Wnioskowanie w przód

Aplikacja zrealizowana w ramach projektu z przedmiotu
Podstawy sztucznej inteligencji w semestrze 16Z

Wiktor Franus
Hubert Jatkowski
Grzegorz Staniszewski

Warszawa, 17 stycznia 2017

1. Definicja

Wnioskowaniem w przód (Modus Ponendo Ponens) nazywa się progresywny algorytm sztucznej inteligencji służący do tworzenia nowych zdań logicznych na podstawie istniejącej bazy faktów i zbioru reguł.

2. Algorytm wnioskowania w przód

```
F – zbiór faktów
R – zbiór reguł
1. Wybierz ze zbioru R te reguły, których wszystkie przesłanki
   znajdują się w zbiorze F.
2. Wybierz jedną regułę i aktywuj ją – dodaj konkluzję wybranej reguły
   do F, usuń wybraną regułę ze zbioru R.
3. Jeśli cel znajduje się z zbiorze F lub jeśli R jest pusty -
   zakończ. W przeciwnym razie idź do punktu 2.
```

3. Struktura pliku wejściowego z bazą wiedzy

Zgodnie z założeniami na dane wejściowe programu składają się fakty pierwotne oraz reguły modelowanego świata przedstawione w koniuncyjno-implikacyjnym rachunku predykatów. Fakty i reguły powinny zostać umieszone w jednym pliku tekstowym. Poprawne działanie algorytmu wnioskowania jest zapewnione, jeśli plik wejściowy będzie utworzony według następujących reguł:

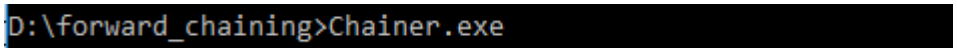
1. Fakty reprezentowane są przez pojedyncze wyrazy.
2. Każdy fakt wpisany jest w osobnej linii pliku lub kilka faktów znajduje się w jednej linii i każdy z nich oddzielony jest od drugiego znakiem koniunkcji ‘&’ (znak ASCII 38)
3. Każda reguła znajduje się w osobnej linii
4. Reguła reprezentowana jest przez przesłanki, znak implikacji ‘=>’ (ASCII 61 i 62) oraz konkluzję. Przesłanki mają taki sam format jak fakty (pojedyńczy predykat lub koniunkcja predykatów). Konkluzję stanowi jeden predykat
5. Kolejność w jakiej zapisane są fakty i reguły jest dowolna

Poniżej zaprezentowana jest zawartość przykładowego pliku z danymi wejściowymi zbudowanego zgodnie z powyższymi regułami.

```
~~a
~~~b
c&d
~b=>f
f=>q
f&z=>w
q=>~a
```

4. Instrukcja użytkownika

Skompilowany program składa się z jednego pliku wykonywalnego, który uruchamiany jest bez żadnych argumentów. Np. na systemie Windows:



Po uruchomieniu uzytkownik zostanie poproszony o podanie ścieżki do pliku zawierającego bazę więdzy.

```
D:\forward_chaining>Chainer.exe
Sciezka do pliku z wiedza:
```

Należy podać ścieżkę do istniejącego pliku. W przypadku problemu z otwarciem pliku lub podania błędnej ścieżki program zwróci komunikat z błędem.

```
D:\forward_chaining>Chainer.exe
Sciezka do pliku z wiedza:
nieistniejacy_plik.txt
Blad odczytu pliku.
```

Następnie użytkownik proszony jest o wpisanie celu/hipotezy, którą algorytm ma spróbować wywnioskować z zadanej bazy wiedzy.

```
D:\forward_chaining>type in2
a&~z
~c
d
e=>f
~z&~c=>e
f&~c=>d
f&~z=>k
k=>l
```

```
D:\forward_chaining>Chainer.exe
Sciezka do pliku z wiedza:
in2
Zapytanie:
```

Rezulatem działania algorytmu jest lista predykatów, które posłużyły do generowania kolejnych faktów oraz stwierdzenie “Tak” lub “Nie” świadczące odpowiednio o sukcesie lub porażce.

```
D:\forward_chaining>Chainer.exe
Sciezka do pliku z wiedza:
in2
Zapytanie:
l
Tak: ~c, ~z, e, f, ~c, f, ~z, k
```

5. Opis struktury programu

Aplikacja napisana została w języku C++, korzysta z bibliotek standardowych tego języka, a jej kompilacja i uruchomienie możliwe jest zarówno na systemach Unix, jak i Windows.
Kod źródłowy podzielony został na 11 plików: 6 plików nagłówkowych, 4 plików zawierających definicje funkcji oraz pliku main.cpp. Wyróżnić można 2 moduły:

- DataReader – moduł odpowiedzialny za wczytanie bazy wiedzy z pliku do przekazanych mu struktur danych. Jego zadaniem jest również uproszczenie predykatów zgodnie z prawem podwójnej negacji – wielokrotne znaki negacji poprzedzające symbol w bazie biedzy zastępowane są pojedynczym symbolem negacji lub symbol usuwany jest całkowicie
- ForwardChainer – zasadniczy moduł aplikacji zawierający funkcję wnioskującą oraz definicje pomocniczych struktur używanych przez ten moduł

6. Wnioski

Aplikacja poprawnie wnioskuje dla przykładowych danych, ostrzega przed sprzecznością w danych wykrytą zarówno podczas wczytywania, jak i w etapie wnioskowania. Jednym z założeń projektowych, które nie zostało ostatecznie zaimplementowane jest możliwość wyboru kryterium wyboru reguł ze zbioru konfliktowego. Obecnie program podczas wnioskowania wybiera zawsze pierwszą regułę ze zbioru reguł, która może zostać "aktywowana" dla aktualnego zbioru odwiedzonych faktów. Umożliwienie użytkownikowi wyboru innego kryterium nie jest skomplikowane i może zostać zaimplementowane w kolejnych wersjach aplikacji.
Innym sugerowanym przez autorów rozwinięciem aplikacji może być operowanie przez algorytm na klauzulach, a nie implikacjach. Wejściowa baza reguł byłaby zawsze najpierw sprowadzana do zbioru klauzul, co w rezultacie, dla pewnych przypadków, może dawać wyższą skuteczność algorytmu.

Dla przykładu:
Fakty: $b, \neg c$
Reguła: $a \wedge b \Rightarrow c$
Cel: $\neg a$
Wnioskowanie z użyciem samych implikacji nie jest w stanie udowodnić celu. Algorytm kończy się porażką. Jeśli z kolei dostępną regułę przedstawimy za pomocą klauzuli to wywnioskowanie celu staje się możliwe.

$$a \wedge b \Rightarrow c \;\Leftrightarrow \neg a \vee \neg b \vee \neg c$$

Uzyskana klauzula generuje dwie implikacje nieobecne w początkowym zbiorze reguł:

$$b \wedge \neg c \Rightarrow \neg a$$
$$a \wedge \neg c \Rightarrow \neg b$$

Kolejnym udogodnieniem możliwym do realizacji w przyszłości jest akceptacja więcej niż jednego celu. Skupić można się również na prezentacji rezultatów. Obecnie wyświetlane są: binarny werdykt oraz lista predykatów, które wchodzą w skład przesłanek aktywowanych reguł. Aplikacja mogłaby prezentować użytkownikowi kolejne etapy wnioskowania poprzez rysowanie tzw. grafu wnioskowania.