Vysvětlování modelovacích chyb

Petr Křemen

FEL ČVUT

Co nás čeká

1. Vysvětlování modelovacích chyb

2. Black-box metody
   - Algoritmy založené na CS-stromech
   - Algoritmus založený na Reiterově algoritmu

Vysvětlování modelovacích chyb

Motivace

- Představme si, že budujeme ontologii a inferenční stroj nám v určitém okamžiku sdělí, že je tato ontologie nekonzistentní. **Co s tím ?**

Motivace

- Představme si, že budujeme ontologii a inferenční stroj nám v určitém okamžiku sdělí, že je tato ontologie nekonzistentní. **Co s tím ?**
- **Můžeme začít ručně procházet** axiomy a koukat se, "kde je co špatně".

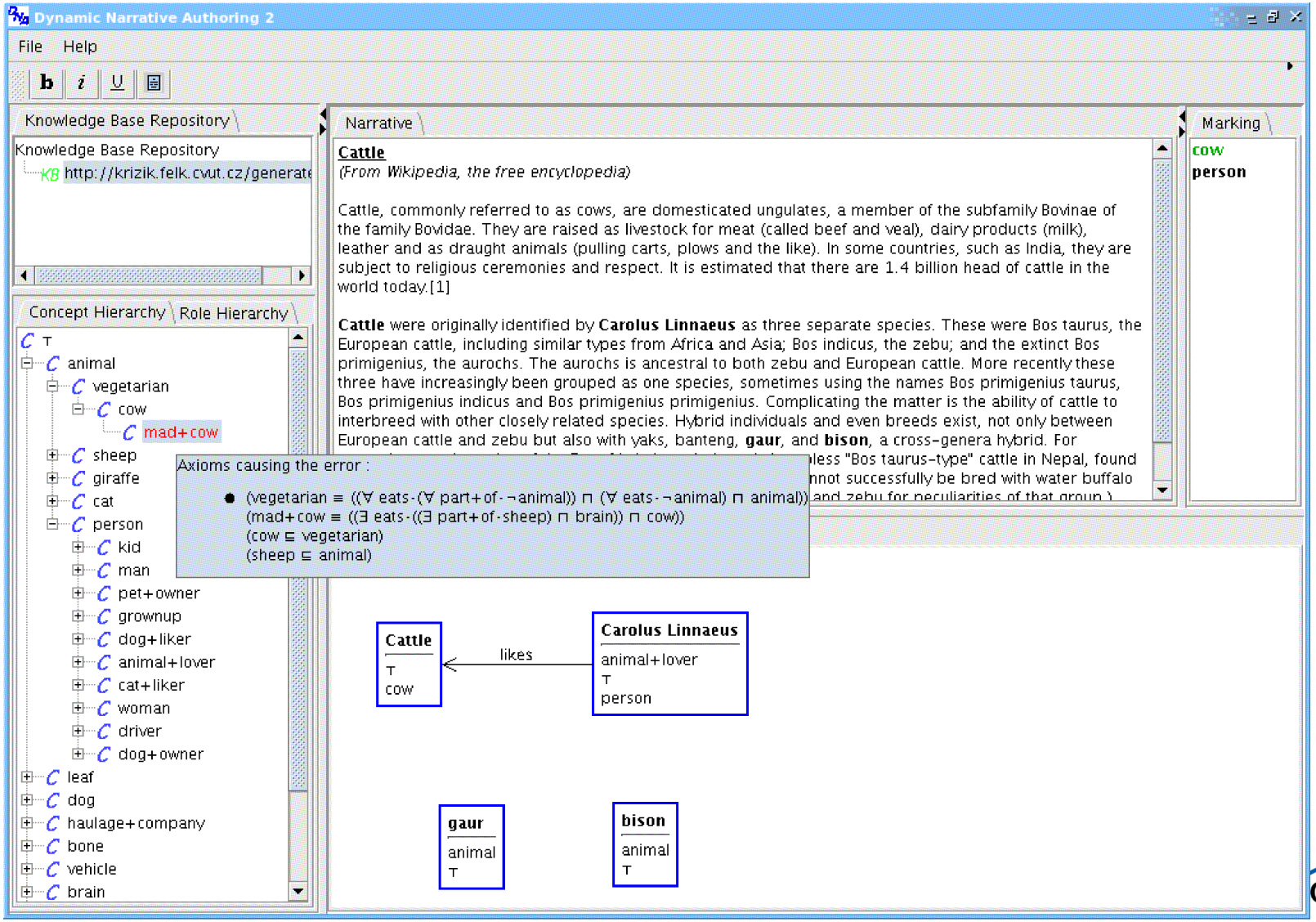
Motivace

- Představme si, že budujeme ontologii a inferenční stroj nám v určitém okamžiku sdělí, že je tato ontologie nekonzistentní. **Co s tím ?**
- **Můžeme začít ručně procházet** axiomy a koukat se, "kde je co špatně".
- Ale co když máme třeba již **stovky tisíc axiomů** ?

Motivace

- Představme si, že budujeme ontologii a inferenční stroj nám v určitém okamžiku sdělí, že je tato ontologie nekonzistentní. **Co s tím ?**
- **Můžeme začít ručně procházet** axiomy a koukat se, "kde je co špatně".
- Ale co když máme třeba již **stovky tisíc axiomů** ?
- Potřebujeme nějaký automatizovaný způsob, kterým každý takový problém "lokalizujeme" do množiny řádově menší.

DNA


Dynamic Narrative Authoring 2
File
Help
Knowledge Base Repository
Knowledge Base Repository
KB http://krizik.felk.cvut.cz/generate
Concept Hierarchy
Role Hierarchy
⊤
animal
vegetarian
cow
mad+cow
sheep
giraffe
cat
person
kid
man
pet+owner
grownup
dog+liker
animal+lover
cat+liker
woman
driver
dog+owner
leaf
dog
haulage+company
bone
vehicle
brain
Narrative
Cattle
(From Wikipedia, the free encyclopedia)
Cattle, commonly referred to as cows, are domesticated ungulates, a member of the subfamily Bovinae of the family Bovidae. They are raised as livestock for meat (called beef and veal), dairy products (milk), leather and as draught animals (pulling carts, plows and the like). In some countries, such as India, they are subject to religious ceremonies and respect. It is estimated that there are 1.4 billion head of cattle in the world today.[1]
Cattle were originally identified by Carolus Linnaeus as three separate species. These were Bos taurus, the European cattle, including similar types from Africa and Asia; Bos indicus, the zebu; and the extinct Bos primigenius, the aurochs. The aurochs is ancestral to both zebu and European cattle. More recently these three have increasingly been grouped as one species, sometimes using the names Bos primigenius taurus, Bos primigenius indicus and Bos primigenius primigenius. Complicating the matter is the ability of cattle to interbreed with other closely related species. Hybrid individuals and even breeds exist, not only between European cattle and zebu but also with yaks, banteng, gaur, and bison, a cross-genera hybrid. For
Marking
cow
person
Axioms causing the error :
(vegetarian ≡ ((∀ eats-(∀ part+of-¬animal)) ⊓ (∀ eats-¬animal) ⊓ animal))
(mad+cow ≡ ((∃ eats-((∃ part+of-sheep) ⊓ brain)) ⊓ cow))
(cow ⊑ vegetarian)
(sheep ⊑ animal)
Cattle
⊤
cow
likes
Carolus Linnaeus
animal+lover
⊤
person
gaur
animal
⊤
bison
animal
⊤

MUPS - příklad

Minimal unsatisfiability preserving subterminology (MUPS) je minimální množina axiomů, které zodpovídají za nesplnitelnost daného konceptu.

MUPS - příklad

Minimal unsatisfiability preserving subterminology (MUPS) je minimální množina axiomů, které zodpovídají za nesplnitelnost daného konceptu.

MUPS - příklad

Minimal unsatisfiability preserving subterminology (MUPS) je minimální množina axiomů, které zodpovídají za nesplnitelnost daného konceptu.

Příklad

Uvažujme ontologii $\mathcal{K}_5 = (\{\alpha_1, \alpha_2, \alpha_3\}, \emptyset)$

$$\begin{aligned}
\alpha_1 &: \mathit{Osoba} \sqsubseteq \exists \mathit{maRodice} \cdot (\mathit{Muz} \sqcap \mathit{Zena}) \sqcap \forall \mathit{maRodice} \cdot \neg \mathit{Osoba}, \\
\alpha_2 &: \mathit{Muz} \sqsubseteq \neg \mathit{Zena}, \\
\alpha_3 &: \mathit{Muz} \sqcup \mathit{Zena} \sqsubseteq \mathit{Osoba}.
\end{aligned}$$

Nesplnitelnost konceptu *Osoba* plyne nezávisle ze dvou množin (MUPS), a sice $\{\alpha_1, \alpha_2\}$ a $\{\alpha_1, \alpha_3\}$. Ověřte si !

MUPS

V současné době existují dva přístupy pro hledání všech MUPS pro daný koncept:

MUPS

V současné době existují dva přístupy pro hledání všech MUPS pro daný koncept:

black-box metody provádějí mnoho testů splnitelnosti s využitím existujícího reasoneru.

- ☺ flexibilní a snadno přepoužitelné pro jinou deskripční logiku
- ☹ časově náročné

MUPS

V současné době existují dva přístupy pro hledání všech MUPS pro daný koncept:

black-box metody provádějí mnoho testů splnitelnosti s využitím existujícího reasoneru.

- ☺ flexibilní a snadno přepoužitelné pro jinou deskripční logiku
- ☹ časově náročné

glass-box metody jsou plně zaintegrovány v existujícím reasoneru.

- ☺ efektivní
- ☹ špatně přepoužitelné pro jinou deskripční logiku.

Glass-box metody

- Pro jazyk $\mathcal{ALC}$ existuje úplný algoritmus, jehož myšlenka je tato:

Glass-box metody

- Pro jazyk $\mathcal{ALC}$ existuje úplný algoritmus, jehož myšlenka je tato:
  - tablový algoritmus pro $\mathcal{ALC}$ je rozšířen o pamět axiomů, které byly použity při konstrukci grafů zúplnění.

Glass-box metody

- Pro jazyk $\mathcal{ALC}$ existuje úplný algoritmus, jehož myšlenka je tato:
  - tablový algoritmus pro $\mathcal{ALC}$ je rozšířen o pamět axiomů, které byly použity při konstrukci grafů zúplnění.
  - pro každý sporný graf zúplnění lze obsah této paměti axiomů snadno převést právě na MUPS.

Glass-box metody

- Pro jazyk $\mathcal{ALC}$ existuje úplný algoritmus, jehož myšlenka je tato:
  - tablový algoritmus pro $\mathcal{ALC}$ je rozšířen o pamět axiomů, které byly použity při konstrukci grafů zúplnění.
  - pro každý sporný graf zúplnění lze obsah této paměti axiomů snadno převést právě na MUPS.
- Úplné glass-box metody bohužel neexistují pro jazyky OWL-DL a OWL2-DL. Pamět axiomů lze využít i pro tyto logiky, ovšem pouze jako předzpracování pro black-box algoritmy.

Black-box metody

Formulace úlohy

- máme *množinu axiomů* X daného jazyka a *reasoner* R pro daný jazyk. Cílem je nalézt vysvětlení pro :

Formulace úlohy

- máme *množinu axiomů* X daného jazyka a *reasoner* R pro daný jazyk. Cílem je nalézt vysvětlení pro :
  1. nesplnitelnost konceptu,

Formulace úlohy

- máme *množinu axiomů* X daného jazyka a *reasoner* R pro daný jazyk. Cílem je nalézt vysvětlení pro :
  1. nesplnitelnost konceptu,
  2. nekonzistenci znalostní báze

Formulace úlohy

- máme *množinu axiomů* X daného jazyka a *reasoner* R pro daný jazyk. Cílem je nalézt vysvětlení pro :
  1. nesplnitelnost konceptu,
  2. nekonzistenci znalostní báze
  3. platnost libovolného axiomu.

Formulace úlohy

- máme *množinu axiomů* X daného jazyka a *reasoner* R pro daný jazyk. Cílem je nalézt vysvětlení pro :
  1. nesplnitelnost konceptu,
  2. nekonzistenci znalostní báze
  3. platnost libovolného axiomu.
- lze ukázat [Kal06], že se můžeme BÚNO zabývat pouze (i).

Formulace úlohy

- máme *množinu axiomů* X daného jazyka a *reasoner* R pro daný jazyk. Cílem je nalézt vysvětlení pro :
  1. nesplnitelnost konceptu,
  2. nekonzistenci znalostní báze
  3. platnost libovolného axiomu.
- lze ukázat [Kal06], že se můžeme BÚNO zabývat pouze (i).
- označíme jako $MUPS(C, Y)$ (Minimal Unsatisfiability Preserving Subterminology) nějakou podmnožinu axiomů z $Y \subseteq X$ zodpovědných za nesplnitelnost konceptu C. Ta je konečná, neboť axiomů máme též konečně konečně mnoho.

Formulace úlohy (2)

- Vyšetřujeme nesplnitelnost konceptu C. Označíme

$$R(C, Y) = \left\{ \begin{array}{ll} \mathit{true} & \text{pokud } Y \not\models (C \sqsubseteq \bot) \\ \mathit{false} & \text{pokud } Y \models (C \sqsubseteq \bot)) \end{array} \right\}$$

Formulace úlohy (2)

- Vyšetřujeme nesplnitelnost konceptu C. Označíme

$$R(C, Y) = \left\{ \begin{array}{ll} \mathit{true} & \text{pokud } Y \not\models (C \sqsubseteq \bot) \\ \mathit{false} & \text{pokud } Y \models (C \sqsubseteq \bot)) \end{array} \right\}$$

- metod je mnoho [dSW03], my uvedeme pouze dvě:

Formulace úlohy (2)

- Vyšetřujeme nesplnitelnost konceptu C. Označíme

$$R(C, Y) = \left\{ \begin{array}{ll} \mathit{true} & \text{pokud } Y \not\models (C \sqsubseteq \bot) \\ \mathit{false} & \text{pokud } Y \models (C \sqsubseteq \bot)) \end{array} \right\}$$

- metod je mnoho [dSW03], my uvedeme pouze dvě:
  - algoritmy založené na CS-stromech

Formulace úlohy (2)

- Vyšetřujeme nesplnitelnost konceptu C. Označíme

$$R(C, Y) = \left\{ \begin{array}{ll} \mathit{true} & \text{pokud } Y \not\models (C \sqsubseteq \bot) \\ \mathit{false} & \text{pokud } Y \models (C \sqsubseteq \bot)) \end{array} \right\}$$

- metod je mnoho [dSW03], my uvedeme pouze dvě:
  - algoritmy založené na CS-stromech
  - Hledání jednoho MUPS[Kal06] + reiterův algoritmus [Rei87].

Algoritmy založené na CS-stromech

- Kdybychom chtěli hledat MUPS slepě, testovali bychom pro každou podmnožiny axiomů z $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, zda nezpůsobuje nesplnitelnost a poté bychom uživateli jako MUPS nabídli množiny nejmenší vzhledem k inkluzi. *Conflict-set stromy (CS-stromy)* systematizují prohledávání všech těchto podmnožin z $\mathcal{K}$. Hlavní myšlenka :

  > *Pokud jsme nalezli množinu axiomů, vzhledem ke které je C splnitelný, potom nemusíme vyšetřovat její podmnožiny – pro ně musí být C rovněž splnitelný.*

- CS-strom je reprezentací stavového prostoru, kde každý stav s má tvar (D, P), kde
  - D je množina axiomů, které *nutně musí být* ve všech MUPSech nalezených v podstromu s kořenem v s.
  - P je množina axiomů, které *mohou být* v některých MUPS nalezených v podstromu s kořenem ve vrcholu s.

Prohledávání CS-stromu

Následující algorimus je exponenciální v počtu aplikací tablového algoritmu.

1. (Inicializace) Kořenem stromu je počáteční stav $s_0 = (\emptyset, \mathcal{K})$, který reprezentuje fakt, že apriori neznáme axiom, které musí být nutně v každém hledaném MUPS ($D_{s_0} = \emptyset$), ale mohou tam být všechny ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Dále položme $Z = (s_0)$ a $R = \emptyset$

Prohledávání CS-stromu

Následující algorimus je exponenciální v počtu aplikací tablového algoritmu.

1. (Inicializace) Kořenem stromu je počáteční stav $s_0 = (\emptyset, \mathcal{K})$, který reprezentuje fakt, že apriori neznáme axiom, které musí být nutně v každém hledaném MUPS ($D_{s_0} = \emptyset$), ale mohou tam být všechny ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Dále položme $Z = (s_0)$ a $R = \emptyset$
2. (Prohledávání do hloubky) Je-li Z prázdný, ukončíme prohledávání. Jinak vyjmeme první prvek s ze Z.

Prohledávání CS-stromu

Následující algorimus je exponenciální v počtu aplikací tablového algoritmu.

1. (Inicializace) Kořenem stromu je počáteční stav $s_0 = (\emptyset, \mathcal{K})$, který reprezentuje fakt, že apriori neznáme axiom, které musí být nutně v každém hledaném MUPS ($D_{s_0} = \emptyset$), ale mohou tam být všechny ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Dále položme $Z = (s_0)$ a $R = \emptyset$
2. (Prohledávání do hloubky) Je-li Z prázdný, ukončíme prohledávání. Jinak vyjmeme první prvek s ze Z.
3. (Test) Je-li $R(C, D_s \cup P_s) = \mathit{true}$, potom ani z žádné podmnožiny $D_s \cup P_s$ nemůže nesplnitelnost plynout – pokračujeme bodem 2.

Prohledávání CS-stromu

Následující algorimus je exponenciální v počtu aplikací tablového algoritmu.

1. (Inicializace) Kořenem stromu je počáteční stav $s_0 = (\emptyset, \mathcal{K})$, který reprezentuje fakt, že apriori neznáme axiom, které musí být nutně v každém hledaném MUPS ($D_{s_0} = \emptyset$), ale mohou tam být všechny ($P_{s_0} = \mathcal{T} \cup \mathcal{A}$). Dále položme $Z = (s_0)$ a $R = \emptyset$
2. (Prohledávání do hloubky) Je-li Z prázdný, ukončíme prohledávání. Jinak vyjmeme první prvek s ze Z.
3. (Test) Je-li $R(C, D_s \cup P_s) = \mathit{true}$, potom ani z žádné podmnožiny $D_s \cup P_s$ nemůže nesplnitelnost plynout – pokračujeme bodem 2.
4. (Nalezení nesplnitelné množiny) Přidáme $D_s \cup P_s$ do R a naopak odstraníme z R všechny $s' \in R$ takové, že $D_s \cup P_s \subseteq s'$. Pro $P_s = \alpha_1, \ldots, \alpha_N$ přidáme na začátek zásobníku Z nový vrchol $(D_s \cup \{\alpha_1, \ldots, \alpha_{i-1}\}, P_s \setminus \{\alpha_1, \ldots, \alpha_i\})$. Pokračujeme bodem 2.

Prohledávání CS-stromu (2)

- Korektnost : Podstatný je bod 4 – zde pokrýváme všechny možnosti. Vždy platí, že $D_s \cup P_s$ se liší právě o jeden prvek oproti $D'_s \cup P'_s$, kde s' je potomek s.

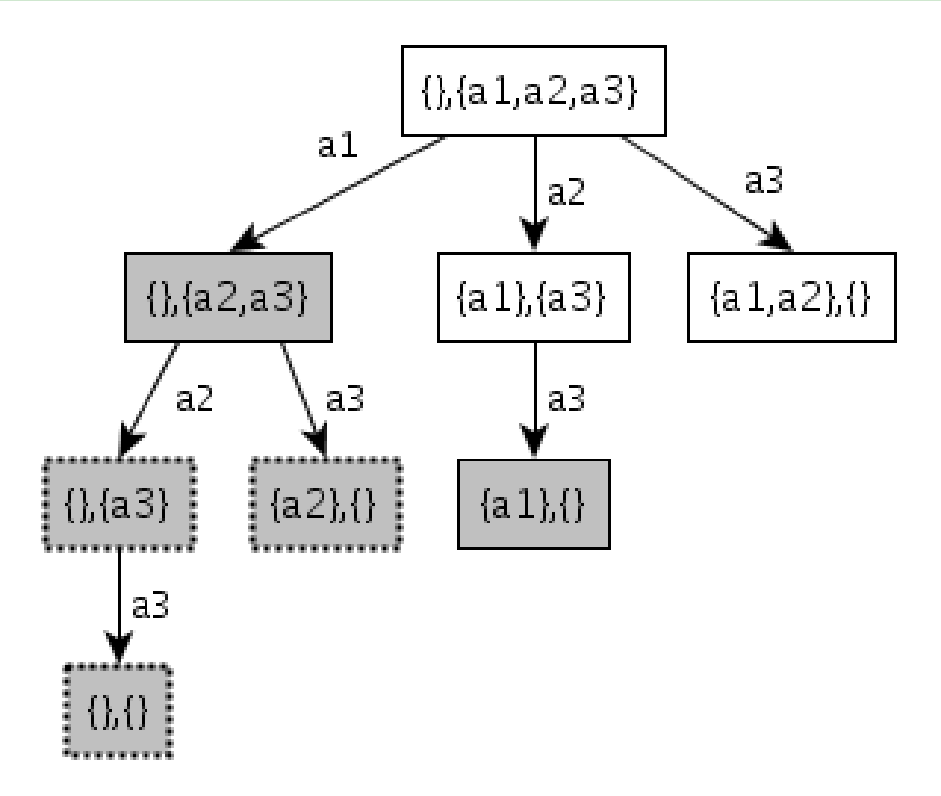
Prohledávání CS-stromu (2)

- Korektnost : Podstatný je bod 4 – zde pokrýváme všechny možnosti. Vždy platí, že $D_s \cup P_s$ se liší právě o jeden prvek oproti $D'_s \cup P'_s$, kde s' je potomek s.
- Konečnost : Množina $D_s \cup P_s$ je na začátku konečná a zmenšuje se s hloubkou stromu. Navíc v bodě 4 generujeme pouze konečný počet vrcholů.

Prohledávání CS-stromu – příklad

Příklad

CS-tree pro $\mathcal{K}_5$ by vypadal takto :



V šedých vrcholech je $R(C, D \cup P) = \mathit{true}$. Vrcholy ohraničené tečkovanou čarou ani nemusely být generovány.

singleMUPS(C, Y) – nalezení jednoho MUPS

Následující algorimus je polynomiální v počtu aplikací tablového algoritmu – efektivita je tedy zásadně podmíněna efektivitou tablového algoritmu samotného.

1. (Inicializace) Označíme $S = \emptyset$, $K = \emptyset$

singleMUPS(C, Y) – nalezení jednoho MUPS

Následující algorimus je polynomiální v počtu aplikací tablového algoritmu – efektivita je tedy zásadně podmíněna efektivitou tablového algoritmu samotného.

1. (Inicializace) Označíme $S = \emptyset$, $K = \emptyset$
2. (Nalezení nadmnožiny MUPS) Dokud je $R(C, S) = \mathit{false}$, potom $S = S \cup \{\alpha\}$ pro nějaké $\alpha \in Y \setminus S$.

singleMUPS(C, Y) – nalezení jednoho MUPS

Následující algorimus je polynomiální v počtu aplikací tablového algoritmu – efektivita je tedy zásadně podmíněna efektivitou tablového algoritmu samotného.

1. (Inicializace) Označíme $S = \emptyset$, $K = \emptyset$
2. (Nalezení nadmnožiny MUPS) Dokud je $R(C, S) = \mathit{false}$, potom $S = S \cup \{\alpha\}$ pro nějaké $\alpha \in Y \setminus S$.
3. (Prořezání nalezené množiny) Pro každý $\alpha \in S \setminus K$ vyhodnoť $R(C, S \setminus \{\alpha\})$. Je-li výsledek *false*, potom $K = K \cup \{\alpha\}$. Výsledná množina K je rovna právě hledanému MUPS.

Nalezení jednoho $MUPS(C, Y)$ - příklad

Příklad

Sledujme běh výpočtu $singleMUPS(Osoba, \mathcal{K}_5)$.

Nalezení jednoho $MUPS(C, Y)$ - příklad

Příklad

Sledujme běh výpočtu $singleMUPS(Osoba, \mathcal{K}_5)$.

1.FÁZE :

$$\mathcal{T}_5 \cup \mathcal{A}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \qquad R(C, \{\alpha_1\}) = true$$
$$S = \{\alpha_1\}$$

Nalezení jednoho $MUPS(C, Y)$ - příklad

Příklad

Sledujme běh výpočtu $singleMUPS(Osoba, \mathcal{K}_5)$.

1.FÁZE :

$$\mathcal{T}_5 \cup \mathcal{A}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \qquad R(C, \{\alpha_1, \alpha_2\}) = false$$
$$S = \{\alpha_1, \alpha_2\}$$

Nalezení jednoho $MUPS(C, Y)$ - příklad

Příklad

Sledujme běh výpočtu $singleMUPS(Osoba, \mathcal{K}_5)$.

1.FÁZE :

$\mathcal{T}_5 \cup \mathcal{A}_5 = \{\alpha_1, \alpha_2, \alpha_3\}$ $\quad R(C, \{\alpha_1, \alpha_2\}) = false$

$S = \{\alpha_1, \alpha_2\}$

2.FÁZE :

$S = \{\alpha_1, \alpha_2\}$ $\quad R(C, \{\alpha_1, \alpha_2\} - \{\alpha_1\}) = true$

$K = \{\alpha_1\}$

Nalezení jednoho $MUPS(C, Y)$ - příklad

Příklad

Sledujme běh výpočtu $singleMUPS(Osoba, \mathcal{K}_5)$.

1.FÁZE :

$$\mathcal{T}_5 \cup \mathcal{A}_5 = \{\alpha_1, \alpha_2, \alpha_3\} \qquad R(C, \{\alpha_1, \alpha_2\}) = false$$
$$S = \{\alpha_1, \alpha_2\}$$

2.FÁZE :

$$S = \{\alpha_1, \alpha_2\} \qquad R(C, \{\alpha_1, \alpha_2\} - \{\alpha_2\}) = true$$
$$K = \{\alpha_1, \alpha_2\}$$

Nalezení všech MUPS - Reiterův algoritmus

- Máme k dispozici algoritmus *singleMUPS*(C, Y), který nám vygeneruje jeden (libovolný) MUPS. Reiterův algoritmus vytváří tzv. "Hitting Set Tree", jehož uzly tvoří dvojice (K_i, M_i), kde K_i je ontologie vzniklá z $\mathcal{K}$ odstraněním některých axiomů a $M_i = \mathit{singleMUPS}(K_i, M_i)$, nebo $M_i =$ "*SAT*", je-li vyšetřovaný koncept splnitelný v K_i..

Nalezení všech MUPS - Reiterův algoritmus

- Máme k dispozici algoritmus $singleMUPS(C, Y)$, který nám vygeneruje jeden (libovolný) MUPS. Reiterův algoritmus vytváří tzv. "Hitting Set Tree", jehož uzly tvoří dvojice (K_i, M_i), kde K_i je ontologie vzniklá z $\mathcal{K}$ odstraněním některých axiomů a $M_i = singleMUPS(K_i, M_i)$, nebo $M_i =$ "SAT", je-li vyšetřovaný koncept splnitelný v K_i..
- Cesty od kořene k listům tvoří tzv. *diagnózy* (tedy množiny axiomů, které když odstraníme z KB, stane se vyšetřovaný koncept splnitelný). Tento strom se prohledává do hloubky.

Nalezení všech MUPS - Reiterův algoritmus

- Máme k dispozici algoritmus $singleMUPS(C, Y)$, který nám vygeneruje jeden (libovolný) MUPS. Reiterův algoritmus vytváří tzv. "Hitting Set Tree", jehož uzly tvoří dvojice (K_i, M_i), kde K_i je ontologie vzniklá z $\mathcal{K}$ odstraněním některých axiomů a $M_i = singleMUPS(K_i, M_i)$, nebo $M_i =$ "SAT", je-li vyšetřovaný koncept splnitelný v K_i..
- Cesty od kořene k listům tvoří tzv. *diagnózy* (tedy množiny axiomů, které když odstraníme z KB, stane se vyšetřovaný koncept splnitelný). Tento strom se prohledává do hloubky.
- Počet volání algoritmu $singleMUPS(C, Y)$ je omezen faktoriálem (a tedy exponenciálou) počtu axiomů na vstupu. Proč ?

Nalezení všech MUPS - Reiterův algoritmus (2)

1 (Inicializace) Nalezneme jeden MUPS pro C v $\mathcal{K}$, ten se staned kořenem $s_0 = (\mathcal{K}, \mathit{singleMUPS}(C, \mathcal{K}))$ stromu. Dále položíme $Z = (s_0)$.

Nalezení všech MUPS - Reiterův algoritmus (2)

1. (Inicializace) Nalezneme jeden MUPS pro C v $\mathcal{K}$, ten se staned kořenem $s_0 = (\mathcal{K}, \mathit{singleMUPS}(C, \mathcal{K}))$ stromu. Dále položíme $Z = (s_0)$.
2. (Prohledávání do hloubky) Je-li Z prázdný, ukončíme prohledávání. Jinak vyjmi první prvek ze Z a označ jej $s_i = (K_i, M_i)$.

Nalezení všech MUPS - Reiterův algoritmus (2)

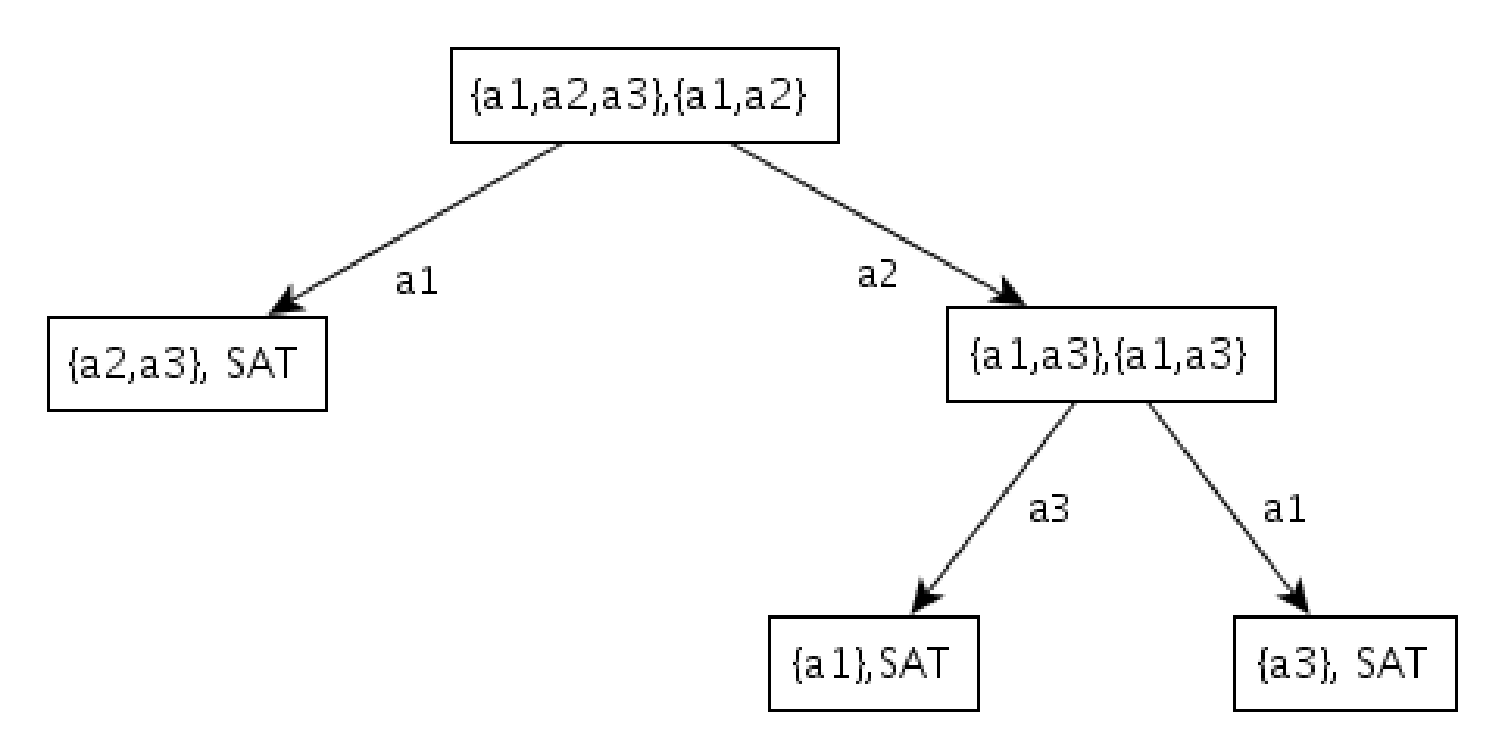
1. (Inicializace) Nalezneme jeden MUPS pro C v $\mathcal{K}$, ten se staned kořenem $s_0 = (\mathcal{K}, singleMUPS(C, \mathcal{K}))$ stromu. Dále položíme $Z = (s_0)$.
2. (Prohledávání do hloubky) Je-li Z prázdný, ukončíme prohledávání. Jinak vyjmi první prvek ze Z a označ jej $s_i = (K_i, M_i)$.
3. (Test) Je-li $M_i =$ "SAT", potom pokračujeme prohledáváním v bodě 2.

Nalezení všech MUPS - Reiterův algoritmus (2)

1. (Inicializace) Nalezneme jeden MUPS pro C v $\mathcal{K}$, ten se staned kořenem $s_0 = (\mathcal{K}, \mathit{singleMUPS}(C, \mathcal{K}))$ stromu. Dále položíme $Z = (s_0)$.
2. (Prohledávání do hloubky) Je-li Z prázdný, ukončíme prohledávání. Jinak vyjmi první prvek ze Z a označ jej $s_i = (K_i, M_i)$.
3. (Test) Je-li $M_i =$ "*SAT*", potom pokračujeme prohledáváním v bodě 2.
4. (Rozklad) Pro každý $\alpha \in M_i$ vložíme do Z nový vrchol ohodnocený $(K_i \setminus \{\alpha\}, \mathit{singleMUPS}(K_i \setminus \{\alpha\}, C))$. Pokračujeme prohledáváním v bodě 2.

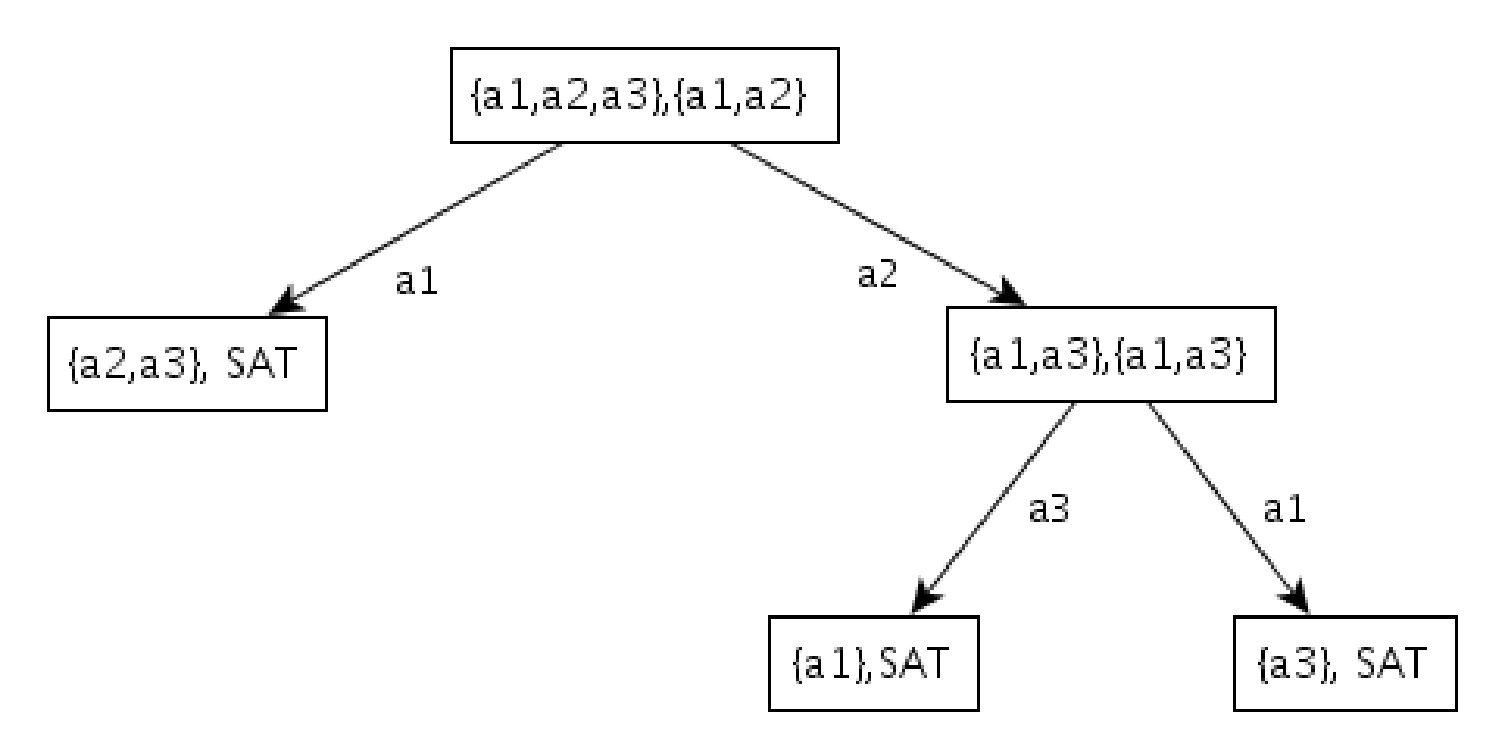
Nalezení všech MUPS - Reiterův algoritmus, příklad

Příklad – pokračování



Nalezení všech MUPS - Reiterův algoritmus, příklad

Příklad – pokračování



Výsledkem jsou dva MUPSy $\{\alpha_1, \alpha_2\}$ a $\{\alpha_1, \alpha_3\}$. "Zdarma" jsme získali též minimální diagnózy $\{\alpha_1\}$ a $\{\alpha_2, \alpha_3\}$.

Vysvětlování modelovacích chyb – shrnutí

- nejoblíbenější přístup je **nalezení množiny axiomů zodpovědných za danou modelovací chybu**, např. MUPS.
- black-box vs. glass box metody
- jiné možnosti - např. inkrementální přístup [dSW03].
- cílem je nalézt především diagnózy – co je třeba udělat, aby daný modelovací problém zmizel.
- výše uvedená metoda je velmi univerzální - lze ji použít na mnoho jiných problémů, které s deskripční logikou vůbec nesouvisejí.