

Lex Augusteijn

**The Diagrams EBNF to PostScript compiler,
release 7.2**

Lex Augusteijn

The Diagrams EBNF to PostScript compiler, release 7.2

ABSTRACT

This document describes the **Diagrams** tool, which is a compiler from EBNF production rules to syntax diagrams in PostScript. Apart from the traditional extended BNF constructions, notations for layout are available. Scaling and positioning is completely automatic.

Contents

1	Introduction	2
2	Syntax definition	2
3	Short-cuts	4
4	Switches	4
5	Cooperation with Elegant	5
6	Font directory	5

1 Introduction

This document describes the **Diagrams** tool, which is a compiler from EBNF production rules to syntax Diagrams in PostScript. Apart from the traditional extended BNF constructions, notations for layout control are available.

2 Syntax definition

The extended BNF syntax has the following form:

```

<specification> ::= { [ <formfeed> ] <rule> }*
<rule>          ::= <ident> <derives> <disjunction> .
<disjunction>  ::= <alternation> // <or>
<alternation>  ::= <conjunction> // <separates>
<conjunction>  ::= [ <basic> // <and> ]
<basic>        ::= ( <disjunction> )
                | [ <disjunction> ]
                | { <disjunction> } [ + ]
                | <ident>
                | <terminal>

<derives>      ::= ::= | =
<or>           ::= | | \ | | |< | |>
<and>          ::= | \ | \< | \>
<separates>    ::= // | \ \
<formfeed>     ::= <the ascii formfeed character>
<ident>        ::= <letter> { <letter> | <digit> | - | _ }*
<terminal>     ::= ' ( <any character> | { <any character without ' or \ | \ ' | \ \ }+ ) '
                | " ( <any character> | { <any character without " or \ | \ " | \ \ }+ ) "

```

The meaning of these constructions is best explained by presenting their compilations into syntax diagrams.

The EBNF rules:

```

disjunction-1 ::= a | b c | d e f .
disjunction-2 ::= a \ | b c \ | d e f .
disjunction-3 ::= a |< b c |< d e f .
disjunction-4 ::= a |> b c |> d e f .

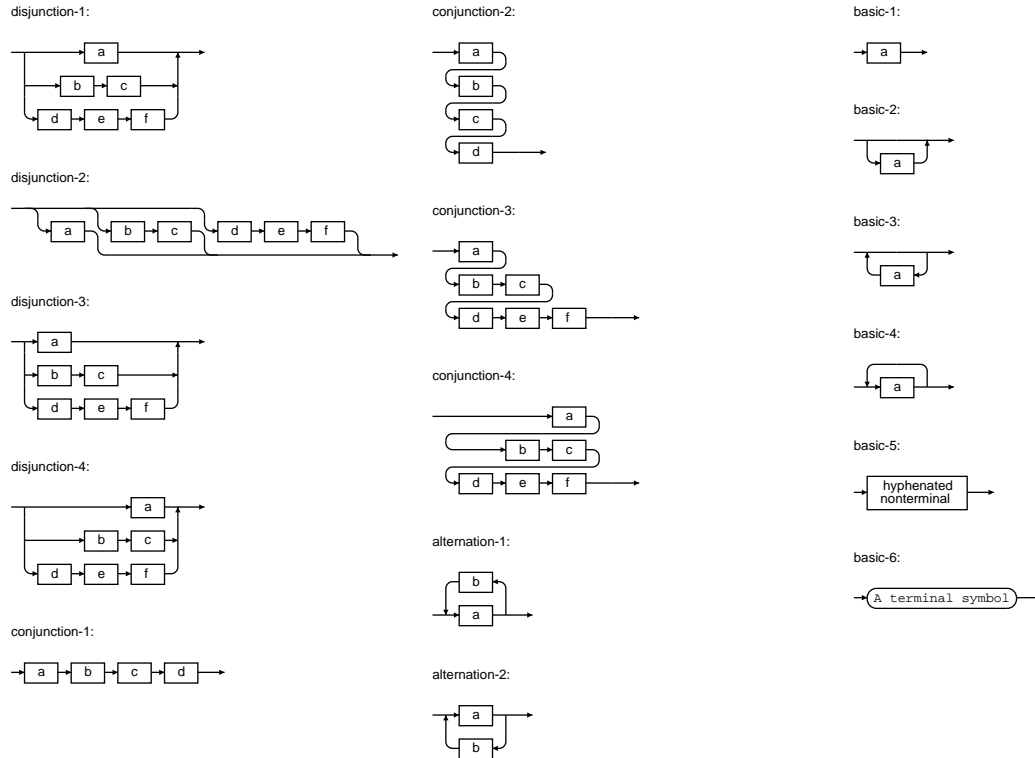
conjunction-1 ::= a b c d .
conjunction-2 ::= a \ b \ c \ d .
conjunction-3 ::= a \< ( b c ) \< ( d e f ) .
conjunction-4 ::= a \> ( b c ) \> ( d e f ) .

alternation-1 ::= a // b .
alternation-2 ::= a \ \ b .

basic-1 ::= ( a ) .
basic-2 ::= [ a ] .
basic-3 ::= { a } .
basic-4 ::= { a }+ .
basic-5 ::= hyphenated-nonterminal .
basic-6 ::= 'A terminal symbol' .

```

are compiled into the following diagrams: It is possible to use two forms of production rules,



with different *derives* symbols. A rule using = does not produce a diagram, but serves as a macro definition. The right hand side of the rule is substituted everywhere where the left hand side occurs. Cyclic macros are not allowed. These macros are not to be regarded as textual ones. Rather, they are structural macros, which means that the right-hand-side of the macro is treated as if it were inclosed by parentheses.

As an example, we give the diagrams for the **Diagrams** tool itself:

```

specification ::= { [formfeed] rule } .
rule          ::= ident derives disjunction '.' .
disjunction  ::= alternation // or .
alternation  ::= conjunction // separates .
conjunction  ::= [ basic // and ] .
basic        ::= '(' disjunction ')'
              | '[' disjunction ']'
              | '{' disjunction '}' [ '+' ]
              | ident
              | terminal
              | 'EMPTY' .

derives      = '::=' | '=' .
or           = '|' | '\\|' | '<' | '>' .
and         = '\\<' | '\\<' | '\\>' .
separates   = '/' | '\\\\' .
ident       ::= letter { letter | digit | '-' | '_' } .
terminal    ::= '\\'' (any | {char | '\\''}+) '\\''
              | '\\'' (any | {char | '\\''}+) '\\'' .
formfeed    = 'ASCII form feed' .

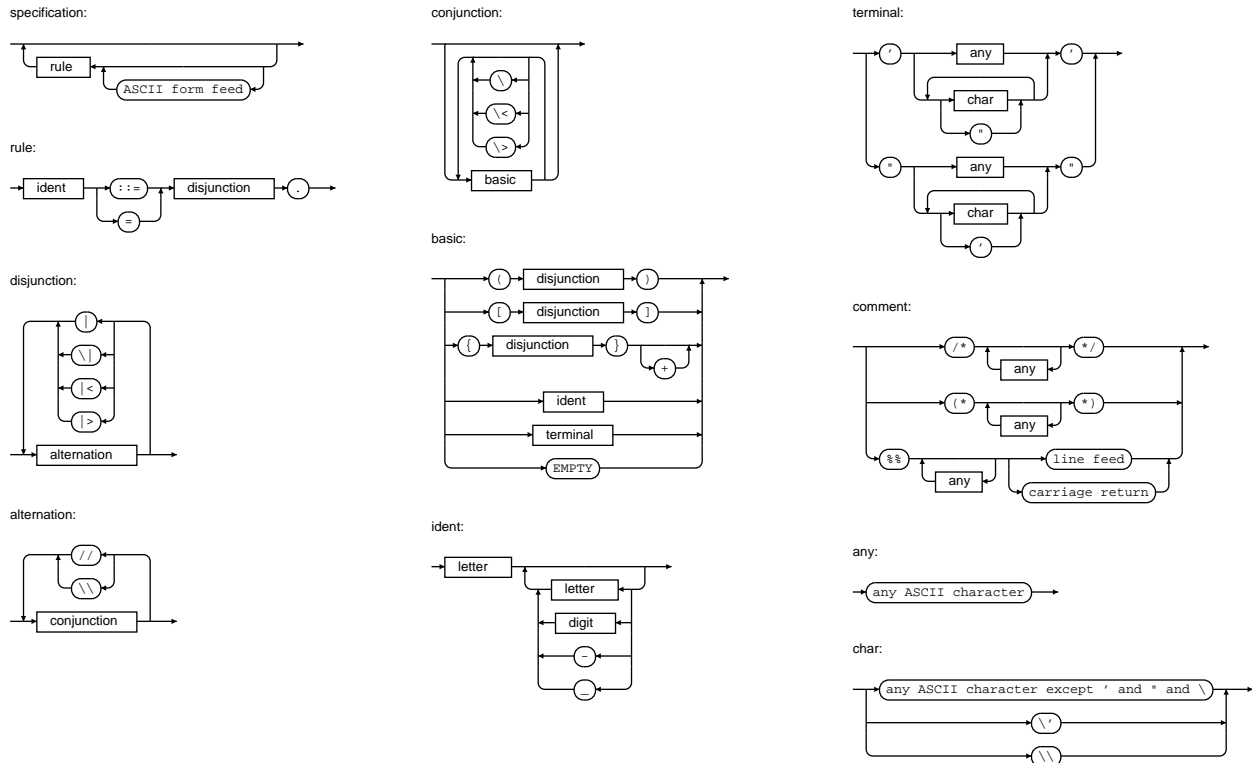
```

```

comment      ::= '/' * { any } '*' /
              | '(' * { any } '*' )
              | '%' * { any } feed .
feed         = 'line feed' | 'carriage return' .
any          ::= 'any ASCII character' .
char        ::= 'any ASCII character except \' and " and \'\' | '\\\' | '\\\\' .

```

This EBNF is compiled into the diagrams:



3 Short-cuts

When elements of a conjunction are drawn above or below each other, two constructions are treated in a special way, as shown by the following example:

```
shortcut-1 ::= [ a ] \ [ b ] \ [ c ] .
```

```
shortcut-2 ::= { a } \ { b } \ { c } .
```

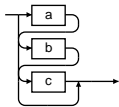
```
shortcut-3 ::= { a }+ \ { b }+ \ { c }+ .
```

This EBNF is compiled into the diagrams:

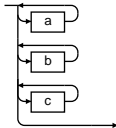
4 Switches

Diagrams supports the following command switches:

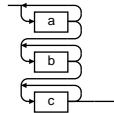
shortcut-1:



shortcut-2:



shortcut-3:



usage: Diagrams switches input ...

Possible switches:

```

-v          : Report version number and creation date
-w          : Suppress warnings
-scale s1 s2 : Apply scale factor s1/s2
-Scale s1 s2 : Apply absolute scale factor s1/s2
-height h   : Set paper height to h cm, default A4
-width w    : Set paper width to w cm, default A4
-Height h   : Set diagram height to h cm, default 25.5 cm
-Width w    : Set diagram width to w cm, default 16.8
-topmargin t : Set page top margin to t cm, default 2.12 cm
-leftmargin l : Set page left margin to l cm, default 2.12 cm
-centerhorizontal      : Center diagram horizontally
-centervertical       : Center diagram vertically
-columns c            : Use c columns per page, default 1
-pagenr n            : Number pages, starting with n
-landscape           : Use landscape format
-fontlib dir        : Use dir as library for font description files
-ebnf file          : Generate proper EBNF in file
-latex              : Generate LaTeX includable PostScript
-singlefile         : Generate all pages in one file
-multifiles         : Generate each page in a separate file (default)

```

`Diagrams` computes itself a scale factor to ensure that the diagrams fit on the pages. An additional scale factor can be supplied by the `-scale` switch. The automatically computed scale factor can be overruled by the use of the `-Scale` switch. The size of the paper, diagrams and margins can be supplied. Specifying inconsistent values for these gives an undefined behavior. `Diagrams` can generate input for the Bnf tool by giving the `-ebnf` switch.

5 Cooperation with Elegant

The tools `Elegant` and `ScanGen` [Aug02a, Aug02b] can generate an EBNF from an attribute grammar and a scanner specification respectively. When these are both passed to `Diagrams`, a complete set of syntax diagrams is obtained.

6 Font directory

PostScript fonts are described in so called *afm-files* (Adobe font metrics files). `Diagrams` attempts to read those files for the two fonts it uses (Courier and Helvetica) in order to be able to determine the sizes of the characters in those fonts. By default the directory `/usr/tran/sparc/lib`

is taken, but this can be overruled by the use of the `-fontlib` switch.

The directory `/${ELEGANTROOT}/lib/afm` contains a copy of these afm-files.

If the afm-files can not be read, **Diagrams** makes an estimate for the sizes of the characters, which can result in too wide or too narrow boxes for terminal or non-terminal symbols.

References

- [Aug02a] Lex Augusteijn. Definition of the programming language **Elegant**, release 7.2. Document <http://www.extra.research.philips.com/ist>, Philips Research Laboratories, Eindhoven, the Netherlands, August 2002.
- [Aug02b] Lex Augusteijn. The **Elegant** scanner generator definition language **ScanGen**, release 7.2. Document <http://www.extra.research.philips.com/ist>, Philips Research Laboratories, Eindhoven, the Netherlands, August 2002.