) )  |  v2 )
```

## 8.7 Generisanje permutacija

**Primer 8.5.** *Razmotrimo problem generisanja svih permutacija brojeva od* $0$ *do* $n-1$.

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
nDim = 5;
bSolution = true;

/* domain */
for (nI=0; nI <= nDim-1; nI++) {
  bSolution &&= (nA[nI] < nDim);
}

/* all different */
for (nI=0; nI <= nDim-2; nI++) {
  for (nJ=nI+1; nJ <= nDim-1; nJ++) {
    bSolution &&= (nA[nI] != nA[nJ]);
  }
}

assert_all(bSolution);
```

*Navedeni kôd govori kada niz* `nA` *zaista sadrži permutaciju brojeva od* $0$ *do* $n-1$, *ali možemo da ga upotrebimo i za generisanje svih permutacija. Sistem URSA poziva se na sledeći način:*

```
\$ ./ursa < permutacije.urs
```

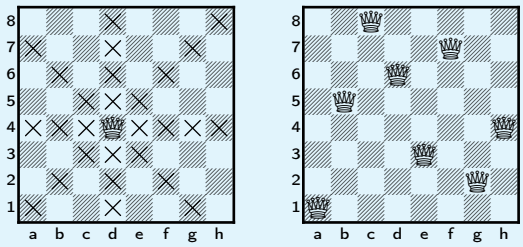
*i daje se sledeći izlaz:*

```
--> Solution 1
nA[0]=1;
nA[1]=3;
nA[2]=0;
nA[3]=4;
nA[4]=2;

--> Solution 2
nA[0]=0;
nA[1]=3;
nA[2]=1;
nA[3]=4;
nA[4]=2;

...

--> Solution 120
nA[0]=1;
nA[1]=4;
nA[2]=2;
nA[3]=0;
nA[4]=3;

[Formula generation: 0.000601s; conversion to CNF: 0.000381s; total: 0.000982s]
[Solving time: 0.003718s]
[Formula size: 125 variables, 380 clauses]
```

## 8.8 Problem $n$ dama

**Primer 8.6.** *Razmotrimo dobro poznati problem $n$ dama: na tabli $n \times n$ potrebno je rasporediti $n$ dama tako da se nikoje dve ne napadaju.*

*Na narednoj slici prikazano je kretanje dame u šahu (levo) i jedno rešenje problema (desno):*




*Na CSP jeziku MiniZinc, ovaj problem može da se modeluje na sledeći način:*

```
int: n;
array [1..n] of var 1..n: q; % queen in column i is in row q[i]

include "alldifferent.mzn";

constraint alldifferent(q);                        % distinct rows
constraint alldifferent([ q[i] + i | i in 1..n]); % distinct diagonals
constraint alldifferent([ q[i] - i | i in 1..n]); % upwards+downwards

% search
solve :: int_search(q, first_fail, indomain_min)
      satisfy;
output [ if fix(q[j]) == i then "Q" else "." endif ++
         if j == n then "\n" else "" endif | i,j in 1..n]
```

*Na jeziku rešavača choco-solver, ovaj problem može da se modeluje na sledeći način:*

```
int n = 8;
Model model = new Model(n + "-queens problem");
IntVar[] vars = new IntVar[n];
for(int q = 0; q < n; q++){
    vars[q] = model.intVar("Q_"+q, 1, n);
}
for(int i  = 0; i < n-1; i++){
    for(int j = i + 1; j < n; j++){
        model.arithm(vars[i], "!=",vars[j]).post();
        model.arithm(vars[i], "!=", vars[j], "-", j - i).post();
        model.arithm(vars[i], "!=", vars[j], "+", j - i).post();
    }
}
Solution solution = model.getSolver().findSolution();
if(solution != null){
    System.out.println(solution.toString());
}
```

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
nDim=8;
bDomain = true;
bNoCapture = true;

for(ni=0; ni<nDim; ni++) {
  bDomain &&= (n[ni]<nDim);
  for(nj=ni+1; nj<nDim; nj++)
    bNoCapture &&= n[ni]!=n[nj] && ni+n[nj]!=nj+n[ni] && ni+n[ni]!=nj+n[nj];
}

assert_all(bDomain && bNoCapture);
```

*Primetimo da se navedeni kôd skoro bez ikakvih izmena može koristiti u C programu za proveru da li je rešenje problema zapisano u nizu* `n` *ispravno.*

*Sistem URSA poziva se na sledeći način:*

```
\$ ./ursa < queens.urs
```

*i daje se sledeći izlaz:*

```
--> Solution 1
n[0]=6;
n[1]=1;
n[2]=5;
n[3]=2;
n[4]=0;
n[5]=3;
n[6]=7;
n[7]=4;

...

--> Solution 92
n[0]=6;
n[1]=3;
n[2]=1;
n[3]=4;
n[4]=7;
n[5]=0;
n[6]=2;
n[7]=5;

[Formula generation: 0.005716s; conversion to CNF: 0.003257s; total: 0.008973s]
[Solving time: 0.026775s]
[Formula size: 1345 variables, 5440 clauses]
```

Interesantno je razmotriti broj promenljivih i broj klauza (kao i njihove količnike) za različite vrednosti parametra $n$.

Ukoliko se URSA pozove sa opcijom `-q` onda ne ispisuje rešenja, pa je izlaz kraći.

## 8.9 Problem SEND MORE MONEY

**Primer 8.7.** *Problem SEND MORE MONEY sastoji se u pronalaženju različitih cifara koje odgovaraju slovima D, E, M, N, O, R, S, Y pri čemu S i M nisu jednake nuli i jednakost*

*SEND + MORE = MONEY*

*mora da važi. Rešenje je jedinstveno: 9567 + 1085 = 10652.*

*Na CSP jeziku MiniZinc, ovaj problem može da se modeluje na sledeći način:*

```
include "alldifferent.mzn";

var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;

constraint           1000 * S + 100 * E + 10 * N + D
                   + 1000 * M + 100 * O + 10 * R + E
       = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;
```

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
bAllDifferent   = (nS!=nE) && (nS!=nN) && (nS!=nD) && (nS!=nM) && (nS!=nO) && (nS!=nR) && (nS!=nY);
bAllDifferent &&= (nE!=nN) && (nE!=nD) && (nE!=nM) && (nE!=nO) && (nE!=nR) && (nE!=nY);
bAllDifferent &&= (nN!=nD) && (nN!=nM) && (nN!=nO) && (nN!=nR) && (nN!=nY);
bAllDifferent &&= (nD!=nM) && (nD!=nO) && (nD!=nR) && (nD!=nY);
bAllDifferent &&= (nM!=nO) && (nM!=nR) && (nM!=nY);
bAllDifferent &&= (nO!=nR) && (nO!=nY);
bAllDifferent &&= (nR!=nY);

bDomain =  (nS<10) && (nE<10) && (nN<10) && (nD<10) && (nM<10) && (nO<10) && (nR<10) && (nY<10);

bNoLeadingZeros = (nM!=0) && (nS!=0);

bSum = (nS*1000+nE*100+nN*10+nD+nM*1000+nO*100+nR*10+nE==nM*10000+nO*1000+nN*100+nE*10+nY);

assert_all(bAllDifferent && bDomain && bNoLeadingZeros && bSum);
```

*Sistem URSA poziva se za dužinu binarnog zapisa* $> 17$*:*

```
\$ ./ursa < send_more_money.urs -l18
```

*i daje se sledeći izlaz:*

```
--> Solution 1
nD=7;
nE=5;
nM=1;
nN=6;
nO=0;
nR=8;
nS=9;
nY=2;
```

## 8.10 Vežbanje 1: IMO 1960/1

**Primer 8.8.** *IMO 1960/1: Find all the three-digit numbers for which one obtains, when dividing the number by 11, the sum of the squares of the digits of the initial number.*

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
assert_all(100*na+10*nb+nc==11*(na*na+nb*nb+nc*nc) & 0<na & na<10 & nb<10 & nc<10);
```

*Sistem URSA mora se pozvati za dužinu binarnog zapisa* $\geq 10$ *(da bi mogli da budu obrađivani brojevi koji su trocifreni u binarnom zapisu):*

```
\$ ./ursa < send_more_money.urs -l18
```

*Sva rešenja su:*

```
--> Solution 1
na=5;
nb=5;
nc=0;

--> Solution 2
na=8;
nb=0;
nc=3;

[Formula generation: 0.000441s; conversion to CNF: 0.00931s; total: 0.009751s]
[Solving time: 0.004778s]
[Formula size: 1601 variables, 5627 clauses]
```

## 8.11 Vežbanje 2: IMO 1981/3

**Primer 8.9.** *IMO 1981/3: Determine the maximum value of* $m^2+n^2$ *where* $m$ *and* $n$ *are integers satisfying* $m, n \in \{1, 2, ..., 1981\}$ *and* $(n^2 - mn - m^2)^2 = 1$

*U sistemu URSA, svi parovi* $(m, n)$ *mogu se opisati na sledeći način (primetimo da je uslov zamenjen uslovima* $n^2 - mn - m^2 = 1$ *i* $n^2 - mn - m^2 = -1$*, a da su oni onda preformulisani u* $n^2 = 1 + mn + m^2$ *i* $n^2 + 1 = mn + m^2$*, kako bi se izbeglo prekoračenje koje je posledica oduzimanja):*

```
assert_all(nn<=1981 && nm<=1981 && (nn*nn==1+nm*nn+nm*nm || nn*nn+1==nm*nn+nm*nm));
```

*Dovoljno je pozvati sistem URSA za dužinu binarnog zapisa* 32*:*

```
\$ ./ursa < IMO1981_3.urs -l32
```

*Svi parovi su:*

```
--> Solution 1
nm=1; nn=2;
--> Solution 2
nm=1; nn=0;
--> Solution 3
nm=1; nn=1;
--> Solution 4
nm=233; nn=377;
--> Solution 5
nm=89; nn=144;
--> Solution 6
nm=377; nn=610;
--> Solution 7
nm=13; nn=21;
--> Solution 8
nm=5; nn=8;
--> Solution 9
nm=21; nn=34;
--> Solution 10
nm=987; nn=1597;
--> Solution 11
nm=3; nn=5;
--> Solution 12
nm=55; nn=89;
--> Solution 13
nm=0; nn=1;
--> Solution 14
nm=2; nn=3;
--> Solution 15
nm=34; nn=55;
--> Solution 16
nm=8; nn=13;
--> Solution 17
nm=144; nn=233;
--> Solution 18
nm=610; nn=987;

[Formula generation: 0.000157s; conversion to CNF: 0.029322s; total: 0.029479s]
[Solving time: 0.985276s]
[Formula size: 9867 variables, 34685 clauses]
```

*pa je tražena vrednost* $987^2 + 1597^2$ *(primetimo da svaki dobijeni par predstavlja neki par susednih Fibonačijevih brojeva).*

## 8.12 Vežbanje 3: IMO 1977/5

**Primer 8.10.** *Let $a$ and $b$ be natural numbers and let $q$ and $r$ be the quotient and remainder respectively when $a^2 + b^2$ is divided by $a + b$. Determine the numbers $a$ and $b$ if $q^2 + r = 1977$.*

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
b = nA<=256 && nB<=256;
assert_all(b && nA*nA+nB*nB==(nA+nB)*nQ+nR && nR<nA+nB && nQ*nQ+nR==1977);
```

*Sistem URSA može se pozvati za dužinu binarnog zapisa 32:*

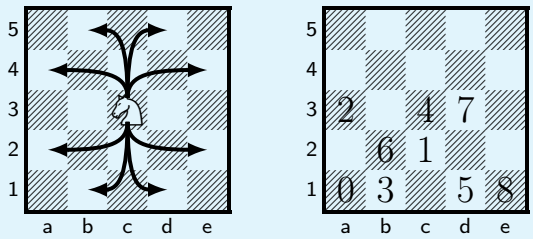
```
\$ ./ursa < IMO1977_5.urs -l32
```

*Izlaz je:*

```
--> Solution 1
nA=50;
nB=7;
nQ=44;
nR=41;

--> Solution 2
nA=50;
nB=37;
nQ=44;
nR=41;

--> Solution 3
nA=37;
nB=50;
nQ=44;
nR=41;

--> Solution 4
nA=7;
nB=50;
nQ=44;
nR=41;

[Formula generation: 0.000416s; conversion to CNF: 0.048192s; total: 0.048608s]
[Solving time: 8.6406s]
[Formula size: 13337 variables, 46015 clauses]
```

## 8.13 Problem skakačev put

**Primer 8.11.** *Na šahovskoj tabli dimenzija $n \times n$ potrebno je pronaći put za skakača iz donjeg levog ugla do gornjeg desnog ugla takav da skakač poseti i sva druga polja tačno po jednom (postoje i druge varijante ovog problema). Na narednoj slici, prikazano je kretanje skakača i jedna putanja od osam poteza koja se ne može nastaviti.*




*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
procedure Legal(nDim, nX1, nY1, nX2, nY2, bLegal) {
  bLegal = ((nX2 == nX1+1 && nY2 == nY1+2) ||
            (nX2 == nX1+2 && nY2 == nY1+1) ||
            (nX2 == nX1+1 && nY1 == nY2+2) ||
            (nX2 == nX1+2 && nY1 == nY2+1) ||
            (nX1 == nX2+1 && nY2 == nY1+2) ||
            (nX1 == nX2+2 && nY2 == nY1+1) ||
            (nX1 == nX2+1 && nY1 == nY2+2) ||
            (nX1 == nX2+2 && nY1 == nY2+1));
  bLegal &&= (nX1<nDim && nX2<nDim && nY1<nDim && nY2<nDim);
}

nDim = 5;
nX[0] = 0;
nY[0] = 0;
nX[nDim*nDim-1] = nDim-1;
nY[nDim*nDim-1] = nDim-1;
bSolution = true;

for (nI=0; nI <= nDim*nDim-2; nI++) {
  for (nJ=nI+1; nJ <= nDim*nDim-1; nJ++) {
    bSolution &&= !(nX[nI] == nX[nJ] && nY[nI] == nY[nJ]);
  }
}
for (nI=0; nI <= nDim * nDim-2; nI++) {
  nx1 = nX[nI];   ny1 = nY[nI];
  nx2 = nX[nI+1]; ny2 = nY[nI+1];
  call Legal(nDim, nx1, ny1, nx2, ny2, b);
  bSolution &&= b;
}
assert(bSolution);
```

*Jedno rešenje je sledeće:*

```
nX[10]=4;
nX[11]=3;
nX[12]=1;
nX[13]=0;
nX[14]=2;
nX[15]=4;
nX[16]=3;
nX[17]=1;
nX[18]=0;
nX[19]=1;
nX[1]=2;
nX[20]=3;
nX[21]=1;
nX[22]=0;
nX[23]=2;
nX[2]=4;
nX[3]=3;
nX[4]=1;
nX[5]=0;
nX[6]=2;
nX[7]=4;
nX[8]=2;
nX[9]=3;
nY[10]=2;
nY[11]=0;
nY[12]=1;
nY[13]=3;
nY[14]=4;
nY[15]=3;
nY[16]=1;
nY[17]=0;
nY[18]=2;
nY[19]=4;
nY[1]=1;
nY[20]=3;
nY[21]=2;
nY[22]=4;
nY[23]=3;
nY[2]=0;
nY[3]=2;
nY[4]=3;
nY[5]=1;
nY[6]=0;
nY[7]=1;
nY[8]=2;
nY[9]=4;

[Formula generation: 0.040536s; conversion to CNF: 0.021388s; total: 0.061924s]
[Solving time: 0.717544s]
[Formula size: 7377 variables, 35938 clauses]
```

*Navedenom rešenju odgovara sledeći obilazak:*

| | a | b | c | d | e |
|---|---|---|---|---|---|
| 5 | 22 | 19 | 14 | 9 | 24 |
| 4 | 13 | 4 | 23 | 20 | 15 |
| 3 | 18 | 21 | 8 | 3 | 10 |
| 2 | 5 | 12 | 1 | 16 | 7 |
| 1 | 0 | 17 | 6 | 11 | 2 |

## 8.14 Mali primerak verifikacije softvera 1

**Primer 8.12.** *Verifikacija softvera ima za zadatak da utvrdi da neki kôd daje rezultat u skladu sa specifikacijom. Jedna varijanta ovog zadatka je da se utvrdi da neki kôd daje isti rezultat kao i neki drugi kôd (tj. smatra se da je specifikacija zadata tim drugim kodom.*

*U ovom primeru biće pokazano da jedan fragment kôda ekvivalentan drugom:*

```
nA = nBitoff - ((nBitoff >> 3) << 3);

nB = nBitoff & 7;

assert (nA != nB);
```

*Uslov u naredbi* `assert` *govori da su vrednosti* `nA` *i* `nB` *različite, tj. dve verzije koda nisu ekvivalentne. Pitanje je da li postoji vrednost* `nv` *(dužine koju URSA koristi za reprezentaciju – default je dužina 8) za koju je to tako. URSA daje naredni izlaz:*

```
No solutions found
[Formula generation: 0.000492s; conversion to CNF: 0.000767s; total: 0.001259s]
[Solving time: 0.000109s]
[Formula size: 20 variables, 38 clauses]
```

*Što znači da su dve verzije koda ekvivalentne.*

## 8.15 Mali primerak verifikacije softvera 2

**Primer 8.13.** *U ovom primeru biće pokazano da je kôd za proveravanje da li broj u svom binarnom zapisu ima tačno jednu jedinicu ispravan (on će biti proveren u odnosu na trivijalnu, ali neefikasnu verziju):*

```
b1 = (nv!=0) && ((nv & (nv-1))==0);

nLen=8;
bOne = false;
bMoreThanOne = false;
for(ni=0; ni<nLen; ni++)  {
  bMoreThanOne ||= bOne && ((nv & 1)!=0);
  bOne ||= ((nv & 1)!=0);
  nv >>= 1;
}
b2 = bOne && !bMoreThanOne;

assert(b1^^b2);
```

*Uslov u naredbi* `assert` *govori da su vrednosti* `b1` *i* `b2` *različite, tj. dve verzije koda nisu ekvivalentne. Pitanje je da li postoji vrednost* `nv` *(dužine koju URSA koristi za reprezentaciju – default je dužina 8) za koju je to tako. URSA daje naredni izlaz:*

```
No solutions found
[Formula generation: 0.001468s; conversion to CNF: 0.000991s; total: 0.002459s]
[Solving time: 0.001713s]
[Formula size: 48 variables, 183 clauses]
```

*Što znači da su dve verzije koda ekvivalentne.*

## 8.16 Množenje polinoma

**Primer 8.14.** *Naredni kôd daje algoritam za množenje dva (konkretna) polinoma (stepena najviše 3):*

```
 nN = 4;
/* p1(x) = x+1 */
nA[0] = 1;
nA[1] = 1;
nA[2] = 0;
nA[3] = 0;

/* p2(x) = (x+2)*(x+3) = x^2 + 5x + 6 */
nB[0] = 6;
nB[1] = 5;
nB[2] = 1;
nB[3] = 0;

for (nI=0; nI<nN; nI++) {
  nP[nI] = 0;
}
for (nI=0; nI<nN; nI++) {
  for (nJ=0; nJ<nN; nJ++) {
    nP[nI+nJ] += nA[nI]*nB[nJ];
  }
}

print nP[0];
print nP[1];
print nP[2];
print nP[3];
```

*Izvršavanjem ovog koda dobija se rezultat:*

```
--> ground number
--> value: 6

--> ground number
--> value: 11

--> ground number
--> value: 6

--> ground number
--> value: 1
```

*Dakle, proizvod polinoma* $x+1$ *i* $x^2+5x+6$ *dobija se polinom* $x^3+6x^2+11x+6$. *No, ključni deo ovog koda može se upotrebiti i za deljenje polinoma: ako su elementi niza* `nA` *nepoznati, onda mogu biti pronađeni koristeći sledeći kôd i to će biti rezultat* deljenja *polinoma* $x^3+6x^2+11x+6$ *polinomom* $x^2+5x+6$ *(uslov sadržan u promenljivoj b blokira rešenja koja se dobijaju zbog modularne aritmetike):*

```
nN = 4;
/* p2(x) = (x+2)*(x+3) = x^2 + 5x + 6 */
nB[0] = 6;
nB[1] = 5;
nB[2] = 1;
nB[3] = 0;

for (nI=0; nI<nN; nI++) {
  nP[nI] = 0;
}
for (nI=0; nI<nN; nI++) {
  for (nJ=0; nJ<nN; nJ++) {
    nP[nI+nJ] += nA[nI]*nB[nJ];
  }
}
b = true;
for (nI=0; nI<nN; nI++) {
  b &&= (nA[nI]<16);
  b &&= (nB[nI]<16);
}
assert_all (nP[0] == 6 && nP[1] == 11 && nP[2] == 6 && nP[3] == 1 && b);
```

*URSA daje izlaz:*

```
--> Solution 1
nA[0]=1;
nA[1]=1;
nA[2]=0;
nA[3]=0;

[Formula generation: 0.00395s; conversion to CNF: 0.00242s; total: 0.00637s]
[Solving time: 0.000448s]
[Formula size: 434 variables, 1339 clauses]
```

*tj. količnik polinoma* $x^3 + 6x^2 + 11x + 6$ *i* $x^2 + 5x + 6$ *je polinom* $x + 1$. *Primetimo da smo na opisani način realizovali deljenje polinoma, samo na osnovu definicije množenja. Možemo da odemo i korak dalje i tražimo sve polinome kojima je deljiv polinom* $x^3+6x^2+11x+6$ *(ili neki drugi polinom, naravno). Navedena specifikacija traži sve faktore oblika* $ax + b$*:*

```
nN = 4;
for (nI=0; nI<nN; nI++) {
  nP[nI] = 0;
}
for (nI=0; nI<nN; nI++) {
  for (nJ=0; nJ<nN; nJ++) {
    nP[nI+nJ] += nA[nI]*nB[nJ];
  }
}

b = true;
for (nI=0; nI<nN; nI++) {
  b &&= (nA[nI]<16);
  b &&= (nB[nI]<16);
}
assert_all (nP[0] == 6 && nP[1] == 11 && nP[2] == 6 && nP[3] == 1 &&
            nA[1] != 0 &&  nA[2] == 0 && nA[3] == 0 && b);
```

*URSA daje sledeći izlaz:*

```
--> Solution 1
nA[0]=2;
nA[1]=1;
nA[2]=0;
nA[3]=0;
nB[0]=3;
nB[1]=4;
nB[2]=1;
nB[3]=0;

--> Solution 2
nA[0]=3;
nA[1]=1;
nA[2]=0;
nA[3]=0;
nB[0]=2;
nB[1]=3;
nB[2]=1;
nB[3]=0;

--> Solution 3
nA[0]=1;
nA[1]=1;
nA[2]=0;
nA[3]=0;
nB[0]=6;
nB[1]=5;
nB[2]=1;
nB[3]=0;
```

*Dakle, polinom* $x^3 + 6x^2 + 11x + 6$ *ima faktore* $x + 2$, $x + 3$, $x + 1$ *i to su jedini linearni faktori (oni čine i faktorizaciju datog polinoma).*

## 8.17 Vežbanje 4: IMO 1963/6

**Primer 8.15.** *IMO 1963: Five students A, B, C, D, and E have taken part in a certain competition. Before the competition, two persons X and Y tried to guess the rankings. X thought that the ranking would be A, B, C, D, E; and Y thought that the ranking would be D, A, E, C, B. At the end, it was revealed that X didn't guess correctly any rankings of the participants, and moreover, didn't guess any of the orderings of pairs of consecutive participants. On the other hand, Y guessed the correct rankings of two participants and the correct ordering of two pairs of consecutive partici- pants. Determine the rankings of the competition.*

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
nX[1]=1; nX[2]=2; nX[3]=3; nX[4]=4; nX[5]=5;
nY[1]=4; nY[2]=1; nY[3]=5; nY[4]=3; nY[5]=2;

bDomain=true;
bAllDifferent=true;
for(ni=1;ni<=5;ni++) {
  bDomain &&= (0<nActual[ni] && nActual[ni]<=5);
  for(nj=1;nj<ni;nj++)
    bAllDifferent &&= (nActual[ni] != nActual[nj]);
}

nX1=0; nX2=0; nY1=0; nY2=0;
for(ni=1;ni<=5;ni++)  {
  nX1 += ite(nX[ni]==nActual[ni],1,0);
  nY1 += ite(nY[ni]==nActual[ni],1,0);
}

for(ni=1;ni<=4;ni++)
  for(nj=1;nj<=4;nj++) {
     nX2 += ite(nX[ni]==nActual[nj] && nX[ni+1]==nActual[nj+1],1,0);
     nY2 += ite(nY[ni]==nActual[nj] && nY[ni+1]==nActual[nj+1],1,0);
  }

assert_all(bDomain && bAllDifferent && nX1==0 && nX2==0 && nY1==2 && nY2==2);
```

*Sistem URSA može se pozvati za dužinu binarnog zapisa 3:*

```
\$ ./ursa < IMO1963_6.urs -l3
```

*Izlaz je:*

```
 --> Solution 1
nActual[1]=5;
nActual[2]=4;
nActual[3]=1;
nActual[4]=3;
nActual[5]=2;

[Formula generation: 0.00489s; conversion to CNF: 0.000886s; total: 0.005776s]
[Solving time: 0.001838s]
[Formula size: 279 variables, 1484 clauses]
```

## 8.18 Vežbanje 5: šahovska KK završnica

**Primer 8.16.** *U šahovskoj završnici beli može da ima samo kralja (King) i crni može da ima samo kralja. U toj završnici siguran je remi. Koliko ima legalnih KK pozicija u kojima je na potezu beli? Koliko ima legalnih KRK pozicija u kojima je na potezu crni? Napisati kôd koji odgovara na navedena pitanja i za tablu dimenzija $n \times n$.*

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
nDim=8;
bDomain = nWKx<8 && nWKy<8 && nBKx<8 && nBKy<8;
bKingsNotAdjacent = (nWKx>nBKx+1 || nBKx>nWKx+1 || nWKy>nBKy+1 || nBKy>nWKy+1);
assert_all(bDomain && bKingsNotAdjacent);
```

*i sistem URSA daje izlaz (ovo je broj pozicija i za belog i za crnog na potezu):*

```
--> Solution 3612
nBKx=2;
nBKy=5;
nWKx=7;
nWKy=7;

[Formula generation: 0.000245s; conversion to CNF: 0.000304s; total: 0.000549s]
[Solving time: 0.082163s]
[Formula size: 80 variables, 229 clauses]
```

## 8.19 Vežbanje 6: šahovska KRK završnica

**Primer 8.17.** *KRK je šahovska završnica u kojoj beli ima kralja (King) i topa (Rook), a crni samo kralja. U toj završnici beli dobija. Koliko ima legalnih KRK pozicija u kojima je na potezu beli? Koliko ima legalnih KRK pozicija u kojima je na potezu crni? Napisati kôd koji odgovara na navedena pitanja i za tablu dimenzija $n \times n$.*

*U sistemu URSA, ovaj problem može da se modeluje na sledeći način:*

```
nDim=8;
bDomain = nWKx<nDim && nWKy<nDim && nBKx<nDim && nBKy<nDim && nWRx<nDim && nWRy<nDim;
bDifferent = !(nWKx == nWRx && nWKy == nWRy) && !(nBKx == nWRx && nBKy == nWRy);
bKingsNotAdjacent = (nWKx>nBKx+1 || nBKx>nWKx+1 || nWKy>nBKy+1 || nBKy>nWKy+1);
bBlackKingAttacked = (nBKx == nWRx || nBKy == nWRy);
bWhiteKingBetween = (nBKx == nWKx && nWKx == nWRx &&
                        ((nBKy < nWKy && nWKy < nWRy) || (nBKy > nWKy && nWKy > nWRy)))
                    ||
                    (nBKy == nWKy && nWKy == nWRy &&
                        ((nBKx < nWKx && nWKx < nWRx) || (nBKx > nWKx && nWKx > nWRx)));
assert_all(bDomain && bDifferent && bKingsNotAdjacent  &&
           (!bBlackKingAttacked || bWhiteKingBetween));
```

*i sistem URSA daje izlaz (ovo je broj pozicija u kojima je beli na potezu):*

```
...

--> Solution 175168
nBKx=7;
nBKy=6;
nWKx=7;
nWKy=3;
nWRx=7;
nWRy=2;

[Formula generation: 0.001551s; conversion to CNF: 0.001638s; total: 0.003189s]
[Solving time: 4.64389s]
[Formula size: 197 variables, 623 clauses]
```

## 8.20 Problem generisanje pseudoslučajnih brojeva

**Primer 8.18.** *Pseudoslučajni generator (PRG) je deterministička procedura koja generiše niz pseudoslučajnih brojeva takav da nijedan statistički test ne može da napravi razliku između izlaza generatora i ravnomerne raspodele. Pseudoslučajni generatori imaju brojne primene u računarstvu, na primer u kriptografiji.*

*Linearni kongruentni generator (LCG) je algoritam koji daje niz pseudoslučajnih brojeva izračunatih linearnom modularnom vezom. Metoda predstavlja jedan od najstarijih i najpoznatijih algoritama za generisanje pseudoslučajnih brojeva. Lako i brzo se implementira, posebno na računarskom hardveru koji može da obezbedi modularnu aritmetiku. Generator je definisan rekurentnom vezom oblika:*

$X_{n+1} = (aX_n + c) \mod m$

*gde je $X$ niz pseudoslučajnih vrednosti a sledeće celobrojne konstante koje specifikuju generator:*

- *$m$ je „moduo“, $0 < m$;*
- *$a$ je „množilac“, $0 < a < m$;*
- *$c$ je „inkrement“, $0 \leq c < m$;*
- *$X_0$ je „seed“, „seme“ ili „početna vrednost“, $0 \leq X_0 < m$.*

*Ako je $c = 0$, generator se često naziva multiplikativni kongruentni generator (MCG). Kompilator* `gcc`*, na primer, koristi vrednosti $m = 2^{31}$, $a = 1103515245$, $c = 12345$ i izlaz čini niz od 31 bita.*

*Naredni URSA kôd, koji implementira jedan LCG:*

```
for(ni=1;ni<=10;ni++) {
  print nx;
  nx=nx*1664525+1013904223;
```

*daje sledeći niz brojeva:*

```
--> value: 1
--> value: 1015568748
--> value: 1586005467
--> value: 2165703038
--> value: 3027450565
--> value: 217083232
--> value: 1587069247
--> value: 3327581586
--> value: 2388811721
--> value: 70837908
```

*Očigledno, jednostavno je od zadate seed vrednosti dobiti, na primer, stoti broj u nizu. No, postavlja se pitanje kako uraditi obratno – kako odabrati seed vrednost da bi stoti broj u nizu bio neki zadati broj? Postoje algebarske tehnike za rešavanje tog problema, ali on se jednostavno može rešiti i korišćenjem sistema URSA. Na primer, naredni kod:*

```
for(ni=1; ni<=100; ni++) {
  nx=nx*1664525+1013904223;
```

*daje seed vrednosti za koju je stoti broj u nizu jednak 2021:*

```
--> Solution 1
nseed=1052564417;

[Formula generation: 0.07963s; conversion to CNF: 0.165807s; total: 0.245437s]
[Solving time: 0.263251s]
[Formula size: 108132 variables, 366532 clauses]
```

## 8.21 Analiza blok algoritama za kriptovanje

**Primer 8.19.** Tiny Encryption Algorithm *(TEA) je blokovski algoritam za kriptovanje, poznat zbog svoje jedinostavnosti i izuzetno kratke implementacije (koja se sastoji od svega nekoliko linija). Algoritam su dizajnirali Wheeler i Needham (Cambridge Computer Laboratory) i algoritam je prvi put predstavljen 1994. godine (*https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm*).*

*Algoritam kriptovanja može se u jeziku C implementirati i pozvati na sledeći način:*

```
#include <stdio.h>
#include <stdint.h>

void encrypt (uint32_t v[2], const uint32_t k[4]) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;           /* set up */
    uint32_t delta=0x9E3779B9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<32; i++) {                         /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}

void decrypt (uint32_t v[2], const uint32_t k[4]) {
    uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i;  /* set up */
    uint32_t delta=0x9E3779B9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<32; i++) {                         /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}

int main()
{
  uint32_t data[2] = { 20, 22 };
  uint32_t key[4] = { 1, 2, 3, 4 };

  encrypt (data, key);
  printf("encrypted: %d %d \n", data[0], data[1]);
}
```

*Navedeni program, za zadati ulaz i zadati ključ, daje sledeći izlaz:*

```
encrypted: 2032961448 1747770815
```

*Algoritam kriptovanja (tj. funkcija* `encrypt`*) na jeziku URSA implementira se u skoro neizmenjenom obliku. Naredni kôd*

```
procedure Encrypt (nv0, nv1, nK0, nK1, nK2, nK3) {
    nDelta = 0x9E3779B9;
    nSum = 0;
    for (nI=0; nI<32; nI++) {
        nSum += nDelta;
        nv0 += (((nv1<<4) + nK0) ^ (nv1 + nSum)) ^ ((nv1>>5) + nK1);
        nv1 += (((nv0<<4) + nK2) ^ (nv0 + nSum)) ^ ((nv0>>5) + nK3);
    }
}

nData0 = 20;
nData1 = 22;
nkey0 = 1; nkey1 = 2; nkey2 = 3; nkey3 = 4;
call Encrypt(nData0, nData1, nkey0, nkey1, nkey2, nkey3);

print nData0;
print nData1;
```

*daje sledeći izlaz:*

```
--> ground number
--> value: 2032961448

--> ground number
--> value: 1747770815
```

*To je proces kriptovanja. Ali URSA kôd se može iskoristiti i za proces dekriptovanja (mada bi, naravno, mogla da se neposredno iskoristi i funkcija* `decrypt`*).*

```
procedure Encrypt (nv0, nv1, nK0, nK1, nK2, nK3) {
    nDelta = 0x9E3779B9;
    nSum = 0;
    for (nI=0; nI<32; nI++) {
        nSum += nDelta;
        nv0 += (((nv1<<4) + nK0) ^ (nv1 + nSum)) ^ ((nv1>>5) + nK1);
        nv1 += (((nv0<<4) + nK2) ^ (nv0 + nSum)) ^ ((nv0>>5) + nK3);
    }
}

nData0 = nInput0;
nData1 = nInput1;
nkey0 = 1; nkey1 = 2; nkey2 = 3; nkey3 = 4;
call Encrypt(nData0, nData1, nkey0, nkey1, nkey2, nkey3);

assert_all (nData0 == 2032961448 && nData1 == 1747770815);
```

*Za izlaz 2032961448, 1747770815, ulaz je 20, 22:*

```
--> Solution 1
nInput0=20;
nInput1=22;

[Formula generation: 0.015468s; conversion to CNF: 0.043182s; total: 0.05865s]
[Solving time: 0.043976s]
[Formula size: 22340 variables, 98132 clauses]
```

*Uz neznatne izmene URSA kôd može da se iskoristi i za otkrivanje ključa, ukoliko je poznat par ulaz-izlaz:*

```
procedure Encrypt (nv0, nv1, nK0, nK1, nK2, nK3) {
    nDelta = 0x9E3779B9;
    nSum = 0;
    for (nI=0; nI<32; nI++) {
        nSum += nDelta;
        nv0 += (((nv1<<4) + nK0) ^ (nv1 + nSum)) ^ ((nv1>>5) + nK1);
        nv1 += (((nv0<<4) + nK2) ^ (nv0 + nSum)) ^ ((nv0>>5) + nK3);
    }
}

nData0 = nInput0;
nData1 = nInput1;
nkey0 = nK1; nkey1 = nK2; nkey2 = nK3; nkey3 = nK3;
call Encrypt(nData0, nData1, nkey0, nkey1, nkey2, nkey3);

assert_all (nData0 == 2032961448 && nData1 == 1747770815 &&
            nInput0==20 && nInput1==22);
```

*No, ovako generisana instanca veoma je teška i zahtevala bi možda i dane računarskog vremena za rešavanje. To nije neočekivano, jer je algoritam TEA dizajniran tako da je teško razbiti ga. No, ono što se može efikasnije uraditi je otkrivanje ključa ukoliko su poznati neki delovi i/ili ukoliko je poznato neko ograničenje koje važi za njegove delove. Na primer, ukoliko znamo prva tri dela ključa (1, 2 i 3) i znamo da je četvrti deo ključa manji od 100, onda to možemo specifikovati na sledeći način:*

```
procedure Encrypt (nv0, nv1, nK0, nK1, nK2, nK3) {
    nDelta = 0x9E3779B9;
    nSum = 0;
    for (nI=0; nI<32; nI++) {
        nSum += nDelta;
        nv0 += (((nv1<<4) + nK0) ^ (nv1 + nSum)) ^ ((nv1>>5) + nK1);
        nv1 += (((nv0<<4) + nK2) ^ (nv0 + nSum)) ^ ((nv0>>5) + nK3);
    }
}

nData0 = nInput0;
nData1 = nInput1;
nkey0 = 1; nkey1 = 2; nkey2 = 3; nkey3 = nK3;
call Encrypt(nData0, nData1, nkey0, nkey1, nkey2, nkey3);

assert_all (nData0 == 2032961448 && nData1 == 1747770815 &&
            nInput0==20 && nInput1==22 && nK3<100);
```

*Navedeni kôd daje sledeći izlaz:*

```
--> Solution 1
nInput0=20;
nInput1=22;
nK3=4;

[Formula generation: 0.013043s; conversion to CNF: 0.044961s; total: 0.058004s]
[Solving time: 0.586713s]
[Formula size: 26311 variables, 102649 clauses]
```

*Slične tehnike mogu se koristiti i za analizu drugih kriptografskih algoritama uključujući kriptografske heš funkcije (pa i druge one-way funkcije).*

# Bibliografija

[1] A. Biere, M. Heule, H. Van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009.

[2] Predrag Janičić. URSA: A System for Uniform Reduction to SAT. *Logical Methods in Computer Science*, 8(3):30, September 2012.