1 Grammar Notation

Grammar symbols are defined as they are introduced in this document. Regular expression notation has been used to make the productions more perspicuous. Symbols in italics are nonterminals. Quoted symbols are terminals. ‘s?’ denotes the symbol s is optional. ‘s*’ denotes the symbol s may be occur zero or more times. ‘s+’ denotes the symbol s may occur one or more times. ‘(s/t)’ denotes a choice between the symbol sequences s and t. Parentheses are also used to group symbols with respect to ‘?’, ‘*’ and ‘+’.

2 Lexical Conventions

A program consists of one or more Petri net specifications stored in files. Programs are written using the Unicode character set. The precise version of Unicode used will be determined by the Java Virtual Machine used to run the Pencil compiler. For more information, refer to your Java Developer’s Kit documentation.

2.1 Line Terminators

The sequence of input characters is divided into lines by line terminators. Lines are terminated by the ASCII characters \texttt{CR} (“carriage return”), \texttt{LF} (“linefeed”), or \texttt{CR LF}. The \texttt{CR LF} combination is counted as one line terminator, not two.

2.2 Comments

Both C- and C++-style comments are supported. A C-style comment begins with the characters /\* and ends with the characters */. Any sequence of characters may appear inside of a C-style comment except the string ‘*/’. C-style comments do not nest. A C++-style comment begins with the characters // and ends with a line terminator.

/\* and */ have no special meaning inside comments beginning with //. // has no special meaning inside comments beginning with /*.

2.3 Whitespace

Whitespace is defined as the ASCII space, horizontal tab and form feed characters, as well as line terminators and comments.

2.4 Tokens

There are five classes of tokens: identifiers, keywords, constants, separators and embedded Java blocks. Whitespace is ignored except as a token separator. Whitespace is sometimes required to separate adjacent tokens that might otherwise be combined into one token (i.e., identifiers, keywords and constants).

Token formation is greedy: the input is searched for the longest string of characters that could constitute a token.
2.5 Identifiers

An identifier is a sequence of letters or digits, the first of which must be a letter. There is no limit on the length of an identifier.

Two identifiers are the same if they have the same Unicode character for every letter and digit. Identifiers that have the same external appearance may not be identical. For example, the Latin capital letter 'A' and the Greek capital letter ‘A’ ("Alpha") are different Unicode characters that have the same appearance when displayed.

\[ \text{Identifier} \rightarrow \text{letter (letter | digit)}* \]

2.6 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

- boolean
- byte
- char
- const
- double
- fire
- float
- immediate
- in
- int
- long
- net
- onCall
- out
- package
- public
- ret
- short
- transition

2.7 Constants

Constants are a sequence of ASCII digits representing an integer literal. Constants are unsigned, and the integer represented must be within the range of the Java primitive type \text{int}.

\[ \text{Constant} \rightarrow \text{digit}+ \]

2.8 Separators

The following ASCII characters are separators:

\{
\} 
[ ] : ; , .

2.9 Embedded Java Blocks

Embedded Java blocks begin with the character sequence <\% and end with the character sequence \%>. The embedded code will be inserted into the generated class file without alteration. The type of code allowed within a Java block depends on the location of the block within the specification, but no embedded Java code may contain the string ‘\%’. The compiler may inspect the Java code and issue errors or warnings, but it is not required to do so.

\[ \text{JavaBlock} \rightarrow \text{<\% JavaCode \%>} \]

3 Petri Net Specifications

A Pencil Petri net specification is a file that contains a net declaration followed by any number of place and transition declarations with an optional concluding Java block. A specification is compiled into a Java class that extends Thread. Calling the method start() on the generated class will begin execution of the Petri Net.
3.1 Names
A name is bound by a declaration and is available at any point in the specification that follows the declaration. A name must be unique within the specification.

3.2 Package Declaration
A specification may begin with an optional package declaration. A package declaration takes the same form as a Java package declaration and has the same semantics for the generated code. A package declaration must be the first declaration in the file.

\[
\text{PackageDeclaration} \rightarrow \text{package QualifiedIdentifier \; ;}
\]
\[
\text{QualifiedIdentifier} \rightarrow \text{Identifier \; (\; .\; \text{Identifier})}\*
\]

3.3 Net Declaration
A net declaration gives a name to the Petri net specified and defines its parameters. The net declaration must appear before any other non-package declaration. There may only be one net declaration in a specification. The declaration may begin with the keyword `public`. This will be used as the access modifier for the generated class and has the same meaning as it would if applied to a Java class. The declaration may take any number of parameters of any valid Java type.

\[
\text{NetDeclaration} \rightarrow \text{public}\? \; \text{net} \; \text{Identifier \; NetParameters}\? \; \text{;}
\]
\[
\text{NetParameters} \rightarrow \text{\; (\; \text{NetParameter} \; (\; .\; \text{NetParameter}) \; *)}
\]
\[
\text{NetParameter} \rightarrow \text{JavaType \; \text{Identifier}
\]
\[
\text{JavaType} \rightarrow \text{boolean}\ |
\text{byte}\ |
\text{char}\ |
\text{double}\ |
\text{float}\ |
\text{int}\ |
\text{long}\ |
\text{QualifiedIdentifier}\ |
\text{JavaType \; (\; \text{[]}\; )}\*
\]

3.4 Constant Declarations
A constant declaration associates a name with an integer constant.

\[
\text{ConstDeclaration} \rightarrow \text{const} \; \text{Identifier \; = \; Constant}
\]
3.5 Place Declarations

A place declaration gives a name to one or more places in the Petri net and defines their attributes. A place may have an integral initial marking, an integral upper bound, input transitions and output transitions. Input and output transition names must be valid transition identifiers at the point of the declaration.

\[
\text{PlaceDeclaration} \rightarrow \text{`place'} \text{ Place (',', Place)* ;'}
\]

\[
\text{Place} \rightarrow \text{Identifier PlaceParams?}
\]

\[
\text{PlaceParams} \rightarrow \text{`(' InitialMarking ')'}\]

\[
\text{InitialMarking} \rightarrow \text{`=' IntegerExpression}
\]

\[
\text{UpperBound} \rightarrow \text{IntegerExpression}
\]

\[
\text{IntegerExpression} \rightarrow \text{(Constant | IntegerVariable)}
\]

\[
\text{IntegerVariable} \rightarrow \text{Identifier}
\]

\[
\text{PlaceDefinition} \rightarrow \text{`{' PlaceAttribute * '}}
\]

\[
\text{PlaceAttribute} \rightarrow \text{PlaceInputs | PlaceOutputs}
\]

\[
\text{PlaceInputs} \rightarrow \text{`in' ;' TransitionList ;';}
\]

\[
\text{PlaceOutputs} \rightarrow \text{`out' ;' TransitionList ;';}
\]

\[
\text{TransitionList} \rightarrow \text{TransitionArc (',', TransitionArc)*}
\]

\[
\text{TransitionArc} \rightarrow \text{TransitionName ArcWeight?}
\]

\[
\text{TransitionName} \rightarrow \text{Identifier}
\]

\[
\text{ArcWeight} \rightarrow \text{`(' IntegerExpression ')'}
\]

3.6 Transition Declaration

A transition declaration gives a name to one or more transitions in the Petri net and defines their attributes. A transition may have input places, output places, a firing rule and an associated Java block. Input and output place names must be valid place identifiers at the point of the declaration.

The firing rule starts with the attribute name `fire`. The attribute may take the values `immediate` or `onCall`. The value `immediate` makes the transition fire immediately on becoming enabled. The value `onCall` causes a public method to be created in the generated Java class with the same name as the transition (the `transition method`). When called, this method will block until the transition is enabled, then fire the transition. The transition will fire only when the transition method is called.

An embedded Java block may appear as the last item in a transition definition. The block should contain one or more valid Java statements, which will be executed when the transition is fired.
4 Pencil Grammar

The following lists all of the grammar productions described in Sections 2 and 3. The start production is `PencilSpecification`.

```
ArcWeight → '(' IntegerExpression ')'  
Constant → digit+  
ConstDeclaration → 'const' Identifier Constant  
Expression → (Constant | Variable)  
FiringRule → 'fire' ':' ('immediate' | 'onCall') ';'  
InitialMarking → '=' IntegerExpression  
IntegerExpression → (Constant | IntegerVariable)  
IntegerVariable → Identifier  
JavaBlock → '<%' JavaCode '%'  
JavaType → boolean  
| byte  
| char  
| double  
| float  
| int  
| long  
| QualifiedIdentifier  
| JavaType ('[]')*  
NetDeclaration → 'public'? 'net' Identifier NetParameters? ';'  
NetParameter → JavaType Identifier  
NetParameters → (' NetParameter (', NetParameter) * ')'
```
PackageDeclaration → package QualifiedIdentifier ;

PencilSpecification → PackageDeclaration?

NetDeclaration
(PlaceDeclaration | TransitionDeclaration)*

Place → Identifier PlaceParams?

PlaceArc → PlaceName ArcWeight?

PlaceAttribute → PlaceInputs | PlaceOutputs

PlaceDeclaration → "place" Place (',' Place)* ;

PlaceDefinition → '{ [PlaceAttribute * ]}'

PlaceInputs → 'in' ':' TransitionList ;'

PlaceList → PlaceArc (',' PlaceArc)*

PlaceName → Identifier

PlaceOutputs → 'out' ':' TransitionList ;'

PlaceParams → '{ InitialMarking }'

' { InitialMarking , UpperBound }'

' { InitialMarking , UpperBound, InitialMarking }'

QualifiedIdentifier → Identifier (',' Identifier)*

ReturnValue → [' ret ' Expression ']

Transition → Identifier

TransitionArc → TransitionName ArcWeight?

TransitionAttribute → TransitionInputs | TransitionOutputs | FiringRule

TransitionDeclaration → "transition" Transition (',' Transition)* ;

| "transition" Transition TransitionDefinition

TransitionDefinition → ReturnValue? '{ [TransitionAttribute + FiringRule? JavaBlock? ]}'

TransitionInputs → 'in' ':' PlaceList ;'

TransitionList → TransitionArc (',' TransitionArc)*

TransitionName → Identifier

TransitionOutputs → 'out' ':' PlaceList ;'

UpperBound → IntegerExpression

Variable → Identifier