Agent wnioskujący

Baza wiedzy (knowledge base): Zbiór faktów o świecie. Fakty te, zwane **zdaniami** lub **formułami**, zapisane są w **języku reprezentacji wiedzy**. Agent komunikuje się z bazą wiedzy (KB), używając operacji TELL i ASK:

- TELL: Agent $\rightarrow$ KB – dodanie obserwacji do KB.
- ASK: KB $\rightarrow$ Agent – pytanie o akcję.

Ogólny model agenta posługującego się bazą wiedzy:

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

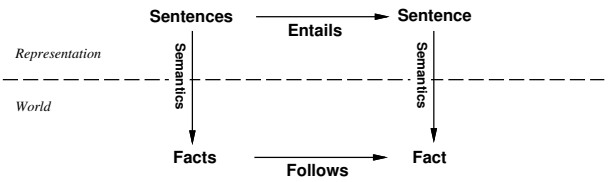

Reprezentacja, wnioskowanie i logika

Reprezentacja wiedzy (knowledge representation: Dziedzina SI zajmująca się metodami reprezentowania wiedzy o świecie.

Język reprezentacji wiedzy tworzą:

- **Syntaktyka** — opisuje budowę zdań.
- **Semantyka** — opisuje związek pomiędzy zdaniami i odpowiadającymi im faktami zachodzącymi w świecie.

Wnioskowanie: proces wyprowadzania nowych zdań ze zdań przyjętych jako prawdziwe (tzn. reprezentujących prawdziwe fakty). Poprawne wnioskowanie powinno zapewniać prawdziwość wyprowadzonych zdań.



Model dla zbioru zdań X: każdy świat w którym prawdziwe są wszystkie zdania ze zbioru X.

Wynikanie (entailment): Zdanie A wynika ze zbioru zdań X, $X \models A$, jeśli A jest prawdziwe w każdym modelu dla X.

Wyprowadzalność (derivability): Zdanie A jest wyprowadzalne ze zbioru zdań X przy użyciu procedury dowodzenia Π, $X \vdash_\Pi A$, wtw gdy Π znajduje dowód zdania A ze zdań zbioru X.

Poprawność procedury dowodzenia: Procedura dowodzenia Π jest poprawna wtw gdy dla każdego zbioru zdań X i każdego zdania A, $X \vdash_\Pi A$ pociąga $X \models A$.

Pełność procedury dowodzenia: Procedura dowodzenia Π jest pełna wtw gdy dla każdego zbioru zdań X i każdego zdania A, $X \models A$ pociąga $X \vdash_\Pi A$.

Logika: Język reprezentacji wiedzy (syntaktyka, semantyka) oraz procedura dowodzenia.

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief 0…1
Fuzzy logic	degree of truth	degree of belief 0…1

Reguła wnioskowania: $\frac{\alpha_1,\ldots\alpha_n}{\beta}$.

Reguła jest **poprawna**, jeśli β jest prawdziwa w każdej interpretacji, w której prawdziwe są $\alpha_1,\ldots\alpha_n$.

Przykłady poprawnych reguł wnioskowania

Reguła **odrywania** (modus ponens): $\frac{\alpha,\ \alpha\Rightarrow\beta}{\beta}$.

Reguła **eliminacji koniunkcji**: $\frac{\alpha_1\wedge\cdots\wedge\alpha_n}{\alpha_i}$

Reguła **wprowadzenia koniunkcji**: $\frac{\alpha_1,\ \ldots,\ \alpha_n}{\alpha_1\wedge\cdots\wedge\alpha_n}$

Reguła **rezolucji**: $\frac{\alpha\vee\beta,\ \neg\beta\vee\gamma}{\alpha\vee\gamma}$.

Poprawność reguły rezolucji ($\frac{\alpha\vee\beta,\ \neg\beta\vee\gamma}{\alpha\vee\gamma}$):

α	β	γ	$\alpha\vee\beta$	$\neg\beta\vee\gamma$	$\alpha\vee\gamma$
False	*False*	*False*	*False*	*True*	*False*
False	*False*	*True*	*False*	*True*	*True*
False	*True*	*False*	*True*	*False*	*False*
False	*True*	*True*	*True*	*True*	*True*
True	*False*	*False*	*True*	*True*	*True*
True	*False*	*True*	*True*	*True*	*True*
True	*True*	*False*	*True*	*False*	*True*
True	*True*	*True*	*True*	*True*	*True*

Monotoniczność: Jeśli formuła A wynika ze zbioru formuł X, to A wynika również z każdego nadzbioru zbioru X.

Złożoność rachunku zdań: Problem stwierdzenia, czy formuła jest tautologią jest NP-zupełny.

Procedury dowodzenia w logice pierwszego rzędu

Podstawienie: Zbiór postaci

$$\{x_1/t_1,\ldots x_n/t_n\} \qquad (1)$$

gdzie $x_1,\ldots,x_n$ są różnymi zmiennymi, natomiast $t_1,\ldots t_n$ są termami. Dopuszczamy podstawienie **puste**, oznaczane ϵ.

Niech θ będzie podstawieniem postaci (1) i niech α będzie wyrażeniem (tzn. formułą lub termem). Piszemy $\alpha\theta$ na oznaczenie wyrażenia powstałego z α w wyniku jednoczesnego zastąpienia wszystkich wolnych wystąpień zmiennych $x_1,\ldots,x_n$ termami $t_1,\ldots t_n$. Na przykład, jeśli $\theta = \{x/Sam, y/Pam\}$, to

$$(Likes(x,y))\theta = Likes(Sam,Pam).$$

Literał: Formuła atomowa lub negacja formuły atomowej.

Zdanie: Formuła bez zmiennych wolnych.

Reguła: Zdanie postaci

$$\forall\vec{x}\ (L_1\wedge\cdots\wedge L_{n-1}\Rightarrow L_n)$$

gdzie $L_1,\ldots,L_n$ $(n>1)$ są literałami różnymi od $True$ i $False$.

Zakładamy, że każda formuła bazy wiedzy jest albo regułą, albo formułą postaci $\forall\vec{x}\ L$, gdzie L jest literałem.

Powyższe założenie pozwala opuścić kwantyfikatory (uniwersalne).

Uogólniona reguła odrywania:

$$\frac{L'_1, \ldots, L'_{n-1},\ (L_1 \wedge \cdots \wedge L_{n-1} \Rightarrow L_n)}{(L_n)\theta}$$

gdzie $L'_1, \ldots, L'_{n-1}$, $L_1, \ldots L_n$ są literałami, natomiast θ jest podstawieniem takim, że dla każdego $i \in \{1, \ldots, n-1\}$, $(L'_i)\theta = (L_i)\theta$.

Złożeniem podstawień θ_1, θ_2 jest podstawienie, oznaczane $\theta_1\theta_2$, którego efekt sprowadza się do kolejnego wykonania θ_1 i θ_2. Zatem:

$$E\theta_1\theta_2 = (E\theta_1)\theta_2.$$

Podstawienie θ jest **unifikatorem** wyrażeń $E_1, \ldots, E_n$ jeśli $E_1\theta = E_2\theta = \cdots = E_n\theta$.

Podstawienie $\{x/a\}$ jest unifikatorem wyrażeń $P(x), P(a)$. Wyrażenia $P(x), Q(b)$ nie są unifikowalne.

Podstawienie θ jest **najogólniejszym** unifikatorem wyrażeń $E_1, \ldots, E_n$, jeśli dla każdego unifikatora tych wyrażeń, γ, istnieje podstawienie λ takie, że $\gamma = \theta\lambda$.

Wyrażenia $P(John, x)$, $P(y, z)$ posiadają nieskończenie wiele unifikatorów:

$$\{y/John, x/z\},\ \{y/John, x/z, w/Freda\},$$
$$\{y/John, x/John, z/John\} \ldots.$$

Najogólniejszy unifikator – $\{y/John, x/z\}$.

Problem unifikacji: Stwierdzić, czy dane wyrażenia $E_1, \ldots, E_n$ są unifikowalne. Jeśli tak, znaleźć ich najogólniejszy unifikator.

Problem unifikacji jest rozstrzygalny.

Zakładamy, że mamy funkcję $UNIFY$, która dla zadanych wyrażeń $E_1, \ldots, E_n$, zwraca ich najogólniejszy unifikator θ lub specjalną stałą $fail$, jeśli unifikator nie istnieje.

Wariant (renaming) — Formuła A jest wariantem formuły B, jeśli A można otrzymać z B, zastępując niektóre zmienne, zmiennymi nie występującymi w B.

$Likes(x, John)$ jest wariantem $Likes(y, John)$. $Likes(x, x)$ nie jest wariantem $Likes(x, y)$.

Formuły naszej bazy wiedzy są takiej postaci, że każda może być zastąpiona swoim wariantem. (Powstała w ten sposób nowa baza wiedzy będzie równoważna starej.)

Dlaczego należy przemianowywać zmienne?

$Knows(x, Elizabeth)$.

$Knows(John, x) \Rightarrow Hates(John, x)$.

Powinniśmy, stosując ogólną regułę odrywania, wywnioskować $Hates(John, Elizabeth)$. Niestety jest to niemożliwe: formuły $Knows(John, x)$ i $Knows(x, Elizabeth)$ nie są unifikowalne. Jeśli przemianujemy zmienne w jednej z formuł, np. formułę $Knows(x, Elizabeth)$ zastąpimy przez $Knows(y, Elizabeth)$, będziemy mogli wyprowadzić $Hates(John, Elizabeth)$.

Możemy założyć: Przesłanki uogólnionej reguły odrywania nie zawierają wspólnych zmiennych.

Procedury dowodzenia oparte na uogólnionej regule odrywania

Metoda progresywna (Forward-chaining).

Stosujemy, kiedy nowy fakt p zostaje dodany do bazy wiedzy KB. Procedura znajduje nowe fakty, które wynikają z $KB \cup \{p\}$.

procedure FORWARD-CHAIN(KB, p)

if there is a sentence in KB that is a renaming of p **then return**
Add p to KB
for each $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ **in** KB such that for some i, UNIFY(p_i, p) $= \theta$ succeeds **do**
FIND-AND-INFER($KB, [p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n], q, \theta$)
end

---

procedure FIND-AND-INFER(KB, *premises*, *conclusion*, θ)

if *premises* $=$ [ ] **then**
FORWARD-CHAIN(KB, SUBST(θ, *conclusion*))
else for each p' **in** KB such that UNIFY(p', SUBST(θ, FIRST(*premises*))) $= \theta_2$ **do**
FIND-AND-INFER(KB, REST(*premises*), *conclusion*, COMPOSE(θ, θ_2))
end

Metoda regresywna (Backward-chaining).

Stosujemy, kiedy chcemy otrzymać odpowiedź na pytanie zadane bazie wiedzy.

function BACK-CHAIN(KB, q) **returns** a set of substitutions

BACK-CHAIN-LIST(KB, [q], {})

---

function BACK-CHAIN-LIST(KB, *qlist*, θ) **returns** a set of substitutions
inputs: KB, a knowledge base
qlist, a list of conjuncts forming a query (θ already applied)
θ, the current substitution
static: *answers*, a set of substitutions, initially empty

if *qlist* is empty **then return** $\{\theta\}$
$q \leftarrow$ FIRST(*qlist*)
for each q'_i **in** KB such that $\theta_i \leftarrow$ UNIFY(q, q'_i) succeeds **do**
Add COMPOSE(θ, θ_i) to *answers*
end
for each sentence $(p_1 \wedge \ldots \wedge p_n \Rightarrow q'_i)$ **in** KB such that $\theta_i \leftarrow$ UNIFY(q, q'_i) succeeds **do**
answers $\leftarrow$ BACK-CHAIN-LIST(KB, SUBST(θ_i, [$p_1 \ldots p_n$]), COMPOSE(θ, θ_i)) $\cup$ *answers*
end
return the union of BACK-CHAIN-LIST(KB, REST(*qlist*), θ) for each $\theta \in$ *answers*

Przykład użycia metody regresywnej

$$American(x) \wedge Weapon(y) \wedge Nation(z) \wedge Hostile(z) \wedge Sells(x, z, y) \Rightarrow Criminal(x). \quad (1)$$

$$Owns(Nono, M1). \quad (2)$$

$$Missile(M1). \quad (3)$$

$$Owns(Nono, x) \wedge Missile(x) \Rightarrow Sells(West, Nono, x). \quad (4)$$

$$Missile(x) \Rightarrow Weapon(x). \quad (5)$$

$$Enemy(x, America) \Rightarrow Hostile(x). \quad (6)$$

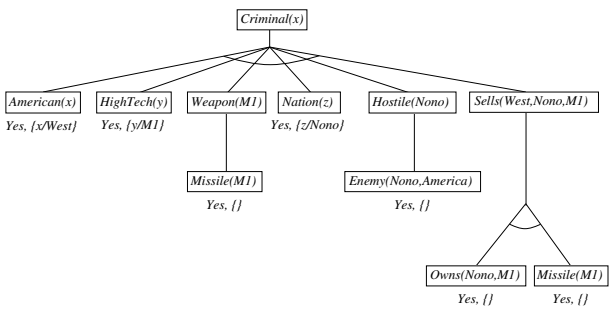
$$American(West). \quad (7)$$

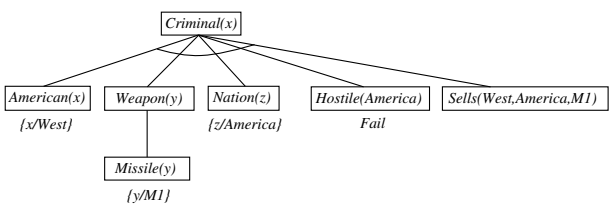
$$Nation(Nono). \quad (8)$$

$$Enemy(Nono, America). \quad (9)$$

$$Nation(America). \quad (10)$$

Dowieść, że $Criminal(West)$.





Metody wnioskowania oparte na uogólnionej regule odrywania są niepełne.

$\forall x\ P(x) \Rightarrow Q(x)$.

$\forall x\ \neg P(x) \Rightarrow R(x)$.

$\forall x\ Q(x) \Rightarrow S(x)$.

$\forall x\ R(x) \Rightarrow S(x)$.

$T(a)$.

Z powyższych formuł wynika $S(a)$. ($S(a)$ jest prawdziwa jeśli $Q(a)$ lub $R(a)$ jest prawdziwa, a jedna z tych formuł musi być prawdziwa, ponieważ $P(a)$ jest prawdziwa lub $\neg P(a)$ jest prawdziwa.) Niestety formuły $S(a)$ nie można wyprowadzić ani metodą progresywną ani regresywną.

Metoda rezolucji (Resolution method)

Metoda rezolucji (Robinson, 1965) jest pełną (i stosunkowo efektywną) procedurą dowodzenia twierdzeń w logice pierwszego rzędu.

Problem dowodzenia twierdzeń: Dla danej bazy wiedzy KB i danej formuły α, stwierdzić, czy $KB \models \alpha$. (Problem półrozstrzygalny.)

Problem dowodzenia twierdzeń jest równoważny **problemowi spełnialności**: Dla danej bazy wiedzy KB i danej formuły α, stwierdzić, czy zbiór $KB \cup \{\neg\alpha\}$ jest niespełnialny.

Ponieważ bazy wiedzy są skończone, $KB \cup \{\neg\alpha\}$ można zawsze traktować jako pojedynczą formułę.

Metoda rezolucji, dla danej formuły β, próbuje stwierdzić, czy β jest niespełnialna.

Metodę rezolucji stosuje się dla formuł w postaci **klauzulowej**.

Klauzula – formuła postaci

$$l_1 \vee \cdots \vee l_n \quad (n \geq 0) \qquad (11)$$

gdzie $l_1, \ldots, l_n$ są literałami. Jeśli $n = 0$, klauzulę (11) nazywamy **kluzulą pustą** i oznaczamy symbolem $\perp$.

Postać kluzulowa:

$$\forall\, \overline{x_1}\, C_1 \wedge \cdots \wedge \forall\, \overline{x_m}\, C_m \quad (m \geq 1)$$

gdzie $C_1, \ldots, C_n$ są klauzulami, natomiast $\overline{x_i}$ jest listą wszystkich zmiennych występujących w C_i.

Twierdzenie Istnieje algorytm, który dla dowolnej formuły A znajduje formułę B taką, że: (1) B jest w postaci klauzulowej. (2) A jest spełnialna wtw gdy B jest spełnialna. (A i B nie muszą być równoważne.)

Sprowadzanie formuły A do postaci klauzulowej

1. Niech $\overline{x}$ będzie listą wszystkich zmiennych wolnych występujących w A. Zastępujemy A formułą $\exists\, \overline{x}\, A$.

2. Eliminujemy wszystkie redundantne kwantyfikatory, tzn. każdy kwantyfikator postaci $\forall x$ lub $\exists x$ taki, że zmienna x nie występuje w obrębie działania tego kwantyfikatora.

3. Przemianowujemy zmienne tak, aby wszystkie kwantyfikowane zmienne były różne.

4. Eliminujemy "$\Rightarrow$" i "$\Leftrightarrow$", zastępując $B \Rightarrow C$ przez $\neg B \vee C$ i $B \Leftrightarrow C$ przez $((\neg B \vee C) \wedge (\neg C \vee B))$.

5. Przesuwamy "$\neg$" w prawo, zastępując $\neg\forall x B$ przez $\exists x\ \neg B$, $\neg\exists x\ B$ przez $\forall x\ \neg B$, $\neg(B \vee C)$ przez $\neg B \wedge \neg C$, $\neg(B \wedge C)$ przez $\neg B \vee \neg C$, $\neg\neg B$ przez B, tak długo aż wszystkie symbole negacji występują bezpośrednio przed formułami atomowymi.

6. (Krok opcjonalny. Poniżej Q oznacza $\forall$ lub $\exists$; $\diamondsuit$ oznacza $\wedge$ lub $\vee$.) Przesuwamy kwantyfikatory w prawo, zastępując $Qx(B \diamondsuit C)$ przez $B \diamondsuit (QxC)$, jeśli x nie jest wolna w B, i przez $(QxB) \diamondsuit C$, jeśli x nie jest wolna w C.

7. Skolemizacja (Usunięcie kwantyfikatorów egzystencjalnych). Powtarzamy poniższy proces, aż wszystkie kwantyfikatory $\exists$ zostaną usunięte.

Bierzemy dowolną podformułę postaci $\exists y\, B(y)$. Niech $x_1, \ldots, x_n$ $(n \geq 0)$ będzie listą wszystkich zmiennych wolnych występujących w $\exists y\, B(y)$, które są uniwersalnie kwantyfikowane w pewnej nadformule formuły $\exists y\, B(y)$. Jeśli $n = 0$, zastępujemy $\exists y\, B(y)$ przez $B(A)$, gdzie A jest **nową** stałą indywiduową. Jeśli $n > 0$, zastępujemy $\exists y\, B(y)$ przez $B(F(x_1, \ldots, x_n))$, gdzie F jest **nowym** symbolem funkcji n-argumentowej.

8. Usuwamy wszystkie kwantyfikatory uniwersalne.

9. Rozdzielamy $\wedge$ względem $\vee$, zastępując $(B \wedge C) \vee D$ przez $(B \vee D) \wedge (C \vee D)$ oraz $B \vee (C \wedge D)$ przez $(B \vee C) \wedge (B \vee D)$.

10. Eliminujemy stałe $True$ i $False$. Aktualna formuła jest koniunkcją klauzul. Usuwamy każdą klauzulę zawierającą literał $\neg False$ lub $True$. Z pozostałych klauzul usuwamy każdy literał $\neg True$ i $False$.

11. Dodajemy kwantyfikatory uniwersalne. Aktualna formuła jest postaci $C_1 \wedge \cdots \wedge C_k$, gdzie $C_1, \ldots, C_k$ są klauzulami. Zastępujemy ją formułą

$$\forall\, \overline{x_1}\, C_1 \wedge \cdots \wedge \forall\, \overline{x_k}\, C_k$$

gdzie $\overline{x_i}$ jest listą wszystkich zmiennych występujących w C_i.

Uwaga: Kroki 1 i 7 nie zachowują równoważności. (Zachowują spełnialność.)

Formułę w postaci kluzulowej

$$\forall\, \overline{x_1}\, C_1 \wedge \cdots \wedge \forall\, \overline{x_m}\, C_m$$

możemy zapisać jako zbiór klauzul, tzn. w postaci $\{C_1, \ldots, C_m\}$. Możemy też zawsze opuścić kwantyfikatory.

Literał dodatni – formuła atomowa.

Literał ujemny – negacja formuły atomowej.

$| l |$ oznacza formułę atomową literału l.

Piszemy $l_1, \ldots, l_n \in C$, aby wyrazić fakt, że literały $l_1, \ldots, l_n$ występują w kluzuli C. Jeśli $l_1, \ldots, l_n \in C$, to $C - \{l_1, \ldots, l_n\}$ oznacza klauzulę otrzymaną z C w wyniku usunięcia $l_1, \ldots, l_n$.

Dla wyrażenia E i podstawienia θ, piszemy $E\theta$ jako skrót zapisu $SUBST(\theta, E)$.

Reguła rezolucji (Resolution rule)

Definicja 1 Niech C_1 i C_2 będą klauzulami bez wspólnych zmiennych. Niech $l_1, \ldots, l_n \in C_1$ $(n > 0)$ i $l'_1, \ldots, l'_k \in C_2$ $(k > 0)$. Załóżmy ponadto, że wszystkie literały $l_1, \ldots, l_n$ są dodatnie i wszystkie literały $l'_1, \ldots, l'_k$ są ujemne, lub odwrotnie. Jeśli θ jest najogólniejszym unifikatorem zbioru $\{| l_1 |, \ldots, | l_n |, | l'_1 |, \ldots, | l'_k |\}$, to kluzulę

$$(C_1\theta - \{l_1\theta, \ldots, l_n\theta\}) \vee (C_2\theta - \{l'_1\theta, \ldots, l'_k\theta\})$$

nazywamy **binarną rezolwentą** klauzul C_1 i C_2.

Przykład Niech

$$C_1 = P(x) \vee P(y) \vee Q(B).$$
$$C_2 = R(A) \vee \neg P(F(A)).$$

Klauzula $Q(B) \vee R(A)$ jest binarną rezolwentą klauzul C_1 i C_2. $(l_1 = P(x), l_2 = P(y), l'_1 = \neg P(F(A));\ \theta = \{x/F(A), y/F(A)\}.)$

Definicja 2 **Rezolwentą** klauzul C_1 i C_2, oznaczaną $RES(C_1, C_2)$, nazywamy dowolną binarną rezolwentę wariantu klauzuli C_1 i wariantu klauzuli C_2.

Definicja 3 **Regułą rezolucji** nazywamy regułę wnioskowania

$$\frac{C_1, C_2}{RES(C_1, C_2)}.$$

Dowód rezolucyjny ze zbioru klauzul CL: Ciąg klauzul $C_1, \ldots, C_n$ taki, że $C_n = \bot$ i dla każdego $1 \leq i \leq n$, $C_i \in CL$ lub C_i jest rezolwentą C_j i C_k, gdzie $j, k < i$.

Twierdzenie Niech CL będzie zbiorem klauzul nie zawierających symbolu "=". CL jest niespełnialny wtw gdy istnieje dowód rezolucyjny z CL.

Aksjomaty równości

CL – zbiór klauzul. Przez $EQ(CL)$ oznaczamy **zbiór aksjomatów równości dla** CL. $EQ(CL)$ zawiera następujące klauzule:

$x = x$.

$x \neq y \vee y = x$.

$x \neq y \vee y \neq z \vee x = z$.

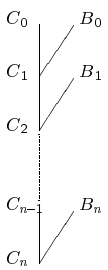
$x_1 \neq y_1 \vee \cdots \vee x_n \neq y_n \vee$
$\neg P(x_1, \ldots, x_n) \vee P(y_1, \ldots, y_n)$,
dla każdego n-argumentowego $(n > 0)$ symbolu predykatowego P występującego w CL.

$x_1 \neq y_1 \vee \cdots \vee x_n \neq y_n \vee$
$(F(x_1, \ldots, x_n) = F(y_1, \ldots, y_n))$,
dla każdego n-argumentowego $(n > 0)$ symbolu funkcyjnego F występującego w CL.

Twierdzenie Niech CL będzie zbiorem klauzul z równością. CL jest niespełnialny wtw gdy istnieje dowód rezolucyjny z $CL \cup EQ(CL)$.

Rezolucja liniowa

Liniowym dowodem rezolucyjnym ze zbioru klauzul CL nazywamy dowód postaci



gdzie:

1. $C_0 \in CL$.
2. Dla $0 \leq i < n$, $B_i \in CL$ lub $B_i = C_j$, dla pewnego $j < i$.
3. Dla $1 \leq i \leq n$, C_i jest rezolwentą C_{i-1} i B_{i-1}.
4. $C_n = \bot$.

Twierdzenie Niech CL będzie zbiorem klauzul nie zawierających symbolu "=". CL jest niespełnialny wtw gdy istnieje liniowy dowód rezolucyjny z CL.

Twierdzenie Niech CL będzie zbiorem klauzul z równością. CL jest niespełnialny wtw gdy istnieje liniowy dowód rezolucyjny ze zbioru $CL \cup EQ(CL)$.

Klauzulę C_0 w liniowym dowodzie rezolucyjnym nazywamy **klauzulą główną**.

Twierdzenie Niech CL będzie niespełnialnym zbiorem klauzul i niech $CL' \subseteq CL$. Jeśli zbiór $CL - CL'$ jest spełnialny, to istnieje liniowy dowód rezolucyjny z CL (ewentualnie z $CL \cup EQ(CL)$) z klauzulą główną $C \in CL'$.

Logiczne systemy wnioskowania

- Udowadniacze twierdzeń (theorem provers) – AURA, OTTER, SCAN.
- Języki programowania w logice (logic programming languages) – PROLOG, LIFE, GÖDEL.
- Systemy produkcyjne (production systems) – OPS-5, CLIPS, SOAR .
- Ramy (frames) – OWL, KODIAK, FRAIL.
- Sieci semantyczne (semantic nets) – CG, NETL, SNEPS