About the WSN2BNF compiler

Radboud University Nijmegen
the Netherlands
Louis ten Bosch

Brief description of the WSN2FSM compiler

This compiler translates an input file in Wirth Syntax Notation (WSN) format into a text file that specifies a weighted Finite State Transducer. The compiler is developed within the SPRAAK project 2007-2008 by Frank Kusters. The compiler, **wsn2fsm.pl**, is written in perl.

WSN

Like BNF, WSN specifies a syntax.
Originally proposed by Wirth in 1977 as an alternative to BNF, it has several advantages over BNF. The most important one is probably that its definition allows fewer alterations and is therefore better described. In contrast, BNF has many variants and extensions - such as EBNF. WSN has been used in several standards (e.g. ISO 10303-21).

Example of a WSN-specified file (example4.wsn):

```
!grammar example;

!start <main>;

<_main_> :== <S> ;

<S> :== <DET> <ADJECTIVE> <OBJECT>  ;

<DET> :== de | een ;

<ADJECTIVE> :==  [0.4] <AA> | [0.5] ( kleine | kleine2 ) | [0.25] rode  ;

<OBJECT> :== auto en [ nog ] { meer } ;


<AA> :== "boze 123" | kwade  ;
```

The call

```
perl wsn2fsm.pl example4.wsn
```

produces the following weighted FSM specification (see also the manual):

```
0 0 3 "de" 0.00
1 0 3 "een" 0.00
2 3 2 "boze 123" -0.92
3 3 2 "kwade" -0.92
4 3 2 "kleine" -0.69
5 3 2 "kleine2" -0.69
```

```
6 3 2 "rode" -1.39
7 2 6 "auto" 0.00
8 6 5 "en" 0.00
9 5 4 epsilon 0.00
10 5 4 "nog" 0.00
11 4 4 "meer" 0.00
12 4 1 epsilon 0.00
```

Description

In the **input** file of wsn2fsm.pl, the following conventions are used.

The entire specification a WSN-formatted input is in terms of lists of production rules (productions). A production is terminated by a semicolon (;) and is of the form

```
<A> :== right-hand side
```

The element on the left is defined to be the combination of elements on the right. Elements at the right-hand side may be terminals and non-terminals. Non-terminals are indicated by brackets (<>).

The algorithm looks for the non-terminal <_main_>. This non-terminal is the single reserved one. The epsilon is a reserved terminal (see manual).

- "a b" means "a and b": "a followed by b"
- "a | b" means "a or b"
- "epsilon" denotes the empty string
- Repetition is denoted by curly brackets: {a} stands for "epsilon | a | a a | a a a | ..."
- Optionality is expressed by square brackets: "[a] b" stands for "a b | b"
- Parentheses serve groupings: "(a | b) c" stands for "a c | b c"
- Without further parentheses, the ordering is: "or < optionality/repetition < and"

Strings between double brackets are treated as single words. See "boze 123" above.

The terms figuring in an "or" expression can be assigned a probability by adding numbers between square brackets, as indicated in the example above. These probabilities translate into arc penalties in the resulting FSM.

The **output** file consists of lines, each line representing one arc.
A line contains an arc id, the id of the begin node of this arc, the id of the end node of this arc, a label and a score. The score corresponds to the natural logarithm of the probability that is specified in the input.
By default, 0 and 1 represent the unique begin node and unique end node of the entire FSM (graph).

Epsilons

Epsilons (empty arcs in output) are avoided as much as possible. In some cases however
they are used by default even if they could be avoided. Examples are provided by the cases

```
{ a } { b } # epsilon cannot be avoided
{ a } b     # epsilon is used but could be avoided
```

For more details see the manual.

Error handling

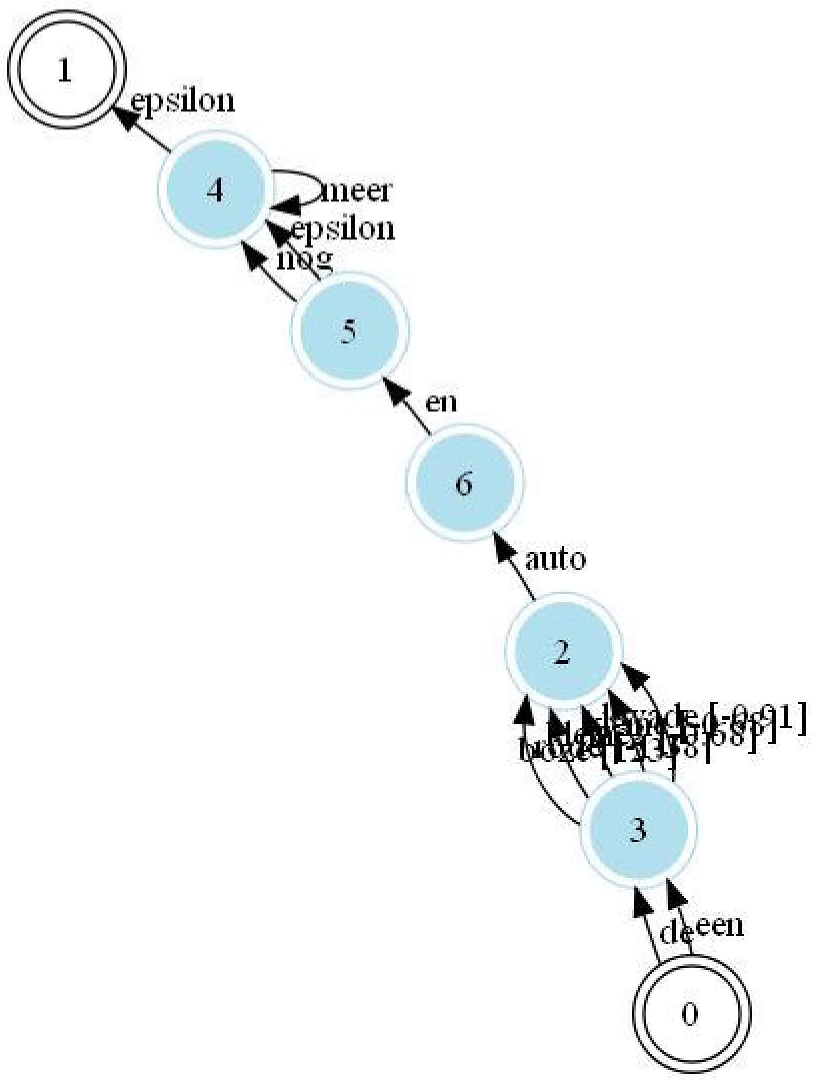
The compiler is able to detect many formatting errors. If appropriate, warnings are printed to stdout.

Graphical representations

The output of wns2fsm can be converted by a perl script wns2fsm2dot.perl into a dot-formatted file, which can be interpreted by graph visualisation tools such as the publicly available tool graphviz (http://www.graphviz.org). The dot-formatted output of this perl script is shown below.

```
digraph X {
size = "8"
node [shape = doublecircle]
node [color=lightblue2, style=filled];
0 -> 3 [label = "de"];
0 -> 3 [label = "een"];
3 -> 2 [label = "boze [123]"];
3 -> 2 [label = "kwade [-0.91]"];
3 -> 2 [label = "kleine [-0.68]"];
3 -> 2 [label = "kleine2 [-0.68]"];
3 -> 2 [label = "rode [-1.38]"];
2 -> 6 [label = "auto"];
6 -> 5 [label = "en"];
5 -> 4 [label = "epsilon"];
5 -> 4 [label = "nog"];
4 -> 4 [label = "meer"];
4 -> 1 [label = "epsilon"];
}
```

The picture, obtained by using graphviz with this dot-formatted file as input, is shown below. The picture can in many ways be optimized to improve intelligibility by using more sophisticated dot-commands. The perl script wsn2fsm2dot.perl only provides an example.



Louis ten Bosch
Radboud University Nijmegen