Kurs języka Prolog 2019

Lista zadań nr 7

Na zajęcia 17 kwietnia 2019

Zadanie 1 (1 pkt). Napisz gramatykę bezkontekstową definiującą język

$$L \;=\; \{a^n b^n \mid n \in \mathbb{N}\}.$$

Przepisz ją w postaci DCG. Pokaż, jak jest ona tłumaczona przez predykat `expand_term/2`. Użyj predykatu `phrase/2` by sprawdzić, który z ciągów

```
""
"aabb"
"aba"
```

należy do języka L. Ile nawrotów wykonuje program podczas sprawdzania? Pokaż, że Twój program także poprawnie generuje wszystkie słowa należące do języka L. Przerób gramatykę DCG tak, by analiza słów działała deterministycznie (bez głębokiego nawracania). Czy nowa gramatyka może być także wykorzystana do generowania słów? Dodaj do niej akcje semantyczne obliczające liczbę wystąpień liter `a` w podanym słowie.

Zadanie 2 (1 pkt). Składnię abstrakcyjną drzew binarnych bez etykiet będziemy reprezentować w postaci struktur zbudowanych z atomu `leaf/0` i binarnego funktora `node/2`. Składnię konkretną takich drzew opisuje gramatyka

$$\begin{aligned} \langle \text{tree} \rangle &\;\to\; \texttt{*} \\ \langle \text{tree} \rangle &\;\to\; \texttt{(}\ \langle \text{tree} \rangle\ \langle \text{tree} \rangle\ \texttt{)} \end{aligned}$$

gdzie `*` oznacza liść, a para drzew ujęta w nawiasy oznacza drzewo, w którym synami korzenia są podane drzewa. Napisz gramatykę DCG odpowiadającą powyższej gramatyce. Czy generuje ona wszystkie słowa należące do języka opisanego tą gramatyką? Przepisz gramatykę DCG tak, by otrzymany program prologowy działał deterministycznie podczas rozpoznawania słów (taki program nie nadaje się oczywiście do generowania słów). Dodaj odpowiednie akcje semantyczne budujące abstrakcyjne drzewa rozbioru.

Zadanie 3 (1 pkt). Oto gramatyka zawierająca binarny operator łączący w lewo:

$$\begin{aligned} \langle \text{expression} \rangle &\;\to\; \langle \text{simple expression} \rangle \\ \langle \text{expression} \rangle &\;\to\; \langle \text{expression} \rangle\ \texttt{*}\ \langle \text{simple expression} \rangle \\ \langle \text{simple expression} \rangle &\;\to\; \texttt{a} \\ \langle \text{simple expression} \rangle &\;\to\; \texttt{b} \\ \langle \text{simple expression} \rangle &\;\to\; \texttt{(}\ \langle \text{expression} \rangle\ \texttt{)} \end{aligned}$$

Napisz odpowiednią gramatykę DCG odpowiadającą powyższej gramatyce. Czy generuje ona wszystkie słowa należące do języka opisanego tą gramatyką? Przepisz gramatykę DCG tak, by otrzymany program prologowy działał deterministycznie podczas rozpoznawania słów (taki program nie nadaje się oczywiście do generowania słów). Dodaj odpowiednie akcje semantyczne budujące abstrakcyjne drzewa rozbioru takich wyrażeń, zbudowane z atomów `a/0` i `b/0` oraz binarnego funktora `*/2`.

Zadanie 4 (2 pkt). Zaprojektuj algorytm, który dla podanej gramatyki bezkontekstowej rozstrzyga, czy język przez nią generowany jest niepusty. Zaimplementuj go w Prologu. Gramatykę reprezentuj w postaci zbioru klauzul DCG.

Zadanie 5 (8 pkt). Oto struktura leksykalna i składnia prostego języka programowania. Klasyfikacja znaków:

- *biały znak* — jeden ze znaków ASCII: HT, LF lub SPACE;
- *cyfra* — jeden ze znaków ASCII: `0`–`9`;
- *litera* — jeden ze znaków ASCII: `a`–`z` lub `A`–`Z`;
- *symbol specjalny* — jeden ze znaków ASCII: `+`, `-`, `*`, `^`, `:`, `=`; `<`, `>`, `(`, `)`, `;`;
- *znak dopuszczalny* — suma powyższych rozłącznych zbiorów.

Komentarz to najdłuższy ciąg dowolnych znaków zaczynający się znakami `(*`, zakończony znakami `*)` i nie zawierający wewnątrz ciągu znaków `*)` (komentarzy nie można zagnieżdżać). *Token* to najdłuższy spójny ciąg znaków dopuszczalnych nie zawierający białych znaków oraz komentarzy. Dopuszczalnymi tokenami są: `+`, `-`, `*`, `^`, `=`, `<`, `>`, `<=`, `>=`, `<>`, `:=`, `;` oraz zbiory tokenów literał-całkowitoliczbowy, słowo-kluczowe i identyfikator opisane poniższą gramatyką:

literał-całkowitoliczbowy	::=	*cyfra* | literał-całkowitoliczbowy *cyfra*
słowo-kluczowe	::=	`if` | `then` | `else` | `fi` | `while` | `do` | `od` | `div` | `mod` | `or` | `and` | `not`
alfanum	::=	*litera* | alfanum *litera* | alfanum *cyfra*
identyfikator	::=	alfanum różny od słowo-kluczowe

Tokeny dopuszczalne są symbolami terminalnymi gramatyki języka:

⟨program⟩	::=	⟨instrukcja⟩ | ⟨program⟩ `;` ⟨instrukcja⟩
⟨instrukcja⟩	::=	identyfikator `:=` ⟨wyrażenie arytmetyczne⟩
	|	`if` ⟨wyrażenie logiczne⟩ `then` ⟨program⟩ `fi`
	|	`if` ⟨wyrażenie logiczne⟩ `then` ⟨program⟩ `else` ⟨program⟩ `fi`
	|	`while` ⟨wyrażenie logiczne⟩ `do` ⟨program⟩ `od`
⟨wyrażenie logiczne⟩	::=	⟨składnik logiczny⟩ | ⟨wyrażenie logiczne⟩ `or` ⟨składnik logiczny⟩
⟨składnik logiczny⟩	::=	⟨czynnik logiczny⟩ | ⟨składnik logiczny⟩ `and` ⟨czynnik logiczny⟩
⟨czynnik logiczny⟩	::=	⟨wyrażenie relacyjne⟩ | `not` ⟨czynnik logiczny⟩
⟨wyrażenie relacyjne⟩	::=	⟨wyrażenie arytmetyczne⟩ ⟨operator relacyjny⟩ ⟨wyrażenie arytmetyczne⟩
	|	`(` ⟨wyrażenie logiczne⟩ `)`
⟨operator relacyjny⟩	::=	`=` | `<` | `>` | `<=` | `>=` | `<>`
⟨wyrażenie arytmetyczne⟩	::=	⟨składnik⟩
	|	⟨wyrażenie arytmetyczne⟩ ⟨operator addytywny⟩ ⟨składnik⟩
⟨składnik⟩	::=	⟨czynnik⟩
	|	⟨składnik⟩ ⟨operator multiplikatywny⟩ ⟨składnik⟩
⟨czynnik⟩	::=	⟨wyrażenie proste⟩ | ⟨wyrażenie proste⟩ `^` ⟨czynnik⟩
⟨wyrażenie proste⟩	::=	`(` ⟨wyrażenie arytmetyczne⟩ `)`
	|	literał-całkowitoliczbowy
	|	identyfikator
⟨operator addytywny⟩	::=	`+` | `-`
⟨operator multiplikatywny⟩	::=	`*` | `div` | `mod`

Napisz parser DCG tego języka, który tworzy abstrakcyjne drzewa rozbioru programów.

Zadanie 6 (7 pkt). Znaczenie programów napisanych w języku z poprzedniego zadania jest opisane za pomocą następujących reguł semantyki naturalnej, w których n oznacza liczbę, x — identyfikator, $\oplus$ — operator arytmetyczny, $\odot$ — operator relacyjny, $\oslash$ — operator logiczny, T i F oznaczają wartości logiczne prawdy i fałszu, a π jest stanem pamięci (funkcją przypisującą identyfikatorom liczby całkowite):

$$\frac{}{\langle n, \pi\rangle \to n} \qquad \frac{}{\langle x, \pi\rangle \to \pi(x)}$$

$$\frac{\langle e_1, \pi\rangle \to n_1 \quad \langle e_2, \pi\rangle \to n_2 \quad \models n = n_1 \oplus n_2}{\langle e_1 \oplus e_2, \pi\rangle \to n} \qquad \frac{\langle e_1, \pi\rangle \to n_1 \quad \langle e_2, \pi\rangle \to n_2 \quad \models l \Leftrightarrow n_1 \odot n_2}{\langle e_1 \odot e_2, \pi\rangle \to l}$$

$$\frac{\langle b, \pi\rangle \to l_1 \quad \models l \Leftrightarrow \neg l_1}{\langle \texttt{not}\ b, \pi\rangle \to l} \qquad \frac{\langle b_1, \pi\rangle \to l_1 \quad \langle b_2, \pi\rangle \to l_2 \quad \models l \Leftrightarrow (l_1 \oslash l_2)}{\langle b_1 \oslash b_2, \pi\rangle \to l}$$

$$\frac{\langle c_1, \pi\rangle \to \pi' \quad \langle c_2, \pi'\rangle \to \pi''}{\langle c_1\ \texttt{;}\ c_2, \pi\rangle \to \pi''} \qquad \frac{\langle e, \pi\rangle \to n}{\langle x\ \texttt{:=}\ e, \pi\rangle \to \pi[x/n]}$$

$$\frac{\langle b, \pi\rangle \to F}{\langle \texttt{if}\ b\ \texttt{then}\ c\ \texttt{fi}, \pi\rangle \to \pi} \qquad \frac{\langle b, \pi\rangle \to T \quad \langle c, \pi\rangle \to \pi'}{\langle \texttt{if}\ b\ \texttt{then}\ c\ \texttt{fi}, \pi\rangle \to \pi'}$$

$$\frac{\langle b, \pi\rangle \to T \quad \langle c_1, \pi\rangle \to \pi'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{fi}, \pi\rangle \to \pi'} \qquad \frac{\langle b, \pi\rangle \to F \quad \langle c_2, \pi\rangle \to \pi'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{fi}, \pi\rangle \to \pi'}$$

$$\frac{\langle b, \pi\rangle \to F}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}, \pi\rangle \to \pi} \qquad \frac{\langle b, \pi\rangle \to T \quad \langle c, \pi\rangle \to \pi' \quad \langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}, \pi'\rangle \to \pi''}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{od}, \pi\rangle \to \pi''}$$

Zaprogramuj predykaty `evalExpr/3`, `evalLog/3` i `evalProg/3` implementujące powyższe reguły semantyki operacyjnej. Pierwszym argumentem tych predykatów powinno być abstrakcyjne drzewo rozbioru, odpowiednio, wyrażenia arytmetycznego, wyrażenia logicznego bądź programu, a drugim — stan pamięci opisany za pomocą listy asocjacji. Ponieważ nie mamy w języku deklaracji zmiennych, to zakładamy, że zmienna, która nie występuje na liście asocjacji ma wartość 0. Trzecim, wyjściowym parametrem powyższych predykatów powinna być, odpowiednio, liczba całkowita, atomy `true` i `false`, bądź lista asocjacji opisująca końcowy stan pamięci. Połącz powyższe predykaty z parserem z poprzedniego zadania aby otrzymać interpreter naszego języka.