

Wykład 8 z Programowania, godzina 2

Paweł Laskoś

zannotowano 10 marca 2005

A zatem, reasumując, jeśli w dwu klauzulach znajdziemy dwa literały o przeciwnych „znakach”, to możemy – usunąwszy literały i połączywszy klauzule – otrzymać nową formułę zdaniową, implikowaną przez wyjściową. Jeśli – odpowiednio wybierając różnoimienne literały w klauzulach – doprowadzimy w ten sposób do pozostania formuły, składającej się z jednej, pustej klauzuli, oznacza to, że pierwotna formuła implikuje fałsz, zatem jest fałszywa.

$$\frac{[p, q] [\neg p, q] [\neg q]}{[q] [\neg q]} \\ []$$

Powyższy przykład świadczy również o indeterminizmie takiego sposobu dowodzenia twierdzeń – wystarczy inaczej dobierać spajane klauzule, by nie dojść do rozwiązania:

$$\frac{[p, q] [\neg p, q] [\neg q]}{[p, q] [\neg p]} \\ [q]$$

Oznacza to, że należy umiejętnie przewidzieć ścieżkę na drzewie możliwych spajań, która zakończy się klauzulą pustą. Jest to zatem proces heurystyczny – ogólne dowodzenie twierdzeń jest problemem NP-zupełnym.

Reguła rezolucji jest podstawą automatycznego dowodzenia twierdzeń. Alan Robinson rozszerzył jej możliwości na rachunek predykatów pierwszego rzędu – który ograniczył do zdań zawierających kwantyfikator, spójniki $\neg \vee \wedge$ oraz atomy postaci $R(t_1, \dots, t_n)$, gdzie R jest n -arnym symbolem relacyjnym. Formuły miały mieć formę preneksową i CNF.

Zgodnie z umową, że wolne zmienne w formułach są skwantyfikowane ogólnie, możemy w formułach opuszczać kwantyfikator ogólnie. Aby opuścić kwantyfikator szczególny zmiennej y należałoby w formule zastąpić wystąpienia tej zmiennej przez stałą c_y , jednak może być ona zależna od zmiennych uprzednio skwantyfikowanych ogólnie – zatem zamiast stałej wprowadzamy symbol funkcyjny $y(x_1, \dots, x_n)$, gdzie x_i to zmienne ogólnie skwantyfikowane. Ten proces opuszczania kwantyfikatorów jest dozwolony, ponieważ prawdziwość twierdzenia nie zależy od interpretacji symboli funkcyjnych i relacyjnych. Od nazwiska jego twórcy – norweskiego matematyka Thoralfa Skolema – proces ten nazywany jest skolemizacją, zaś funkcje, którymi zastępujemy wystąpienia zmiennych – funkcjami Skolema.

$$\forall x \exists y \ x + 1 = y \\ \exists y \ x + 1 = y \\ x + 1 = f(x)$$

Na tak uproszczonych formułach możemy wprowadzić „lepszą” rezolucję. Traktując symbole funkcyjne i relacyjne jak termy (mają tę samą strukturę), jeśli możemy w dwu klauzulach C_1 , C_2 wyróżnić dwa różnoimienne termy (odpowiednio t_1 , t_2), to jeśli ϑ jest ich najogólniejszym unifikatorem, to wyjściowa formuła implikuje

$$(C_1 \setminus \{t_1\}, C_2 \setminus \{t_2\}, \dots) \vartheta$$

Idea dalszego postępowania jest ta sama, jak w pierwszym przypadku. Binarne unifikacja (przedstawiony model operujący tylko na parze termów na raz) łatwo może się „zatkać”, lepiej działają analogiczne procesy na większej liczbie klauzul i termów. Nie ulega zmianie fakt, że proces rezolucji jest heurystyczny – i tu nie ma najlepszego algorytmu dobierającego rezolucje.

Tu kończy się dygresja – wracamy do Marsylii. Twórcy Prologu zdecydowali się na dalsze ograniczenie form, jakie przyjmują formuły rachunku predykatów, tym razem stratnie. Poniżej mamy ciąg równoważnych przekształceń przykładowej klauzuli:

$$\neg t_1 \vee t_2 \vee \neg t_3 \vee t_4 \vee t_5$$

$$\neg (t_1 \wedge t_3) \vee t_2 \vee t_4 \vee t_5$$

$$(t_1 \wedge t_2) \Rightarrow t_2 \vee t_4 \vee t_5$$

Jeśli t_1 i t_3 będą prawdziwe, to prawdziwy będzie również któryś z t_2 , t_4 , t_5 . Aby uniknąć pytania „który?”, Colmeramer i Kowalski postanowili ograniczyć ilość termów w następniku implikacji do co najwyżej jednego. Klauzule zatem miały zawierać co najwyżej jeden literal pozytywny. Ograniczenie to jest stratne, ponieważ choć dla każdej formuły istnieje formuła w CNF równoważna jej, to ograniczenie tylko do takich klauzul (zwanym hornowskimi) ogranicza również zbiór formuł, które można w ten sposób przedstawić. Zaletą tego ograniczenia jest szybki algorytm dowodzenia twierdzeń – o złożoności wielomianowej.

Klauzule hornowskie można podzielić na kilka podklas: klauzule postaci

$$\top \Rightarrow t$$

nazywamy faktami, bo koniunkcja zera elementów jest zawsze prawdziwa, zatem t jest zawsze prawdziwy; postaci

$$(t_1 \wedge \dots \wedge t_n) \Rightarrow t$$

nazywamy regułami; zaś postaci

$$(t_1 \wedge \dots \wedge t_n) \Rightarrow \perp$$

nazywamy celami. Wtedy algorytm dowodzenia twierdzeń sprowadza się do następujących kroków: skoro następnik faktu jest prawdziwy, to jeśli występuje w poprzedniku jakiejś reguły, to nie wpływa na jego prawdziwość (bo jest koniunkcją), zatem można go zeń usunąć; wyeliminowanie w taki sposób wszystkich predykatów z poprzednika pewnej reguły sprawia, że staje się ona faktem. W podobny sposób przetwarza się cele, do uzyskania pustej listy predykatów.

By uzyskać determinizm algorytmu, uznano, że predykaty tworzące poprzedniki celów i reguł przetwarza się od lewego do prawego, co jednak sprawia, że ten wariant rezolucji nie jest zupełny. Nie da się tym algorytmem dowieść twierdzenia prawdziwego, jednak jego niewątpliwą zaletą jest łatwość implementacji.

O ile w Lispie wszystko należało przetworzyć na pary kropkowe, to Prolog ma sygnaturę Σ składającą się z funkcyjnych i predykatów (tj. symboli funkcyjnych i relacyjnych o określonej arności) oraz atomów (stałych). Oznacza to, że napisy

ident/0.
ident/1.
ident/7.

reprezentują inne funktory (pierwszy z napisów to stała). W Prologu operujemy na $\mathcal{T}(\Sigma, V)$, czyli zbiorze termów o sygnaturze Σ i nośniku (zbiorze zmiennych) V . Wiadomo, że zmienna $v \in V$ jest termem, stała $c \in \Sigma_0$ jest termem, oraz $f(t_1, \dots, t_n)$ jest termem dla termów $t_1, \dots, t_n \in \mathcal{T}(\Sigma, V)$ i funktora $f \in \Sigma_n$.

Z elementów składni Prologu: identyfikatory funktorów, predykatów i stałych to ciągi znaków zaczynające się małą literą, zmiennych – wielką literą. Napisy

p(X).
p(X) :- q(Y), r(c).
?- q(Y), r(c).

oznaczają odpowiednio fakt, regułę i cel. Implikacje zostały „odwrócone” i zastąpione $:-$ z powodu klawiaturowego ubóstwa tamtych czasów. Pomiedzy identyfikatorem predykatu lub funktora a nawiasem otwierającym nie może być spacji.

Działanie Prologu – podobnie jak Lispu – to przetwarzanie napisów. Odpowiednikiem typowego programu ”Hello, World!” są „związki rodzinne”.

Gosia Krzyś
 \
 / Jan Ala Grześ
 \
 / Henio Kasia

Niech atomami (których jedyną własnością jest to, że są różne) będą **Gosia**, **Krzyś** etc. Jeśli zdefiniujemy predykat `rodzic/2`. to diagram zapiszemy za pomocą sześciu następujących faktów:

rodzic(Gosia, Jan).
rodzic(Ala, Henio).
etc...

Jeśli następnie pojawi się wywołanie faktu

?- rodzic(Ala, Kasia).

to Prolog będzie próbował zaprzeczyć twierdzeniu drogą unifikacji. Algorytm kończy pracę, gdy otrzyma pusty poprzednik implikacji, tj. formułę $\top \Rightarrow \perp$.