Език SIMPLA за описване на крайни автомати
и
Реализация на SIMPLA->C++ компилатор на SIMPLA

Димитър Тошков Тошев
ФН 44144
Специалност Информатика, II-ри курс.

Езика SIMPLA (SIMPle Language for Automata)

Езика разглежда програмите като поредица идентификатори, разделени с шпации, табулации или нови редове (whitespace).

Всяка програма на simpla представлява поредица преходи от вида

```
<state_name> <input> <output> <state_name>
            [<input> <output> <state_name>...] ,
```

Заградени в къдрави скоби. input, output и state_name са думи над знаците, обозначени от ASCII символите 33 до 126 (всички символи след шпацията). За input и output има някои специални случаи:

'	input: приема за вход whitespace. На практика преставлява епсилон-преход. Изпълнява се само ако има празно пространство преди следващата входна дума и няма дефиниран преход за нея или за '+'.
+	input: приема за вход произволна дума
*	input: приема за вход произволна дума или whitespace. Изпълнява се само ако няма друга възможност
&	output: input (прочетената входна дума)
#	output: Нищо
#файл	output: съдържанието на файла “файл”. Това се налага, тъй като C++ има ограничение за максимална дължина на низове, а и подобрява четимостта на програми с по-обемен изход.

За да се приеме като вход буквално ', + или *, трябва да се напишат \', \+ и * . Това са единствените думи, които имат специално значение за входа, от гледна точка на компилатора.

SIMPLA се компилира до C++, а входните и изходните низове директно стават C++ низове. Поради това, за изход имаме възможност да изкарваме нов ред с \n, шпация с \x20 и т.н. Това също така значи и че двойните кавички трябва да се предхождат от \ .

Входната азбука на компилатора и множеството състояния се извеждат от дефинираните състояния и преходи. За завършителни се третират тези състояния, за които не са дефинирани преходи. При попадане в такова състояние автомата спира своята работа. Автомата трябва да има дефинирано състояние “start”, което се третира за начално.

Пример (hello world):

```
{
start + Hello\x20World! end ,
}
```

Този автомат разпознава всички непразни думи и извежда “Hello World!” .

Пример - разпознава дали сме въвели вярно шестнадесетично число, последвано от празен символ:

```
{
start - # zero
        0 # x ,
zero 0 # x ,
x       x # digit ,
digit 0 # digit
        1 # digit
.............
        9 # digit
        A # digit
        a # digit
..........
        f # digit
        ' yes end
        * no end ,
}
```

Пример - изчиства всичкия whitespace. Този автомат приключва при край на входа:

```
{
start ' # start
        + & start ,
}
```

Компилиране до C++ код

За да се компилира SIMPLA програма до C++ код, ще използваме начален и краен C++ код, дефиниращ начало на C++ програма, клас симулиращ автомат и край на програмата. Между тях ще вмъкнем код, инициализиращ автомата с нужните състояния.
Началния и крайния код за програмата са съответно във файловете simpla_header.h и simpla_end.h . Тук е даден само кода на самата SIMPLA програма, който също се намира и във файла simpla.sla:

```
{
start { #simpla_header.h stmt ,
stmt ' \ta.addState\x20(\" state
     } #simpla_end.h end ,
state + & state_close ,
state_close ' \");\n\ta.addTransition\x20(\" input ,
input + & comma1 ,
comma1 ' \",\x20\" output ,
output + & comma2 ,
comma2 ' \",\x20\" next ,
next + & stmt_end ,
stmt_end , \");\n stmt
         ' \");\n\ta.addTransition\x20(\" input ,
}
```

Симулиращия C++ автомат е дефиниран като променлива a от тип automat. Той има методи addState (std::string) и addTransition (std::string, std::string, std::string).
addState задава състояние, за което ще дефинираме преходи, а addTransition задава пореден преход за даденото състояние.

Ето и автоматна граматика, дефинираща езика SIMPLA:
X са всички непразни думи, състоящи се от ASCII символите 33 до 126. Със символа + ще бележа произволна такава дума. Нетерминалите са написани на кирилица, за да не съвпадат с терминалите.
Г < {старт, израз, съст, вх, изх, следв}, X, старт, P >
P = {
старт -> {съст ,
израз -> 'съст | } ,
съст -> +вх ,
вх -> +изх ,
изх -> +следв ,
следв -> +вх | ,израз
}