Data Structures in Coco/R

H. Mössenböck, Aug. 2002

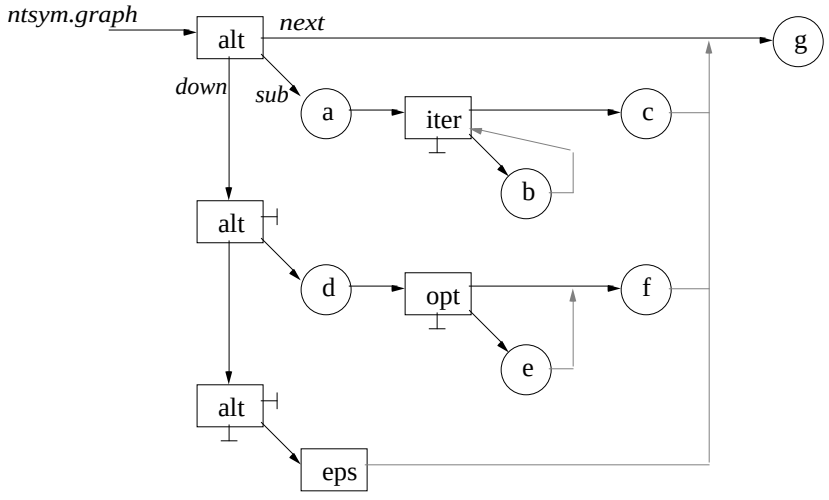
This technical note describes the data structures in the C# implementation of the compiler generator *Coco/R* (http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/CSharp). The major data structures are:

- The **Symbol table** (Classes: *Symbol*). All terminals, pragmas and nonterminals in linear sequence. This data structure is trivial and therefore not further described.
- The **Syntax graph** (Classes: *Node*, *Graph*). The productions of the grammar as separate subgraphs. For every nonterminal *sym* there is a pointer *sym.graph* to the root of this symbol's syntax graph. A snapshot of this data structure is described in Section 1.
- The **Scanner automaton** (Classes: *State*, *Action*, *Target*, *Melted*). The DFA generated from token declarations. The token declarations are first translated to a syntax graph which is then transformed into a deterministic finite automaton. These steps are shown in Section 2.
- The **Character classes** (Class: *CharClass*). The character sets declared in the grammar stored in a linear list. This data structure is trivial and therefore not further explained.

1. Syntax Graph

Production: A = (a {b} c | d [e] f |) g.

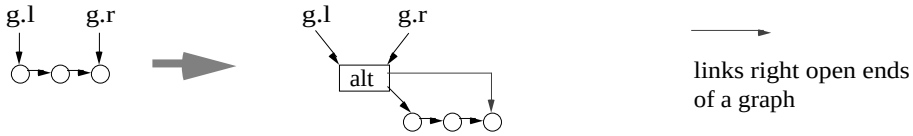
Graph:



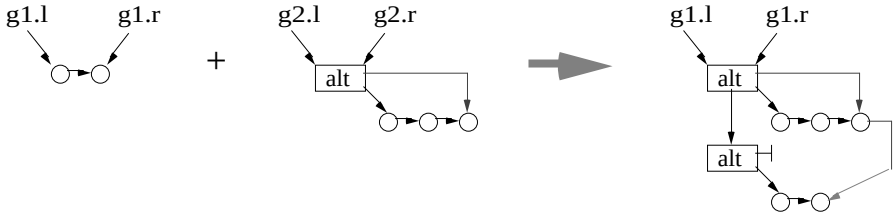
Dotted lines denote *next* pointers that point upwards. For any node *n*, if *n.next* points upwards, then *n.up* is true.

Operations to build the syntax graph

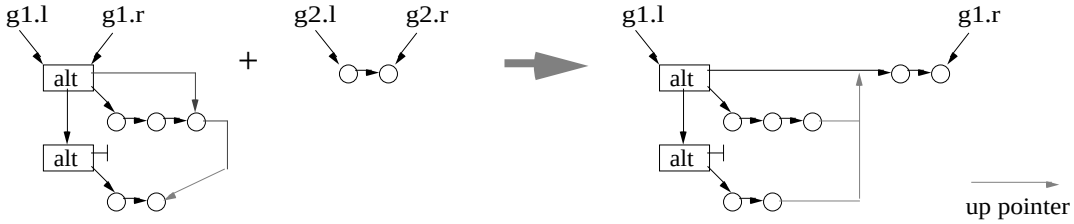
Graph.MakeFirstAlt(g)



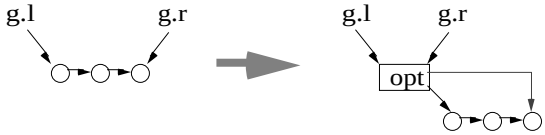
Graph.MakeAlternative(g1, g2)



Graph.MakeSequence(g1, g2)

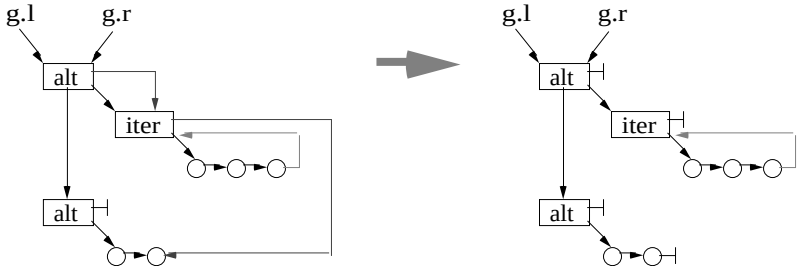


Graph.MakeOption(g)



Graph.MakeIteration(g)

Graph.Finnish(g)

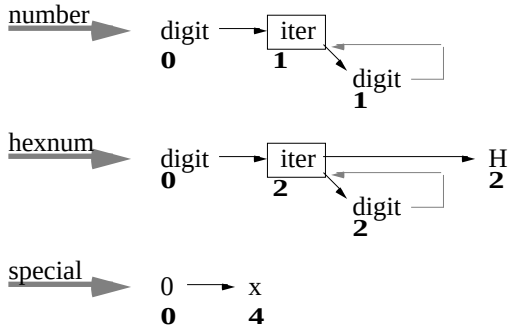


2. Scanner automaton

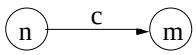
Declarations:

```
CHARACTERS
   digit = '0'..'9'.
   hex   = digit + 'a'..'f'.
TOKENS
   number  = digit {digit}.
   hexnum  = digit {digit} 'H'.
   special  = "0x".
```

Syntax graph for the tokens

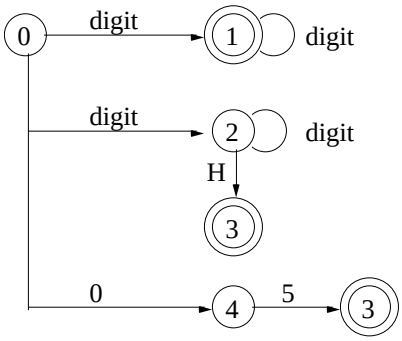


The bold numbers denote the states that were assigned to the nodes by *DFA.NumberNodes*. They are used to derive the automaton from the graph as follows: if a node for a character or a character class *c* has the number *n* and its *next* pointer points to a node with number *m*, then this leads to a transition



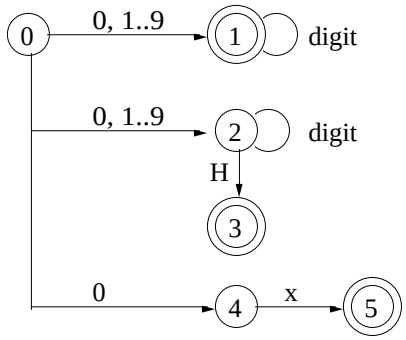
If there is no next node, the transition leads to a new state.

Nondeterministic automaton



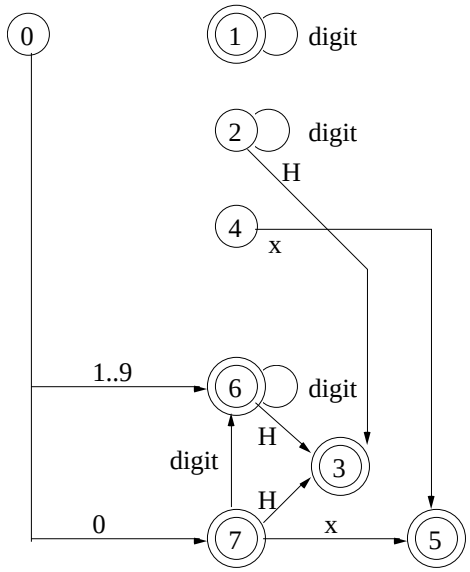
The automaton is nondeterministic since there are three transitions with '0' in state 0 and two with *digit* in state 0. As a first step in making the automaton deterministic overlapping character ranges are split. This is done by *DFA.MakeUnique*.

After MakeUnique



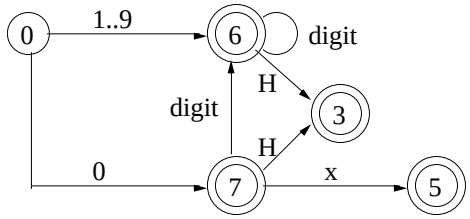
The next step is to melt those states that can be reached by a transition with the same symbol from the same state. This is done in *DFA.MeltStates*.

After MeltStates



The only remaining task now is to delete the redundant states (here 1, 2 and 4).

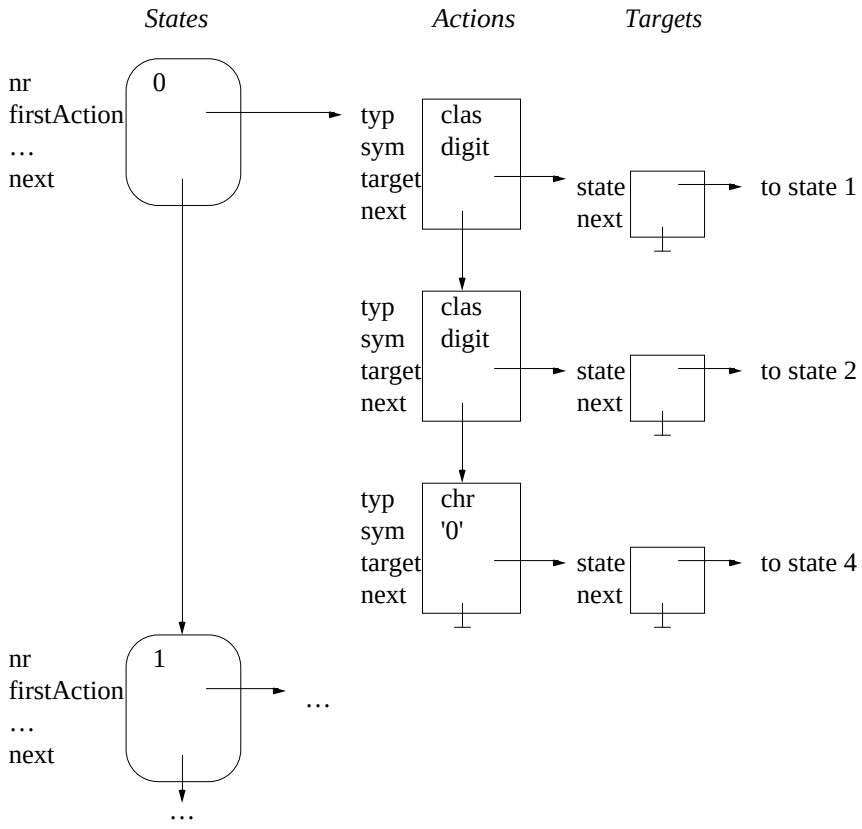
After DeleteRedundantStates



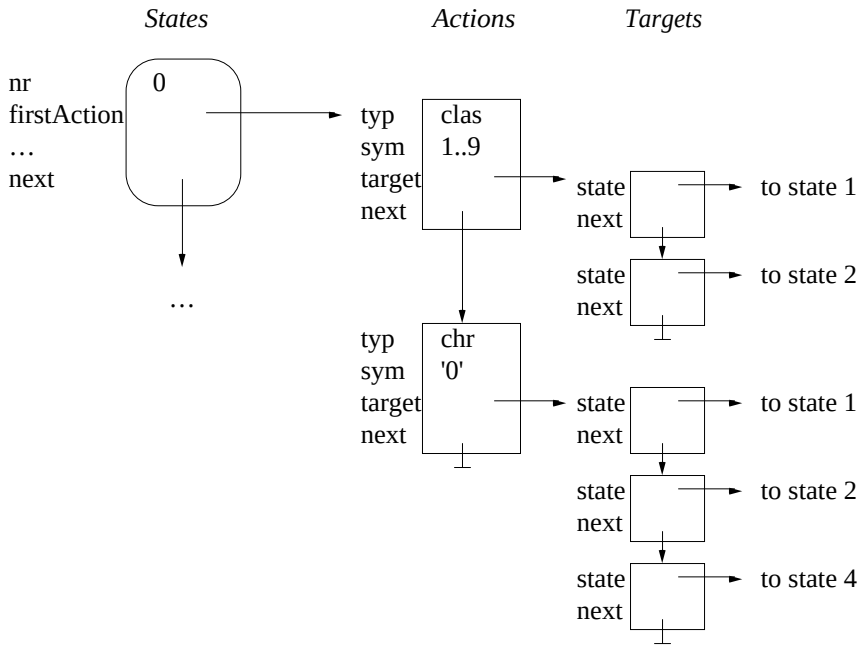
This is the resulting deterministic finite automaton from which the scanner is generated.

Concrete data structures

Nondeterministic automaton



After MakeUnique



After MeltStates