Obecná rezoluce (s proměnnými)

- mohli bychom generovat základní instance klauzulí a jejich rezolventy
- ale tím bychom toho prohledávali příliš mnoho

Proto:

- Pokud necháme proměnné neurčené tak dlouho, jak jen to půjde, prohledáváme vlastně pro mnoho základních klauzulí naráz.
- Když si literály v různých klauzulích nebudou odpovídat, zkusíme je **unifikovat**.

Definition (Substituce)

Substituce je konečná množina tvaru $\{t_1/v_1, \ldots, t_k/v_k\}$, kde pro každé $i = 1, \ldots, k$ je v_i proměnná a t_i term různý od v_i a pro $i \neq j$ je $v_i \neq v_j$.
Jsou–li $t_1, \ldots, t_k$ základní termy, mluvíme o **základní substituci**.
Prázdnou substituci označíme ϵ.

Příklady substitucí:

$$\theta_1 = \{[1,2]/x, y/z\}$$
$$\theta_2 = \{u(u(a))/x, u(r(z))/y\}$$
$$\theta_3 = \{x/y, y/x\}$$

Definition (Instance klauzule, základní instance)

Je–li $\theta = \{t_1/v_1, \ldots, t_k/v_k\}$ substituce a E výraz (term či klauzule), označíme $E\theta$ výraz vzniklý z E nahrazením všech výskytů v_i termem t_i, $i = 1, \ldots, k$. Výraz $E\theta = E_{v_1,\ldots,v_k}[t_1, \ldots, t_k]$ se nazývá **instancí klauzule** (termu) E. Instanci, která neobsahuje žádné proměnné, nazveme **základní** instancí.

Example

Pro formuli: $E \equiv \neg wumpus(x) \vee smelly(y) \vee \neg adjacent(x, y)$ a substituce $\theta_1 = \{[1, 2]/x, y/z\}$ $\theta_2 = \{u(u(a))/x, u(r(z))/y\}$ $\theta_3 = \{x/y, y/x\}$ dostaneme:

$$
\begin{aligned}
E\theta_1 &\equiv \neg wumpus([1, 2]) \vee smelly(y) \vee \neg adjacent([1, 2], y) \\
E\theta_2 &\equiv \neg wumpus(u(u(a))) \vee smelly(u(r(z))) \vee \neg adjacent(u(u(a)), u(\\
E\theta_3 &\equiv \neg wumpus(y) \vee smelly(x) \vee \neg adjacent(y, x)
\end{aligned}
$$

Definition (Složení substitucí)

Jsou–li $\theta = \{t_1/v_1, \ldots, t_k/v_k\}$ a $\lambda = \{u_1/x_1, \ldots, u_k/x_l\}$ substituce, pak **složením** θ a λ je substituce $\theta \circ \lambda$, která vznikne z množiny $\{t_1\lambda/v_1, \ldots, t_k\lambda/v_k, u_1/x_1, \ldots, u_l/x_l\}$ škrtnutím všech členů $t_j\lambda/v_j$ pro každé $t_j\lambda = v_j$ a všech členů u_i/x_i takových, že x_i je jedna z proměnných $v_1, \ldots v_k$.

Example

$\theta = \{f(y)/x, z/y\}$, $\lambda = \{a/x, b/y, y/z\}$
Množina pro škrtání je $\{f(b)/x, y/y, a/x, b/y, y/z\}$
tedy $\theta \circ \lambda = \{f(b)/x, y/z\}$.
Snadno se ukáže, že $(\theta \circ \lambda) \circ \mu = \theta \circ (\lambda \circ \mu)$ a $\epsilon \circ \theta = \theta \circ \epsilon$ pro každé θ, λ, μ.

Definition (Unifikace)

Substituce θ je **unifikační substituce** (unifikace) pro množinu výrazů $\{E_1, \ldots, E_k\}$, jestliže $E_1\theta \equiv E_2\theta \equiv \ldots \equiv E_k\theta$ (tj. termy či klauzule jsou identické).

Definition (Nejobecnější unifikace)

Unifikační substituce σ pro $\{E_1, \ldots, E_k\}$ se nazývá **nejobecnější unifikace**, jestliže pro každou unifikaci θ množiny $\{E_1, \ldots, E_k\}$ existuje substituce λ tak, že $\theta = \sigma \circ \lambda$.

Example

Substituce $\theta = \{a/x, f(b)/y\}$ je unifikace pro množinu $\{p(a, y), p(x, f(b))\}$.
Substituce $\theta = \{a/x, f(b)/y, f(b)/z\}$ je unifikace pro množinu $\{p(a, y), p(x, z)\}$, ale nejobecnější unifikace této množiny je $\sigma = \{a/x, z/y\}$ (i $\sigma_1 = \{a/x, y/z\}$).
Chceme–li unifikovat množinu výrazů, musíme nejprve najít, v čem se od sebe liší a pak se snažit vhodnou substitucí tento rozdíl odstranit.

Definition (Rozdílová množina)

Mějme neprázdnou množinu výrazů E. Najdeme první (počítáno od leva) místo, na kterém nemají všechny výrazy z E stejný symbol. Všechny podvýrazy prvků v E, začínající symbolem napsaným na tomto místě, tvoří tzv. rozdílovou množinu pro E.

Example

Rozdílová množina pro $\{p(x, f(y, z)), p(x, a), p(x, g(h(k(x))))\}$ je množina $\{f(y, z), a, g(h(k(x)))\}$.

Unifikační algoritmus

1. Polož $k = 0, E_k = E, \sigma_k = \epsilon$
2. Je-li E_k jednoprvková množina, STOP, σ_k je nejobecnější unifikátor E.
   Není-li E_k jednoprvková, najdi množinu nesouhlasu D_k pro E_k.
3. Jestliže v D_k existuje proměnná v_k a term t_k tak, že t_k neobsahuje v_k, jdi do (4). Jinak STOP - E není unifikovatelné.
4. Utvoř $\sigma_{k+1} = \sigma_k \circ \{t_k/v_k\}, E_{k+1} = E_k\sigma_{k+1}$.
5. Polož $k \leftarrow k + 1$ a jdi do (2).

Pozn. V kroku (4) je samozřejmě $E_{k+1} = E\sigma_{k+1}$.

Unifikační algoritmus – příklad 1

$E = \{p(a, x, f(g(y))),\ p(z, f(z), f(u))\}$.

1. $\sigma_0 = \epsilon, E_0 = E$, E není jednoprvková, $D_0 = \{a, z\}$.
2. V D_0 je proměnná z, která není obsažena v termu a.
3. $\sigma_1 = \sigma_0 \circ \{a/z\} = \epsilon \circ \{a/z\} = \{a/z\}$.
   $E_1 = E_0\{a/z\} = \{p(a, x, f(g(y))), p(a, f(a), f(u))\}$.
4. E_1 není jednoprvková, $D_1 = \{x, f(a)\}$
5. V D_1 najdeme proměnnou x a term $f(a)$.
6. $\sigma_2 = \sigma_1 \circ \{f(a)/x\} = \{a/z\} \circ \{f(a)/x\} = \{a/z, f(a)/x\}$.
   $E_2 = E_1\{f(a)/x\} = \{p(a, f(a), f(g(y))), p(a, f(a), f(u))\}$.
7. E_2 není jednoprvková, $D_2 = \{g(y), u\}$.
8. V D_2 najdeme u a $g(y)$.
9. $\sigma_3 = \sigma_2 \circ \{g(y)/u\} = \{a/z, f(a)/x\} \circ \{g(y)/u\} =$
   $\{a/z, f(a)/x, g(y)/u\}$.
10. E_4 je jednoprvková, tedy $\sigma_3\{a/z, f(a)/x, g(y)/u\}$ je nejobecnější unifikátor E.

Unifikační algoritmus – příklad 2

$E = \{q(f(a), g(x)), q(y, y)\}$.

1. $\sigma_0 = \epsilon, E_0 = E$, E není jednoprvková, $D_0 = \{f(a), y\}$.
2. V D_0 je proměnná y, která není obsažena v termu $f(a)$.
3. $\sigma_1 = \sigma_0 \circ \{f(a)/y\} = \epsilon \circ \{f(a)/y\} = \{f(a)/y\}$.
   $E_1 = E_0\{f(a)/y\} = \{q(f(a), g(x)), q(f(a), f(a))\}$.
4. E_1 není jednoprvková, $D_1 = \{g(x), f(a)\}$. D_1 však neobsahuje žádnou proměnnou, E není unifikovatelná.

Theorem (Unifikační věta)

Je-li E konečná neprázdná unifikovatelná množina výrazů, pak unifikační algoritmus se vstupem E se zastaví v kroku (2) a výstup σ_k je nejobecnější unifikátor pro množinu E.

Důkaz:

- Algoritmus zastaví po konečném počtu kroků, neboť se v každém kroku buď zvýší počet písmen shody nebo sníží počet prvků rozdílové množiny.
- Pokud existuje unifikační substituce, pak se algoritmus nemůže zastavit v kroku 4.
- Že algoritmus vydá nejobecnější unifikaci se dokazuje indukcí pro každý krok unifikačního algoritmu.

pokračování důkazu unifikační věty

Nechť θ je unifikátor pro E. Indukcí podle k se dokáže, že existuje substituce λ_k tak, že $\theta = \sigma_k \circ \lambda_k$.
Položíme počáteční $\lambda_0 = \theta$.
Je-li $E\sigma_k$ jednoprvková, končíme v kroku (2), jinak máme rozdílovou množinu D_k. Protože z indukčního předpokladu $\theta = \sigma_k \circ \lambda_k$ je unifikace, tedy λ_k musí unifikovat D_k. Protože je D_k rozdílová množina, musí obsahovat rozdíl, tj. obsahuje proměnnou (označíme jí v_k) a term t_k různý od v_k. Tento term nemůže obsahovat v_k, jinak by neexistovala žádná unifikace.
Položme $\lambda_{k+1} = \lambda_k - \{t_k\lambda_k/v_k\}$. Protože se v_k nevyskytuje v t_k je: $t_k\lambda_{k+1} = t_k(\lambda_k - \{t_k\lambda_k/v_k\}) = t_k\lambda_k$.

pokračování důkazu unifikační věty

Odtud:

$$\begin{aligned}
\{t_k/v_k\} \circ \lambda_{k+1} &= \{t_k\lambda_{k+1}/v_k\} \cup \lambda_{k+1} \\
&= \{t_k\lambda_k/v_k\} \cup \lambda_{k+1} \\
&= \{t_k\lambda_k/v_k\} \cup (\lambda_k - \{t_k\lambda_k/v_k\}) \\
&= \lambda_k
\end{aligned}$$

Je tedy: $\theta = \sigma_k \circ \lambda_k = \sigma_k \circ \{t_k/v_k\} \circ \lambda_{k+1} = \sigma_{k+1} \circ \lambda_{k+1}$.
Unifikační algoritmus se musí zastavit v bodě (2), pak je σ_k unifikace a indukcí jsme dokázali, že je obecnější než kterákoli unifikace θ.

Definition (Binární rezolventa)

Nechť C_1 a C_2 jsou klauzule nemající společné proměnné, L_1 literál z C_1, L_2 literál z C_2. Mají-li L_1 a $\neg L_2$ nejobecnější unifikátor σ, nazveme klauzuli $(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$ **binární resolventou** C_1 a C_2.

Definition (faktor klauzule)

Jestliže pro alespoň dva literály klauzule C (stejného "znaménka") existuje nejobecnější unifikátor σ, nazveme $C\sigma$ **faktorem** C. Je-li $C\sigma$ klauzule o jediném literálu, říkáme jí jednotkový faktor.

Definition (resolventa = obecné rezoluční pravidlo)

Resolventou klauzulí C_1 a C_2 nazveme:

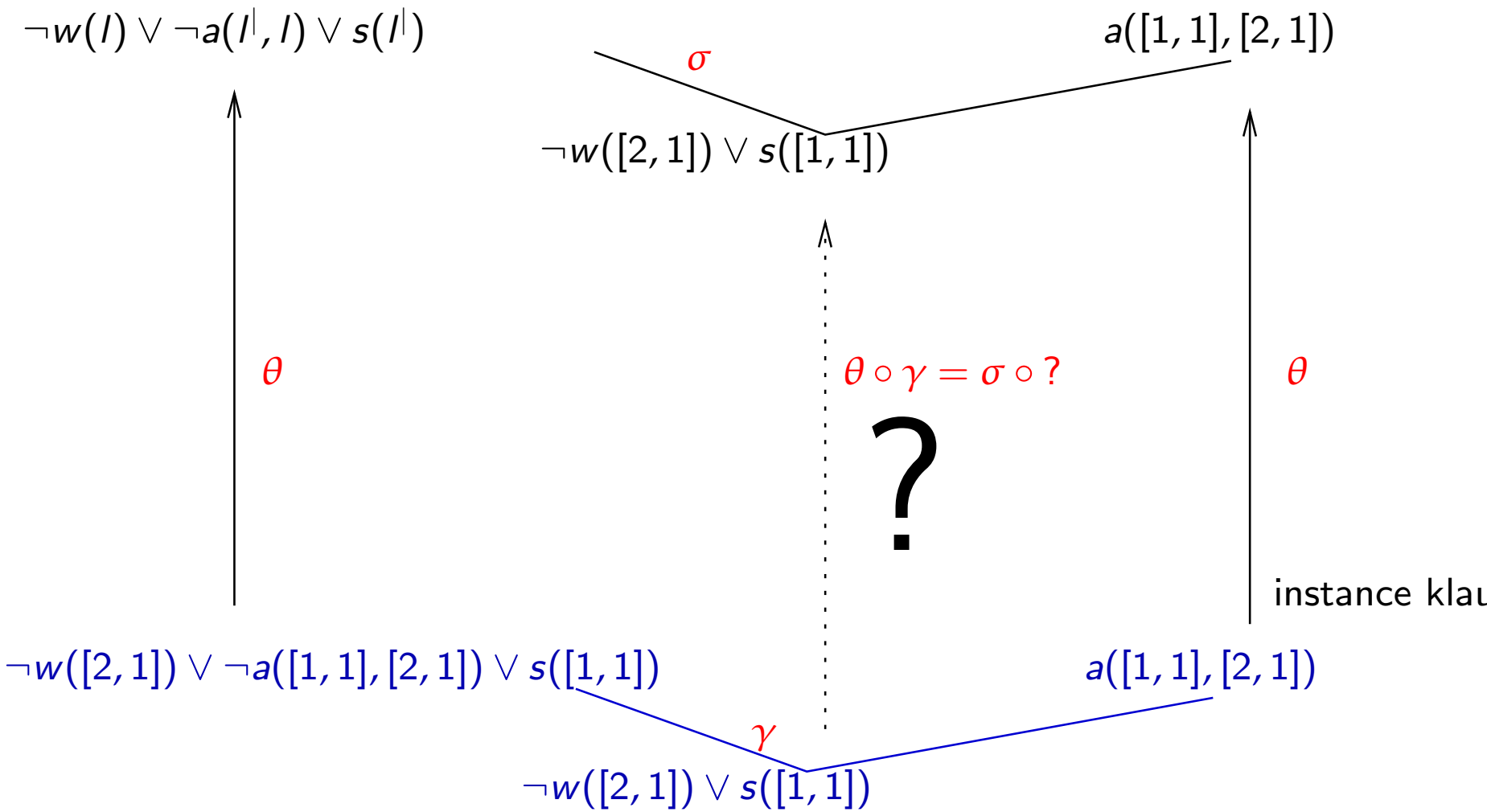
1. binární resolventu C_1 a C_2
2. binární resolventu C_1 a faktoru C_2
3. binární resolventu faktoru C_1 a C_2
4. binární resolventu faktoru C_1 a faktoru C_2

Resolventa je logický důsledek. Ale: najdeme vždy odvození $\square$?

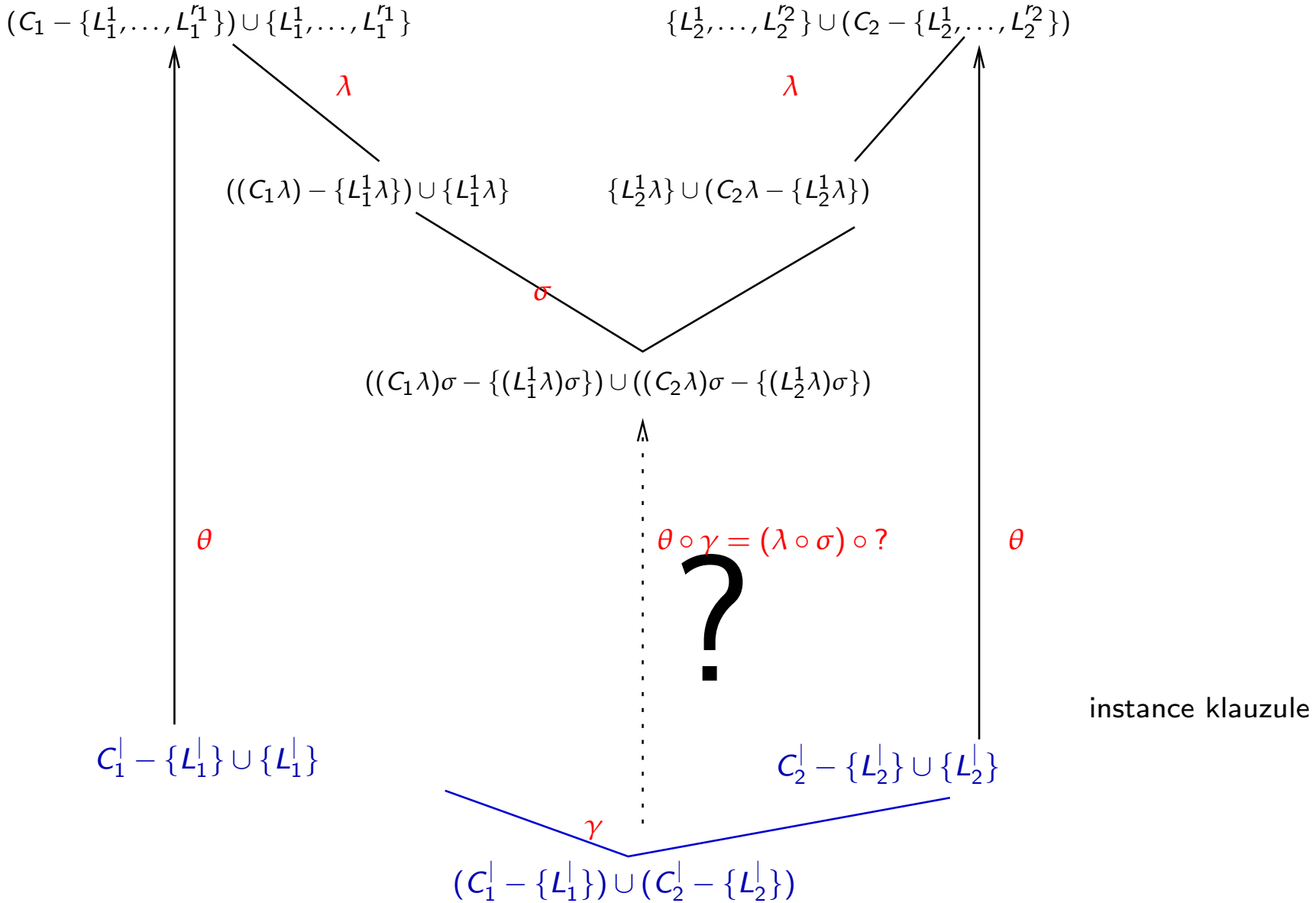
Lemma (Lifting lemma)

Nechť $C_1^|$ a $C_2^|$ jsou instance klauzulí C_1 a C_2, nechť $C^|$ je resolenta $C_1^|$ a $C_2^|$. Pak existuje resolventa C klauzulí C_1 a C_2 taková, že $C^|$ je instance C.

Lifting lemma



Lifting lemma



instance klauzule

Lifting lemma

Je-li zapotřebí, přejmenujeme proměnné v C_1 a C_2 tak, aby C_1 a C_2 neměly společné proměnné.
Nechť $L_1^|$ a $L_2^|$ jsou literály z $C_1^|$ a $C_2^|$, pomocí nichž se vytvořila resolventa $C^|$, nechť γ je nejobecnější unifikátor $L_1^|$ a $\neg L_2^|$. Tedy $C^| = (C_1^| - L_1^|\gamma) \cup (C_2^| - L_2^|\gamma)$.
Protože $C_1^|$ a $C_2^|$ jsou instance C_1 a C_2, existuje substituce θ tak, že $C_1^| = C_1\theta$ a $C_2^| = C_2\theta$. Nechť $L_i^1, \ldots L_i^{r_i}$ jsou literály z C_i odpovídající $L_i^|$ (pro $i = 1, 2$), které se ztotožnily po provedení substituce θ, tj. $L_i^1\theta = \ldots = L_i^{r_i}\theta = L_i^|$ $(i = 1, 2)$.

Lifting lemma

Je-li $r_i > 1$, vezmeme nejobecnější unifikátor λ_i pro $\{L_i^1, \ldots, L_i^{r_i}\}$ a položíme $L_i = L_i^1\lambda_i$. Pak L_i je literál ve faktoru $C_i\lambda_i$ složky C_i.
Je-li $r_i = 1$, položíme $\lambda_i = \epsilon$ a $L_i = L_i^1\lambda_i$.
Dále položíme $\lambda = \lambda_1 \cup \lambda_2$.
Zřejmě je $L_i^|$ instancí L_i. Protože $L_1^|$ a $\neg L_2^|$ jsou unifikovatelné, jsou i L_1 a $\neg L_2$ unifikovatelné, nechť σ je nejobecnější unifikátor L_1 a $\neg L_2$. Položme

$$\begin{aligned} C &= ((C_1\lambda)\sigma - L_1\sigma) \cup ((C_2\lambda)\sigma - L_2\sigma) \\ &= ((C_1\lambda)\sigma - (\{L_1^1, \ldots, L_1^{r_1}\}\lambda)\sigma) \cup ((C_2\lambda)\sigma - (\{L_2^1, \ldots, L_2^{r_2}\}\lambda)\sigma) \\ &= (C_1(\lambda \circ \sigma) - \{L_1^1, \ldots, L_1^{r_1}\}(\lambda \circ \sigma)) \cup (C_2(\lambda \circ \sigma) - \{L_2^1, \ldots, L_2^{r_2}\}(\lambda \end{aligned}$$

C je rezolventa C_1 a C_2. $C^|$ je instancí C, neboť

Lifting lemma

$$
\begin{aligned}
C^{|} &= (C_1^{|}\gamma - L_1^{|}\gamma) \cup (C_2^{|}\gamma - L_2^{|}\gamma) \\
&= ((C_1\theta)\gamma - (\{L_1^1, \ldots, L_1^{r_1}\}\theta)\gamma) \cup ((C_2\theta)\gamma - (\{L_2^1, \ldots, L_2^{r_2}\}\theta)\gamma) \\
&= (C_1(\theta \circ \gamma) - \{L_1^1, \ldots, L_1^{r_1}\}(\theta \circ \gamma)) \cup (C_2(\theta \circ \gamma) - \{L_2^1, \ldots, L_2^{r_2}\}(\theta
\end{aligned}
$$

a $\lambda \circ \sigma$ je obecnější než $\theta \circ \gamma$.

Theorem (Robinsonova věta:)

Libovolná množina klauzulí S je sporná právě tehdy, je-li z ní odvoditelná prázdná klauzule po konečném počtu aplikací obecného rezolučního pravidla.

Důkaz: $\Longleftarrow$ Nechť existuje dedukce $\square$. Protože resolventa je logickým důsledkem a odvodili jsme FALSE, množina S je sporná, tj. nemá model.

$\Longrightarrow$ Z Herbrandovy věty víme, že S je sporná právě když každý úplný semantický strom má konečný uzavřený podstrom.
Zkonstruujeme úplný semantický strom, najdeme k němu uzavřený podstrom a k tomu podstromu vytvoříme rezoluční odvození $\square$.
Protože N_1 a N_2 jsou uzly selhání ale N není uzel selhání, musí existovat základní instance $C_1^|$ a $C_2^|$ složek C_1 a C_2 takové, že jsou nepravdivé v $I(N_1)$ a $I(N_2)$, ale obě nejsou nepravdivé v $I(N)$.
Odtud plyne, že $C_1^|$ musí obsahovat literál $L_1^| = m_{n+1}$ a $C_2^|$ literál $L_2^| = \neg m_{n+1}$. Resolventou $C^|$ složek $C_1^|$ a $C_2^|$ je $C^| = (C_1^| - L_1^|) \cup (C_2^| - L_2^|)$. $C^|$ je nepravdivá v $I(N)$, protože jak $C_1^| - L_1^|$ tak $C_2^| - L_2^|$ jsou nepravdivé v $I(N)$. Podle Lifting lemmatu existuje resolventa C složek C_1 a C_2 tak, že $C^|$ je instancí C. Označme $T^{||}$ uzavřený semantický strom pro $S \cup \{C\}$, vzniklý z $T^|$ odstraněním všech uzlů a hran ležících pod prvním (od kořene) uzlem M takokvým, že $C^|$ je nepravdivá v $I(M)$. Počet uzlů $T^{||}$ je jistě menší než počet uzlů $T^|$.
Opakováním tohoto postupu dostaneme uzavřený semantický strom obsahující jediný vrchol (kořen). To je ale možné pouze tak, že postupným odvozováním rezosvent jsme dostali $\square$.

Získání odpovědi

- nalezení rezolučního zamítnutí negace dotazu a příslušných unifikačních množin
- nové proměnné místo skolemovských funcí získaných z negace dotazu
- složky získané z negace dotazu se doplní na tautologie, tj. přidá se k nim jejich negace
- modifikovaný strom zamítnutí, kde se každá rezolventa získá pomocí stejné unifikační množiny
- klauzule v kořeni stromu je odpovědí

Maximální délku důkazu nelze odhadnout

Pro libovolný algoritmus, který odpovídá na otázky, zda zvolená formule je důsledkem vstupní množiny formulí, neexistuje žádné časové omezení na délku jeho práce.
Důsledek Gödlovy věty o neúplnosti.

Rezoluční strategie

- prohledávání do šírky (úplná)
- podpůrné množiny (úplná když ...)
- jednotková (není úplná)
- vstupní (není úplná)
- filtrace předchůdných složek (lze dodef. na úplnou)
- lineární (není úplná, hrozí zacyklení)

Strategie prohledávání do šířky

Nejprve generujeme všechny rezolventy, které lze získat z výchozí množiny klauzulí jednou aplikací rezolučního pravidla. Těmto rezolventám se říká rezolventy prvního řádu. Teprve po vytvoření všech rezolvent prvního řádu jsou generovány rezolventy druhého řádu z rodičovských párů, v nichž aspoň jedna z klauzulí je rozelventou prvního řádu.
Podobně jsou generovány rezolventy vyšších řádů.

Strategie podpůrné množiny

Strategie podpůrné množiny vychází ze skutečnosti, že v každé výchozí množině klauzulí je možní určit podmnožinu, která je sama bezesporná. Samozřejmě, že rezolventy dvojic klauzulí z této podmnožiny nemohou vést ke sporu (např. Prologovský program bez dotazu). Teprve přidáním dotazu vznikne sporná množina klauzulí.

Strategie podpůrné množiny generuje rezolventy jen z takových rodičovských párů, že jeden z rodičů je odvozen od negace dokazovaného tvrzení, je totiž

- buď přímo klauzulí, která vznikla při negaci dokazovaného tvrzení
- nebo má takovou klauzuli za svého předchůdce.

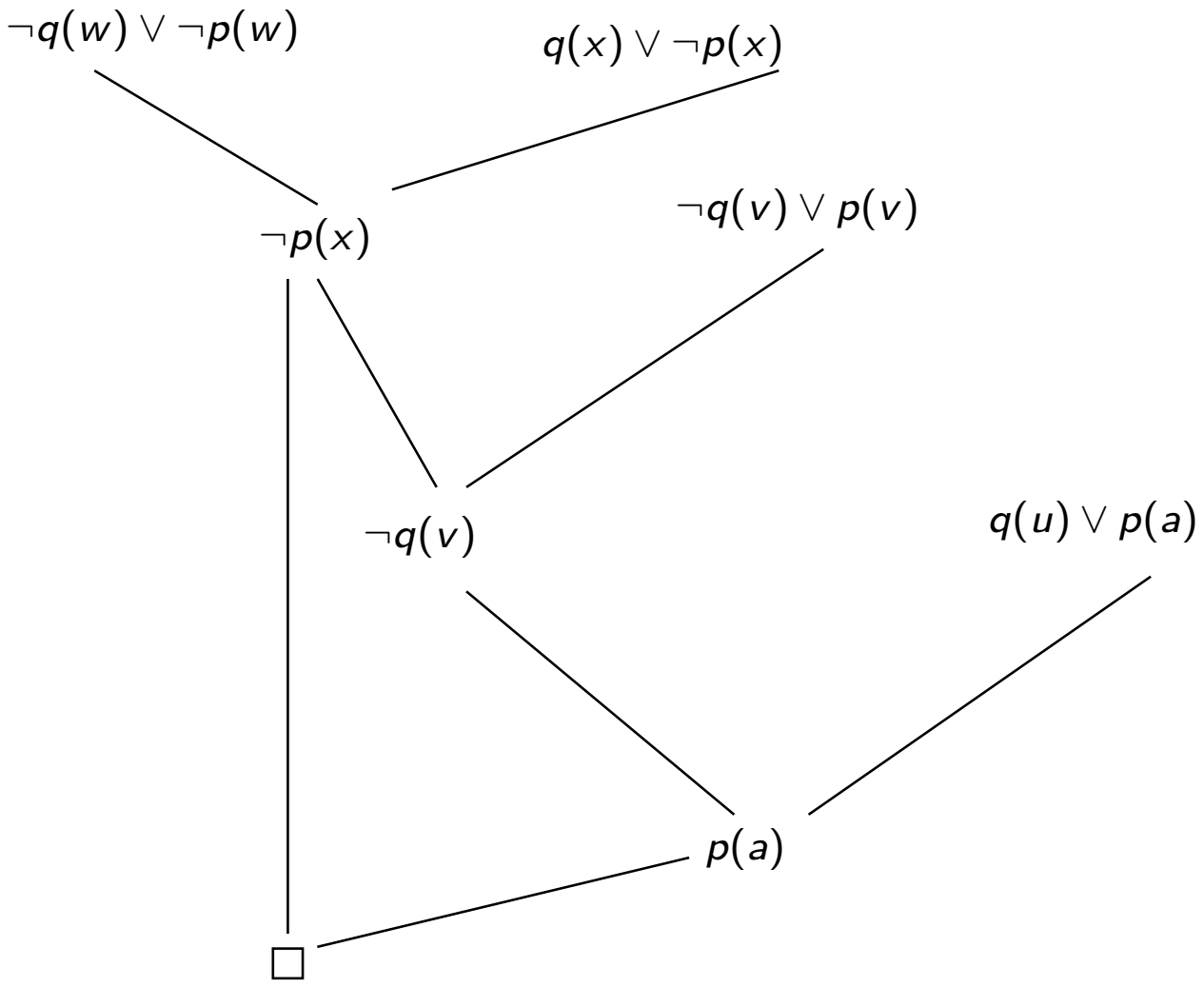
V rámci tohoto omezení opět prohledáváme do šířky.

Vstupní strategie

Jednotková strategie generuje rezolventy z párů, ve kterých aspoň jeden z rodičů je klauzule tvořená jediným literálem. Výsledná rezolventa má vtomto případě jistě menší počet literálů, než delší z obou výchozích klauzulé.
Bohužel nejde o úplnou strategii.
aspoň jeden z prvů je klauzule přímo z výchozí množiny klauzulí, jejíž spornost dokazujeme.
Ani vstupní strategie není úplná.

vstupní ani jednotková strategie není úplná

$$KB^+ = \{\neg q(w) \vee \neg p(w), q(x) \vee \neg p(x), \neg q(v) \vee p(v), q(u) \vee p(a)\}$$



Lineární strategie

Metoda filtrace předchůdných složek - každá rezolventa má rodičovskou složku, která je

- buď ze vstupní množiny S
- nebo je předchůdcem druhé rodičovské složky.

Lineární strategie využívají vždy poslední generovanou klauzuli jako jednu ze složek páru, ze kterého bude generována rezolventa.
Spolu s pevnou volbou literálu k rezoluci užívaná Prologem.
Není úplná.

Omezování množiny rezolvent

- klauzule–tautologie se může škrtnout
- klauzule obsahující pravdivý literál se může šktrtnout
- klauzule, která je instancí jiné klauzule, se může škrtnout

Definition (Zahrnování (subsumce))

Klauzule C zahrnuje klauzuli C_2, pokud pro nějakou substituci σ platí: $C\sigma \subset C_2$ (podmožina, pokud chápeme klauzuli jakožto množinu literálů).

Systémy založené na logickém odvozování

Jazyky pro dokazování vět (SAM, OTTER, AURA) dokazují věty v plné logice prvního řádu, často pro úkoly matematiky či vědeckého usuzování.
Jazyky pro logické programování (Prolog, MRS,Life) typicky omezují jazyk, omezujícím použití negace, disjunkce a-nebo rovnosti.
Produkční systémy (OPS-5, TOPS-5, CLIPS, SOAR) užívají implikace jako jejich základní reprezentaci. Důsledek implikace se chápe jako akce či doporučení, nejen logický závěr. Akce obsahují vklad či výmaz z dataváze znalostí, vstup, výstup. Produkční systémy užívají dopředné řetězení, některé mají zabudovanou strategii řešení konfliktu více aktivních pravidel.

Reprezentace znalostí

- Situační kalkul (predikátová logika)
- Sémantické sítě, rámce (umožňují udržet nekonzistenci lokálně)
- Nejistá znalost
  - logika defaultů (kvalitativně – nyní)
  - kvantitativně –

Ontologie — základní pojmy reprezentace

Jak reprezentovat znalosti?

- přímým plánem, co dělat — asi ne, potřebujeme reagovat na vjemy
- ve výrokové logice — lze, ale příliš mnoho formulí
- v predikátové logice
  - reprezentovat každé políčko zvláštní konstantou
  - nebo `location=[1,2]`

Reprezentace v predikátové logice

Komunikace s bází znalostí

- řeknu $Tell(KB, percept([smell, breeze, none], 5))$
- $Ask(KB, \exists x\ action(x, 5))$ a doufám v odpověďní substituci $Yes,\ \{shoot/x\}$

Vjemy

- $\forall b, g, t\ percept([smell, b, g], t) \Rightarrow smelt(t)$
- $\forall s, b, t\ percept([s, b, glitter], t) \Rightarrow atGold(t)$

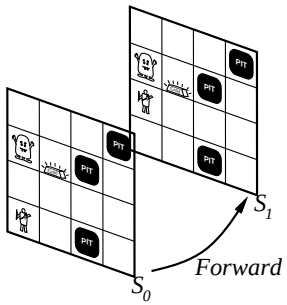
Reflexy — jednoduché odvození akcí

- Reflex: $\forall t\ atGold(t) \Rightarrow action(grab, t)$
- Reflex s vnitřním stavem:
  $\forall t\ atGold(t) \wedge \neg holding(gold, t) \Rightarrow action(grab, t)$
- $holding(gold, t)$ nemůžeme pozorovat, není vjem
  $\Rightarrow$ proto si to musíme pamatovat ve vlastní paměti (bázi znalostí)
- s reflexy nevystačíme, musíme přemýšlet o tom, kde je volno a kde nebezpečně.

Zachycení času — Situační kalkul

Jedna možnost zachycení času je Situační kalkul.

- Fakta platí v situacích místo obecné platnosti např. $holding(gold, now)$ místo $holding(gold)$
- Situace jsou propojeny predikátem $result(a, s)$, který definuje výsledek akce a v situaci s.



Problém rámců (Frame problem)

Nabízelo by se popsat efekt akcí axiomy efektu, např.:

- $\forall s\ atGold(s) \Rightarrow holding(gold, result(grab, s))$

Problém rámců je, jak elegantně popsat, že se nic jiného nezmění, např.

- $\forall s\ haveArrow(s) \Rightarrow haveArrow(result(grab, s))$

Axiomy o predikátech místo o akcích

Řešením problému rámců, co se týče vyhnutí se "no change" axiomům, je psát axiomy o predikátech, např. pro $holding(gold)$

$\forall a, s\ holding(gold, result(a, s)) \Leftrightarrow$

$$[(a = grab \wedge atGold(s)) \vee (holding(gold, s) \wedge a \neq release)]$$

Zůstává problém, jak se vyhnout kopírování všech vlastností mezi situacemi při inferenci.

Uspořádat akce podle výhodnosti

- **Výborná** — zvednout zlato
- **Dobrá** — navštívit pole, které je OK a nebylo navštíveno
- **Střední** — pohyb na pole OK, které už známe
- **Riskantní** — pohyb na pole, pro které neumíme rozhodnout, zda je OK nebo ne
- **Smrtelná** — pohyb na pole, kde je díra nebo Wumpus.

To nám ale nezaručí optimální chování. Potřebujeme plán, tj. posloupnost akcí.

Tvorba plánu (v situačním kalkulu)

Vložme do báze znalostí počáteční podmínky (předpokládáme, že S_0 je počáteční situace popsaná v bázi znalostí)

- $At(agent, [1,1], S_0)$
- $At(gold, [1,2], S_0)$

a zeptejme se

- $Ask(KB, \exists s\ holding(gold, s))$

tj. v jaké situaci budeme držet zlato?
Doufáme v odpověď typu:

- $\{result(grab, result(forward, S_0))/s\}$

tj. jdi Krok vpřed a Zvedni zlato.

Plán jakožto seznam

Lepší přístup je reprezentovat plán jakožto seznam $[a_1, a_2, \ldots, a_n]$ a definovat funkci $planresult(p, s)$ vracející situaci po provedení plánu p v situaci s.
Dotaz a odpověď pak budou vypadat:

- $Ask(KB, \exists p\ holding(gold, planresult(p, S_0)))$
- $\{[forward, grab]/p\}$

Definice *planresult* je následující:

- $\forall s\ planresult([], s) = s$
- $\forall a, p, s\ planresult([a|p], s) = planresult(p, result(a, s))$

Plánovací systémy

Pro tvorbu složitějších plánů existují specializované plánovací systémy, kterými se budeme zabývat koncem semestru.