

Marcin Woliński

Dokumentacja techniczna platformy gramatycznej Birnam

(wersja bardzo wstępna)

15 lipca 2010

Tutaj powinien być opis, jak to implementowane przez nas gramatyki uwzględniają wszystkie możliwości DCG z ograniczeniami wynikającymi z analizy wstępującej, czyli nie może być reguł z pustą prawą stroną. Na razie jednak jest tylko to, co ciekawe:

1. Elementy opcjonalne reguł

Metanieterminal `optional` pozwala wprowadzić opcjonalny nieterminal do reguły. Z powodów technicznych nie można tego zrobić z inicjalnym nieterminalem reguły. Metanieterminal ma 3 argumenty: nieterminal, warunki obliczane, jeśli nieterminal wystąpił i warunki obliczane w przeciwnym wypadku. Warunki zapisujemy w nawiasach klamrowych, pusta para nawiasów sygnalizuje brak warunków.

W następującym przykładzie mamy opcjonalny przecinek. Przy obecności nieterminala `przec` zmienna `Pk3` jest zadawana przez ten nieterminal i nie są potrzebne dodatkowe warunki. Jeżeli go brak, zmienna `Pk3` przyjmuje wartość wskazaną w warunku.

```
fl(..., Pk) -->
przec(Pk1),
fno(..., Pk2),
optional( przec(Pk3), {}, { Pk3=(+|-) } ),
{ obl_pk(Pk, [Pk1,Pk2,Pk3]) }.
```

2. Sekwencje nieterminali i warunki iterowane

Istotnym rozszerzeniem w stosunku do DCG jest możliwość zadania po prawej stronie reguły, że pewne nieterminale mogą wystąpić zero lub więcej razy. Zauważmy, że ma to w szczególności istotne znaczenie dla kształtu powstających drzew analizy, które mogą zawierać wierzchołki dowolnego rzędu. Nie jest to możliwe w czystym DCG.

2.1. Proste sekwencje

Sekwencje nieterminali zadaje się za pomocą metanieterminala `sequence_of`. Na przykład następująca reguła głosi, że zdanie składa się z jednostki `ff`, po której następuje zero lub więcej jednostek `fw`:

```

zdanie -->
    ff,
    sequence_of([
        fw
    ]).

```

Argumentem `sequence_of` jest lista nieterminali. Jeżeli zawiera ona więcej niż jeden element, każdy z nich może występować w dowolnej liczbie i kolejności. Na przykład reguła

```

zdanie -->
    ff,
    sequence_of([
        fw,
        fl
    ]).

```

pozwala wygenerować m.in. ciąg `ff fw fw fl fw`, albo `ff fl fl fl fl fl fl`, jak również samo `ff`.

2.2. Warunki w sekwencji

Istotną cechą gramatyk DCG jest użycie argumentów nieterminali i nakładanie warunków na nie. Wypada zacząć od uwagi, że domyślnie zmienne występujące w argumentach `sequence_of` są dzielone z otaczającą regułą. Tak więc w następującym przykładzie pierwszy argument jednostki `ff` musi być taki sam jak pierwszy argument w każdym wystąpieniu jednostki `fw` w sekwencji (jeżeli takie są):

```

zdanie -->
    ff(A),
    sequence_of([
        fw(A)
    ]).

```

Nakładanie warunków na sekwencje jest możliwe w dwóch wariantach: dla każdego elementu sekwencji może być wymagany ten sam warunek lub też na całą sekwencję może być nałożony warunek iterowany.

2.2.1. Warunki lokalne dla wystąpienia nieterminala

Pierwszą możliwość ilustruje następujący przykład:

```

zdanie -->
    ff(A),
    sequence_of([
        fw(A, Neg)
        ^[sprawdź_neg, Neg]
    ]).

```

```

sprawdź_neg(ani).
sprawdź_neg(nie).

```

Warunek `sprawdź_neg` sprawdza, że jego argument jest równy `ani` lub `nie`. Zapis `^[sprawdź_neg, Neg]` wskazuje, że warunek ten ma być zastosowany do zmiennej `Neg` dla każdego wystąpienia `fw` w sekwencji z osobna. W szczególności zmienna `Neg` nie jest uzgadniana z otaczającą regułą ani z innymi

elementami sekwencji. Dla każdego wystąpienia fw jest to nowa zmienna swobodna.

2.2.2. Warunki iterowane po wszystkich elementach sekwencji

Warunki iterowane są nieco bardziej skomplikowane. Można za ich pomocą wyrazić warunek, który dla sekwencji o dowolnej długości da się obliczyć krokowo. Zaczynamy od pewnej wartości początkowej, a następnie dla każdego elementu sekwencji na podstawie tej wartości i wskazanej wartości specyficznej dla danego elementu obliczamy pewien wynik. Wynik ten staje się wartością początkową dla następnego wystąpienia elementu. Wynik obliczenia dla ostatniego elementu sekwencji staje się wynikiem dla całości.

Dla przykładu wyobraźmy sobie, że mamy jednostkę suma rozpisującą się na niepusty ciąg składników. Każdy składnik ma argument wyrażający jego wartość, chcielibyśmy obliczyć wartość sumy, czyli sumę składników:

```
suma(S) -->
  składnik(S1),
  sequence_of([
    składnik(S2)
    ^[dodaj,S1,S2,S]
  ]).
```

```
dodaj(Skł1,Skł2,Suma) :- Suma is Skł1 + Skł2.
```

Zapis `^[dodaj,S1,S2,S]` definiuje warunek iterowany. Pierwszy element nazywa predykat, który wykonuje krok obliczenia — w tym wypadku `dodaj`. Na mocy konwencji musi to być predykat trójargumentowy. Następne zmienne staną się odpowiednimi argumentami tego predykatu. `S1` — na początku iteracji. `S2` jest zmienną „lokalną” dla elementu, która nie uzgadnia się z otoczeniem, podobnie jak `Neg` w poprzednim przykładzie. `S` jest zmienną, która zostanie uzgodniona z wynikiem obliczenia po ostatnim kroku iteracji.

Oto przykład realizacji prawej strony tej reguły (pierwszy element jest poza `sequence_of`) i lista warunków, które zostaną kolejno obliczone:

```
składnik(S1) składnik(S21) składnik(S22) składnik(S23)
```

```
dodaj(S1,S21,W1)
dodaj(W1,S22,W2)
dodaj(W2,S23,S)
```

gdzie `S` jest zmienną uzgadniającą się z nagłówkiem reguły `suma(S)`, a `W1` i `W2` nowymi zmiennymi wprowadzonymi na potrzeby obliczenia.

2.2.3. Warunki wspólne

Czasami chcielibyśmy nałożyć pewien warunek na wszystkie elementy sekwencji, niezależnie od tego, które konkretnie nieterminale wystąpią. Można to uczynić za pomocą definicji warunków iterowanych podanej po liście nieterminali (po nawiasie zamykającym listę):

```
zdanie(Dest) -->
  ff(W, Dest1),
  sequence_of([
    fw(W1, Dest2)
    ^[oblwym,W,W1,_],
```

```

        fl(Dest2)
    ]
    ^[obldest, Dest1, Dest2, Dest]
).

```

W tym przykładzie warunek oblwym jest obliczany dla elementów fw, a warunek obldest jest wspólny dla wszystkich elementów sekwencji. Zauważmy, że w związku z tym zmienna Dest2 występuje zarówno w jednostce fw jak i fl. Jest to nowa zmienna w każdym elemencie sekwencji dzieląca wartość tylko z odpowiednim warunkiem w tym kroku iteracji.

2.2.4. Zmienne wolne

Ostatnia konwencja notacyjna związana z metanieterminalem `sequence_of` pozwala po prostu wskazać, że pewna zmienna występująca w nieterminale w sekwencji ma się nie uzgadniać z otoczeniem, co oznacza, że dla każdego wystąpienia dopuszczalna jest inna wartość, na którą nie nakładamy żadnych ograniczeń:

```

zdanie -->
    ff,
    sequence_of([
        fw(X)
        ^[X]
    ]).

```

2.3. Sekwencje sekwencji

Za pomocą `sequence_of` można też zapisać powtarzanie sekwencji nieterminali. Na przykład reguła

```

s -->
    sequence_of([
        [a,b],
        [c,d,e],
        f
    ]).

```

opisuje konstrukcje złożone z wystąpień ciągu a,b lub ciągu c,d,e lub nieterminali f. Na przykład a b a b f c d e f f. W wypadku tego zapisu warunki podaje się po zamykającym nawiasie obejmującym daną podsekwencję:

```

s(K) -->
    sequence_of([
        [a(X),b(Y)]
        ^[dobrze(X)]
        ^[policz, 0, Y, K],
        [c,d,e],
        f
    ]).

```

2.4. Przykład reguły

Oto przykład realnej reguły gramatycznej zapisanej za pomocą warunków iterowanych wraz z definicją (działającą!) jednego z nich — oblpnw

ustalającego, czy zdanie jest neutralne, pytajne, względne, czy pytajnozależne. Reguła ta opisuje zdanie złożone z frazy finitywnej, po której następuje ciąg fraz wymaganych i luźnych.

```

zdanie(Wf, A, C, T, Rl, O, Neg, I, Pnw, Sub) --> s(e1),
  ff(Wf, A, C, T, Rl, O, Wym, K, Neg, I, Pnw1, na),
  sequence_of([
    fw(W2, K, A, C, Rl, O, Neg, ni, Pnw2, po)
      ^[oblpwym, Wym, W2, ResztaWym]
      ^[K],
    fl(A, C, Rl, O, Neg, I, Pnw2, po)
  ])
  ^[oblpnw, Pnw1, Pnw2, Pnw]
),
{ czy_dopuszczalna_reszta(ResztaWym) }.

```

```

oblpnw(neut,neut,neut).
oblpnw(neut,pyt,pyt).
oblpnw(pyt,neut,pyt).
oblpnw(pyt,pyt,pyt).
oblpnw(wz,neut,wz).
oblpnw(pz,neut,pz).
oblpnw(pz,pyt,pz).

```

Dla parametru Pnw zdania bardzo istotna jest jego wartość w pierwszym elemencie, w tej regule — frazy finitywnej. Staje się ona inicjalną wartością Pnw w iteracji. Jeżeli wartość ta jest neut, to warunek oblpnw dopuszcza w kroku sekwencji wartość neut lub pyt dla każdego elementu i taką samą wartość przyjmuje wynik obliczenia w danym kroku. Podobnie wartość pyt na wejściu dopuszcza neut lub pyt w kroku, ale wynik pozostaje pyt. Dla wz na wejściu dopuszczalne jest tylko neut i wynik pozostaje wz. Podobnie zachowuje się wynik dla inicjalnego pz, ale w kroku dopuszczalne jest pyt i neut.

Następny przykład przedstawia regułę, w której inicjalna fraza jest wymagana. Ponieważ opisujemy zdanie, musi w nim wystąpić również fraza finitywna. Ponadto przewidujemy dwa miejsca na opcjonalne sekwencje fraz wymaganych i luźnych: między inicjalną fw i ff oraz po ff.

```

zdanie(Wf, A, C, T, Rl, O, Neg, I, Pnw, Sub) --> s(e2),
  fw(W1, K, A, C, Rl, O, Neg, I, Pnw1, po),
  sequence_of([
    fw(W2, K, A, C, Rl, O, Neg, ni, Pnw2, po)
      ^[zbierzwym, [W1], W2, PreFfWym]
      ^[K],
    fl(A, C, Rl, O, Neg, I, Pnw2, po)
  ])
  ^[oblpnw, Pnw1, Pnw2, PnwPreF]
),
ff(Wf, A, C, T, Rl, O, Wym, K, Neg, I, Pnw4, na),
{ oblpnw(PnwPreF, Pnw4, PnwPostF),
  wyjmijlistę(PreFfWym, Wym, PostFfWym) },
sequence_of([
  fw(W5, K, A, C, Rl, O, Neg, ni, Pnw5, po)

```

```

      ^[wyjmijwym, PostFfWym, W5, ResztaWym]
      ^[K],
      fl(A, C, Rl, O, Neg, I, Pnw5, po)
    ]
    ^[oblpnw, PnwPostF, Pnw5, Pnw]
  ),
  { czy_dopuszczalna_reszta(ResztaWym) }.

```

zbierzwym(WW, W, [W|WW])).

Warunki w tej regule są nieco skomplikowane przez konieczność odpowiedniego przekazania wartości do i z dwóch sekwencji. Popatrzmy na warunki dotyczące parametrów wymagań. W sekwencji poprzedzającej frazę finitywną, nie znając jeszcze reakcji czasownika, tylko gromadzimy zaobserwowane typy fraz wymaganych w listę za pomocą warunku zbierzwym (przytoczonego). Ostateczna lista staje się wartością zmiennej PreFfWym. Gdy zanalizujemy frazę finitywną, sprawdzamy, czy cała ta lista jest zgodna z wymaganiami Wym danego czasownika (warunek wyjmijlistę). Dla następnych fraz wymaganych zgodność badamy już na bieżąco za pomocą warunku wyjmijwym. Wreszcie na zakończenie opis niewypełnionych wymagań ResztaWym trafia do warunku `czy_dopuszczalna_reszta`, który może sprawdzić, czy dopełnione zostały ewentualne warunki na (nie)eliptyczność.

Przedstawione reguły stanowią dwie z trzech reguł na zdanie (elementarne) używanych przez obecną wersję parsera Świgr.