# The Complete Syntax for PAL

This appendix provides a complete listing of PAL's syntax. The technique used here was developed in the form shown by James Morris. (This section was last modified on 02/17/68 at 15:24 by Evans.)

Although it is possible to provide a complete, unambiguous syntax for PAL using only the BNF notation used in the rest of this manual, doing so is impractical: It appears that in excess of one hundred definitions would be necessary. In order to achieve a more concise (and therefore more useful) set of productions, we proceed in two steps:

1. We first define a language which we call PAL-1, using the usual BNF notation. PAL-1 differs from PAL only in that more parentheses are required in PAL-1.

2. We next extend PAL-1 to PAL by adding a set of simple context-dependent productions whose effect is to allow the omission of parentheses in certain contexts.

A context free production is a BNF definition of the form that we have been using. A context dependent production is one in which there is not only a class name but also one or more terminal symbols on the left side. We now give the complete syntax of PAL-1, using the usual notation:

<program> ::= <E0> % | { def <D0> }$_{1}^{\infty}$ %

## Expressions

<E0> ::= <E2> where <D0> | <E1>
<E1> ::= let <D0> in <E1> | <E2>
<E2> ::= <E3> { ; <E3> }$_{0}^{\infty}$
<E3> ::= { <variable> : }$_{0}^{\infty}$ <E4>
<E4> ::= ll <B0> . <E3> | <E5>
<E5> ::= <E6> := <E5> | res <E6> | <E6>

```
<E6>   ::=  <E7> { , <E7> }∞₀
<E7>   ::=  goto <E8> | <E8>
<E8>   ::=  <E9> -> { <variable> : }∞₀ <E8>
                  ! { <variable> : }∞₀ <E8>
            | <E9>
<E9>   ::=  <E9> % <variable> <E10> | <E10>
<E10>  ::=  <E10> aug <E11> | <E11>
<E11>  ::=  <E12> { logor <E12> }∞₀
<E12>  ::=  <E13> { & <E13> }∞₀
<E13>  ::=  not <E14> | <E14>
<E14>  ::=  <E14> { ls | = | gr } <E15> | <E15>
<E15>  ::=  <E15> { + | - } <E16> | + <E16> | - <E16> | <E16>
<E16>  ::=  <E16> { * | / } <E17> | <E17>
<E17>  ::=  <E17> ** <E18> | <E18>
<E18>  ::=  $ <E19> | val <E19> | <E19>
<E19>  ::=  <E19> <E20> | <E20>
<E20>  ::=  ( <E0> ) | <variable> | <quotation> | <numeric> | jj
       | true | false | nil | dummy
```

Definitions

```
<D0>   ::=  <D1> within <D0> | <D1>
<D1>   ::=  <D2> { and <D2> }∞₀
<D2>   ::=  rec <D2> | <B2> = <E1> | <variable> <B0> = <E1>
       | ( <D0> )
```

Bound Variable Lists

```
<B0>   ::=  { <B1> }∞₁
<B1>   ::=  <variable> | ( ) | ( <B2> )
<B2>   ::=  <variable> { , <variable> }∞₀
```

The syntax just presented defines a <program>. The three classes left undefined by this syntax, <variable>, <quotation> and <numeric>, are assumed to be as defined on page 1.4/F-1. The mark "ϰ" shown is assumed to be inserted automatically at the end of each source file.

As remarked earlier, any member of this class is also legal PAL. As an example of the difference between PAL and PAL-1,

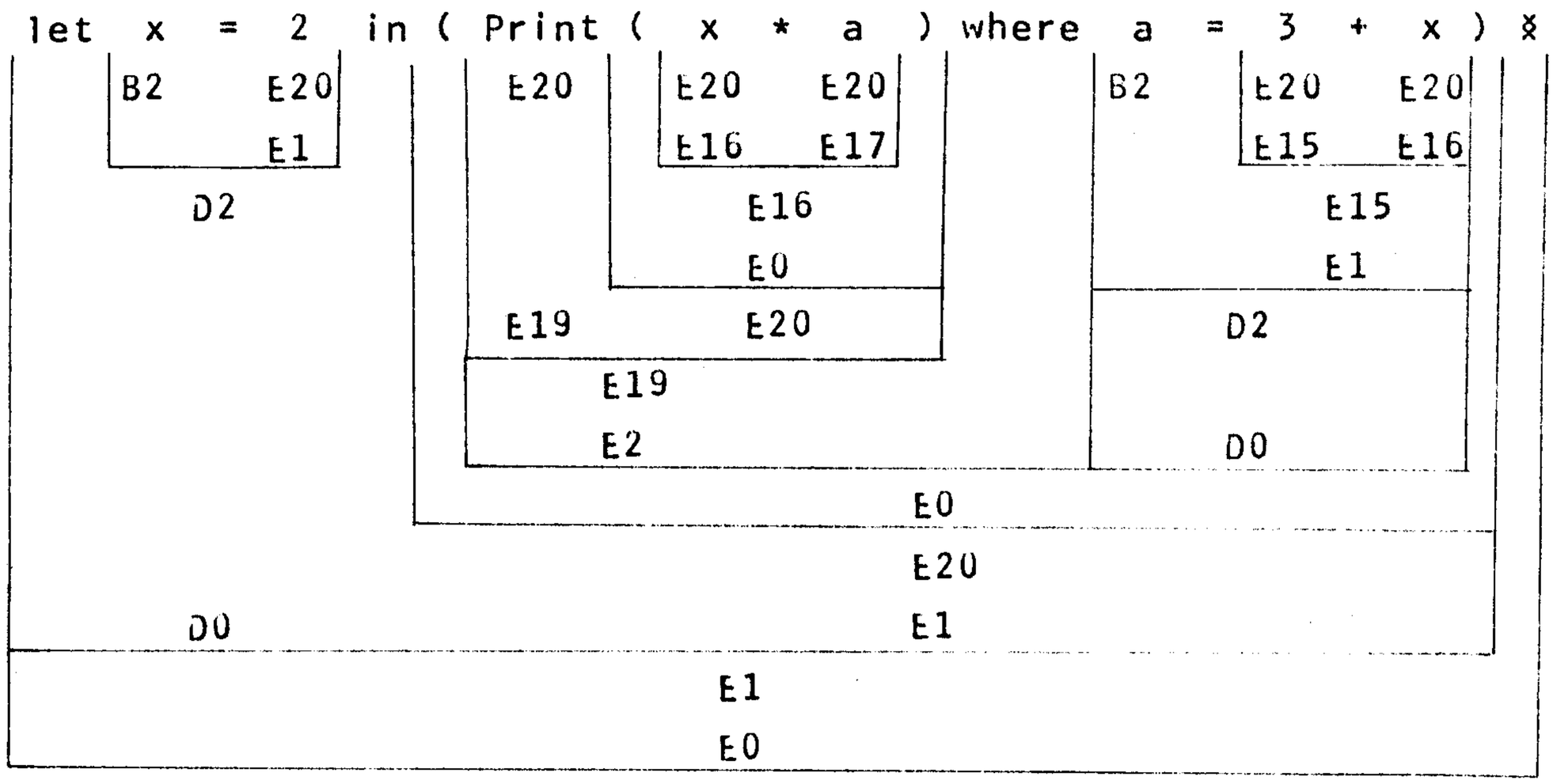
consider the following:

```
let x = 2
in
Print (x * a)
where a = 3 + x
```

This is legal PAL but is not legal in PAL-1. Legal PAL-1 would be

```
let x = 2
in
(    Print (x * a)
     where a = 3 + x
)
```

We now show in detail that this second sample is actually a legal PAL-1 program, providing only that the terminator "ϰ" is appended to it. Note the following diagram:




A few words about this notation are in order. Starting from the left, we observe that x is a <variable> and is therefore in <B2>. (It is of course also true that x is <E20>, but that fact is not useful.) 2 is a <numeric> and thus in <E20>. Anything that is an <E20> is also an <E1>, with the intervening steps omitted from the diagram for the sake of brevity. Then "<B2> = <E1>" is a

<D2>. It is left as an exercise for the reader to verify the correctness of the rest of the derivation.

It is instructive to note the effect of removing the parentheses that surround the where expression. The parse would proceed as far as

let <D0> in <E2> where <D0>

and then to

let <D0> in <E0>

and be unable to proceed further.

We now define six sets: A0, A1, A2, A3, A4, A5 and A6.

A0 ~ in | ) | ⍝ | !
A1 ~ A0 | and | within
A2 ~ A1 | where
A3 ~ A2 | := | , | aug | %
A4 ~ A3 | -> | & | logor
A5 ~ A4 | + | - | = | ls | gr

Finally we get to the critical part of this section: six context dependent productions. We first give them, and then we explain their meaning:

<E1> A0 :::= <E2> where <D0> A0
<E17> A1 :::= let <D0> in <E1> A1
<E17> A2 :::= ll <B0> . <E3> A2
<E17> A3 :::= goto <E8> A3
<E17> A4 :::= not <E12> A4
<E17> A5 :::= + <E14> A5 | - <E14> A5

We first note that ":::=" has been used instead of "::=", to indicate that these definitions are different from what we have seen heretofor. We now proceed by example, showing that the PAL program shown above is legal by this syntax. We have shown that the expression can be parsed as

let <D0> in <E2> where <D0>

Now we make use of the first context dependent production. The set A0 includes the right terminator "ϰ". and the production indicates that the text

<E2> where <D0> A0

may be replaced by

<E1> A0

In the present instance, we then have

let <D0> in <E1> ϰ

and this clearly parses as a <program>.