Techniki kompilacji - koncepcja wstępna

Prowadzący - dr inż. Piotr Gawkowski

Tomasz Nowak

1 Opis języka

1.1 Wstęp

"LOGO z L-systemem" - interpreter języka pozwalającego definiować L-system i opisywać elementarne operacje w tym systemie w języku "grafiki żółwia". Proponowana nazwa własna języka: LTurtle.
Podstawowe elementy języka:

- Funkcje wbudowane, zmieniające stan rysującego żółwia
- Zbiór symboli, każdy z nich reprezentuje pewien ciąg operacji żółwia
- Zbiór produkcji, generujących ciągi symboli w L-systemie

Dla uproszczenia, jedynym rodzajem zmiennych występujących w języku są symbole L-systemu (nie ma np. zmiennych liczbowych czy napisów). Są one jednocześnie jedynym rodzajem funkcji tworzonych przez użytkownika LTurtle. Ten język można nazwać językiem funkcyjnym, z wyjątkiem wbudowanych operacji wykonywanych przez żółwia. Składnia języka jest połączeniem konwencji z C (klamry, średniki) z Pythonem (przypisanie wartości do zmiennej jest równoznaczne z jej definicją, nie ma deklaracji typu; są jednolinijkowe komentarze rozpoczynające się #).

1.2 Operacje żółwia

Niektóre z poniższych funkcji przyjmą argumenty liczbowe. Liczby muszą być podane jawnie (nie ma zmiennych liczbowych). Możliwe "typy danych": int, float. Operacje dostępne są wewnątrz definicji symbolu L-systemu, a także poza nią.

- *forward(float+)* - przesuwa żółwia, rysując linię
- *rotate(float[-180.0:180.0])* - obrót (kąt dodatni - zgodnie z ruchem wskazówek zegara, ujemny - przeciwnie)
- *penup()* - wyłącza pisak (ruch żółwia nie powoduje rysowania)
- *pendown()* - odwrotność penup()

- *pencolour(int[0:255],* int[0:255], int[0:255]) - ustawia kolor (RGB)
- *goto(float, float)* - przesuwa żółwia na współrzędne określone bezwzględnie
- *pensize(float+)* - ustawia średnicę pisaka
- *scale(float+)* - wszystkie wielkości rysowane przez żółwia będą skalowane
- *pushstate()* - zapisuje na stosie obecny stan żółwia
- *popstate()* - zdejmuje ze stosu obecny stan żółwia

Operacja *scale* jest potrzebna, ponieważ funkcje definiowane w L-systemie nie przyjmują żadnych argumentów - wszystkie długości są zapisane jawnymi stałymi liczbowymi. Dzięki tej operacji można wielokrotnie wykorzystywać raz zdefiniowaną funkcję do rysowania obiektów różnych rozmiarów (do zmiany położenia na obrazie wynikowym i obrotu można wykorzystać operacje *rotate* i *forward*). W przyszłości można dodać inne operacje do rysowania bardziej skomplikowanych kształtów, np. *draw_circle*.

1.3 Operacje na symbolach L-systemu

- Definicja nowego symbolu, np.: line = { forward(1.0); rotate(60.0); };
- Zdefiniowanie reguły podstawienia, np. X −> X + line;
- Ewaluacja wyrażenia określoną liczbę iteracji, podstawiane pod nowy symbol, np.: operation = evaluate(5, X+X+X);
- Wykonanie operacji przez żółwia, np.: execute(operation);

Operacje przypisania znaczenia danego symbolu (=) i zdefiniowania podstawienia (−>) mogą nadpisywać poprzednią wartość tylko gdy zostaną poprzedzone słowem kluczowym *redefine*, np.: *redefine line = { forward(1.0); rotate(30.0); };*. Operacja "=" musi być zdefiniowana dla każdego używanego symbolu, natomiast "−>"nie musi być zdefiniowana - symbol bez zdefiniowanych podstawień jest uznawany za terminalny.

1.4 Proste przykłady kodu

Rysowanie kwadratu (jedna z możliwych implementacji - z demonstracją komentarzy i symbolu bez instrukcji):

```
step = { forward(1.0); rotate(90.0); };
sqaure_step = {}; # empty instruction list - allowed
sqaure_step -> sqaure_step + step;
square = evaluate(4, sqaure_step);
pencolour(255, 0, 0); # direct execution of turtle command
execute(square);
# line with comment
```

Przykład 2 z angielskiej Wikipedii o L-systemach - *Fractal (binary) tree* (zamienione nazwy symboli: s0 zamiast 0, s1 zamiast 1, left zamiat [, right zamiast]):

```
s0 = { forward(1.0); };
s1 = { forward(2.0); };
left = { pushstate(); rotate(-45.0); };
right = { popstate(); rotate(45.0); };
s1 -> s1 + s1;
s0 -> s1 + left + s0 + right + s0;
tree3 = evaluate(3, s0);
execute(tree3);
```

2 Formalna specyfikacja

2.1 Zdefiniowane tokeny

Słowa kluczowe:
forward rotate penup pendown pencolour goto pensize scale pushstate popstate evaluate execute redefine
Operatory:
= ->
Separatory:
+ { } ( ) ; ,
Składniki nazw zmiennych:
(litery) (cyfry) _
Składniki stałych liczbowych:
(cyfry) . -
Wyznaczające komentarz:
(koniec linii)

2.2 Gramatyka

Gramatyka opisana po usunięciu komentarzy (jednolinijkowy komentarz zaczynający się od #). Konwencja - kolejne symbole definiowane są w takiej kolejności, w jakiej pojawiły się w poprzednich defnicjach (tak jakby były odkładane na stos).

```
program = { statement ";" }
statement = definition | execution
definition = redefinition | createDefinition
execution = literalExecution | turtleStatement
literalExecution = "execute" "(" literalString ")"
redefinition = "redefine" createDefinition
createDefinition = operation | production | evaluation
literalString = literal | literal "+" literalString
```

```
operation = literal "=" "{" { turtleStatement ";" } "}"
production = literal "->" literalString
evaluation = literal "=" "evaluate" "(" int
             "," literalString ")
literal = letter { letter | "_" | digit }
turtleStatement = forwardSt | rotateSt | penupSt | pendownSt
                  | pencolourSt | gotoSt | pensizeSt
                  | scaleSt | pushstateSt | popstateSt
forwardSt = "forward" "(" float ")"
rotateSt = "rotate" "(" float ")"
penupSt = "penup" "(" ")"
pendownSt = "pendown" "(" ")"
pencolourSt = "pencolour" "(" int "," "int" "," "int" ")"
gotoSt = "goto" "(" float "," float ")"
pensizeSt = "pensize" "(" float ")"
scaleSt = "scale" "(" float ")"
pushstateSt = "pushstate" "(" ")"
popstateSt = "popstate" "(" ")"
int = "0" | negativeInt | positiveInt
letter = "a".."z" | "A".."Z"
negativeInt = "-" positiveInt
positiveInt = nonzeroDigit { digit }
nonzeroDigit = "1".."9"
digit = "0".."1"
float = int | int "." { digit }
```

Uwaga dot. int i float - na poziomie gramatyki nie ma rozróżnienia między liczbami ujemnymi/nieujemnymi, dlatego parser zaakceptuje np. ujemną wartość koloru (zostanie ona odrzucona na poziomie semantyki/wykonania).

3 Wymagania i sposób uruchamiania

3.1 Wymagania funkcjonalne

- Interpreter LTurtle przyjmuje jako argumenty wywołania pojedynczy plik tekstowy z kodem i nazwę pliku wyjściowego.
- Jako wyjście iterpreter produkuje obraz, stworzony zgodnie z opisem.
- Rozdzielczość obrazu wyjściowego jest ustalana dynamicznie - na podstawie skrajnych współrzędnych, do których dotarł żółw.
- W przypadku wystąpienia błędu (na poziomie parsowania / semantyki), wypisywany jest odpowiedni komunikat, wraz z podaniem, gdzie błąd wystąpił.

3.2 Wymagania niefunkcjonalne

- Komunikaty błędu powinny być możliwie przejrzyste dla użytkownika.
- W interpreterze zostanie wprowadzone ograniczenie na długość tokenu.
- Interpreter w miarę możliwości będzie usuwał z pamięci niepotrzebne dane tymczasowe.

4 Wstępny opis sposobu realizacji

4.1 Środowisko uruchomieniowe

Interpreter zostanie napisany w języku C++ (standard 11). Będzie korzystał z biblioteki SFML (do tworzenia grafiki). System operacyjny - Linux (ale program powinien być przenośny). Kompilator - gcc 7.3 (ale program powinien dać się skompilować też przy użyciu innego kompilatora). Sposób budowania - CMake 3.10.

4.2 Moduły interpretera

- Moduł obsługi źródła - zwraca kolejne znaki dla leksera, zapamiętuje numer wiersza i numer znaku w wierszu.
- Lekser - odczytuje tekst znak po znaku, zwraca kolejne tokeny na żądanie parsera.
- Parser - analizuje gramatykę kodu, zstępując w głąb.
- Analizator semantyczny - zamienia drzewo rozbioru gramatycznego na sekwencję instrukcji żółwia.
- Moduł wykonawczy.

4.3 Ogólna idea sposobu wykonania

Kod zapisany w LTurtle można sprowadzić do liniowej sekwencji operacji żółwia, dlatego kolejne instrukcje podczas analizy semantycznej będą zapisywane na liście. Wykonanie kodu będzie podzielone na dwie części. Pierwsza, następująca wraz z analizą semantyczną, zapamiętuje kolejne instrukcje żółwia i współrzędne, na których się znalazł, ale niczego nie rysuje. Jest to spowodowane koniecznością spełnienia wymagania funkcjonalnego nr 3 - rozdzielczość obrazu jest dostosowywana dynamicznie do zakresu współrzędnych, po których przemieszczał się żółw. Rozdzielczość musi być obliczona przed rasteryzacją, ponieważ przyjęcie odgórnie narzuconej rozdzielczości wpływałoby negatywnie na jakość tworzonych obrazów albo znacząco ograniczałoby kreatywność użytkownika w tworzeniu obrazu. W drugiej fazie wykonania (w tym momencie plik z kodem programu może już być zamknięty) następuje rasteryzacja.

4.4 Testowanie

- Testy jednostkowe poszczególnych modułów programu, zgodnie z ich interfejsami.
- Testowanie intuicyjne - "czy program działa", wraz z badaniem zachowania interpretera dla błędnie napisanego kodu.
- Porównanie wyników działania interpretera LTurtle z wynikami działania programu rysującego L-systemy, dostępnego w Internecie.