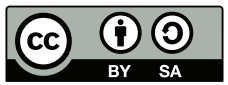
brmiversity: Umělá inteligence a teoretická informatika

Přednáška č. 13

Petr Baudiš ⟨pasky@ucw.cz⟩

brmlab 2011



brmlab hackerspace prague

Outline

Reprezentace znalostí

- Doteď spíš emergentní, automaticky naučené znalosti
- Dnes explicitní logická reprezentace
- Predikátová logika, SAT
- Výroková logika, temporální logika atd.

- Prvotní motivace: Dokazování vět ve formálních systémech
- Teď: Odvozování logických důsledků v bázi znalostí — *uvažování*

- Popis (jednoduchého) světa a logikou řízený agent
- Deduktivní databáze a produkční systémy, logické programování
- Formální verifikace (software, hardware, . . .)
- Strojové dokazování vět (matematická tvrzení, automatický vědec)

brmlab hackerspace prague

Znalostní báze

- Znalosti o světě i možné akce můžeme vyjádřit formálně (třeba *logické predikáty* — logické výrazy s proměnnými)
- Zajímá nás, jak implementovat **bázi znalostí** (množinu všeho, co víme)
- Operace *TELL* přidá do báze nový poznatek
- Operace *ASK* bázi položí *dotaz*
  Rozdíl vůči databázi: Testujeme splnitelnost logického výrazu, báze musí sama *odvodit důkaz*
- Typicky nevíme vše — neúplná informace, vzhledem ke znalostem máme několik *možných modelů* světa; pak musíme použít další heuristiky

Predikátová a výroková logika

- Proměnné: Booleovsky ohodnocené (true, false) popisují "mapu světa"
- Predikáty: Vztahy mezi proměnnými
- Formule: Řetězení podformulí operátory $\wedge, \vee, \neg$ popisuje "zákony světa"
- $a \Rightarrow b$ je $\neg b \vee a$, $a \Leftrightarrow b$ je $(a \wedge b) \vee (\neg a \wedge \neg b)$

- **Syntax:** Definuje, co vše je formule
- **Sémantika:** Říká, jestli je v daném světě (modelu) formule pravdivá

- Výroková logika (zatím odložíme stanou): funkce (výroba "podproměnných")
- Existenční $\exists$ a univerzální $\forall$ kvalifikátor

Odvozovací metody

- $x_0 = 0 \quad x_1 = 1 \quad x_2 = 0 \quad x_3 = 1 \quad x_4 = 1$
- $x_0 \vee \neg x_1 \vee x_2 \vee \neg x_3 \vee x_4 \Rightarrow x_5$
- Zajímá nás, jestli platí $x_3 \vee x_5$ Odvozování: Modus ponens
- **Ověřování modelu:** Spočítáme x_5 a dosadíme
- **Odvozovací pravidla:** Odhalíme tautologii $(x_3 \vee \neg x_3)$
- Korektnost: Nikdy neschválíme neplatný důsledek
- Úplnost: Vždy schválíme platný důsledek

- Věta je **splnitelná** $\Leftrightarrow$ je pravdivá v nějakém modelu $(x_0 \vee x_1)$
- Věta je **nesplnitelná** $\Leftrightarrow$ nepravdivá v každém modelu $(x_1 \wedge x_5)$
- Máme $ASK(KB, \alpha)$ (je α logický důsledek báze KB?); můžeme převést na testování splnitelnosti
- **Enumerace:** Generuj a testuj — zkus všechny kombinace výrokových proměnných, existuje nějaká splňená v KB i α?

Příklad: Wumpus World

- Nákres a pravidla
- Příklad znalostí a dotazu
- (Slajdy R. Bartáka.)

Výroková logika

- Výrokovou logiku lze redukovat na predikátovou
- **Grounding:** Za proměnné dosadíme všechny možné termy a z nich pak vyrobíme výrokové proměnné
- **Skolemizace:** Výsledky existenčních kvantifikátorů nám bude vyrábět Skolemova konstanta, resp. funkce
- Grounding generuje spoustu irelevantních výrazů, odvozujme přímo v PL!

Odvozování ve výrokové logice

- **Lifting:** Kvantifikátory substituujeme až v modus ponens
- Jak najdeme substituci kvantifikátorů?

$$\text{Nechť } Knows(Karel, Jarda)$$
$$UNIFY(Knows(Karel, x), Knows(y, z)) = \{x/Jarda, y/Karel, z/Jarda\}$$
$$UNIFY(Knows(Karel, x), Knows(y, z)) = \{y/Karel, x/z\}$$

- **Unifikace:** Substiticí je více možných, chceme najít *nejobecnější unifikátor*
- **Standardizace stranou:** Proměnné z univerzálních kvalifikátorů nazveme unikátně

Konjuktivní normální forma

CNF: konjunkce klauzulí, klauzule je disjunkce proměnných

$$(x_0 \vee \neg x_1) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

Všechny výrazy lze převést na CNF (i ve výrokové logice):

$$\forall x : (\forall y : Animal(y) \Rightarrow Loves(x, y)) \Rightarrow (\exists y : Loves(y, x))$$

$$\forall x : (\neg \forall y : (\neg Animal(y) \vee Loves(x, y))) \vee (\exists y : Loves(y, x))$$

$$\forall x : (\exists y : (Animal(y) \wedge \neg Loves(x, y))) \vee (\exists y : Loves(y, x))$$

$$\forall x : (\exists y : (Animal(y) \wedge \neg Loves(x, y))) \vee (\exists z : Loves(z, x))$$

$$((Animal(F(x)) \wedge \neg Loves(x, F(x)))) \vee (Loves(G(x), x))$$

$$(Animal(F(x)) \vee Loves(G(x), x)) \wedge (\neg Loves(x, F(x)) \vee Loves(G(x), x))$$

Testování splnitelnosti CNF je stále NP-úplné (SAT)

Nejtěžší formule mají $klauzul/promnnch = 4.3$

Ověřování modelů

$ASK(KB, \alpha)$; už jsme zkoušeli enumerovat

DPLL (Davis, Putnam, Logemann, Loveland)

- Rekurzivní backtracking s heuristikami
- Brzká terminace, známe-li hodnotu některých klauzulí
- H. ryzí proměnné: dosadíme, mají-li $\forall$ výskyty stejné znaménko
- H. jednotkové klauzule: dosadíme, vystupuje-li proměnná samostatně

WalkSAT

- Hill-climbing random walk (rychlý, není úplný)
- Iterativně vybíráme neplatné klauzule, s pravděpodobností p přepeneme náhodnou proměnnou, jinak proměnnou, která opraví co nejvíce klauzulí

Obecná rezoluce

- $ASK(KB, \alpha)$ pomocí odvozovacích pravidel (ne dosazováním hodnot)
- Obecná rezoluce: Důkaz sporem!
- Začneme s $KB \wedge \neg\alpha$, převedeme na CNF
- Aplikujeme rezoluční pravidlo: $(l_1 \vee l_2), (\neg l_2 \vee l_3) | (l_1 \vee l_3)$
- Tedy hledáme a sjednocujeme klauzule s komplementárním (vzájemně negovaným) výskytem proměnné
- Pokud se dvě klauzule zcela vymlátí, dokázali jsme spor (a platí α)
- Pokud už nelze vytvářet nové klauzule, spor není (a platí $\neg\alpha$)

Hornovské klauzule

- **Hornovská klauzule:** Klauzule speciálního tvaru, kde je maximálně jeden pozitivní literál
- $C \wedge (\neg B \vee A) \wedge (\neg C \vee \neg D \vee B)$
- Jak KB v takové formě může vzniknout?

Hornovské klauzule

- **Hornovská klauzule:** Klauzule speciálního tvaru, kde je maximálně jeden pozitivní literál
- $C \wedge (\neg B \vee A) \wedge (\neg C \vee \neg D \vee B)$
- Jak KB v takové formě může vzniknout?
  Implikace s jedním důsledkem!
- $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
- Logické programování (Prolog)

- Jednoduché (rychlé a srozumitelné) odvozování
- Rozhodovací problém má složitost pouze $O(N)$

brmlab hackerspace prague

Řetězení

Dopředné řetězení

- Idea: Z faktů generujeme postupně důsledky, dokud nedokážeme, co jsme chtěli
- Pro každou klauzuli máme čítač nesplněných předpokladů
- Postupně bereme důsledky splněných klauzulí (čítač na nule) a dekrementujeme čítače dalších klauzulí
- **Rete algoritmus:** Síť závislostí, inkrementální dokazování

Zpětné řetězení

- Idea: Hledáme, jaké předpoklady musíme splnit pro dokázání důsledků, dokud nejsou všechny splněné

Reprezentace znalostí

- Kde začít?
- Příklad: Elektrické obvody
- Slajdy R. Bartáka
- Kategorie a taxonomie

Otázky?

Příště: Automatické plánování (dokazování v situačním kalkulu) a splňování omezujících podmínek.

Outline

Rekapitulace — Třídy složitosti

- **Třídy složitosti:** Skupina algoritmů se stejnou řádovou složitostí v závislosti na délce vstupu (P, NP, LSPACE, PSPACE, EXPTIME, ...)
- P: Všechny algoritmy, které běží v *polynomiálním čase* na "obyčejném" Turingově stroji (DTS)
- NP: Algoritmy, které běží v polynomiálním čase na nedeterministickém Turingově stroji (NTS)
- PSPACE: Všechny algoritmy, které sežerou *polynomiálně mnoho pásky*

- NP-úplný problém

Rekapitulace — Třídy složitosti

- **Třídy složitosti:** Skupina algoritmů se stejnou řádovou složitostí v závislosti na délce vstupu (P, NP, LSPACE, PSPACE, EXPTIME, ...)
- P: Všechny algoritmy, které běží v *polynomiálním čase* na "obyčejném" Turingově stroji (DTS)
- NP: Algoritmy, které běží v polynomiálním čase na nedeterministickém Turingově stroji (NTS)
- PSPACE: Všechny algoritmy, které sežerou *polynomiálně mnoho pásky*

- NP-úplný problém: Je v NP a lze na něj všechny NP problémy převést v polynomiálním čase

Pseudopolynomiální algoritmy

- Kódování vstupu může být unární nebo binární (či jiné)
- Někdy to není jedno; časová složitost vzhledem k délce vstupu vs. k velikosti čísla na vstupu
- Buď n číslo na vstupu; $l = \log_2 n$ délka zakódování ($n \simeq 2^l$)
- **Pseudopolynomiální algoritmus** má dobu běhu omezen polynomem v n, ne v l

- Každý polynomiální algoritmus je i pseudopolynomiální
- Naopak to neplatí! (Např. součet podmnožiny.)

- **Číselný problém** je takový, kde není nutně $n \leq p(l)$
- Není-li NP-úplný problém číselný, nelze jej řešit pseudopolynomiálním algoritmem
- Je-li NP-úplný problém číselný, může být ještě **silně NP-úplný** a také jej tak nelze řešit (např. TSP)

Aproximační algoritmy

- Povolme suboptimální řešení NP-těžkých optimalizačních problémů
- Budeme chtít polynomiální čas a odhad kvality vzhledem k optimu
- Odhad kvality: Poměrová chyba A/O nebo relativní chyba $|A - O|/O$
- Např. vrcholové pokrytí lze aproximovat s poměrovou chybou max. 2

- **Aproximační schéma:** $AS_X(Y, e)$ řeší $X(Y)$ s max. relativní chybou e
- **Úplně polynomiální aproximační schéma:** AS časem omezené v polynomu $l(Y)$ i e
- Pro silně NP-úplné úlohy ÚPAS neexistuje (pokud $P \neq NP$)

Otázky?

To je o výpočetní složitosti vše!

Nemluvili jsme o řadě krásných věcí . . .

TS s přímým přístupem, RAM stroje, booleovské obvody.

L a NL, interaktivní dokazovací systémy,
třídy pravděpodobnostních algoritmů,
praktický nedeterminismus (kvantové počítače).

Outline

Datové struktury ve vnější paměti

- Vnější paměť: Pomalá, *blokový* přenos (po stránkách)
- Můžeme chtít měřit složitost v počtu I/O operací

- B-sort
- Externí hashování (Fagin, Cormack, Larson-Kalja)

Externí hashování

Faginovo hashování

- Parametr tabulky: Prvních k bitů hashe
- 2^k řádků v tabulce, každý obsahuje ukazatel na stránku s množinou prvků
- Pokud se nějaká stránka naplní, rozštěpí se a zvýší se k

Perfektní hashování

- Cormack: Mám hashovací funkce $h(x), g(i, x)$
- Adresář je hashovaný přes h a obsahuje i pro g, hlavní soubor adresovaný přes g
- Larson-Kalja: Jen h, tabulka stránek se separátorem (horní mez h)

Otázky?

Příště: Třídění ve vnitřní a vnější paměti.

Děkuji vám

pasky@ucw.cz

Příště: Umělá inteligence. Neuronové sítě (zbytky). Datové struktury. Vyčíslitelnost (Gödelovy věty).

brmlab hackerspace prague