

Zadanie domowe 3

Parser:

W tym zadaniu użyjecie ML-Yacc do zbudowania parsera dla języka Fun. Sporo przydatnej teorii znajdziecie w podręczniku Appela, a dokładniej w rozdziale 3, w szczególności w sekcji 3.4. Dokumentację do Yacc znajdziecie tutaj:

<http://www.smlnj.org/doc/ML-Yacc/>

a specyfikację języka Fun tam, gdzie ostatnio:

www.math.us.edu.pl/~pgladki/teaching/2013-2014/tk_fun.html

Założcie nowy katalog **lab3** i skopiujcie do niego całą zawartość katalogu **lab2**. Następnie ściągnijcie plik

www.math.us.edu.pl/~pgladki/teaching/2013-2014/tk2_lab3.zip

i rozpakujcie go; w szczególności znajdziecie tam pliki `sources.cm` `fun.grm` `parse.sml` `compile.sml` `all.fun` `testcases.sml` – zastąpcie starą wersję `sources.cm` nową wersją.

Do zadania potrzebny Wam będzie działający lexer `fun.lex` z poprzedniego zadania. Jeżeli Wasz nie działa wystarczająco dobrze, zgłóście się do prowadzącego po (wystarczająco dobrze) działającą wersję.

Modyfikujcie plik **fun.grm** tak długo, dopóki nie będzie specyfikował parsera dla języka Fun. Na początku nie przejmujcie się przesadnie semantyką, a skupcie się na parserze i wyjaśnieniu ewentualnych dwuznaczności w gramatyce. Oto kilka rzeczy, na które trzeba zwrócić uwagę:

- Specyfikacja języka Fun zawiera reguły gramatyczne, które opisują strukturę syntaktyczną języka Fun.
- Będziecie musieli wyspecyfikować nazwy końcowe odpowiadające nazwom tokenów generowanych przez lexer. ML-Yacc automatycznie generuje strukturę, która zbuduje tokeny. Innymi słowy, **fun.grm** będzie musiał zawierać następujący kod (rozszerzony o inne nazwy końcowe):

```
%term EOF
| INT of int
| ID of string
| COMMA | SEMICOLON | COLON
| LPAREN | RPAREN | ...
```

Dla nieterminali możecie używać dowolnych nazw, jakie przypadną Wam do gustu, na przykład:

```
%nonterm foo of Absyn.exp | ...
```

- Definicja języka Fun nie jest w stu procentach jasna odnośnie do gramatyki, jakiej powinniście użyć, wszelako plik `testcases.sml` powinien okazać się pomocny.
- Jeżeli będziecie używali wskaźników do poprzedników, róbcie to z głową. Innymi słowy, nie wrzucajcie wskaźnika wszędzie tam, gdzie wyskoczy jakiś błąd w Waszej gramatyce, ale bądźcie w stanie wyjaśnić w Waszym pliku `README` w jaki sposób użycie wskaźnika ma osiągnąć zamierzony efekt, być może dodając odnośnik do pliku `fun.grm.desc` wygenerowanego przez `ml-yacc`.

- Skompilujcie kod z użyciem **CM.make "sources.cm"**; komenda ta wywoła w razie potrzeby zarówno **ml-lex** jak i **ml-yacc**. Przetestujcie Wasz parser wywołując **Compile.compile "nazwa"**. Ta komenda wywoła **Parse.parse** a następnie uruchomi interpreter na zadanym pliku. Na przykład możecie użyć "all.fun" jako "nazwa".
- Innym sposobem na testowanie jest wykonanie `val = Compile.go()`; w ten sposób dla każdej pary ciągów w `testcases.sml` uruchomiony zostanie parser i porównany abstrakcyjny syntaks.

Po uporaniu się z gramatyką, zajmijcie się semantyką. W efekcie powinniście otrzymać abstrakcyjny syntaks dla kodu pośredniego Fun. Oto kilka uwag, na co trzeba zwrócić uwagę:

- Niektóre operacje z poziomu wejścia nie pojawiają się w kodzie pośrednim. Na przykład "and", "or" muszą zostać wkompirowane w inne istniejące konstrukcje. Na przykład "e1 and e2" może być przetłumaczone jako "if e1 then (if e2 then 1 else 0) else 0".
- Każdy obiekt jaki wygenerujecie w abstrakcyjnym syntaksie powinien być uwzględniony w konstruktorze Fun.Pos, który oznacza początek i koniec danej konstrukcji w kodzie źródłowym.

Dodajcie jakąś obsługę do znajdowania i poprawiania potocznych błędów z wykorzystaniem konstrukcji ML-Yacc takich jak "%value" oraz "%change" opisanych na stronach 80-81 MClinML oraz w manualu do ML-Yacc. Dodajcie niezbędne wyjaśnienia do Waszego pliku README.

Plik **all.fun** testuje konstrukcje syntaktyczne w sposób raczej trywialny, tak więc napiszcie kilka własnych plików .fun. Jeżeli chcecie potem oddać je razem z Waszym zadaniem, to zróbcie to.

To zadanie składa się z dwóch części. Są w zasadzie całkowicie niezależne jedna od drugiej, możecie więc pracować nad nimi w dowolnej kolejności.

Prześlijcie mailem pliki README, fun.grm oraz wszelkie pliki .fun, jakie dodatkowo napisaliście. Termin zadania mija w **czwartek, 9 stycznia**. Proszę pamiętać o oznaczeniu maila tagiem [aghtk2].