Group	: Computational Linguistics
Topic	: formalism

Title	: **Surface Rule Compiler**
Author	: René Leermakers

Doc.Nr.	: 0164
Date	: 4-12-86
Status	: concept
Supersedes	: -
Distribution	: Project
Clearance	: Project
Keywords	: Surface grammar, notation, compiler

Institute for Perception Research

Contents

1 Introduction

In this section I describe the surface rule compiler. It was devised in order to enable the rule-writers to test the rules quickly. In a later stage, Carel will implement a more ambitious compiler. This will involve some minor changes of notation, as I had to deviate a little from the notation Carel proposed in his 'engineeral' thesis.

The present compiler truly compiles only the rule regular expressions, and transduces the remaining part of the rule in a simple way.

For rule-writers, the main part of this document is the example of section 2. Section 3 gives the basis grammar for the regular expressions of surface rules, section 4 lists the macros used in the condition/action parts of the rules and section 5 displays the output as it is generated from the rule of section 2. Lastly the last section gives the attribute grammar actually used in the compiler.

2 Syntax by Example

Because of the transducing part of the compiler, no complete syntax of the rule notation can be given. However, the next section contains a description that is as formal as possible. Here we give a sample input file, with a Rosetta2 surface rule that should contain all relevant features. The file is to be called SURFRULES.SUR.

```
%ENGLISH
&

%PPrule
RegularExpression:
  PP=[PREP/1].NP/2.{preporpart}
  preporpart = PREP/4|PART/3
ConditionsAndActions:
  VAR moodvar::moodtype;
      prepfound::BOOLEAN;
      exppostpkey::baskeytype;
      prepkeyvar::baskeytype;

BEGIN
<*
 HINIT:BEGIN
         moodvar:=omegamood;prepfound:=false;exppostpkey:=0;prepkeyvar:=0
       END;
 1    :<*
```

```
        LOCALCONDITION:%PREP.soort in [gewoneprep,splitprep]
        GLOBAL: #CONDITION: TRUE
                #ACTION:  BEGIN
                          SYNREL:=headrel;
                          IF %PREP.soort=splitprep THEN exppostkey:=postpkey;
                          prepkeyvar:=%PREP.key;
                          prepfound:=TRUE
                          END
       *>
 2     :<*
        LOCALCONDITION: (%NP.soort<>hetpro) AND
                        (%NP.cases * [dative,accusative] <> [])
        GLOBAL: #CONDITION: TRUE
                #ACTION: BEGIN
                          SYNREL:=objrel;
                          moodvar:=%NP.mood
                          END
       *>
 3     :<*
        LOCALCONDITION: TRUE
        GLOBAL: #CONDITION: prepfound AND (exppostkey = %PART.key)
                #ACTION: BEGIN
                          SYNREL:=postpreprel
                          END
       *>
 4     :<*
        LOCALCONDITION: (%PREP.soort=postprep)
        GLOBAL: #CONDITION: not(prepfound)
                #ACTION: BEGIN
                          SYNREL:=headrel;
                          prepkeyvar:=%PREP.key;
                          prepfound:=true
                          END
       *>
 HFINAL: #CONDITION: prepfound
         #ACTION: BEGIN
                   MAKET_PP;
                   IF moodvar <> omegamood THEN $PP.mood:=moodvar
                       ELSE $PP.mood:=omegamood;
                   $PP.advsoort:=anderevar;
                   $PP.prepkey:=prepkeyvar;
                   $PP.tense:=omegatense
                   END
*>
```

```
END;
&
```

Some comment is due. The if-then-else construct in the final action makes no sense, in this case. It is written to illustrate that the construct may be used in the final (or other, but this can always be avoided) action.

The file SURFRULES.SUR has to start with the name of the language between the symbols '%' and '&'/. These symbols, when they are the first character of a line, also signal the beginning and the end of a surface rule.

The attributes of categories appearing in the regular expression are prefixed by %, followed by the category name, followed by a period. A new top node, with the category PP in the example, is created by the statement MAKET_, followed by the category name. It attributes are referred to by prefixing them by &, followed by the category name and a period.

3 Syntax grammar

The regular expression part of a rule satisfies a simple syntax, which is given here.

```
utt =rulename."RegularExpression:".ident.'='.graph.{helpgraph}.
                                                    "ConditionsAndActions:"
graph = concgraph.{"|".concgraph}
concgraph = elementarygraph.{".elementarygraph}
elementarygraph = "(".graph.")" | "[".graph."]" | "{".graph."}" |
                   ident | ident."/".number
helpgraph = ident."=".graph
number = '1'|'2'| .. |'99'
```

4 Compilation and transduction

In the translation of the conditions and actions, various macros are translated as follows:

```
LOCALCONDITION: -> loccond:assignstatus(
GLOBAL: -> );globcond: BEGIN
#CONDITION: -> assignstatus(
#ACTION: -> );IF status THEN
HFINAL: -> Hfinal: BEGIN
*> -> END; END;
<* -> CASE a OF {the first one} | CASE mode OF {the other ones}
%'cat'. -> b^.ls^.'cat'field.
```

```
$'cat'. -> top^.ls^.'cat'field.
MAKET_'cat' -> MAKET_'cat'(top);addnewtop(top)
:: -> :[STATIC]
```

The translation of these macros takes place in the scanner module of the compiler.

5 Output files

Below I list the output files corresponding to the above rule, just to give an idea. Details will change in future of course. First the surface graph defining module, which represents the regular expression in binary form.

```
[INHERIT('GENERAL:listree','ENGLISH:lsdomaint','ENGLISH:maket'
,'ENGLISH:copyt','GENERAL:surfrulesgraphs','ENGLISH:lsstree')]
MODULE surfrulesgraphs;

{rule:}
{:PPrule}
function preporpartgraph:psurfgraph;
BEGIN
preporpartgraph:=
alt(atom(PREP,4),
    atom(PART,3)
    )
END;
procedure PPrulegraph(i:INTEGER);
BEGIN
prod(i,PP,
    conc(opt(atom(PREP,1)
            ),
        conc(atom(NP,2),
            star(preporpartgraph
                )
            )
        )
    )
END;
{:PPrule}
[[GLOBAL] procedure SFG(i:INTEGER
BEGIN
CASE i OF
  1:PPrulegraph(  1);
```

```
END
END;
END.
```

The condition/action parts are contained in another module. As it comes out of the compiler, lay-out is terrible, so I edited it somewhat.

```
[INHERIT('GENERAL:listree','ENGLISH:lsdomaint','ENGLISH:maket'
,'ENGLISH:copyt','GENERAL:surfrules','ENGLISH:lsstree')]
MODULE surfrules;
PROCEDURE PPrule(a:nodeid;b:psnode;mode:surfrulemode);
VAR moodvar:[STATIC]moodtype;
prepfound:[STATIC]BOOLEAN;
exppostpkey:[STATIC]baskeytype;
prepkeyvar:[STATIC]baskeytype;
BEGIN
CASE a OF
HINIT:BEGIN
      moodvar:=omegamood;prepfound:=false;exppostpkey:=0;prepkeyvar:=0
      END;
1 :CASE mode OF
     loccond:assignstatus(b^.ls^.PREPfield.soort in [gewoneprep,splitprep]
   );globcond: BEGIN assignstatus( TRUE
               );IF status THEN BEGIN
               SYNREL:=headrel;
               IF b^.ls^.PREPfield.soort=splitprep THEN exppostkey:=postpkey;
               prepkeyvar:=b^.ls^.PREPfield.key;
               prepfound:=TRUE
               END
               END;
   END;
2 :CASE mode OF
     loccond:assignstatus( (b^.ls^.NPfield.soort<>hetpro) AND
            (b^.ls^.NPfield.cases * [dative,accusative] <> [])
   );globcond: BEGIN assignstatus( TRUE
              );IF status THEN BEGIN
              SYNREL:=objrel;
              moodvar:=b^.ls^.NPfield.mood
              END
              END;
   END;
3 :CASE mode OF
     loccond:assignstatus( TRUE
```

```
   );globcond: BEGIN assignstatus( prepfound AND (exppostpkey = b^.ls^.PARTfield.key)
               );IF status THEN BEGIN
               SYNREL:=postpreprel
               END
               END;
   END;
4 :CASE mode OF
     loccond:assignstatus( (b^.ls^.PREPfield.soort=postprep)
   );globcond: BEGIN assignstatus( not(prepfound)
              );IF status THEN BEGIN
                SYNREL:=headrel;
                prepkeyvar:=b^.ls^.PREPfield.key;
                prepfound:=true
                END
                END;
  END;
Hfinal: BEGIN assignstatus( prepfound
             );IF status THEN BEGIN
               MAKET_PP(top);addnewtop(top);
               IF moodvar <> omegamood THEN top^.ls^.PPfield.mood:=moodvar
                 ELSE top^.ls^.PPfield.mood:=omegamood;
               top^.ls^.PPfield.advsoort:=anderevar;
               top^.ls^.PPfield.prepkey:=prepkeyvar ;
               top^.ls^.PPfield.tense:=omegatense
               END
        END;
  END;
END;
[GLOBAL] procedure surfrule(rnr:INTEGER;a:nodeid;
                            b:psnode;mode:surfrulemode);
BEGIN
CASE rnr OF
  1:PPrule(a,b,mode);
END
END;
END.
```

6 Attributed grammar

```
1: utt = rulename/1.Regexp/1.ident/1.iscat/1.graph/1.{helpgraph/2}.CAs/1
parameters:numofhelpgraphs:integer:0
```

```
1:void
2:local:true
  global:numofhelpgraphs:=numofhelpgraphs+1;
Hfinal:mkutt;numofhelpgraphs:=numofhelpgraphs

2: graph = concgraph\2.{vertline\1.concgraph\2}
parameters:numofconcgraphs:integer:0
1:void
2:local:true
  global:numofconcgraphs:=numofconcgraphs+1;
Hfinal:mkgraph;numofconcgraphs:=numofconcgraphs

3: concgraph = elementarygraph\2.{dot\1.elementarygraph\2}
parameters:numofelementarygraphs:integer:0
1:void
2:local:true
  global:numofelementarygraphs:=numofelementarygraphs+1;
Hfinal:mkgraph;numofelementarygraphs:=numofelementarygraphs

4: elementarygraph = roundopen/2.graph/1.roundclose/1 |
                  squareopen/3.graph/1.squareclose/1 |
                  curlyopen/4.graph/1.curlyclose/1 |
                   ident/5 | ident/1.slash/6.number/1
parameters:graphtype:(enclosedgraph,optgraph,stargraph,helpgraphident,
                                                 atomgraph):atomgraph
1:void
2:local:true
  global:graphtype:=enclosedgraph
3:local:true
  global:graphtype:=optgraph
4:local:true
  global:graphtype:=stargraph
5:local:true
  global:graphtype:=helpgraphident
6:local:true
  global:graphtype:=atomgraph
Hfinal:mkelementarygraph;graphtype:=graphtype

5: helpgraph = ident\2.iscat\1.graph\1
parameters:str:string:''
1:void
2:local:true
  global:str:=str
Hfinal:mkhelpgraph;str:=str
```

```
6: number = charstring/1
1:local:checknumber(str)
  global:true
Hfinal:mknumber

7: CAs = charstring/1
1:local:checkCAs(str)
  global:true
Hfinal:mkCAs

8: RegExp = charstring/1
1:local:checkRegExp(str)
  global:true
Hfinal:mkRegExp

9: ident = charstring/1
parameters:str:string:''
1:local:true
  global:str:=str
Hfinal:mkident;str:=str
```