POLSKA AKADEMIA NAUK
Instytut Badań Systemowych

WSPOMAGANIE DECYZJI
SYSTEMY EKSPERCKIE

pod redakcją

Romana Kulikowskiego i Lucyny Bogdan

Warszawa 1995

WSPOMAGANIE DECYZJI

SYSTEMY EKSPERCKIE

pod redakcją

Romana Kulikowskiego i Lucyny Bogdan

Warszawa 1995

Wydano z wykorzystaniem dotacji
KOMITETU BADAŃ NAUKOWYCH

Materiały konferencji: "Analiza Decyzyjna, Systemy Eksperckie, Zastosowania Systemów Komputerowych", Warszawa, 25-27 maja 1994r.

Komitet Programowy Konferencji:

Andrzej Ameljańczyk, Zdzisław Bubnicki, Wiesław Grudzewski, Olgierd Hryniewicz, Janusz Kacprzyk, Lech Kruś, Roman Kulikowski (przewodniczący), Kazimierz Mańczak, Ireneusz Nykowski, Zdzisław Pawlak, Roman Słowiński, Andrzej Straszak, Andrzej Weryński, Andrzej Wierzbicki.

Wykonano z oryginałów tekstowych dostarczonych przez autorów

ISBN 83-85847-85-5

AUTOMATYCZNE WNIOSKOWANIE W OPARCIU O PROLOGOWĄ TECHNOLOGIĘ DOWODZENIA TWIERDZEŃ

Adam Meissner
Politechnika Poznańska, Katedra Automatyki, Robotyki i Informatyki

System wykonawczy języka Prolog (ang. *Prolog engine*) jest z punktu widzenia logiki formalnej mechanizmem wnioskującym w zbiorze klauzul Horna. Ze względu na wysoką efektywność oraz oszczędną gospodarkę pamięcią komputera podejmowane są różnorakie próby wykorzystania systemów prologowych do dowodzenia dowolnych twierdzeń klasycznego rachunku predykatów. Na przeszkodzie temu stoi jednak niepełność podstawowych algorytmów w oparciu o które Prolog przeprowadza wnioskowania. Konkretnie chodzi tu o:

- algorytm unifikacji;
- regułę wnioskowania;
- strategię wyboru przesłanek.

Prologowa technologia dowodzenia twierdzeń (ang. *Prolog Technology Theorem Proving*) [5] jest metodą dostosowania powyższych procedur do wnioskowań w pełnej logice pierwszego rzędu. Polega ona na przekształcaniu wejściowego zbioru formuł, zawierającego aksjomaty danej teorii oraz dowodzone twierdzenie, w równoważny modelowo program prologowy. Program ów, oprócz reprezentacji formuł wejściowych, zawiera także predykaty rozszerzające mechanizmy wchodzące w skład standardowego systemu prologowego. Jeżeli wykonanie programu zakończy się sukcesem, oznacza to, że dowodzona hipoteza jest twierdzeniem badanej teorii.

Transformacja wejściowego zbioru formuł przebiega w pięciu etapach. Etapy pierwszy i piąty wprowadzone zostały przez autora referatu jako propozycja rozszerzenia metody *PTTP*. Celem operacji przeprowadzanych w trakcie etapu pierwszego jest sprowadzenie formuł wejściowych do standardowej postaci, jaką przyjmują dane na których działają algorytmy przedstawione w [5]. Etap piąty rozszerza program wynikowy o procedury generujące tekst dowodu rezolucyjnego przyjętej hipotezy.

Podczas pierwszej fazy przekształceń tworzona jest prologowa reprezentacja zbioru wejściowego. Proces ten przebiega w dwóch krokach. W pierwszym kroku zbiór

aksjomatów rozszerza się o negacje dowodzonej hipotezy; jest to konieczne do poprawnego działania procedur wnioskujących. Następnie aksjomaty sprowadzone zostają do *standardowej postaci Skolema* (ang. *Skolem normal form*). Szczegółowy opis tego zabiegu znaleźć można w [2]. Formuła jest w standardowej postaci Skolema, o ile jest formułą postaci $\forall x_1 \ldots \forall x_n \Phi$, gdzie Φ jest w *normalnej postaci koniunkcyjnej* (ang. *conjunctive normal form*) [2] i nie zawiera zmiennych wolnych oraz żadnych kwantyfikatorów (zmienne związane we formule źródłowej kwantyfikatorami szczegółowymi zastąpione zostały przez *funkcje Skolema*). Podobnym przekształceniom podlega hipoteza; jedyna pojawiająca się tu różnica jest konsekwencją automatycznego negowania dowodzonej formuły przez system prologowy. Postać zanegowana musi być zgodna z ogólną postacią aksjomatów; ostatecznie przyjmuje ona formę $\exists x_1 \ldots \exists x_n \Phi$, gdzie Φ jest w *normalnej postaci alternatywnej* (ang. *disjunction normal form*) i nie zawiera kwantyfikatorów ani zmiennych wolnych. W drugim kroku pierwszego etapu przekształceń dla każdej formuły konstruuje się reprezentujący ją zbiór klauzul prologowych. Proces ten ilustruje poniższy przykład:

przykład

aksjomat: $(\forall x)(\forall y)\ ((A(x) \vee B(x)) \wedge (\neg C(y) \vee D(y))$

prologowa reprezentacja:

```
a(X) :- not_b(X).
b(X) :- not_a(X).
not_c(Y) :- not_d(Y).
d(Y) :- c(Y).
```

hipoteza: $(\exists x)(\exists y)\ (A(x) \wedge B(x)) \vee (\neg C(y) \wedge D(y))$

prologowa reprezentacja:

```
query :- a(X), b(X).
query :- not_c(X), d(X).
```

Pojawiający się w nazwach predykatów przedrostek *not_* funkcjonuje jako metaoperator negacji. Predykat *query* ma charakter sztuczny i pełni rolę głównego celu w programie wynikowym. W rezultacie opisanych powyżej przekształceń powstaje ostatecznie zbiór klauzul stanowiący pod względem syntaktycznym poprawny program prologowy. Program taki nie nadaje się jednak do bezpośredniego wykonania ze względu na występujący w nim przedrostek *not_*, którego poprawna interpretacja wykracza poza semantykę Prologu i wymaga rozszerzenia prologowej reguły wnioskowania.

Celem drugiego etapu przekształceń jest wprowadzenie do zbioru wejściowego reguł rozszerzających algorytm prologowej unifikacji o tzw. *test wystąpień* (ang. *occurs check*). Polega on na sprawdzaniu, czy dana zmienna występuje w unifikowanym z nią termie złożonym; jeśli tak jest - to unifikacja kończy się niepowodzeniem. Należy zaznaczyć, że test wystąpień stanowi jeden z kroków klasycznego algorytmu unifikacji. W praktyce jednak, większość istniejących systemów prologowych nie przeprowadza go. Decydują o tym przede wszystkim względy efektywnościowe (duża złożoność obliczeniowa). Poza tym brak testu umożliwia generowanie termów cyklicznych, które chociaż z punktu widzenia logiki nie są danymi poprawnymi (gdyż nie należą do

uniwersum Herbranda), to jednak pozwalają na łatwą reprezentację struktur cyklicznych pojawiających się w wielu zastosowaniach Prologu. Uproszczony algorytm unifikacji może jednak popaść w nieskończoną pętlę lub też zwrócić odpowiedź niepoprawną (tj. udowodnić formułę fałszywą). Istnieje kilka sposobów usunięcia tej wady; PTTP wykorzystuje w tym celu metodę *linearyzacji* klauzul, zaproponowaną przez D.A.Plaisteda [4]. Przekształceniu podlegają wyłącznie te klauzule, w których nagłówkach pewna zmienna występuje więcej niż jednokrotnie. Zasadę przekształcenia ilustruje poniższa tabela:

klauzula wejściowa	klauzula wyjściowa
p(X,...,X) :- body(X,...,X).	p(X, X1,...,Xn-1) :- unify(X, X1), ..., unify(X, Xn-1), body(X,X1,...,Xn-1).

Konieczne jest oczywiście dołączenie do zbioru wynikowego definicji predykatu *unify/2*, realizującego unifikację z testem wystąpień.

Podczas trzeciego etapu przekształceń rozszerza się prologową regułę wnioskowania. Umożliwia to poprawną interpretację przedrostka *not_*, który może wystąpić w nazwach predykatów. Standardowy system prologowy wnioskuje posługując się tzw. *uporządkowaną rezolucją źródłową* (ang. *input ordered resolution*). Podstawową zaletę tej strategii stanowi wysoka efektywność, zakres jej wykorzystania ogranicza się jednak wyłącznie do zbioru klauzul Horna; tylko dla tego rodzaju formuł rezolucja źródłowa jest strategią pełną. Istnieje jednak prosta metoda pozwalająca rozszerzyć ją, kosztem niewielkiego zmniejszenia efektywności, do innej odmiany zasady rezolucji zwanej *eliminacją modeli* (ang. *model elimination*). Strategia ta, zaproponowana przez D.W.Lovelanda [3], posiada własność pełności w zbiorze dowolnych formuł logiki pierwszego rzędu. Wprowadzenie jej do programu prologowego uzyskuje się poprzez przekształcenie wszystkich klauzul według wzoru:

	klauzula wejściowa	klauzula wyjściowa
reguła	p(X, Y) :- q(X, X1), ..., ..., r(Xn, Y).	p(X, Y, Anc) :- NewAnc = [p(X,Z) | Anc], q(X, X1, NewAnc), ..., ..., r(Xn, Y, NewAnc).
fakt	p(X, Y).	p(X, Y, Anc).

Argumenty *Anc* oraz *NewAnc* implementują tzw. *listę przodków* (ang. *ancestors list*) [5]. W dalszej kolejności do zbioru klauzul definiujących każdy z predykatów, dodaje się klauzulę której zadaniem jest sprawdzenie, czy negacja tego predykatu znajduje się na liście przodków. Klauzula ta dla dowolnego literału pozytywnego *p* ma postać:

p(X1,..., Xn,Anc) :- unifiable_member(not_p(X1,...,Xn),Anc);

natomiast w przypadku literałów negatywnych:

not_p(X1,...,Xn,Anc):- unifiable_member(p(X1,...,Xn),Anc).

Predykat *unifiable_member(Elem, List)* jest prawdziwy, o ile *Elem* unifikuje się uwzględniając test wystąpień) z pewnym elementem listy *List*. Jego definicję należy dołączyć do programu wynikowego.

Podczas czwartego etapu przekształceń program wynikowy rozszerza się o mechanizmy modyfikujące technikę przeszukiwania grafu rezolucji. W Prologu, do powyższego celu zastosowano algorytm *DFS* (ang. *Depth First Search*), czyli "najpierw w głąb". Fakt ten sprawia, że jeśli dany predykat zdefiniowany jest przez kilka klauzul, to będą one wywoływane zawsze w porządku określonym przez kolejność występowania w tekście programu. Zasadniczym powodem wyboru tej strategii była efektywność oraz oszczędna gospodarka pamięcią komputera. Wynika to z faktu, że w toku działania DFS, konstruuje się zawsze tylko jedną rezolwentę. Przeszukiwanie grafu w głąb nie jest niestety strategią pełną. Jeżeli graf rezolucji zawiera cykl, to Prolog może popaść w nieskończoną pętlę pomimo, że dowód badanego celu istnieje i zostałby znaleziony w przypadku obrania innej drogi w grafie. Powyższej wady nie posiada strategia *CB-DFS* (ang. *Consequtively Bounded Depth First Search),* czyli przeszukiwania w głąb z iteracyjnym pogłębianiem. Jest ona przy tym optymalna ze względu na długość znalezionego wywodu, czas poszukiwania, oraz wykorzystanie pamięci, czego dowód znaleźć można w pracy R.E. Korfa [1]. Wprowadzenie do programu wynikowego algorytmów realizujących CB-DFS odbywa się poprzez przekształcenie wszystkich klauzul według następującego wzoru:

	klauzula wejściowa	**klauzula wyjściowa**
reguła	p(X1,...,Xn) :- q1(X11,...,X1n), ..., ...,....., qm(Xm1,...,Xmn).	p(X, Y, DpIn, DpOut) :- DpIn > m, DpIn1 is DpIn - m, q1(X11,...,X1n, DpIn1, DpIn2), ..., ..., qm(Xm1,...,Xmn,DpInm, DpOut).
fakt	p(X,Xn).	p(X, Y, DpIn, DpIn).
główny cel	query :- p(X1,...,Xn).	query :- gen_depth(D), p(X1,...,Xn, D, _).

Ponadto, do programu należy również dołączyć definicję predykatu *gen_depth/1*, generującego podczas nawrotu kolejną wartość głębokości.

W trakcie etapu piątego program wynikowy ulega przekształceniom umożliwiającym wygenerowanie dowodu rezolucyjnego badanej hipotezy. Jest nim ciąg nagłówków klauzul, których wywołanie doprowadziło do pomyślnego wykonania programu. Dowód o powyższej strukturze uzyskuje się poprzez zapamiętywanie wywołanych nagłówków na odpowiednio zorganizowanej liście. Listę tę przekazuje się za pomocą dwóch

dodatkowych argumentów, o które rozszerza się predykaty z programu. Przekształcenie to można opisać za pomocą poniższej tabeli:

	klauzula wejściowa	klauzula wyjściowa
reguła	p(X1,...,Xi) :- q1(A1,...,Aj), q2(B1,...,Bk), ..., ..., qm(Z1,...,Zl).	p(X1,...,Xi,L, [(p(X1,...,Xj) :- B)|L]) :- q1(A1,...,Aj, [], Pass1), q2(B1,...,Bk, Pass1, Pass2), ..., ..., qm(Z1,...,Zl, PassM, B).
fakt	p(X,...,Xi).	p(X,...Xi, L, [p(X,...,Xi) | L]).
główny cel	query :- ....., ..., p(X1,...,Xi).	query :- ....., ..., p(X1,...,Xn, [],P), print_proof(P).

Formatowanie i wydruk dowodu realizuje predykat *print_proof/1*, którego definicję dołącza się do programu wynikowego. Zaletą powyższej metody jest przede wszystkim to, że poza unifikacją dodatkowych argumentów w momencie wywoływania nagłówka celu bieżącego, nie wymaga ona żadnych dodatkowych działań podczas wykonywania programu.

Podsumowując, przedstawiony tutaj system służy do dowodzenie twierdzeń logiki pierwszego rzędu. Stanowi on rozszerzenie konstrukcji opisanej w [5], w porównaniu z którą posiada możliwość przyjmowania, jako danych wejściowych, dowolnych formuł rachunku predykatów bez konieczności wstępnego przekształcenia ich do specjalnej postaci. System wykorzystuje również nową metodę pozyskiwania i formowania tekstu dowodu, która jest znacznie bardziej efektywna od poprzedniej, ponadto pozwala na uzyskanie pełnego tekstu rezolucyjnego dowodu badanej hipotezy. Dość istotny problem stanowi niestety stosunkowo mała czytelność takiego dowodu. Problem ten wydaje się trudny do usunięcia, gdyż jest on konsekwencją podstawowych założeń, na których opiera się metoda PTTP.

LITERATURA

[1] Korf R. E.: Depth-first iterative-deepening:an optimal admissible tree search, *Artificial Intelligence* **27**, (1985) 97 - 109.

[2] Loveland D. W.: Automated Theorem Proving: A Logical Basis, *North-Holland Publishing Company*, Amsterdam 1978.

[3] Loveland D. W.: A simplified format for the model elimination procedure, *Journal of the AMC* **16**, 1969, 349-363.

[4] Plaisted D. A.: Non-Horn clause logic programming without contrapositives, *Journal of Automated Reasoning* **4, 3**, September 1988, 287 - 325.

[5] Stickel M. E.: A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog, *SRI Technical Note* **464**, June 1989.

ISBN 83-85847-85-5

W celu uzyskania bliższych informacji i zakupu dodatkowych egzemplarzy prosimy o kontakt
z Instytutem Badań Systemowych PAN
ul. Newelska 6, 01-447 Warszawa
tel. 36-19-01 w. 241 e-mail: kotuszew@ibspan.waw.pl