Procedura Davisa-Putnama

Rafał Kluszczyński

Luty 2004

1 Wprowadzenie

Jednym z zagadnień badanych w ramach informatyki teoretycznej jest automatyzacja procedury dowodzenia. W teorii rachunku zdań sprowadza się to do problemu rozważenia czy dane zdanie p jest tautologią, czyli takim zdaniem, które niezależnie od wartości logicznych występujących w nim zdań prostych, przyjmuje zawsze wartość logiczną $TRUE$. Jest to problem rozstrzygalny, gdyż jednym z najprostszych sposobów sprawdzenia jest wygenerowanie wszystkich możliwych bool'owskich wartościowań dla skończonej ilości zdań prostych, występujących w p (niech ich będzie n). Ponieważ dowolne zdanie proste może przyjać tylko jedno z dwóch wartości ($TRUE$ lub $FALSE$), wnioskujemy, iż wszystkich możliwych wartościowań będzie 2^n. Zatem widzimy, że problem tautologii ma złożoność wykładniczą, a ponieważ problem $P = NP$ wciąż należy do nierozwiązanych, nie ma algorytmu o mniejszej złożoności.

Sprawdzanie jednak wszystkich wartościowań jest podejściem "naiwnym" i niewątpliwie bardzo czasochłonnym, szczególnie dla dużych n. W tym celu powstało wiele algorytmów działających znacznie szybciej. Innym podejściem do dowodzenia tautologii jest aksjomatyczny system Hilberta, który opiera się na kilku aksjomatach i regule *"modus ponens"*, która okazała się niezwykle trudna do realizacji automatycznej. Dzieje się to dlatego, iż w regule tej należy wybrać podcel. Wybór ten niejednokrotnie przesądza o sukcesie wyprowadzenia dowodu, zatem jego trafność zależy głównie od inteligencji człowieka.

W tym momencie może się nam nasunąć watpliwość czy isnieje jakiś system aksjomatyczny, który da się zautomatyzować ? Okazuje sie, że odpowiedź jest twierdząca, a system taki jest wykorzystywany w rachunku sekwencyjnym. Operuje on na wyrażeniach nazywanych sekwentami i okazuje się, że rachunek tych sekwentów pozwala nam dowodzić tautologii. Jednak po

zapoznaniu sie z nim zauważamy, że, podobnie jak w systemie Hilberta, występuje tu trudna do automatyzacji reguła *"cut rule"*, której użycie wymaga najcenniejszego czynnika ludzkiego – inteligencji. Z pomocą przychodzi nam tutaj twierdzenie Gentzena, które mówi, iż każdy sekwent posiadający dowód w systemie sekwencyjnym, jeśli wykorzystuje regułę *"cut rule"*, to posiada również taki dowód, w którego wyprowadzeniu reguły tej nie ma.

Okazuje się zatem, że problem, czy dane zdanie jest tautologią, jesteśmy w stanie zautomatyzować. Innymi metodami wykorzystywanymi w tym celu są: metoda Semantic Tableaux i Rezolucja. Są to metody redukcyjne i odrzucające, tzn. zamiast pokazywać, że p jest tautologią, pokazują, że $\neg p$ jest niespełnialne. Obie metody opierają się nie na klasycznym podziale zdań, tylko na podziale na: literały, podwójne negacje, α-zdania i β-zdania. *Literały* są to dowolne zdania proste lub ich zaprzeczenia, *podwójnymi negacjami* będziemy określać zdania postaci $\neg(\neg p)$, natomiast z dowolnym *α-zdaniem* wiążą się dwa zdania α_1 i α_2 takie, że wartość logiczna α-zdania zależy od koniunkcji wartości logicznych zdań α_1 i α_2. Analogicznie jest dla *β-zdania*, tylko że jego wartość logiczna zależy od alternatywy wartości logicznych zdań β_1 i β_2.

Metoda Semantic Tableaux opiera się na analizie zdania w jego alternatywnej postaci normalnej (tzn. w postaci alternatywy pewnych koniunkcji, równoważnej danemu zdaniu). Strukturą danych jest drzewo, często określane "tablicą semantyczną", którego etykietami wierzchołków są zdania. Dowolną gałąź utożsamiamy z jedną z koniunkcji, natomiast całe drzewo ze zdaniem. W metodzie tej mamy cztery główne reguły dopisujące do struktury, a celem jest uzyskanie tzw. tablicy domkniętej, czyli takiej, aby wśród etykiet każdej gałęzi można było znaleźć parę zdań: p i $\neg p$. Widzimy wówczas, że każda koniunkcja, przy dowolnym wartościowaniu będzie niespełnialna (bo zawiera zdanie p i $\neg p$), a zatem alternatywa takich koniunkcji musi być niespełniona. Stąd wnioskujemy, iż tablicy domkniętej odpowiada zdanie niespełnione. Zatem sprawdzenie czy zdanie p jest tautologią sprowadza się w metodzie ST do zbadania, czy z jednowierzchołkowego drzewa o etykiecie $\neg p$ da się utworzyć drzewo domknięte.

W metodzie rezolucji sytuacja przedstawia się podobnie. Metoda ta opiera się na strukturze danych w postaci "listy list", która reprezentuje koniunktywną postać normalną badanego zdania p, czyli koniunkcję klauzul, gdzie przez pojęcie *klauzuli* rozumiemy alternatywę pewnych zdań. Reguły tej metody są regułami rozszerzającymi, natomiast celem jest doprowadzić przy użyciu tych reguł listę postaci $< [\ \neg p\ ] >$ do jej domkniętego rozwinięcia rezolucyjnego. Jest to "lista list" zawierająca listę pustą. Wartość logiczna klauzuli pustej [ ] wynosi $FALSE$, a zatem koniunkcja zawierająca taką klauzulę będzie również wartościowana na $FALSE$. Zatem zdanie reprezentowane przez tą "listę list" nie jest spełnialne.

2 Metoda *Davisa-Putnama*

Zanim powstała stosowana dzisiaj metoda rezolucji, oraz zanim jeszcze była znana notacja unifikacyjna, w roku 1960 została przedstawiona procedura *Davisa-Putnama*. Miała to być technika dowodząca odpowiednia dla efektywnej automatyzacji. Pomimo istnienia obecnie metody rezolucji, jest ona nadal jedną z efektywniejszych i szybszych wśród znanych nam metod.

Procedura *Davisa-Putnama*, podobnie jak metoda rezolucji, jest metodą odrzucającą. Chcąc udowodnić zdanie p, zaczynamy od jego negacji: $\neg p$. W pierwszym etapie sprowadzamy zdanie do koniunktywnej postaci normalnej, czyli do równoważnej zdaniu p postaci koniunkcji klauzul. Drugi etap procedury jest testem na niespełnialność tej właśnie koniunkcji.

Na potrzeby naszej procedury wprowadzimy jeszcze jedno dodatkowe pojęcie, mianowicie pojęcie *bloku*.

Definicja 1 *Blokiem nazywamy alternatywę zbioru (koniunkcji) klauzul.*

Stosując metodę *Davisa-Putnama* częścią strategii jest utrzymywanie jak najmniejszej ilości koniunkcji klauzul w bloku, preferowanie jednej. Samo działanie procedury natomiast polega na przekształcaniu bloku poprzez

proste reguły przepisywania. Syntaktycznie, wszystkie te przekształcenia polegają na zastępowaniu zbiorów klauzul. Semantycznie, każde przekształcenie nie powinno oddziaływać na spełnialność bloku, tzn. przekształcony blok musi być spełnialny wtedy i tylko wtedy, gdy oryginalny był spełnialny. Celem jest wyprowadzenie bloku, który w sposób oczywisty byłby spełnialny lub nie.

W tym celu dysponujemy dwoma rodzinami reguł. Pierwszą są reguły przygotowawcze, które nie są *stricte* potrzebne, ale mogą znacznie przyspieszyć działanie procedury. Drugą grupą są reguły główne, które stanowią esencję metody *Davisa-Putnama*.

3 Reguły przygotowawcze

Reguła *"Recurrent"* :

Usuwamy wszystkie powtórzenia w klauzulach w bloku i ewentualnie porządkujemy literały w pewnym porządku.

Oczywiście łatwo zauważyć, iż ten krok procedury nie wpływa na spełnialność, ponieważ na mocy rachunku zdań wiemy, że

$$p \vee p \Leftrightarrow p$$

jest tautologią.

Definicja 2 *Dla dowolnego zdania prostego* q *, definiujemy* $\overline{q} = \neg q$ *oraz* $\overline{\neg q} = q$*. Parę literałów* L *i* $\overline{L}$ *nazywamy komplementarnymi.*

Reguła *"Complementary"* :

Usuwamy ze zbioru klauzul wszystkie te, które zawierają literały komplementarne. Usuwamy również każdą klauzulę zawierającą $\top$, oraz każde wystąpienie $\bot$ w pozostałych klauzulach.

Tutaj także pominiemy formalny dowód zachowywania spełnialności. Łatwo bowiem widać, iż każda klauzula zawierająca komplementarne literały jest zawsze spełnialna na mocy tautologii $p \vee \neg p$. Wynika to z binarności wartościowania bool'owskiego, w wyniku którego zawsze przynajmniej jedno ze zdań p, albo $\neg p$ bedzie spełnione. Dodatkowo, każde zdanie prawdziwe (oznaczane symbolem $\top$) powoduje wartościowanie klauzuli w której występuje na $TRUE$, zatem w koniunkcji tych klauzul wystarczy rozważyć pozostałe, z których usuwamy zdania nieprawdziwe (oznaczane $\bot$), ponieważ te natomiast nie wpływają na wartość bool'owską alternatywy.

4 Reguły główne

Teraz zajmiemy się główną rodziną reguł metody *Davisa-Putnama*. Przedstawimy te reguły oraz podstawowe fakty dotyczące ich semantyki. Następnie pokażemy przykład działania procedury, by na zakończenie udowodnić jej zgodność i zupełność.

Reguła *"One-Literal"* :

> Załóżmy, że blok B zawiera koniunkcję klauzul S, w której występuje jednoliterałowa klauzula $[L]$. Modyfikujemy B usuwając z S wszystkie klauzule zawierające L, oraz wymazując wystąpienia $\overline{L}$ z pozostałych klauzul w S. (Przy zastosowaniu tej reguły mówimy, że została ona użyta *na* literale L).

Zauważmy, że aby udowodnić, iż reguła ta nie łamie spełnialności bloków, wystarczy udowodnić, że modyfikacje dokonane na jednej koniunkcji klauzul posiadają taką własność.

Twierdzenie 3 *Załóżmy, że koniunkcja klauzul S zawiera jednoliterałową klauzulę $[L]$. Niech S^* będzie wynikiem zastosowania reguły "One-Literal" na literale L. Wówczas S jest spełnialne wtedy i tylko wtedy, gdy S^* jest spełnialne.*

Dowód.

Ze względu na przejrzystość i łatwość notacji użyjemy konwencji języka Prolog do zapisu klauzul. Przez $[A|B]$ będziemy oznaczać klauzulę, której pierwszym elementem jest A, a B jest klauzulą zawierającą resztę elementów.

Przejdźmy teraz do sedna dowodu. Załóżmy, że S jest zbiorem (koniunkcją) klauzul postaci:

$$< [L], [L|C_1], \ldots, [L|C_n], [\neg L|D_1], \ldots, [\neg L|D_k], E_1, \ldots, E_j >,$$

gdzie wyróżnione literały L i $\neg L$ są zarazem ich jedynymi wystąpieniami w klauzulach. Wtedy S^* jest następującym zbiorem klauzul:

$$< D_1, \ldots, D_k, E_1, \ldots, E_j >.$$

Załóżmy najpierw, że S jest spełnialne. Oznacza to, że istnieje bool'owskie wartościowanie v przekształcające każdy element S na $TRUE$. W szczególności $v(L) = TRUE$, a także $v([\neg L|D_1]) = TRUE$. Ponieważ jednak $v(\neg L) = FALSE$, to musi zachodzić warunek, że $v(D_1) = TRUE$. Analogicznie dochodzimy do takiego samego wniosku dla kazdego D_i. Oczywiscie $v(E_1) = TRUE, \ldots, v(E_j) = TRUE$. Zatem każdy element S^* jest wartościowany na $TRUE$ przez v, więc S^* jest spełnialny.

Teraz załóżmy, że S^* jest spełnialne. Niech zatem wartościowanie bool'owskie v przekształca każdy element S^* na $TRUE$. Definujemy nowe bool'owskie wartościowanie w. Dla każdego literału P, różnego od L, kładziemy $w(P) = v(P)$. Natomiast dla literału L: $w(L) = TRUE$. Jest oczywiste, że dla każdej klauzuli K nie zawierającej wystąpienia L lub $\neg L$, $v(K) = w(K)$. Zatem $w(E_1) = TRUE, \ldots, w(E_j) = TRUE$, ponieważ $v(E_1) = TRUE, \ldots, v(E_j) = TRUE$. Rozumując tak dalej, ponieważ $v(D_1) = TRUE$, to $w(D_1) = TRUE$, a stąd $w([\neg L|D_1]) = TRUE$. Podobnie dla $[\neg L|D_2], \ldots, [\neg L|D_k]$. Analogicznie, jeśli $w(L) = TRUE$, zatem $w([L|C_1]) = TRUE, \ldots, w([L|C_n]) = TRUE$, oraz oczywiście $w([L]) = TRUE$. W ten sposób wartościowanie w przekształca każdy element S na $TRUE$, więc S jest spełnialne, co kończy dowód.

□

Reguła *"Affirmative-Negative"* :

Załóżmy, że blok B zawiera koniunkcję klauzul S, w której niektóre klauzule zawierają literał L, a żadna nie zawiera $\overline{L}$. Modyfikujemy B poprzez usunięcie z S wszystkich klauzul zawierających L. (Przy zastosowaniu tej reguły mówimy, że została ona użyta *na* literale L).

Twierdzenie 4 *Załóżmy, że koniunkcja klauzul S zawiera niektóre klauzule z literałem L, a pozostałe bez literału $\overline{L}$. Niech S^* będzie koniunkcją powstałą po zastosowaniu reguły "Affirmative-Negative" na literale L. Wówczas S jest spełnialne wtedy i tylko wtedy, gdy S^* jest spełnialne.*

Dowód.

Załóżmy, że S jest zbiorem (koniunkcją) klauzul w postaci:

$$< [L|C_1], \ldots, [L|C_n], E_1, \ldots, E_j >,$$

gdzie wyróżniony literał L jest zarazem jedynym jego wystąpieniem w klauzulach, oraz żadna z klauzul $E_1, \ldots, E_j$ nie zawiera $\overline{L}$. Wtedy S^* jest następującym zbiorem klauzul:

$$< E_1, \ldots, E_j >.$$

Załóżmy, że S jest spełnialne. Oznacza to, że istnieje wartościowanie v przekształcające każdy element (klauzulę) S na $TRUE$. W szczególności mamy, że $v(E_1) = TRUE, \ldots, v(E_j) = TRUE$, a zatem koniunkcja klauzul S^* jest spełnialna.

Załóżmy teraz, że S^* jest spełnialne. Zatem niech v bedzie wartościowaniem przekształcającym każdy element S^* na $TRUE$. Definujemy nowe wartościowanie w, takie, że dla literałów z klauzul $E_1, \ldots, E_j$ zachodzi tożsamość $w \equiv v$. Natomiast dla literału L kładziemy $w(L) = TRUE$. Wynika stąd, iż dla dowolnej klauzuli C_1 mamy $w([L|C_1]) = TRUE$. Analogicznie otrzymujemy, że $w([L|C_2]) = TRUE, \ldots, w([L|C_n]) = TRUE$, więc istnieje wartościowanie w, przy którym S jest spełnialne.

□

Zanim przejdziemy do kolejnej reguły *Davisa-Putnama*, wprowadzimy pojęcie zawierania się klauzul.

Definicja 5 *Klauzula C_1 zawarta jest w klauzuli C_2, jeśli każdy literał z C_1 występuje również w C_2.*

Niech przykładowe klauzule będą postaci:

$$C_1 = [A_1, \ldots, A_n],$$
$$C_2 = [A_1, \ldots, A_n, D_1, \ldots, D_m].$$

Widzimy, że C_1 zawiera się w C_2, co w skrócie będziemy zapisywać $C_1 \leqslant C_2$. Ponieważ wiemy, że klauzula jest alternatywą literałów, stąd jeśli C_1 jest niespełnialne, to C_2 także. Nasuwa się więc wniosek, iż wystarczy badać tylko klauzulę zawartą w drugiej. Właśnie to spostrzeżenie wykorzystuje następna reguła.

Reguła *"Subsumption"* :

Załóżmy, że blok B zawiera koniunkcję klauzul S, w której występują klauzule C_1 i C_2 takie, że $C_1 \leqslant C_2$. Modyfikujemy B poprzez usunięcie z S klauzuli C_2.

Twierdzenie 6 *Załóżmy, że koniunkcja klauzul S zawiera klauzule C_1 i C_2 takie, że $C_1 \leqslant C_2$. Niech S^* będzie wynikiem zastosowania reguły "Subsumption" na S. Wówczas S jest spełnialne wtedy i tylko wtedy, gdy S^* jest spełnialne.*

Dowód.

Załóżmy, że S jest następującym zbiorem (koniunkcją) klauzul:

$$< C_1, C_2, E_1, \ldots, E_j >,$$

gdzie $C_1 \leqslant C_2$. Wówczas S^* wygląda następująco:

$$< C_1, E_1, \ldots, E_j >.$$

Załóżmy, że S jest spełnialne. Oznacza to, że istnieje wartościowanie v przekształcające każdy element (klauzulę) S na $TRUE$. Mamy zatem

$v(C_1) = TRUE, v(E_1) = TRUE, \ldots, v(E_j) = TRUE$, a zatem koniunkcja klauzul S^* jest spełnialna.

Załóżmy teraz, że S^* jest spełnialne. Niech v bedzie wartościowaniem przekształcającym każdy element S^* na $TRUE$. Wynika stąd, że dla klauzul $C_1, E_1, \ldots, E_j$ zachodzą następujące równości

$$v(C_1) = TRUE,\ v(E_1) = TRUE,\ \ldots\ ,\ v(E_j) = TRUE.$$

Natomiast dla dowolnej klauzuli C_2 takiej, że $C_1 \leqslant C_2$, skoro $v(C_1) = TRUE$, to również $v(C_2) = TRUE$. Zatem wartościowanie v spełnia także koniunkcję S.

□

Reguła *"Splitting"* :

Załóżmy, że blok B zawiera koniunkcję klauzul S, w której niektóre klauzule zawierają literał L, a inne $\overline{L}$ (mogą także wystąpić klauzule zarówno bez L jak i $\overline{L}$). Modyfikujemy B poprzez zastąpienie S dwoma koniunkcjami S_L i $S_{\overline{L}}$, gdzie:

- S_L to koniunkcja klauzul powstała z S poprzez usunięcie wszystkich klauzul w S zawierających L, oraz wymazanie $\overline{L}$ z pozostałych klauzul,
- $S_{\overline{L}}$ to koniunkcja klauzul powstała z S poprzez usunięcie wszystkich klauzul w S zawierających $\overline{L}$, oraz wymazanie L z pozostałych klauzul.

(Przy zastosowaniu reguły mówimy, że została ona użyta *na* literale L).

Twierdzenie 7 *Załóżmy, że w koniunkcji klauzul S występują klauzule zawierające literał L, oraz inne zawierające $\overline{L}$. Niech S_L i $S_{\overline{L}}$ będą koniunkcjami klauzul powstałymi w wyniku zastosowania reguły "Splitting" na literale L. Wówczas S jest spełnialne wtedy i tylko wtedy, gdy jest spełnialne S_L lub $S_{\overline{L}}$.*

Dowód.

Załóżmy, że S jest następującym zbiorem (koniunkcją) klauzul:

$$< [L|C_1], \ldots, [L|C_n], [\neg L|D_1], \ldots, [\neg L|D_m], E_1, \ldots, E_j >,$$

gdzie wyróżnione literały L i $\neg L$ są zarazem ich jedynymi wystąpieniami, czyli nie występują w żadnej z klauzul $C_1, \ldots, C_n, D_1, \ldots, D_m, E_1, \ldots, E_j$. Wówczas S_L i $S_{\overline{L}}$ są następującymi zbiorami klauzul:

$$S_L =< D_1, \ldots, D_m, E_1, \ldots, E_j >,$$
$$S_{\overline{L}} =< C_1, \ldots, C_n, E_1, \ldots, E_j >.$$

Załóżmy, że S jest spełnialne. Oznacza to, że istnieje bool'owskie wartościowanie v przekształcające każdy element z S na $TRUE$. Mamy zatem:

$$v([L|C_1]) = TRUE \ , \ \ldots \ , \ v([L|C_n]) = TRUE,$$
$$v([\neg L|D_1]) = TRUE \ , \ \ldots \ , \ v([\neg L|D_m]) = TRUE,$$
$$v(E_1) = TRUE \ , \ \ldots \ , \ v(E_j) = TRUE.$$

Teraz istnieją dwie możliwości, albo literał L jest wartościowany przez v na $TRUE$, albo na $FALSE$:

1. jeśli $v(L) = TRUE$, to skoro $v([\neg L|D_1]) = TRUE$, zatem $v(D_1) = TRUE$. W ten sam sposób otrzymujemy, że $v(D_2) = TRUE, \ \ldots \ , \ v(D_m) = TRUE$, co sprowadza nas do wniosku, iż koniunkcja klauzul S_L jest spełnialna,

2. jeśli natomiast $v(L) = FALSE$, to wówczas analogicznie otrzymujemy, że $v(C_1) = TRUE, \ \ldots \ , \ v(C_n) = TRUE$. Zatem wówczas koniunkcja $S_{\overline{L}}$ jest spełnialna.

Przejdźmy teraz do implikacji w drugą stronę. Załóżmy, że przynajmniej jedna z koniunkcji S_L, $S_{\overline{L}}$ jest spełnialna. Na początek niech to będzie S_L. Istnieje zatem bool'owskie wartościowanie v przekształcające każdy element S_L na $TRUE$. Definujemy nowe wartościowanie w takie, że dla każdego literału P, występującego w klauzulach $D_1, \ldots, D_m, E_1, \ldots, E_j$ kładziemy $w(P) = v(P)$. Dla literału L natomiast, określamy $w(L) = TRUE$.

Stąd otrzymujemy, że $w([L|C_1]) = TRUE, \ \ldots \ , w([L|C_n]) = TRUE$. Wiedząc z założenia, iż $v(D_1) = TRUE, \ \ldots \ , v(D_m) = TRUE$ wnioskujemy następujące równości $w([\neg L|D_1]) = TRUE, \ \ldots \ , w([\neg L|D_m]) = TRUE$. Oczywiście mamy, że $w(E_1) = TRUE, \ \ldots \ , w(E_j) = TRUE$. Zatem nowe wartościowanie w przekształca wszystkie elementy S na $TRUE$, więc S jest spełnialne.

W przypadku, gdy koniunkcja S_L będzie niespełnialna, czyli spełnialna będzie $S_{\overline{L}}$, wówczas rozumowanie będzie analogiczne. Definiujemy również nowe wartościowanie bool'owskie w, które dla literałów z klauzul $C_1, \ldots, C_n, E_1, \ldots, E_j$ działa tak jak isniejące wartościowanie przekształcające każdy element $S_{\overline{L}}$ na $TRUE$. Dodatkowo dla literału L określamy $w(L) = FALSE$, skąd otrzymujemy, że $w([\neg L|D_1]) = TRUE, \ \ldots \ , w([\neg L|D_m]) = TRUE$. Oczywiście ponieważ $w(C_1) = TRUE, \ \ldots \ , w(C_n) = TRUE$, to również $w([L|C_1]) = TRUE, \ \ldots \ , w([L|C_n]) = TRUE$. Zatem widzimy, że także w przypadku spełnialności tylko koniunkcji $S_{\overline{L}}$, istnieje wartościowanie spełniające S, co kończy dowód.

□

Była to ostatnia reguła metody *Davisa-Putnama*. Na mocy przedstawionych tu również twierdzeń wnioskujemy, iż żadna z tych reguł nie łamie spełnialności, co wkrótce będzie nam niezwykle przydatne. Teraz natomiast zajmiemy się sposobem ich używania.

5 Stosowanie metody *Davisa-Putnama*

Znajomość samych reguł jednak może okazać się niewystarczająca. Musimy bowiem wiedzieć jaki stan bloku odpowiada w naszym przypadku za sukces, a jaki za porażkę. W tym celu przydatna okazuje się następująca terminologia.

Definicja 8 *Niech B będzie blokiem. Wyprowadzeniem Davisa-Putnama dla B nazywamy skończoną sekwencję bloków $B_1, \ldots, B_n$, gdzie $B_1 = B$. Dodat-*

kowo każdy blok w sekwencji pochodzi od jego poprzednika przy użyciu jednej z czterech reguł przepisywania.

Mówimy, że wyprowadzenie zakończyło się sukcesem, jeśli zakończyliśmy pracę z blokiem, w którym każda koniunkcja klauzul zawiera klauzule pustą. Natomiast wyprowadzenie kończy się porażką, jeśli kończymy pracę z blokiem zawierającym przynajmniej jedną, pustą koniunkcję klauzul.

Chcąc więc udowodnić zdanie p metodą *Davisa-Putnama*, zaczynamy od jego zaprzeczenia $\neg p$. Następnie przekształcamy je do koniunktywnej postaci normalnej S. Z otrzymanej koniunkcji klauzul formujemy blok $[S]$, prawdopodobnie upraszczamy używając reguł przygotowawczych. Na koniec pokazujemy, że istnieje wyprowadzenie Davisa-Putnama, które kończy się sukcesem.

Zanim przejdziemy do przykładów przedstawiających działanie procedury *Davisa-Putnama*, przypomnijmy tylko, że symbolem [ ] oznaczamy listę reprezentującą alternatywę, natomiast symbolem $< >$ koniunkcję.

Przykład 1.

Na początek zastosujmy nowopoznaną technikę dla znanego wszystkim jednego z praw *de Morgana* :

$$(p \wedge q) \Leftrightarrow \neg(\neg p \vee \neg q).$$

1. Dokonujemy zaprzeczenia formuły : $\neg(\ (p \wedge q) \Leftrightarrow \neg(\neg p \vee \neg q)\ )$.

2. Przekształcamy do koniunktywnej postaci normalnej:

$$[< [p,p],[p,q],[p,\neg q,\neg p],[q,p],[q,q],[q,\neg q,\neg p],$$
$$[p,\neg q,\neg p],[q,\neg q,\neg p],[\neg q,\neg p,\neg q,\neg p] >].$$

3. Usuwamy powtórzenia w klauzulach:

$$[< [p],[p,q],[p,\neg q,\neg p],[q,p],[q],[q,\neg q,\neg p],$$
$$[p,\neg q,\neg p],[q,\neg q,\neg p],[\neg q,\neg p] >].$$

4. Usuwamy klauzule z komplementarnymi literałami:

$$[< [p], [p, q], [q, p], [q], [\neg q, \neg p] >].$$

5. Stosujemy regułę *"One-Literal"* na literale p :

$$[< [q], [\neg q] >].$$

6. Raz jeszcze używamy regułę *"One-Literal"*, tym razem na literale q, otrzymując ostateczny blok:

$$[< [\,] >].$$

Zakończyliśmy zatem pracę z blokiem zawierającym jedną koniunkcję klauzul, w której występuje tylko klauzula pusta. Stąd wniosek, iż dla naszej formuły istnieje takie wyprowadzenie *Davisa-Putnama*, które kończy sie sukcesem. Zatem pokazaliśmy, że podane wyżej prawo *de Morgana* jest rzeczywiście tautologią.

Przykład 2.

Spróbujmy teraz rozstrzygnąć czy następujące zdanie :

$$(p \vee q) \Rightarrow (p \wedge q)$$

jest tautologią.

1. W tym celu rozważamy zaprzeczenie zdania: $\neg(\ (p \vee q) \Rightarrow (p \wedge q)\ )$.

2. Sprowadzamy do postaci koniunkcji klauzul i tworzymy odpowiedni blok :

$$[< [p, q], [\neg p, \neg q] >].$$

3. Stosujemy regułę *"Splitting"* na literale q :

$$[< [p] >, < [\neg p] >].$$

4. Dla każdej koniunkcji klauzul w bloku możemy teraz zastosować, albo regułę *"One-Literal"*, albo *"Affirmative-Negative"*. W obu przypadkach uzyskamy postać bloku :

$$[< >, < >].$$

Otrzymany blok jest blokiem końcowym. Niektóre z występujących w nim koniunkcji klauzul (w naszym przypadku obie) są puste. Zatem wyprowadzenie *Davisa-Putnama* zakończyło sie porażką, stąd zdanie $(p \vee q) \Rightarrow (p \wedge q)$ nie jest tautologią.

Przykład 3.

Posługując się już biegle poznanymi regułami przepisywania, rozważymy jeszcze jedno zdanie :

$$(p \Leftrightarrow q) \vee (p \Leftrightarrow \neg q).$$

1. Zaczynamy oczywiście od negacji zdania : $\neg(\ (p \Leftrightarrow q) \vee (p \Leftrightarrow \neg q)\ )$.

2. Przekształcamy w blok reprezentujący koniunktywną postać normalną:

$$[< [p, q], [p, \neg p], [\neg q, q], [\neg q, \neg p], [p, \neg q], [p, \neg p], [q, \neg q], [q, \neg p] >].$$

3. Po zastosowaniu reguły przygotowawczej *"Complementary"* mamy:

$$[< [p, q], [\neg q, \neg p], [p, \neg q], [q, \neg p] >].$$

4. Następnie stosujemy regułę *"Splitting"* na literale p :

$$[< [\neg q], [q] >, < [q], [\neg q] >].$$

5. Korzystając teraz z reguły *"One-Literal"* na literale q dla obu koniunkcji klauzul w bloku, otrzymyjemy ostatecznie :

$$[< [\,] >, < [\,] >].$$

Każda koniunkcja klauzul w bloku zawiera klauzulę pustą, więc wyprowadzenie sie powiodło. Stąd wniosek, że badane zdanie jest tautologią.

6 Zgodność i zupełność procedury

Zapoznając się z regułami metody *Davisa-Putnama* zauważyliśmy na pewno, że nie określiliśmy porządku, w jakim te reguły mają być stosowane. Czyli w procedurze tej istnieje pewien stopień niedeterminizmu, który zawsze jest przeszkodą w automatyzacji zadań. Co ciekawe, pokażemy, że ten niedeterminizm nie wypływa na wynik działania metody, czyli że procedura *Davisa-Putnama* jest *de facto* procedurą decyzyjną.

Dysponując regułami oraz pojęciem wyprowadzenia *Davisa-Putnama* pojawiają sie, podobnie jak przy innych znanych metodach, dwa podstawowe pytania.

1. Czy przypadkiem nie udowadniamy rzeczy nieprawdziwych ? (pytanie o zgodność),

2. Czy udowadniamy wszystkie rzeczy prawdziwe ? (pytanie o zupełność).

Z pomocą odpowiedzi na te pytania przychodzi następujące twierdzenie.

Twierdzenie 9 *Załóżmy, że podjęto próbę dowodu formuły ϕ stosując procedurę Davisa-Putnama i próba ta jest kontynuowana dopóki można zastosować jedną z reguł.*

Każda taka próba musi się zakończyć, albo sukcesem, albo porażką. Jeśli zakończy się sukcesem, to ϕ jest tautologią, w przeciwnym przypadku nie jest.

Dowód.

Na początek pokażemy, że każda zakończona próba dowodu, kończy się, albo sukcesem (czyli każda koniunkcja klauzul zawiera klauzule pustą), albo porażką (czyli istnieje pewna pusta koniunkcja klauzul). Dowiedziemy tego przez sprzeczność. Załóżmy, że mamy blok B zawierający jakąś koniunkcję klauzul S, która nie jest pusta i nie zawiera klauzuli pustej. Ponieważ S

nie jest puste, możemy z niej wybrać klauzulę C. Ponadto, skoro S nie zawiera klauzuli pustej, C samo w sobie nie jest puste. Wybieramy zatem z C literał L. Jeśli $\neg L$ nie pojawia się w którejkolwiek z klauzul w S, możemy zastosować regułe *"Affirmative-Negative"*. Jeśli natomiast $\neg L$ pojawia się w niektórych klauzulach w S, to możemy zastosować regułę *"Splitting"* (lub regułę *"One-Literal"*). W obu z tych przypadków nie mamy zakończenia wyprowadzenia.

Kolejną własnością, którą powinniśmy dowieść dla naszej metody, to wykazanie, że każda próba dowodu musi się zakończyć. Wystarczy zauważyć, że każda reguła, jedynie poza *"Subsumption"*, redukuje liczbę literałów, które pojawiają się w koniunkcjach klauzul. Skoro natomiast rozpoczynamy z ich skończoną liczbą, to reguły można zastosować tylko skończoną liczbę razy. Reguła *"Subsumption"* mianowicie usuwa klauzule, których również jest skończona liczba, więc ją także można zastosować tylko skończoną ilość razy.

Ostatnim punktem naszego dowodu będzie pokazanie, że każde zakończenie procedury sukcesem oznacza, iż ϕ jest tautologią, natomiast zakończenie porażką oznacza, że nie jest. Przypomnijmy, że ϕ jest tautologią wtedy i tylko wtedy, gdy $\neg\phi$ nie jest spełnialne. Każda próba dowodu zaczyna się od przekształcenia formuły $\neg\phi$ do koniunktywnej postaci normalnej S, a następnie sformułowania bloku $[S]$. Ponieważ każde wartościowanie bool'owskie przypisze tą samą wartość prawdy ($TRUE$) lub fałszu ($FALSE$) dla $\neg\phi$ i $[S]$, zatem ϕ jest tautologią wtedy i tylko wtedy, gdy blok $[S]$ jest niespełnialny. Wiedząc natomiast, że reguły nie wpływają na spełnialność, $[S]$ będzie spełnialne wtedy i tylko wtedy, gdy ostateczny blok w wyprowadzeniu *Davisa-Putnama* jest spełnialny. Ale blok zawierający pustą koniunkcję klauzul jest spełnialny, podczas gdy blok, gdzie każda koniunkcja klauzul zawiera klauzule pustą nie jest.

□

7 Podsumowanie

Pokazaliśmy zatem, że reguły *Davisa-Putnama* są niedeterministyczne. Według powyżej udowodnionego twierdzenia, dowolny porządek zastosowania tych reguł wyprodukuje dowód, jeśli tylko jest on wyprowadzalny. Oczywiście, niektóre próby będą szybsze niż inne. Łatwo daje się zauważyć, że reguła *"Splitting"* zwiększa liczbę przypadków do rozważania, więc jej użycie powinno być jak najdłużej odkładane. Z drugiej strony, reguła *"One-Literal"* ogranicza liczbę klauzul, które musimy rozważyć bez wprowadzania komplikacji, powinna być zatem preferowana przed użyciem pozostałych reguł.

Reasumując, porządek w jakim reguły te zostały przedstawione, jest dobrym porządkiem dla ich stosowania.

Literatura

[1] Fitting, M. (1996) *First–Order Logic and Automated Theorem Proving.* Graduate texts in Computer Science, Springer, Berlin, 2nd edition.

[2] Notatki z wykładów pt. „*Logika Matematyczna*" (2003/2004) prowadzonych przez prof. dr hab. G. Jarzembskiego na Wydziale Matematyki i Informatyki UMK w Toruniu.