Zadanie z implementacji języków EDSL w Haskellu

Piotr Krzemiński

24 listopada 2013

Rozwiązania należy wysyłać na adres pio.krzeminski@gmail.com do 31 grudnia 2013 r.

1 Zadanie

Zadanie polega na zaprogramowaniu poniższego języka EDSL w dwóch wersjach — płytkiej i głębokiej — podobnie do języka omawianego na seminarium.

Zaimplementuj język EDSL realizujący polecenia grafiki żółwia. Stan żółwia reprezentowany jest przez jego współrzędne w kartezjańskim układzie współrzędnych, orientację oraz stan pisaka (opuszczony lub podniesiony). Składnia języka wygląda następująco:

```
program ::= [command]
command ::= forward n
          | backward n
          | turnLeft n
          | turnRight n
          | penUp
          | penDown
          | repeat n program
```

Każde polecenie modyfikuje stan żółwia, a komendy `forward` i `backward` generują dodatkowo efekty uboczne, zgodnie z poniższą specyfikacją:

- `forward n` – przesuwa żółwia o n kroków naprzód, zgodnie z orientacją; jeśli pisak jest opuszczony, żółw rysuje na płaszczyźnie odcinek
- `backward n` – przesuwa żółwia o n kroków w tył, zgodnie z orientacją; jeśli pisak jest opuszczony, żółw rysuje na płaszczyźnie odcinek
- `turnLeft n` – zmienia orientację żółwia o n stopni w lewo
- `turnRight n` – zmienia orientację żółwia o n stopni w prawo
- `penUp` – podnosi pisak
- `penUp` – opuszcza pisak
- `repeat n program` – n-krotnie powtarza podprogram

Zaprojektuj składnię języka tak, aby w czytelny sposób można było zapisywać sekwencje instrukcji. Do tego celu wygodne może okazać się wykorzystanie haskellowej do-notacji. Wówczas program, który rysuje dwunastokąt foremny, mógłby wyglądać mniej więcej następująco:

```
p = do
      turnRight 90
      repeat 12 $ do
        forward 50
        turnLeft (360 `div` 12)
      turnLeft 90
```

Przyjmujemy założenie, że stanem początkowym żółwia jest punkt $(0, 0)$, orientacja w kierunku północnym i opuszczony pisak. Dla programów w powyższym języku możemy rozważyć kilka różnych semantyk wykonania.

1. **Ewaluacja stanu końcowego** – ignorujemy efekty uboczne polegające na rysowaniu linii, interesuje nas tylko stan końcowy, w którym żółw znajdzie sie po wykonaniu całego programu.
2. **Ewaluacja zbioru odcinków** – rezultatem wykonania programu powinna być lista odcinków, które żółw narysował podczas wykonywania poleceń; na podstawie takiej listy wygeneruj plik z grafiką wektorową w formacie SVG[1].
3. **Interfejs graficzny** – wykorzystaj jedną z bibliotek graficznych (np. SDL) do zaimplementowania graficznego interfejsu, który podczas wykonywania programu rysuje na ekranie obrazek.
4. **„Pretty print" programu** – wydrukowanie programu w postaci ładnie sformatowanego ciągu znaków; postaraj się, aby tak wygenerowany string można było skopiować bezpośrednio do programu haskellowego.

1.1 Wersja płytka (*shallow embedding*)

Wybierz co najmniej dwie spośród powyższych semantyk i zaimplementuj je jako język EDSL w stylu płytkiej integracji, bezpośrednio kodując polecenia żółwia w docelowej semantyce.

1.2 Wersja głęboka (*deep embedding*)

Zdefiniuj haskellowy typ danych do reprezentacji poleceń żółwia. Zaimplementuj polecenia żółwia w taki sposób, aby wyliczały się do abstrakcyjnej reprezentacji programu. Następnie wybierz co najmniej trzy spośród powyższych semantyk i napisz odpowiednie ewaluatory.

1.3 Uwagi

- Postaraj się zaprojektować jednolitą składnię zarówno dla wersji płytkiej, jak i głębokiej.
- Naturalnie chcielibyśmy wykorzystać haskellowe funkcje, aby definiować podprogramy w języku poleceń żółwia. Na jaki problem możemy wówczas trafić podczas implementacji semantyki (4)? W jaki sposób można próbować go rozwiązać?
- Zauważ, że komendę `backward n` można zaimplementować na dwa sposoby: jako polecenie mające reprezentację w składni lub jako cukier syntaktyczny dla pewnej sekwencji poleceń. Jakie zalety i wady mają oba podejścia?

[1] Więcej o rysowaniu grafiki w formacie SVG można przeczytać tutaj: http://oreilly.com/catalog/svgess/chapter/ch03.html

2 Ocenianie

Ocenie podlega:

- poprawność implementacji wybranych semantyk w obu wersjach,
- styl i czytelność kodu,
- zbiór przykładowych programów w języku grafiki żółwia.

Dodatkowe punkty można uzyskać za rozmaite rozszerzenia podstawowej specyfikacji zadania, które mogą polegać na przykład na:

- rozbudowie stanu żółwia np. o kolor lub grubość pisaka i instrukcji pozwalających na zmianę tego stanu,
- rozszerzeniu interfejsu graficznego o debugger, który pozwala wykonywać instrukcje krok po kroku (np. po wciśnięciu klawisza) i jednocześnie wypisuje w konsoli rysowane odcinki i kolejne stany żółwia.