WORKING NOTE AC79

Alan Creak
6 March 1991

PFL SYNTAX, YET AGAIN.

This time it's PFL syntax as modified after John Garvey's go at it in an assignment for the Robotics and real-time computing course (John Garvey : *Parser for PFL*, Assignment 2, 1989). There's still no guarantee that it's complete, consistent, compilable, or anything but rubbish, but I think it's an improvement on the previous Working Note (Alan Creak : *PFL Syntax*, Unpublished working note AC72, 1989).

```
% PFL INFORMAL FORMAL SYNTAX.
%
%  +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%  +                                                                     +
%  +          ABSOLUTELY NOT GUARANTEED IN ANY WAY WHATEVER !            +
%  +                                                                     +
%  +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%
% NOTATION :
%
%    < ... >                  A non-terminal symbol.
%    AWORDINCAPITALS          A terminal symbol.
%    Any symbol not defined here
%                             Itself.
%    ::=                      Produces.
%    =>                       There follows a note in English on the
%                             implications of finding the production.
%    |                        Or.
%    |> s LIST <|             The items of the LIST, without duplication, in
%                             any order, separated if need be by the mark s.
%    | A1 | A2 ...            An enumerated list of items A1, A2, ....
%    |*                       A list of zero or more of whatever follows.
%    |+                       A list of one or more of what follows.
%    |!                       A list of at most one of what follows.
%    [ ... ]                  Optional.
%
<pflprogramme>
     ::=  <identifier> [IS] [A] PROGRAMME [WITH] < programmeparts>
                                END [ [OF] <identifier> ] .
               =>   <identifier> names a programme;
                    both <identifier>s are the same.
<programmeparts>
     ::=  |> . | <image>
               | <components>
               | <startup>
               | <shutdown>
               | <emergency>
               | <operations>
          <|
%
```

```
<image>
     ::=  IMAGE : <imagedetails>
<imagedetails>
     ::=  |> ; |* <linedetails>
          <|
<linedetails>
     ::=  <identifier> [IS] <linedescription>
               => <identifier> names a line.
<linedescription>
     ::=  |> , | <linewidth>
               | <linecontinuity>
               | <lineconditionnames>
          <|
<linewidth>
     ::=  SINGLE
     |    BYTE
     |    CHANNEL <expression>
               =>   <expression> evaluates to give an integer.
<linecontinuity>
     ::=  INTERRUPT
     |    CONTINUOUS
<lineconditionnames>
     ::=  |> , | ON [IS] <identifier>
               =>   <identifier> names a line state.
               | OFF [IS] <identifier>
               =>   <identifier> names a line state.
          <|
%
<components>
     ::=  COMPONENTS : <componentspart>
<componentspart>
     ::=  |> , |* <identifier>
               =>   <identifier> names a machine or procedure.
          <|
%
<startup>
     ::=  STARTUP : <identifier>
               =>   <identifier> names a procedure.
%
<shutdown>
     ::=  SHUTDOWN : <identifier>
               =>   <identifier> names a procedure.
%
<emergency>
     ::=  EMERGENCY : <identifier>
               =>   <identifier> names a procedure.
%
<operations>
     ::=  OPERATIONS : <operationsbody>
<operationsbody>
     ::=  |> \ |*  <sentencesequence>
          <|
<sentencesequence>
     ::=  |> ; |* <sentence>
          <|
% <sentence>s within <sentencesequence>s must be executed serially;
% different <sentencesequence>s may be executed in parallel.
```

```
<sentence>
     ::=  <declaration>
     |    <instruction>
%
<declaration>
     ::=  <datadeclaration>
     |    <proceduredeclaration>
     |    <machinedeclaration>
<datadeclaration>
     ::=  <identifierlist> <propertieslist>
     |    <structuredeclaration>
<structuredeclaration>
     ::=  <identifier> [IS] [A] STRUCTURE [CONTAINING] <structureparts>
                                 END [ [OF] <identifier> ]
<structureparts>
     ::=  |> , |* <datadeclaration>
          <|
<identifierlist>
     ::=  |> , |* <identifier>
               => <identifier> names a variable of type T.
<propertieslist>
     ::=  |> , |! [IS] <typedetails>
               => Type T is defined by <typedetails>.
               |! <- <constant>
               => Type T is the type of the <constant>.
          <|
<typedetails>
     ::=  <datatype>
     |    FILE [OF] <datatype>
<datatype>
     ::=  <simpletype>
     |    ARRAY [OF] <expression> <simpletype>
               =>   <expression> evaluates to an integer.
<simpletype>
     ::=  CHAR
     |    INTEGER
     |    LOGICAL
     |    NUMBER
     |    STRING
%
<proceduredeclaration>
     ::=  <identifier> [IS] AN OPERATION [WITH] <procedureparts>
                                 END [ [OF] <identifier> ]
               =>   <identifier> names a procedure;
                    both <identifier>s are the same.
<procedureparts>
     ::=  |> . | <inputlist>
               =>   the procedure uses input parameters.
               | <outputlist>
               =>   the procedure uses output parameters.
               | <valuespecification>
               =>   the procedure is a function.
               | <startup>
               | <shutdown>
               | <emergency>
               | <operations>
          <|
<machinedeclaration>
     ::=  <identifer> [IS] [A] MACHINE [WITH] <machineparts>
                                   END  [ of <identifier> ]
               =>   <identifier> names a machine;
                    both <identifier>s are the same.
<machineparts>
     ::=  |> . |! <image>
               | <components>
               |! <startup>
               |! <shutdown>
               |! <emergency>
               | <operations>
          <|
<inputlist>
     ::=  USING |> , |+ <identifier> <|
               =>   the number of input parameters, and their order,
                    are known.
<outputlist>
    ::= GIVING |> , |+ <identifier> <|
               =>   the number of output parameters, and their order,
                    are known.
<valuespecification>
    ::=  RETURNING <simpletype>
               =>   the type of the function is known.
%
<instruction>
     ::=  <compoundinstruction>
     |    <conditionalinstruction>
     |    <interruptinstruction>
     |    <iterativeinstruction>
     |    <simpleinstruction>
<compoundinstruction>
     ::=  GROUP <operationsbody> [ ; ] END
<conditionalinstruction>
     ::=  IF <expression> THEN <instruction> [ ELSE <instruction> ]
                                   END IF
               =>   <expression> evaluates to give a logical value.
<interruptinstruction>
     ::=  <wheninstruction>
     |    <wheneverinstruction>
<wheninstruction>
     ::=  WHEN <indicator> DO <instruction>
<wheneverinstruction>
     ::=  WHENEVER <indicator> DO <instruction>
<indicator>
     ::=  <expression>
               =>   <expression> evaluates to give a logical value.
     |    <identifier>
               =>   <identifier> names a line with the INTERRUPT attribute.
<iterativeinstruction>
     ::=  REPEAT <iterationcontrol> : <instruction> END REPEAT
```

```
<iterationcontrol>
     ::=  |> : | WHILE <expression>
                =>   <expression> evaluates to give a logical value.
                | UNTIL <expression>
                =>   <expression> evaluates to give a logical value.
                | [ up to ] <expression> TIMES
                =>   <expression> evaluates to give an integer expression.
                | FOR EACH <identifier>
                =>   <identifier> names an array.
          <|
<simpleinstruction>
     ::=  <assignment>
     |    <procedurecall>
     |    <stopinstruction>
     |    <returninstruction>
     |    <hearinstruction>
     |    <sayinstruction>
%
<assignment>
     ::=  <variable> <- <expression>
                =>   the types of <identifier> and <expression> are
                     the same.
<procedurecall>
     ::=  CALL <identifier> [ <actualparts> ]
                =>   <identifier> names a procedure which is not a
                     function.
<actualparts>
     ::=  |> , | USING |>, |+ <expression> <|
                =>   information on the input parameters.
                | GIVING |> , |+ <variable> <|
                =>   information on the output parameters.
          <|
<stopinstruction>
     ::=  STOP
<returninstruction>
     ::=  RETURN <expression>
                =>   the current scope corresponds to a function;
                     the type of <expression> is the same as that of
                     the function.
<hearinstruction>
     ::=  HEAR <identifier>
                =>   <identifier> names a string.
<sayinstruction>
     ::=  SAY <expression>
                =>   <expression> evaluates to give a string.
%
```

```
<expression>
     ::=  <term>
     |    <term> <binaryarithmeticoperator> <term>
               =>   both <terms> are numeric values.
     |    <term> <binaryrelationaloperator> <term>
               =>   both <term>s are numeric values, or both are string;
                    the <expression> is logical.
     |    <term> <binarylogicaloperator> <term>
               =>   both <term>s are logical;
                    the <expression> is logical.
     |    <term> & <term>
               =>   both <term>s are strings;
                    the <expression> is a string.
<binaryarithmeticoperator>
     ::=  **
     |    *
     |    \
     |    +
     |    -
<binaryrelationaloperator>
     ::=  >
     |    <
     |    =
     |    >=
     |    <=
     |    <>
<binarylogicaloperator>
     ::=  AND
     |    OR
<term>
     ::=  <primary>
     |    - <primary>
               =>   the <primary> is numeric;
                    the <term> is numeric.
     |    NOT <primary>
               =>   the <primary> is logical;
                    the <term> is logical.
     |    # <primary>
               =>   the <primary> is a string;
                    the <term> is numeric.
     |    $ <primary>
               =>   the <primary> is numeric;
                    the <term> is a string.
<primary>
     ::=  <constant>
     |    <identifier>
     |    ( <expression> )
     |    <identifier> [ <actualparts> ]
               =>   <identifier> names a function;
                    the function type and <primary> type are the same.
%<constant>
     :=  a number
     |    a string between quotation marks
     |    TRUE
     |    FALSE
<identifier>
     ::=  any string of letters and digits beginning with a letter which
          is not a <constant> nor a reserved word.
```