Narative for character and token level grammars

August 2025

A grammar consists of a set of terminals, a set of nonterminals, a start nonterminal and a set of grammar rules.

Typically there is also an underlying set of 'characters' and each terminal has an associated pattern which is a set of non-empty character strings. An element of a temrinal's pattern is called a lexeme of the terminal.

For ART specifications this character set is the UNICODE character set, or more accurately the code points associated with each UNICODE character.

Basic terminals - UNICODE code points

The UNICODE characters form the *base terminal set* of an ART specification. They are written in the form

```
‘\uNMPQ
```

where N,M.P,Q are hexadecimal digits.
Where a character has an ASCII glyph this can also be used in an ART specification

```
‘\u0062    ‘b
‘\u002B    ‘+
```

The following is an ART specification which defines the language of strings of the form b, b+b, b+b+b etc

```
S ::= ‘\u0062  |  ‘b ‘+ S
```

Whitespace terminals

The usual whitespace characters, space, tab, newline etc can be specified and used to build a nonterminal that generates strings of whitespace characters.

```
S ::= WT T
T ::= ‘b WT  |  ‘b WT ‘+ WT T
WS ::= ‘  | ‘\t | ‘\n  | ‘\r
WT ::= #  | WS WT
```

We can specifier identifiers as strings of certain letters and didits that begin with a letter.

```
S ::= WT T
T ::= ID WT  |  ID WT ‘+ WT T
A ::= ‘a | ‘b | ‘c | ‘1 | ‘2
ID ::= ‘a | ‘b | ‘c | ID A
WS ::= ‘  | ‘\t | ‘\n  | ‘\r
WT ::= #  | WS WT
```

Stropped terminals

ART specifications permit terminals to be specified as stropped strings of Unicode characters. The pattern of such a terminal is the set of strings which are the characters in the stropped string concatenated with any string of whitespace characters.

```
'b'                    `b WT
'\u0026 a'             `b `a WT
```

This allows an ART specification to avoid the need for explicit whitespace handling.
When reading an input string of code points an ART application will include all the whitespace characters immediately to the right in the lexeme that is matched to a stropped terminal.
Stropped terminals facitiate the use of ART for theory oriented examples where the grammar terminals do not have specified patterns. The input to the generated translator can be a space deliminted sequence of terminal names.

```
S ::= E '+' S | E
E := 'iden' | 'num'
input:  iden + iden + num
```

```
S ::= 't1' S 't1' | 't2' S 't2' | 't3' S 't3'
                  | 't4' S 't4' | 't5' S 't5' | #
input:  t1 t1 t2 t3 t1 t3 t4 t2 t2 t4 t3 t1 t3 t2 t1 t1
```

From stropped to pure character level

We may hope that there is a clear relationship between an ART specification with stropped nonterminals and a pure character level specification. But this is impacted by the behaviour of the ART front end processor which takes a string/stream of code points and returns a sequence of terminals. The front end processor uses a form of longest match for stropped terminals and by default is sensitive to whitespace.
We can simply replace a stropped terminal with is characters and a whitespace nonterminal.

```
S ::= E `+ WT S | E
E := `i`d`e`n WT | `n`u`m WT
WT ::= (`\ | `\t | `\r | `n )*
```

We can simply replace a stropped terminal with is characters and a whitespace nonterminal.

```
S ::= t1 S t1 | t2 S t2 | t3 S t3
                | t4 S t4 | t5 S t5 | #
t1 ::= `t`1 WT
t2 ::= `t`2 WT
t3 ::= `t`3 WT
t4 ::= `t`4 WT
t5 ::= `t`5 WT
```

Mixed stropped and character terminals

It is permitted to have both base character and stropped terminals in an ART specification but this must be handled with care.

```
S ::= ‘a‘n‘d‘b | ’and’
```

```
S ::= A | B | C
A ::= ‘a‘n‘d‘b | ’and’
B ::= ‘x‘y‘z  ‘w | ‘x‘y‘z
C ::= ‘p‘q‘r‘s | pqr
pqr ::= ‘p‘q‘r
```

```
!try "andb"
!try "and b"
!try "xyzw"
!try "xyz w"
!try "pqrs"
!try "pqr s"
```

```
S ::= A | B | C
A ::= ‘a‘n‘d‘b | ’and’
B ::= ‘x‘y‘z  ‘w | ‘x‘y‘z
C ::= ‘p‘q‘r‘s | D
D ::= pqr | D pqr
pqr ::= ‘p‘q‘r WT
WT ::= (‘\ | ‘\t | ‘\r | ‘n )*
```

SML

For SML we have a grammar whose terminals are stropped ’keywords’ form from alpha/digits/punctuation sybmols, and const, vid, funid, strid, tyvar, tycon, lab, d

`smlBasicToken.art`

It requires input strings to have been tokenised, with whitespace and comments removed.
We also have a pure character level grammar whose terminals are unicode code points.

`smlBasicChar.art`

Both grammars are still being developed and need checking.