

# FRI in izgubljeni urniki

## Seminarska naloga 1 pri predmetu Funkcijsko programiranje

Oktober, 2019

### Uvod – pogrešani urničarji in urniki

Študentje fakultete FRI ne vejo, kateri cikel naj izbrati, saj osebni urniki “ne delajo”. Panično si dopisujejo z asistenti. V tem času pa urničarjev nikjer ni, pripadajoči urniki pa so izginili neznano kam.

V vsesplošni zmedbi so asistenti FRI problem priprave urnikov predali študentom programa BMAG-RI. Le-ti so se hitro posvetovali s kolegi programa BMAG-RM. Od njih so izvedeli, da problem priprave osebnih urnikov spada v razred NP problemov.<sup>1</sup> Zvito so jim predlagali, da naj ta problem rešijo s prevodbo na problem SAT.<sup>2</sup> Rešitev le-tega naj poiščejo z Davis–Putnam–Logemann–Lovelandovim (DPLL) algoritmom.<sup>3</sup> Za programski jezik naj izberejo Standardni ML.

### Prvi del – logične formule in algoritem DPLL

Poljubne logične formule (izraze oz. izjave) lahko opišemo z naslednjim podatkovnim tipom:

```
datatype 'a expression = Not of 'a expression
                        | Or of 'a expression list
                        | And of 'a expression list
                        | Implies of 'a expression * 'a expression
                        | Equiv of 'a expression * 'a expression
                        | Variable of 'a
                        | True
                        | False
```

Konstruktor **Not** predstavlja negacijo, **Or** predstavlja disjunkcijo, **And** konjunkcijo, **Implies** logično implikacijo in **Equiv** logično ekvivalenco. Podatkovni tip `'a expression` je polimorfen, ker je tip imen logičnih spremenljivk (**Variable**) poljuben.

Za izpis “dolgi in globoki” izrazov lahko v SML uporabiš naslednjo nastavitev:  
`Control.Print.printDepth := 100;`

V programskem jeziku SML implementiraj naslednje funkcije:

- (3 t.) `eval (fn: ('a * bool) list -> 'a expression -> bool)`  
Sprejme seznam parov (*ime\_spremenljivke*, *vrednost*) in *logično\_formulo*. Uporablja naj *currying*. Vrne izračunano vrednost podane logične formule, v kateri uporabi tiste vrednosti logičnih spremenljivk, ki so določene v podanem seznamu. Če seznam spremenljivk vsebuje več vrednosti iste spremenljivke, naj funkcija uporabi njeno prvo vrednost v podanem seznamu.

Ker konjunkcija (**And**) in disjunkcija (**Or**) komutirata se posplošita iz dveh argumentov na poljubno mnogo – za argument prejmeta seznam. V primeru praznega seznama je pri funkciji **Or** rezultat **False**, v primeru funkcije **And** pa **True**. Pomisli, kaj je rezultat za ti dve funkciji, ko ima seznam le en element (v pomoč naj ti bo primer, ko imaš prazen seznam). Če formula vsebuje spremenljivko, ki je ni v seznamu, funkcija proži izjemo **InvalidVariable**.

<sup>1</sup><https://en.wikipedia.org/wiki/NP-hardness>

<sup>2</sup>[https://en.wikipedia.org/wiki/Boolean\\_satisfiability\\_problem](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem)

<sup>3</sup>[https://en.wikipedia.org/wiki/DPLL\\_algorithm](https://en.wikipedia.org/wiki/DPLL_algorithm)

Funkcijo implementirajte z minimalnim številom rekurzivnih klicev.

Primer.

```
- eval [(1, false), (2, true)] (And [True,
                                   Or [Variable 1, Not (Not (Variable 2))],
                                   Implies (Variable 1, Variable 2)]);

val it = true : bool
```

- (3 t.) **removeEmpty** (fn: 'a expression -> 'a expression)

Poenostavi logično formulo tako, da s konstantami (True in False) smiselno zamenja izraze logičnih funkcij Or in And, katerih argument je prazen seznam. Odstrani logične funkcije Or in And, če je argument le-teh seznam z enim samim elementom. Funkcijo implementirajte z minimalnim številom rekurzivnih klicev.

- (6 t.) **removeConstants** (fn: 'a expression -> 'a expression)

Poenostavi logično formulo tako, da odstrani vse konstante (True in False) na smiseln način, tj. upošteva lastnosti izjavnega računa (npr. Or [Variable "a", True] se poenostavi v True, And [Variable "a", True] se poenostavi v Variable "a", Implies (True, <expr>) se poenostavi v <expr>, itd.).

- (6 t.) **pushNegations** (fn: 'a expression -> 'a expression)

Poenostavi logično formulo tako, da "potisne" vse negacije Not do listov drevesa logične formule, tj. do spremenljivk oz. konstant. Odpravi naj dvojne (ali večkratne) negacije. V pomoč naj ti bodo De Morganovi zakoni. Pazi, da minimalno spreminjaš topologijo drevesa. V primeru negacije ekvivalence "potisni" negacijo po levi strani oz. veji. Primera:

```
- Not (Implies (Not (Not (Variable "a")), Variable "b")) se poenostavi v
  And [Variable "a", Not (Variable "b")],
- Not (Equiv (True, <right_expr>)) se poenostavi v Equiv (Not True, <right_expr>), itd.
```

- (6 t.) **removeVars** (fn: 'a expression -> 'a expression)

Poenostavi logično formulo tako, da združi oz. odstrani ponavljajoče se spremenljivke na smiseln način, tj. izkorišča lastnost idempotentnosti v primeru konjunkcije, disjunkcije, ekvivalence in implikacije (npr. Or [Variable "a", Variable "a", Not (Variable "b"), Not (Variable "b")] se poenostavi v Or [Variable "a", Not (Variable "b")], Implies (Variable "a", Variable "a")) se poenostavi v True, itd.).

- (6 t.) **simplify** (fn: 'a expression -> 'a expression)

Poenostavi oz. zmanjša število logičnih operatorjev, spremenljivk in konstant v logični formuli z uporabo do sedaj definiranih funkcij. Uporabi tudi druga pravila povezana z izjavnim računom.<sup>4</sup> Primera:

```
- Or [Variable "a", Not (Variable "a")] se poenostavi v True,
- Implies (Variable "a", Not (Variable "a")) se poenostavi v Not (Variable "a"), itd.
```

- (15 t.) **tseytinTransformation** (fn: 'a list -> 'a expression -> 'a expression)

Sprejme seznam *imena\_spremenljivk* in *logična\_formula*. Uporablja *currying*. Prejeto formulo naj transformira s Tseytinovo transformacijo.<sup>5</sup> Za imena novovpeljanih spremenljivk naj uporablja imena v seznamu *imena\_spremenljivk*. Če v logični formuli že obstaja spremenljivka, katere ime je v seznamu *imena\_spremenljivk* ali pa le-ta vsebuje podvojena imena, funkcija **tseytinTransformation** proži izjemo **InvalidVariable**. Če ima seznam *imena\_spremenljivk* premalo imen (tj. manj, kot je število prisotnih logičnih funkcij v vhodni logični formuli), funkcija proži izjemo **InsufficientVariables**. Katera imena spremenljivk iz vhodnega seznama uporabi, ni pomembno.

<sup>4</sup>glej. skripto prof. dr. Riste Škrekovskega – <https://www.fmf.uni-lj.si/~skreko/Gradiva/DS1-skripta.pdf> in skripto prof. dr. Gašperja Fijavža – <http://matematika.fri.uni-lj.si/ds/ds.pdf>

<sup>5</sup>[https://en.wikipedia.org/wiki/Tseytin\\_transformation](https://en.wikipedia.org/wiki/Tseytin_transformation)

Paziti moraš le, da vsaki funkciji v formuli *logična\_formula* prirediš (povežeš z `Equiv`) spremenljivko (`Variable`) z novim imenom (vsako ime v vhodnem seznamu imen uporabi le enkrat!).

Primer. Logična formula `Implies (And [Or [Variable "p", Variable "q"], Variable "r"], Not (Variable "s"))` se z uvedbo novih spremenljivk z imeni `["x4", "x3", "x2", "x1"]` (vhodni prameter) transformira v `And [`

```
Variable "x4",
Equiv (Variable "x4", Implies (Variable "x3", Variable "x1")),
Equiv (Variable "x3", And [Variable "x2", Variable "r"]),
Equiv (Variable "x2", Or [Variable "p", Variable "q"]),
Equiv (Variable "x1", Not (Variable "s"))].
```

Po uvedbi novih spremenljivk konjunkcijo ekvivalenc pretvori v konjunktivno normalno obliko – KNO (angl. *conjunctive normal form*, kratica CNF).<sup>6</sup>

V pomoč naj ti bodo naslednje povezave med formulami:

- Formula `Equiv (Variable "x", Not (Variable "a"))` je ekvivalentna formuli `And [Or [Not (Variable "a"), Not (Variable "x")], Or [Variable "a", Variable "x"]]`.
- Formula `Equiv (Variable "x", And [Variable "a", Variable "b"])` je ekvivalentna formuli `And [Or [Not (Variable "a"), Not (Variable "b"), Variable "x"], Or [Variable "a", Not (Variable "x")], Or [Variable "b", Not (Variable "x")]]`.
- Formula `Equiv (Variable "x", Or [Variable "a", Variable "b"])` je ekvivalentna formuli `And [Or [Variable "a", Variable "b", Not (Variable "x")], Or [Not (Variable "a"), Variable "x"], Or [Not (Variable "b"), Variable "x"]]`.
- Formula `Equiv (Variable "x", Implies (Variable "a", Variable "b"))` je ekvivalentna formuli `And [Or [Not (Variable "a"), Variable "b", Not (Variable "x")], Or [Variable "a", Variable "x"], Or [Not (Variable "b"), Variable "x"]]`.
- Formula `Equiv (Variable "x", Equiv (Variable "a", Variable "b"))` je ekvivalentna formuli `And [ Or [Not (Variable "a"), Not (Variable "b"), Variable "x"], Or [Not (Variable "a"), Variable "b", Not (Variable "x")], Or [Variable "a", Not (Variable "b"), Not (Variable "x")], Or [Variable "a", Variable "b", Variable "x"]]`.

Slednje formule predpostavljajo, da funkciji `And` in `Or` sprejemata seznama z dvema elementoma. Za le-te posploši zgornje relacije tudi za primere poljubnega števila "argumentov".

Izhod je drevo z višino največ 3, ki predstavlja logično formulo v obliki KNO (morebitne konstante smiselno odstrani).

- (15 t.) **SATsolver** (`fn: 'a expression -> ('a * bool) list option`)

Za vhodno logično formulo poskuša poiskati take vrednosti spremenljivk (seznam parov (*ime\_spremenljivke, vrednost*)), da se bo formula evalvirala v vrednost `True`. Ker tak nabor vrednosti spremenljivk ne obstaja vedno, funkcija vrača rezultat, ki je tipa `option`.

Funkcijo **SATsolver** implementiraj s pomočjo algoritma DPLL (ali pa z navadnim izčrpnim preiskovanjem v globino). Le-ta za vhod potrebuje logično formulo v obliki KNO. Za pretvorbo vhodne formule v obliko KNO lahko uporabiš funkcijo **tseytinTransformation**. Smiselno je, najprej preveriti, ali je logična formula že v obliki KNO.

Namen algoritma DPLL je izbrati vrednosti spremenljivk tako, da se bodo vsi `Or` izrazi v vhodni logični formuli evalvirali v `True`. Algoritem skuša poiskati tako rešitev s preiskovanjem v globino. Pri tem izbira vrednosti spremenljivke tako, da zadosti vsaj enemu izmed `Or` izrazov v trenutnem izrazu oblike KNO. Ko se odloči za spremenljivko in njeno vrednost, trenutni izraz poenostavi. Če se izraz poenostavi v `False`, algoritem poskuša z negirano vrednostjo izbrane spremenljivke. Če tudi to ne uspe, vrne *ni-rešitve*. Če se izraz ne poenostavi niti v `False` niti v `True` potem algoritem rekurzivno nadaljuje z preiskovanjem v globino.

<sup>6</sup>[https://en.wikipedia.org/wiki/Conjunctive\\_normal\\_form](https://en.wikipedia.org/wiki/Conjunctive_normal_form)

Algoritem DPLL naj vodi evidenco o “odstranjenih” spremenljivkah in njihovih vrednostih (kar služi kot izhod funkcije **SATsolver**). Odstranjene spremenljivke so tiste, ki imajo trenutno določeno vrednost, v samem izrazu pa ne nastopajo več. Koraki algoritma so naslednji.

- **Korak 1.** Poišči spremenljivke, ki stojijo same (so oblike `Or [Variable <name>]` ali `Or [Not (Variable <name>)]`). V primeru `Or [Variable <name>]` v seznam “odstranjenih” spremenljivk dodamo par `(<name>, true)`, v primeru `Or [Not (Variable <name>)]` pa `(<name>, false)`. Trenutno logično formulo poenostavimo z izbranimi vrednostnimi odstranjenih spremenljivk. Ta korak ponavljamo, dokler nimamo več izoliranih spremenljivk.
- **Korak 2.** Če je trenutna logična formula enaka `And []` oz. `True` potem smo z izvajanjem algoritma končali (smo našli take vrednosti spremenljivk, da se je formula evalvirala v vrednost `True`) in vrnemo seznam parov (*ime\_spremenljivke, vrednost*). Opomba: seznam spremenljivk ne obsega nujno vseh spremenljivk, ki so na začetku izvajanja algoritma nastopale v logični formuli.

Če logična formula vsebuje izraz `Or []` oz. je vrednost trenutne logične formule enaka `False`, vrnemo `NONE`. V nasprotnem primeru nadaljujemo s korakom 3.

- **Korak 3.** Izberemo eno poljubno spremenljivko v logični formuli ter jo nastavimo na tako vrednost, da se bo logična funkcija `Or`, ki vsebuje izbrano spremenljivko, evalvirala v logično vrednost `True`. Z izbiro vrednosti izbrane spremenljivke formulo poenostavimo (na podoben način kot v 1. koraku). Nad dobljeno formulo rekurzivno izvedemo algoritem DPLL.

Če je rezultat rekurzivnega klica `NONE`, potem izvedemo še en rekurziven klic, ampak v tem primeru izberemo negirano vrednost izbrane spremenljivke (logično formulo tudi ustrezno poenostavimo). Če je tudi v tem primeru rekurziven klic neuspešen, vrnemo `NONE`.

- (5 t.) **equivalentExpressions** (fn: `''a expression -> ''a expression -> bool`)  
Preveri, ali sta vhodna izraza ekvivalentna. (Namig: izraza `<expr1>` in `<expr2>` nista ekvivalentna, če in samo če DPLL algoritem najde “rešitev” za izraz `Not (Equiv (<expr1>, <expr2>))`.)

## Drugi del – reševanje problema urnikov

Asistenti so študentom posredovali naslednja podatka:

- Termine vaj za posamezen predmet.

Primer (dva predmeta, skupno pet ciklov vaj):

```
{day = "torek", time = 7, course = "DS"},
{day = "sreda", time = 10, course = "DS"},
{day = "petek", time = 14, course = "DS"},
{day = "torek", time = 7, course = "P2"},
{day = "ponedeljek", time = 12, course = "P2"}] : timetable.
```

- Predmetnik za vsakega študenta.

Primer (dva študenta z različnima predmetnikoma):

```
{studentID = 63170000, curriculum = ["DS", "P2", "OPB", "RK"]},
{studentID = 63160000, curriculum = ["P2", "RK", "ARS"]} : student list.
```

Sinonima `timetable` in `student` predstavljata naslednja tipa:

```
type timetable = {day : string, time: int, course: string} list
type student = {studentID : int, curriculum : string list}
```

Implementiraj naslednje funkcije:

- (15 t.) **problemReduction** (fn: `timetable -> student list -> <custom_type> expression`)  
Problem urnikov predstavi z logično formulo v obliki KNO. Predpostaviš lahko, da so na vsakih vajah natanko 3 prosta mesta (lahko pa število mest poljubno povečate) in da se vaje izvajajo le eno uro.

Izbiri cikla vaj pri posameznem predmetu lahko predstavimo z logičnimi spremenljivkami  $x_{s,c,t,p}$ . Spremenljivka  $x_{s,c,t,p}$  ima štiri indekse. Indeks  $s$  določa študenta, indeks  $c$  določa predmet, indeks  $t$  določa cikel vaj in indeks  $p$  mesto oz. sedež na vajah.

Če ima spremenljivka  $x_{s,c,t,p}$  logično vrednost *true*, to pomeni, da študent  $s$ , pri predmetu  $c$ , obiskuje cikel vaj  $t$  in sedi na sedežu  $p$  (in obratno, če ima logično vrednost *false*).

Želimo, da logična formula v obliki KNO opiše naslednje lastnosti:

- Vsak študent obiskuje (vsaj) en cikel vaj za vsak premet njegovega predmetnika. To nam da naslednji del formule v CNF.

$$\bigwedge_{s \in S} \bigwedge_{c \in C_s} \bigvee_{t \in T_c} \bigvee_{p \in P_t} x_{s,c,t,p}$$

Znak  $\vee$  predstavlja disjunkcijo, znak  $\wedge$  pa konjunkcijo. Možica  $S$  prestavlja študente, množica  $C_s$  predstavlja predmete študenta  $s$ , množica  $T_c$  predstavlja možne cikle vaj pri predmetu  $c$  in množica  $P_t$  sedišča pri ciklu vaj  $t$ .

- Izbrani cikli vaj se nobenemu izmed študentov ne prekrivajo.

$$\bigwedge_{s \in S} \bigwedge_{\substack{c_1, c_2 \in C_s \\ t_1 \in T_{c_1}, t_2 \in T_{c_2} \\ \text{time}(t_1) = \text{time}(t_2) \\ p_1 \in P_{t_1}, p_2 \in P_{t_2} \\ (c_1, t_1, p_1) \neq (c_2, t_2, p_2)}} (\neg x_{s,c_1,t_1,p_1} \vee \neg x_{s,c_2,t_2,p_2})$$

Oznake imajo enak pomen kot v prejšnji točki. Oznaka  $\neg$  predstavlja negacijo. Funkcija `time` za dan cikel vaj vrne začetek izvajanja. Opisana logična formula v obliki KNO pravi, da noben študent ni na dveh ciklih vaj istočasno in da ne sedi pri istemu ciklu na dveh ali več sedežnih. Opomba: ti pogoji študenta ne ovirajo pri obiskovanju več ciklov vaj za isti premet.

- Na vsakem stolu sedi največ en študent.

$$\bigwedge_{c \in C} \bigwedge_{t \in T_c} \bigwedge_{p \in P_t} \bigwedge_{\substack{s_1, s_2 \in S_c \\ s_1 \neq s_2}} (\neg x_{s_1,c,t,p} \vee \neg x_{s_2,c,t,p})$$

Množica  $S_c$  predstavlja množico študentov, ki so vpisani v predmet  $c$ . Logična formula pa pravi, da noben par študentov ne sedi na istem stolu pri istih vajah istega predmeta.

Ker je podatkovni tip logičnih izrazov polimorfen (**'a expression**), lahko podatkovni tip imen spremenljivk izbereš poljubno (npr. ime predstavimo kot četverko (`s:int, c:string, t:int, p:int`)).

- (5 t.) **solutionRepresentation** (fn: (`<custom_type> * bool`) list -> student \* timetable list)

Rezultat funkcije **SATsolver** smiselno predstavi kot seznam parov (*študent, urnik*).

## Oddaja seminarske naloge

Poleg ene datoteke `sem1-<vpisna_stevilka>.sml` oddaj tudi poročilo kot en PDF dokument. Poročilo naj vsebuje opis implementacije posamezne funkcije ter rezultate in opis testiranja pravilnosti le-teh. Vse implementirane funkcije (tudi pomožne) naj bodo dokumentirane v samem poročilu. Za nadgradnje boš dobil največ 5 točk, za sam slog kode in optimizacijo pa največ 5. Skupno število točk ne more preseči 100.

Oddana seminarska naloga bo ocenjena glede na naslednji točkovnik:

funkcija <b>eval</b>	3 točke
funkcija <b>removeEmpty</b>	3 točke
funkcija <b>removeConstants</b>	6 točk
funkcija <b>pushNegations</b>	6 točk
funkcija <b>removeVars</b>	6 točk
funkcija <b>simplify</b>	6 točk
funkcija <b>tseytinTransformation</b>	15 točk
funkcija <b>SATsolver</b>	15 točk
funkcija <b>equivalentExpressions</b>	5 točk
funkcija <b>problemReduction</b>	15 točk
funkcija <b>solutionRepesesntation</b>	5 točk
nadgradnje	5 točk
slog kode in optimizacija	5 točk
ustni zagovor	5 točk
<b>skupaj</b>	100 točk