Syntax Definition and Explanation of SEN

Zirak

March 6, 2013

Contents

1 The Complete Syntax

⟨*program*⟩ → ⟨*value*⟩
| ϵ

⟨*value*⟩ → ⟨*atom*⟩
| ⟨*symbol*⟩
| ⟨*string*⟩
| ⟨*list*⟩
| ⟨*plist*⟩

⟨*atom*⟩ → '`nil`'
| '`t`'
| ⟨*anything*⟩

⟨*symbol*⟩ → '`:`' ⟨*atom*⟩

⟨*string*⟩ → '`"`' ⟨*chars*⟩ '`"`'

⟨*chars*⟩ → ⟨*unicode-char*⟩ ⟨*chars*⟩
| ⟨*escaped-char*⟩ ⟨*chars*⟩
| ϵ

⟨*escaped-char*⟩ → '`\b`'
| '`\f`'
| '`\n`'
| '`\r`'
| '`\t`'
| '`\u`' ⟨*hex-digit*⟩ ⟨*hex-digit*⟩ ⟨*hex-digit*⟩ ⟨*hex-digit*⟩

⟨*list*⟩ → '`(`' ⟨*list-values*⟩ '`)`'

⟨*list-values*⟩ → ⟨*value*⟩ ⟨*list-values*⟩
| ϵ

⟨*plist*⟩ → '`(`' ⟨*pairs*⟩ '`)`'

⟨*pairs*⟩ → ⟨*symbol*⟩ ⟨*value*⟩ ⟨*pairs*⟩
| ϵ

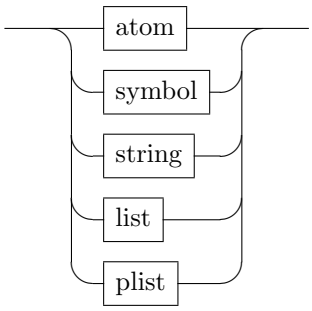
⟨*comment*⟩ → '`;`' ⟨*comment-chars*⟩

⟨*comment-chars*⟩ → ⟨*not-line-terminator*⟩ ⟨*comment-chars*⟩
| ϵ

2 Individual Components

2.1 Values

value

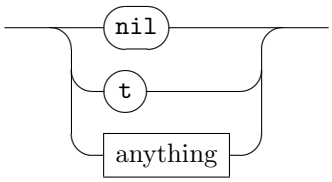


⟨*value*⟩ → ⟨*atom*⟩
| ⟨*symbol*⟩
| ⟨*string*⟩
| ⟨*list*⟩
| ⟨*plist*⟩

A *value* is any of the possible SEN structures.

2.2 Atoms

atom



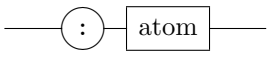
⟨*atom*⟩ → '`nil`'
| '`t`'
| ⟨*anything*⟩

An *atom* is any of the special constructs *nil* or *t*, or any combination of characters, excluding the space character and parentheses (). In addition, an *atom* may not begin with the colon, :.

The *nil* value is akin to *null* or *none* is many other programming languages. It is also used as the de-facto $false$. *t* is akin to *true*.

2.3 Symbols

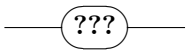
symbol



⟨*symbol*⟩ → '`:`' ⟨*atom*⟩

A *symbol* is a literal value. While an *atom* may be subject to interpretations (for example, *t* may turn to *true* in a target language), a *symbol* will always appear as-is.

2.4 Strings



⟨*string*⟩ → '"' ⟨*char*⟩ '"'

⟨*char*⟩ → ⟨*unicode-char*⟩ ⟨*char*⟩
| ⟨*escaped-char*⟩ ⟨*char*⟩
| ϵ

⟨*escaped-char*⟩ → '\b'
| '\f'
| '\n'
| '\r'
| '\t'
| '\u' ⟨*hex-digit*⟩ ⟨*hex-digit*⟩ ⟨*hex-digit*⟩ ⟨*hex-digit*⟩

A *string* is what you may be familiar with from the C family. *Strings* are delimited by the double-quote character. Any Unicode character may be written inside the double-quotes, with the exception of the double-quote and the backslash, which must be escaped. To escape a character, one writes the backslash character, followed by the desired character. For example: \", which results in the literal double-quote character; $\backslash p$, which results in the character p.

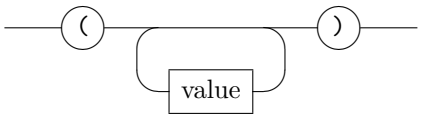
Several common characters which may be difficult to write directly are represented by an escape-sequence:

Sequence	Meaning	Unicode
$\backslash b$	backspace	U+0008
$\backslash f$	form-feed	U+000C
$\backslash n$	line-feed	U+000A
$\backslash r$	carriage-return	U+000D
$\backslash t$	tab	U+0009

Another escape-sequence, $\backslash u$, may be used to represent unicode values inside the range U+0000 → U+FFFF, as such $\backslash uXXXX$ where each X is a hexadecimal digit (case insensitive). For instance, the tab character U+0009 may be written as $\backslash u0009$.

2.5 Lists

list



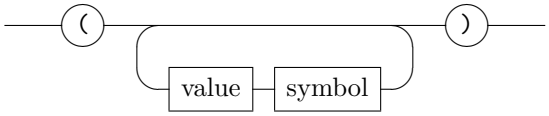
⟨*list*⟩ → '(' ⟨*list-values*⟩ ')'

⟨*list-values*⟩ → ⟨*value*⟩ ⟨*list-values*⟩
| ϵ

A *list* is one or more *value*s, separated by spaces. They do not have to be homogeneous; that is, you can mix up the value types. You may arbitrarily nest lists to easily create complex structures.

2.6 Property-Lists

plist



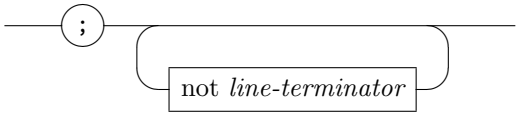
⟨*plist*⟩ → '(' ⟨*pairs*⟩ ')'

⟨*pairs*⟩ → ⟨*symbol*⟩ ⟨*value*⟩ ⟨*pairs*⟩
| ϵ

$p-lists$, or $property-lists$, can be considered a poor man's hash-table. They are made of one or more $key => value$ pairs, where the key must be a *symbol*, and the value may be any *value* allowed in the language. The *key* and *value* are separated by a space, and so are each pair. Like regular *lists*, $p-lists$ are heterogenous.

2.7 Comments

comment



⟨*comment*⟩ → ';' ⟨*comment-chars*⟩

⟨*comment-chars*⟩ → ⟨*not-line-terminator*⟩ ⟨*comment-chars*⟩
| ϵ

A *comment* may be inserted at any point in the program, except inside a string. Its contents are ignored by the parser. The comment spans from the beginning of the semi-colon ; until a line-terminator is met (EOL or EOF).