Vybraná témata z algoritmů

Abstract

Poznámky k semináři VYTAL v letním semestru 2024. Poslední úprava March 6, 2024.

1 První seminář

1.1 Formule, přiřazení, SAT-problém

Booleovská formule je definována induktivně:

- konstanty 0 a 1 jsou formule,
- proměnná x (z nekonečné spočetné množiny proměnných $\{x_1, x_2, \ldots\}$) je formule,
- pokud jsou F a G formule, pak i $\neg F$ (někdy zapisujeme i $\overline{F}$), $(F \wedge G)$, a $(F \vee G)$.

Lze přidat další možnosti jako zkratky

- $(F \to G)$ je zkratka za $\neg F \vee G$,
- $(F \leftrightarrow G)$ je zkratka za $(F \to G) \wedge (G \to F)$,
- $(F \oplus G)$ je zkratkou za $(F \vee G) \wedge (\neg F \vee \neg G)$

Množinu proměnných ve formuli G budeme značit $\textsc{var}(G)$.

Ohodnocení α je částečné zobrazení z množiny proměnných do množiny $\{0, 1\}$. Budeme jej zapisovat také

$$\alpha = \{x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k\},$$

kde $x_1, \ldots$ jsou proměnné a $a_1, \cdots \in \{0, 1\}$. Množinu proměnných ohodnocených α budeme značit $\textsc{var}(\alpha)$.

Výsledkem aplikace ohodnocení α na formuli F, značeno $F\alpha$ je formule, kterou dostaneme následujícím postupem

1. Ve formuli F provedeme symbolickou substituci za proměnné $\textsc{var}(F) \cap \textsc{var}(\alpha)$,
2. Aplikujeme zjednodušovací pravidla, např.
   - $(G \vee 0)$ a $(0 \vee G)$ zjednodušíme na G,
   - $(G \vee 1)$ a $(1 \vee G)$ zjednodušíme na 1,
   - $(G \wedge 0)$ a $(0 \wedge G)$ zjednodušíme na 0,
   - $(G \wedge 1)$ a $(1 \wedge G)$ zjednodušíme na G,
   - $\neg\neg G$ zjednosušíme na G,
   - $\neg 0$ zjednodušíme na 1,
   - $\neg 1$ zjednodušíme na 0.

Formule F je *splnitelná*, když existuje ohodnocení α tak, že $F\alpha = 1$. Pokud F není splnitelná, je to *kontradikce.* F je tautologie, pokud pro každé ohodnocení α, pro které je $\text{VAR}(F) \subseteq \text{VAR}(\alpha) =$, platí $F\alpha = 1$.

Pozorování 1. Pro libovolnou formuli F a proměnnou $x \in \text{VAR}(F)$ platí:

- F je tautologie, právě když $\neg F$ je kontradikce.
- F je splnitelné, právě když $F\{x = 1\}$ nebo $F\{x = 0\}$ je splnitelná.
- F je kontradikce, právě když $F\{x = 1\}$ a $F\{x = 0\}$ jsou kontradikce.

Definujeme množiny formulí

$$\text{SAT} = \{F \mid F \text{ je splnitelná}\}$$
$$\text{TAUT} = \{F \mid F \text{ je tautologie}\}$$

Hned vidíme TAUT $\subset$ SAT.

Formule F a G jsou *sémanticky ekvivalentní* pokud pro každé ohodnocení α takové, že $\text{VAR}(\alpha) = \text{VAR}(F) \cup \text{VAR}(G)$, máme $F\alpha = G\alpha$. Formule F a G jsou *sat ekvivalentní* pokud F je splnitelná právě když G je splnitelná.

Známe už transformace, které zachovávají sémantickou ekvivalenci, například De Morganovi zákony:

$$\neg(F \wedge G) = (\neg F \vee \neg G)$$
$$\neg(F \vee G) = (\neg F \wedge \neg G)$$

S jejich pomocí (a s použitím zákona dvojí negace) můžeme převést libovolnou formuli na sémanticky ekvivalentní formuli, která má negace pouze u proměnných. Tomuto tvaru se říká *negativní normální forma.*

SAT-problém je problém rozhodnout, zda daná formule F patří do množiny SAT (tedy jestli je F splnitelná). Tento problém je důležitý z teoretického (teorie NP-úplnosti atd.) a praktického pohledu (použití SAT solverů a převod algoritmických problémů na SAT-problém). Triviální algoritmus pro SAT-problém zkouší všechna ohodnocení a má exponenciální složitost vzhledem k počtu proměnných ve formuli.

1.2 Konjunktivní normální forma

Formule F je v *konjunktivní normální formě*, pokud je ve tvaru

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$

kde podformule $C_1, C_2, \ldots$ nazýváme *klausule.* Klausule je ve tvaru

$$(u_1 \vee u_2 \vee u_3 \ldots u_k),$$

kde $u_i \in \{x_1, x_2, \ldots\} \cup \{\neg x_1, \neg x_2, \ldots\}$. Říkáme jim *literály.*

Zavedeme množiny

$$\text{CNF} = \{F \mid F \text{ v konjunktivní normální formě}\}$$
$$\text{CNF} - \text{SAT} = \text{SAT} \cap \text{CNF}$$
$$k\text{–SAT} = \{F \in \text{CNF} - \text{SAT} \mid \text{ každá klasule v } F \text{ má nejvýše } k \text{ literálů}\}$$

Věta 1. Ke každé formuli F existuje sématicky ekvivalentní formule G v konjunktivní normální formě. Převod F na G obecně zabere exponenciální čas vhledem k $|\textsc{var}(F)|$.

Proof. Klíčové myšlenky: formule je reprezentací booleovské funkce. Obecně, k booleovské funkci existuje formule v CNF která ji reprezentuje. $\square$

Pozn. existuje formule, pro kterou je výsledná formule v CNF exponenciálně větší. Nemůže tak existovat algoritmus, který by převáděl do konjunktivní normální formy a pracoval v polynomickém čase. (Detaily na semináři).

Algoritmus pro převod formule do CNF je opakováním z prvního ročníku.

1.3 Splnitelnost zachovávající redukce

Existuje například redukce z obecného CNF tvaru na 3—SAT tvar. (Detaily na semináři).

Probereme redukce převádějící formuli v obecném tvaru do CNF. První je *Tseitinova redukce.* Tady si představíme proces výpočtu pravdivosti formule F strukturální indukcí podle definice formule (tady se objevil Booleovský výpočetní obvod jako další reprezentace booleovské funkce). Správnost tohoto výpočtu zakódujeme do CNF formule G tak, že G je splnitelná právě když je F splnitelná.

Pokud je F v negativní normální formě obsahuje odpovídající booleovský obvod pouze monotoní brány a drobnou úpravou Tseitinovi redukce dostaneme formuli, která má méně klauzulí.

Další metoda redukce vyžadující F v negativní normální form je založena na konstrukci speciálního grafu a procházení cest v něm.

2 Seminář 2

Příklady redukcí problémů na CNF formuli. (Tvorbu výsledné formule popisujeme tak, že řekneme jaké vytvoříme klausule. Že tyto klausule spojíme pomocí konjunkce už explicitně neopakujeme.)

2.1 Barvení grafu

Instancí problému barvení je neorientovaný graf s množinou vrcholů V a množinou hran H a přirozené číslo d. Cílem je zjistit, jestli existuje zobrazení $c : V \to \{1, 2, \dots, d\}$ takové, že pro každou $(v, w) \in H$ máme $c(v) \neq c(w)$. (*Představíme si, že vrcholy grafu obarvujeme barvami* $1, 2, \dots, d$. *Funkce* c *určuje barvu vrcholu. Chceme aby sousední vrcholy měli různé barvy.*)

K vytvoření formul potřebujeme $|V| \cdot d$ proměnných. Po každý vrchol v vytvoříme proměnné $v_1, v_2, \dots, v_d$. Pokud je v_i pravdivá, bere me to tak, že vrchol v lze obarvit barvou i. Dále vytvoříme klausuli

$$(v_1 \vee v_2 \vee \ldots \vee v_d)$$

a pro každé $i < j$ klausuli

$$(\neg v_i \vee \neg v_j)$$

Tím zajistíme, že každý vrchol lze (pro dané ohodnocení proměnných) obarvit pouze jednou barvou. Zbývá dodat pro každou hranu (w, v) a pro každé $i \in \{1, 2, \dots, d\}$ klausuli

$$(\neg w_i \vee \neg v_j).$$

Tak zajistíme, že sousední vrcholy nebudou obarveny stejnou barvou.

2.2 Bounded Model Checking

Studujeme posloupnosti

$$X_0 \to X_1 \to \dots X_r, \tag{1}$$

kde X_i je vektor booleovských hodnot, který popisuje stav nějakého systému a zápis $X_i \to X_{i+1}$ značí, že systém přešel jedním krokem ze stavu X_i do stavu X_j. (Pod systémem si můžeme představit třeba program, jehož stav je dán hodnotami proměnných.) Chování systému popíšeme tak, že pomocí formule φ zachytíme platné situace $X \to X'$: Formule musí samozřejmě obsahovat proměnné, které odpovídají booleovským hodnotám v X a X', ohodnocení proměnných dá skutečné vektory booleovských hodnot, formule φ je pro toto ohodnocení pravdivá, právě když pro tyto skutečné vektory můžeme danou změnu stavu provést.

Typicky pak formulemi popíšeme i X_0 a X_r a ptáme se, jestli je v systému proveditelná posloupnost (1) tak, že se ptáme na splnitelnost o tyto nové formule obohacené φ. X_r je například nějakých chybový stav a chceme zjistit, jestli se do něj může systém dostat.

Příklad: program pro problém kritické sekce.

proces A:
$A0$: Maybe goto $A1$
$A1$: If l goto $A1$; else goto $A2$
$A2$: Set $l \leftarrow 1$; goto $A3$
$A3$: Critical; goto $A4$
$A4$: Set $l \leftarrow 0$; goto $A0$

proces B:
$B0$: Maybe goto $B1$
$B1$: If l goto $B1$; else goto $B2$
$B2$: Set $l \leftarrow 1$; goto $B3$
$B3$: Critical; goto $B4$
$B4$: Set $l \leftarrow 0$; goto $B0$

Procesy A,B se libovolně střídají v provádění kroků, sdílejí proměnnou l. Chceme zjistit, jestli se mohou oba procesy současně dostat do stavu 3 (A do $A3$, B do $B3$) ze stavu 0 ($A0$, $B0$). (Pokud ano, náš protokol pro kritickou sekci je chybný).

Stav programu zachytíme pomocí proměnných $A0, \dots, A4, B0, \dots, B4$ s odpovídajícími indexy. Význam: Pokud je například $A2_k$ pravdivě ohodnocená, interpretujeme to tak, že po provedení k kroků (celkem pro oba procesy) je proces A ve stavu $A2$. Způsobem známým z předchozí kapitoly zajistíme, že je vždy (pro daný index) pravdivá právě jedna proměnná procesu A a jedna proměnná procesu B. Dále použijeme proměnnou l a proměnnou @, která bude pravdivá, pokud další krok provede proces A a nepravdivá, pokud jej provede proces B.

Všimneme si, že v procesech jsou 4 typy příkazů:

1. Maybe goto s
2. Critical; goto s
3. Set $l \leftarrow b$; goto s
4. If l goto $s1$; else goto $s2$

Nyní si ukážeme, jak do formule zakódovat korektnost provádění obou procesů. Budeme uvažovat přechod $X \to X'$, proměnné pro X budou bez čárky, proměnné pro X' s čárkou (*Popíšeme tedy princip tvorby formule, ve skutečnosti musíme tuto formuli vytvořit pro každé* $X_i \to X_{i+1}$*, které v posloupnosti máme.*)

Stavy procesu A budeme značit pomocí α, stavy procesu B pomocí β. Napíšeme postup pro proces A, pro proces B je postup stejný, jenom se zamění @ za $\neg$@ a β za α.

- Pokud α odpovídá příkazu typu 1, potom vytvoříme klausuli
$$(\neg @ \vee \neg \alpha \vee \neg \alpha' \vee s')$$
- Pokud α odpovívá příkazu typu 2, potom vytvoříme klausuli
$$(\neg @ \vee \neg \alpha \vee s')$$
- Pokud α odpovívá příkazu typu 3, potom vytvoříme klausule
$$(\neg @ \vee \neg \alpha \vee s'),$$
$$\begin{cases} (\neg @ \vee \neg \alpha \vee l') & \text{pokud } b = 1 \\ (\neg @ \vee \neg \alpha \vee \neg l') & \text{pokud } b = 0 \end{cases}$$
- Pokud α odpovívá příkazu typu 4, potom vytvoříme klausule
$$(\neg @ \vee \neg \alpha \vee \neg l \vee s_1'),$$
$$(\neg @ \vee \neg \alpha \vee l \vee s_2'),$$

Nakonec musíme říci, že proměnnou l mohou měnit pouze ty α, které odpovídají příkazu 3. Označíme-li takovou množinu $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, potom to odpovídá vytvoření klausulí
$$(\neg @ \vee l \vee \alpha_1 \vee \ldots \vee \alpha_k \vee \neg l'),$$
$$(\neg @ \vee \neg l \vee \alpha_1 \vee \ldots \vee \alpha_k \vee l'),$$

Poznámka: Vytvořeným klausulím budete lépe rozumět, pokud si představíte jako implikace. Například klasule $(\neg @ \vee \neg \alpha \vee s')$ je vlastně implikace $(@ \wedge \alpha) \to s'$, která říká *Pokud A provádí další krok a je ve stavu* α*, pak přejde do stavu* s.

Počáteční stav programu je potom dán formulí $A0_0 \wedge B0_0$, a poslední stav je dán $A3_r \wedge B3_r$.

3 Seminář 3

Kombinatorické úvahy o splnitelnosti, jsou obsaženy v ručně psaných poznámkách.

4 Seminář 4

Backtracking. DPLL algoritmus, detaily v ručně psaných poznámkách. Ukázka SAT solveru.

Představení zápočtového úkolu.