Generating syntax diagrams with SYNGEN

Jens Klöcker

kloecker@springer.de

November 16, 1996

Contents

1 Introduction

A very important, but unfortunately mostly neglected job in software engineering is the production of good documentation. One often arising problem in this field is the documentation of user languages, i. e. programming languages, data base languages or macro languages in text processing systems. Possible criterions for a good presentation are

1. clearness,
2. compactness,
3. completeness and
4. efficiency in production.

At this time nearly all authors use the so called *Backus-Naur-Form BNF*, which original was invented to design and not to document or teach languages. For an example look at [1]. This form of documentation satisfies points 2–4. In particular, criterion 4 is given, perhaps the BNF can be taken directly or with slight modifications from parser specifications such as YACC. But clearness, especially for beginners, is not given. Because language documentation must be useful for experts as well as for beginners, the BNF is not the right technique for that purpose.

One possibility for a user friendly design of language documentations is a method, which is called *syntax diagrams*. Good examples can be found in [2] and [4], where they are used to explain the syntax of GOFER and CAML LIGHT. This kind of graphical representation is clear (1), compact (2) and the language can be documented completely.

SYNGEN combines the advantages of efficient production of BNF with the clearness of syntax diagrams. The program takes a BNF-like specification of the language as input and the output is the corresponding syntax diagram in LaTeX form. For that, every left hand side of the BNF must be converted in a diagram with a description similar to the right hand side. The result of the SYNGEN-run on this description can then be included without additional macros in every LaTeX2e document.

As a first application, SYNGEN was used to produce the syntax diagrams for the whole CAML LIGHT language. The result can be found in the previously refered book on functional programming [4].

2 How to make syntax diagrams

In this section we will show how to make syntax diagrams. Therefore the description language ARA (this stands for Enriched Regular Expressions and was derived from the german “Angereicherte Reguläre Ausdrücke”) has to be explained and the usage of SYNGEN must be described.

2.1 Basic elements

2.1.1 Blanks

The characters space, horizontal tabulation and newline are considered as blanks. Blanks are ignored, but they separate adjacent non-terminals.

2.1.2 Non-terminals

Non-terminals are sequences of lowercase and uppercase letters, digits, underscore and the minus sign. All characters are meaningful and can stand on every position in the string. The length is not limited. In the corresponding diagram non-terminals are represented by rectangular boxes. Diagram 1 gives a summary of the rules.

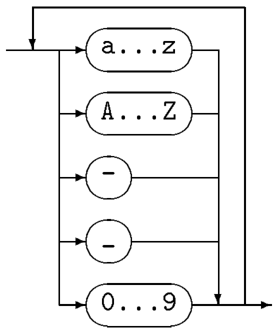


Diagram 1: Non-terminals (`nonterm`)

2.1.3 Terminals

In ARA terminals are enclosed in `''` characters. They can contain any ASCII character from 33 (`!`) to 126 (`~`). Exceptions are the character `'` and the backslash—they must be quoted with the backslash. After compilation with SYNGEN, terminals appear in oval boxes. The complete syntax is given in Diagram 2.

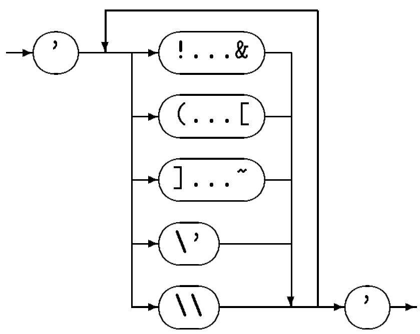


Diagram 2: Terminals (`term`)

2.1.4 Parameters

To get a compact representation, one can use a diagram to represent more then one language element. For every represented non-terminal a *parameter element* can be introduced. With that element a path can be locked for all non-terminals not mentioned in it. Drawn with SYNGEN, parameter elements are enclosed in dashed boxes.

Parameters may consist of all ASCII characters from 33 (`!`) to 126 (`~`). Exceptions are the `>` sign and again the backslash, which is used to represent this characters. Parameters has to be enclosed in `<` and `>` signs. The complete rules are shown in Diagram 3.

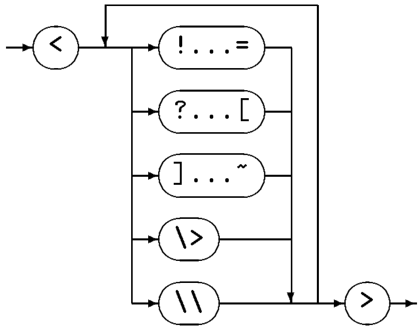


Diagram 3: Parameter (`param`)

2.2 Building up diagrams

The basic elements of the last section can be combined to syntax diagrams according to the following rules. The operators are listed in reversed order of their priority. A summary of the rules is given in the (recursive) Diagram 4.

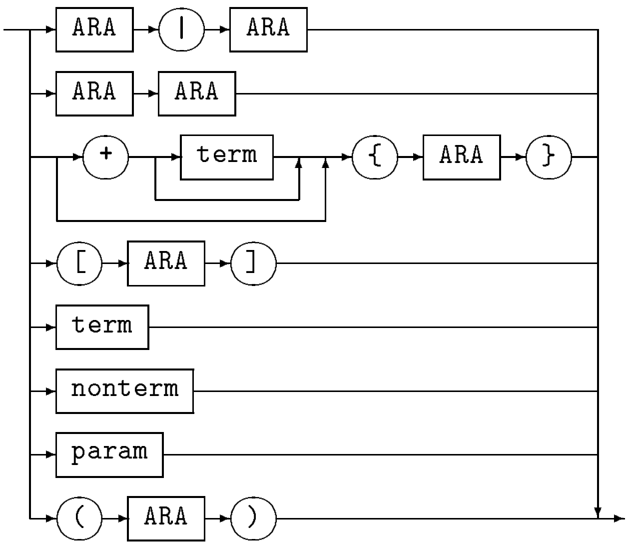


Diagram 4: Enriched Regular Expressions (`ARA`)

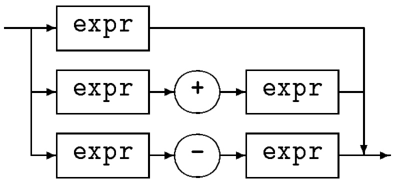
2.2.1 Choice

If there are several possible ARA's from which one can choose the right, then the cases have to be separated by | characters.

Example

```
    expr
  | expr '+' expr
  | expr '-' expr
```

⇒



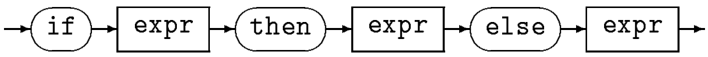
2.2.2 Concatenation

To get a sequence of ARA's, they are written directly one behind the other.

Example

```
'if' expr 'then' expr 'else' expr
```

⇒



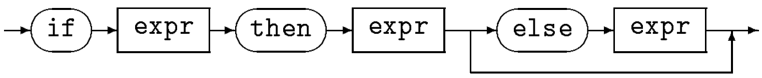
2.2.3 Optional components

If an ARA is optional, the appropriate expression has to be enclosed in square brackets.

Example

```
'if' expr 'then' expr ['else' expr]
```

⇒



2.2.4 Repetitions

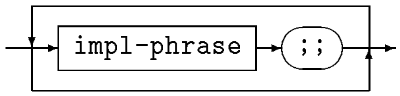
There are three ways to repeat an expression in ARA.

- Zero or several repetitions—denoted with curly brackets.

Example

```
{impl-phrase ';;'}
```

⇒

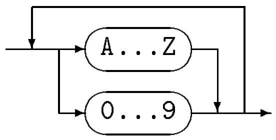


- One or several repetitions—denoted in curly brackets with a leading plus sign.

Example

```
+{ 'A...Z' | '0...9' }
```

⇒

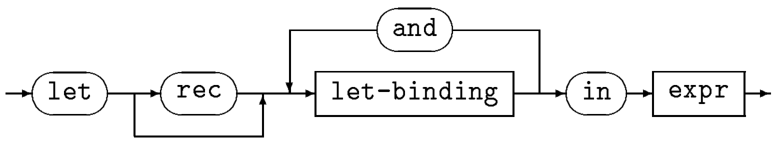


- One or several repetitions with a separating terminal. This construct is like the former one, except that the additional terminal has to be inserted between the plus sign and the opening bracket.

Example

```
'let' ['rec'] + 'and' { let-binding } 'in' expr
```

⇒

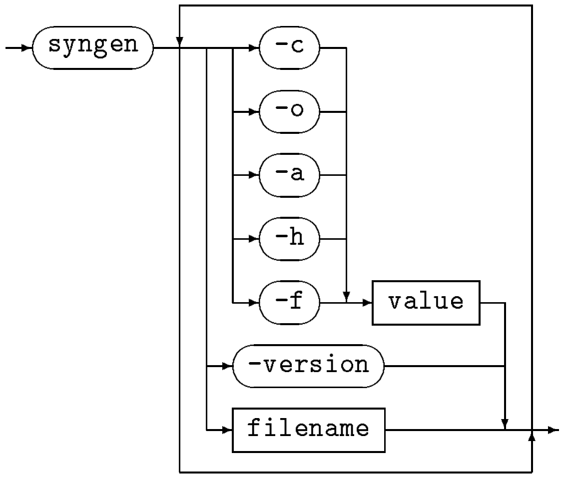


2.2.5 Grouping

Every expression in ARA can be enclosed in parentheses to form groups. For an example look at the concluding Diagram 5 in this section.

2.3 Options

The following diagram shows the syntax of a SYNGEN call.



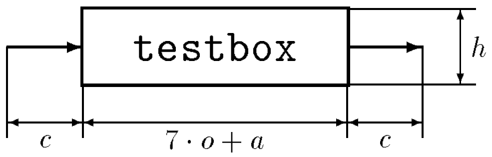
The look of the output can be controlled by the several options. Because the syntax diagrams are build up from the basic elements, the options are mostly deal with the size of terminal, non-terminal and parameter boxes. All values, except f, have to be given in terms of `\unitlength`, which is by default equal to 1 pt. The options are:

option	meaning
`-c` c	Length of connect lines on basic boxes.
`-o` o	Length of a character.
`-a` a	Additional offset to the box length.
`-h` h	Height of basic boxes.
`-f` f	Vertically stacked boxes are separated by $h \cdot f$.

The next values are a good choice for the normal Computer Modern Typewriter Type font family. They are listed in dependency of the font size and under the assumption, that `\unitlength` is equal to 1 pt.

font size	c	o	a	h	f
10 pt	8	5.24995	10.5	14	0.5
11 pt	10	5.69145	11.4	16	0.5
12 pt	12	6.175	12.4	18	0.5

The values for a 10 pt font are predefined and used, if no options are given. The next picture shows the effect of the values c, o, a and h on a non-terminal box with a seven character string.



2.4 A complete example

The problem is to make a diagram for an enumeration definition and declaration of the language C++ (taken from [3]) and to include it in a 11 pt LaTeX-document. The description in ARA is:

```
'enum' (   <def> [name] '{' + ',' {name ['=' expr]} '}'
         | <dekl> name
        )
```

This input is compiled by SYNGEN with the call

```
syngen -c 10 -h 16 -o 5.69145 -a 11.4 example.ara
```

and can be included in the document with

```
\begin{figure}[ht]
 \centerline{\input{example.syn}}
\caption{$<$dekl, def$>$ enumeration}
\end{figure}
```

The result is given in Diagram 5.

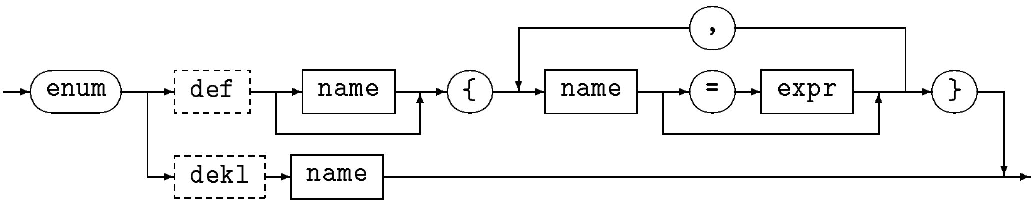


Diagram 5: <dekl, def> enumeration

3 How to read syntax diagrams

The rules for reading syntax diagrams are shown language independent in the following points.

- Every diagram represents one language element or a group of them. If a group is given, the diagram is parameterized with the individual elements.
- Non-terminals are represented by rectangular boxes, terminals by oval boxes and parameters are enclosed in dashed boxes.
- The run through the diagram starts at the leftmost point.
- The direction of reading is given by the arrows on the end of every line.

- The run stops, if the rightmost point of the diagram has reached.
- If a parameter element appears on the way through the diagram, then this point can only be passed while producing the corresponding language element.

References

[1] Martin Reiser and Niklaus Wirth. *Programming in Oberon: Steps beyond Pascal and Modula.* Addison-Wesley, Wokingham, 1992.

[2] Peter Thiemann. *Grundlagen der funktionalen Programmierung.* Teubner, Stuttgart, 1994.

[3] J. Wolff von Gudenberg. *Objektorientiert programmieren von Anfang an.* BI-Wiss.-Verl., Mannheim, 1993.

[4] J. Wolff von Gudenberg. *Algorithmen, Datenstrukturen, Funktionale Programmierung. Eine praktische Einführung mit CAML LIGHT.* Addison-Wesley, Bonn, 1996.