MPC API Documentation

Max Rottenkolber

Friday, 29 January 2016

Table of Contents

1 mpc

Monadic parser combinators. This package contains many different parser combinators ranging from axioms to high level tools. It also contains the RUN entry function and various macros that help integrate parser combinators into the common lisp ecosystem. Functions and starting with the =-prefix construct *parsers*. Their documentation is written from the perspective of the resulting parsers. Refer to the *MPC Manual* (`manual.html`) for a general introduction.

1.1 =and (Function)

Syntax:

— Function: **=and** `&rest` *parsers*

Arguments and Values:

parsers—*parsers*.

Description:

`=and` applies *parsers* sequentially. If all *parsers* succeed, `=and` succeeds with the last *parser*'s result. Otherwise `=and` fails.

1.2 =at-least (Function)

Syntax:

— Function: **=at-least** *n parser* `&key` *limit*

Arguments and Values:

n, limit—bounding index designators. The default for *limit* is `nil`.

parser—a *parser*.

Description:

`=at-least` applies *parser* repeatedly until it fails and succeeds with a list of the results unless *parser* succeeded less than *n* times or, if *limit* is not `nil`, more than *limit* times.

1.3 =bind (Function)

Syntax:

— Function: **=bind** *parser make-parser*

Arguments and Values:

parser—a *parser*.

make-parser—a *function designator* for a one-argument *function* which returns a *parser*.

Description:

`=bind` applies *parser* to the input. For each resulting `(value . input)` pair `=bind` calls *make-parser* with each *value* and applies the resulting parser to each *input*. `=bind` succeeds with the concatenated results or fails if *parser* fails.

1.4 =end-of-input (Function)

Syntax:

— Function: **=end-of-input** <*no arguments*>

Description:

`=end-of-input` succeeds only if the input is empty.

1.5 =eql (Function)

Syntax:

— Function: **=eql** *x*

Arguments and Values:

x—an *object*.

Description:

`=eql` consumes the next item and succeeds with that item as its result if the item is `eql` to *object*.

1.6 =exactly (Function)

Syntax:

— Function: **=exactly** *n parser*

Arguments and Values:

n—an non-negative *integer*.

parser—a *parser*.

Description:

`=exactly` applies *parser* repeatedly until it fails and succeeds with a list of the results unless *parser* succeeded not exactly *n* times.

1.7 =fail (Macro)

Syntax:

— Macro: **=fail** `&body` *handling*

Arguments and Values:

handling—*forms* to run when the parser is invoked.

Description:

`=fail` always fails. If supplied, the *handling forms* will be run when `=fail` is applied. The *handling forms* may call `get-input-position`.

1.8 =funcall (Function)

Syntax:

— Function: **=funcall** *parser function*

Arguments and Values:

parser—a *parser*.

function—a *function designator*.

Description:

`=funcall` applies *parser*. If successful, `=funcall` applies *function* on its result and succeeds with the return value.

1.9 =handler-case (Macro)

Syntax:

— Macro: **=handler-case** *parser* `&rest` *handlers*

Arguments and Values:

parser—a *parser*.

handlers—handler clauses for `handler-case`.

Description:

`=handler-case` establishes *handlers* as if by `handler-case` before applying *parser* to the input. *Handlers* must return *parsers*. If *parser* signals an *error* matched by a *handler*, the *parser* returned by the *handler* will be applied to the input.

1.10 =if (Function)

Syntax:

— Function: **=if** *test-parser then-parser* `&optional` *else-parser*

Arguments and Values:

test-parser—a *parser*.

then-parser—a *parser*.

else-parser—a *parser*. The default is `(=fail)`.

Description:

`=if` applies *then-parser* if *test-parser* would succeed and *else-parser* otherwise. Note that *test-parser* is not actually applied to the input.

1.11 =item (Function)

Syntax:

— Function: **=item** <*no arguments*>

Description:

`=item` consumes the next item and succeeds with that item as its result unless the input is empty.

1.12 =let* (Macro)

Syntax:

— Macro: **=let*** *bindings* `&body` *forms*

bindings::= `({(`*symbol parser*`)}*)`

Arguments and Values:

forms—*forms* of which the last *form* must return a *parser*.

symbol—a *symbol*.

parser—a *form* whose result is a *parser*.

Description:

`=let*` applies the *parsers* in *bindings* as if by `=and` and evaluates *forms* in an implicit `let` environment where the results are bound to the *symbols* in *bindings*. Finally, `=let*` applies the parser returned by the last *form*.

1.13 =list (Function)

Syntax:

— Function: **=list** `&rest` *parsers*

Arguments and Values:

parsers—*parsers*.

Description:

`=list` applies *parsers* sequentially. If all *parsers* succeed, `=list` succeeds with a list of their results. Otherwise `=list` fails.

1.14 =maybe (Function)

Syntax:

— Function: **=maybe** *parser*

Arguments and Values:

parser—a *parser*.

Description:

`=maybe` applies *parser*. If *parser* succeeds `=maybe` will succeed with its result, otherwise it will succeed with `nil`.

1.15 =none-of (Function)

Syntax:

— Function: **=none-of** *list*

Arguments and Values:

list—a *list* of *objects*.

Description:

`=none-of` consumes the next item and succeeds with that item as its result unless the item is `eql` to one of the *objects* in *list*.

1.16 =not (Function)

Syntax:

— Function: **=not** *parser*

Arguments and Values:

parser—a *parser*.

Description:

`=not` consumes the next item and succeeds with that item as its result if *parser* would fail. E.g. it negates *parser*. Note that *parser* is not actually applied to the input.

1.17 =one-of (Function)

Syntax:

— Function: **=one-of** *list*

Arguments and Values:

list—a *list* of *objects*.

Description:

`=one-of` consumes the next item and succeeds with that item as its result if the item is `eql` to any *object* in *list*.

1.18 =one-or-more (Function)

Syntax:

— Function: **=one-or-more** *parser*

Arguments and Values:

parser—a *parser*.

Description:

`=one-or-more` applies *parser* repeatedly until it fails and succeeds with a list of the results if *parser* succeeded at least one time.

1.19 =one-to (Function)

Syntax:

— Function: **=one-to** *n parser*

Arguments and Values:

n—a positive *integer*.

parser—a *parser*.

Description:

`=one-to` applies *parser* repeatedly until it fails and succeeds with a list of the results unless *parser* succeeded less than once or more than *n* times.

1.20 =or (Function)

Syntax:

— Function: **=or** `&rest` *parsers*

Arguments and Values:

parsers—*parsers*.

Description:

`=or` applies *parsers* until one *parser* succeeds, in which case it succeeds with the result of that *parser*. If no *parser* succeeds `=or` fails.

1.21 =plus (Function)

Syntax:

— Function: **=plus** `&rest` *parsers*

Arguments and Values:

parser—a *parser*.

Description:

`=plus` is the non-deterministic choice combinator. It applies *parsers* to input and succeeds with the result of every successful *parser*. `=plus` fails if every *parser* fails.

1.22 =prog1 (Function)

Syntax:

— Function: **=prog1** *parser* `&rest` *parsers*

Arguments and Values:

parser—a *parser*.

parsers—*parsers*.

Description:

`=prog1` applies *parser* and *parsers* sequentially. If they all succeed, `=prog1` succeeds with *parser*'s result. Otherwise `=prog1` fails.

1.23 =prog2 (Function)

Syntax:

— Function: **=prog2** *parser1 parser2* `&rest` *parsers*

Arguments and Values:

parser1—a *parser*.

parser2—a *parser*.

parsers—*parsers*.

Description:

`=prog2` applies *parser1*, *parser2* and *parsers* sequentially. If they all succeed, `=prog2` succeeds with *parser2*'s result. Otherwise `=prog2` fails.

1.24 =range (Function)

Syntax:

— Function: **=range** *from to* `&key` *parser predicate*

Arguments and Values:

from—an *object*.

to—an *object*.

parser—a *parser*. The default is `(=item)`.

predicate—a *function designator* for a three-argument predicate *function*. The default is `char<=`.

Description:

`=range` applies *parser* and, if it succeeds, applies *predicate* to *from*, its results and *to*. `=range` succeeds with the result of *parser* if *predicate* is *true* and fails otherwise.

1.25 =restart-case (Macro)

Syntax:

— Macro: **=restart-case** *parser* `&rest` *restarts*

Arguments and Values:

parser—a *parser*.

restarts—restart clauses for `restart-case`.

Description:

`=restart-case` establishes *restarts* as if by `restart-case` before applying *parser* to the input. *Restarts* must return *parsers*. If *parser* signals an *error* matched by a *restart*, the *parser* returned by the *restart* will be applied to the input.

1.26 =result (Function)

Syntax:

— Function: **=result** *value*

Arguments and Values:

value—an *object*.

Description:

`=result` always succeeds with *value* as its result.

1.27 =satisfies (Function)

Syntax:

— Function: **=satisfies** *predicate*

Arguments and Values:

predicate—a *function designator* for a one-argument predicate *function*.

Description:

`=satisfies` consumes the next item and succeeds with that item as its result if the result satisfies *predicate*.

1.28 =unless (Function)

Syntax:

— Function: **=unless** *test-parser* `&rest` *parsers*

Arguments and Values:

test-parser—a *parser*.

parsers—*parsers*.

Description:

`=unless` applies *parsers* as if by `=and` if *test-parser* would fail. Note that *test-parser* is not actually applied to the input.

1.29 =when (Function)

Syntax:

— Function: **=when** *test-parser* `&rest` *parsers*

Arguments and Values:

test-parser—a *parser*.

parsers—*parsers*.

Description:

`=when` applies *parsers* as if by `=and` if *test-parser* would succeed. Note that *test-parser* is not actually applied to the input.

1.30 =zero-or-more (Function)

Syntax:

— Function: **=zero-or-more** *parser*

Arguments and Values:

parser—a *parser*.

Description:

`=zero-or-more` applies *parser* repeatedly until it fails and succeeds with a list of the results.

1.31 =zero-to (Function)

Syntax:

— Function: **=zero-to** *n parser*

Arguments and Values:

n—an non-negative *integer*.

parser—a *parser*.

Description:

`=zero-to` applies *parser* repeatedly until it fails and succeeds with a list of the results unless *parser* succeeded more than *n* times.

1.32 get-input-position (Function)

Syntax:

— Function: **get-input-position** <*no arguments*>

→ *position*

→ *position, line, column*

Arguments and Values:

position, *column*—non-negative *integers*.

line—a positive *integer*.

Description:

`get-input-position` returns the number of items read from the input. Additionally, *line* and *column* positions are returned if the input's *element type* is `character`. Lines are counted starting at 1 while columns are counted starting from 0.

`get-input-position` may only be called from within the body of `=fail`, the handlers of `=handler-case` or the restarts of `=restart-case`.

Exceptional Situations:

`get-input-position` signals an *error* of *type* `simple-error` unless called within `=fail`, `=handler-case` or `=restart-case`.

1.33 run (Function)

Syntax:

— Function: **run** *parser input-source* `&key` *result*

Arguments and Values:

parser—a *parser*.

input-source—an *array*, an *input stream* or a *list*.

result—a *function designator* to a one-argument *function*. The default is `caar`.

Description:

`run` applies *parser* to *input-source* and returns the result of calling the *result* function on the resulting list of `(value . input)` pairs.

2 mpc.characters

This package includes parsers specialised for character input. Covered are case sensitivity, strings, whitespace and lines.

2.1 *whitespace* (Variable)

Initial Value:

```
(#\Tab #\Newline #\PageUp #\Page #\Return #\ )
```

Value Type:

a *list* of *characters*.

Description:

The *value* of `*whitespace*` is a *list* of *characters* considered to be *whitespace characters*.

2.2 =character (Function)

Syntax:

— Function: **=character** *character* `&optional` *case-sensitive-p*

Arguments and Values:

character—a *character*.

case-sensitive-p—a *generalized boolean*. The default is *true*.

Description:

`=character` consumes the next item and succeeds with that item as its result if the item is equal to *character*. `=character` is case sensitive unless *case-sensitive-p* is *false*.

2.3 =line (Function)

Syntax:

— Function: **=line** `&optional` *keep-newline-p*

Arguments and Values:

keep-newline-p—a *generalized boolean*. The default is *false*.

Description:

`=line` consumes a sequence of zero or more *characters* terminated by a `#\Newline` *character* and succeeds with the *characters* coerced to a *string*. The terminating `#\Newline` *character* is not included in the result unless *keep-newline-p* is *true*.

2.4 =newline (Function)

Syntax:

— Function: **=newline** <*no arguments*>

Description:

`=newline` consumes the next item and succeeds with that item as its result if the item is the `#\Newline` *character*.

2.5 =skip-whitespace (Function)

Syntax:

— Function: **=skip-whitespace** *parser*

Arguments and Values:

parser—a *parser*.

Description:

`=skip-whitespace` consumes a sequence of zero or more items which are members of `*whitespace*` and then applies *parser* and, if successful, succeeds with its result.

2.6 =string (Function)

Syntax:

— Function: **=string** *string* `&optional` *case-sensitive-p*

Arguments and Values:

string—a *string*.

case-sensitive-p—a *generalized boolean*. The default is *true*.

Description:

`=string` consumes a non-empty sequence of *characters* and succeeds with the *character sequence* coerced to a *string* if the result is equal to *sting*. `=string` is case sensitive unless *case-sensitive-p* is *false*.

2.7 =string-of (Function)

Syntax:

— Function: **=string-of** *parser*

Arguments and Values:

parser—a *parser*.

Description:

`=string-of` repeatedly applies *parser* to the input and succeeds with the resulting *character sequence* coerced to a *string*. `=string-of` fails unless *parser* succeeds at least once.

2.8 =whitespace (Function)

Syntax:

— Function: **=whitespace** <*no arguments*>

Description:

`=whitespace` consumes the next item and succeeds with that item as its result if the item is a member of `*whitespace*`.

3 mpc.numerals

This package includes parsers for string numerals. Covered are single digits, natural numbers and signed integers with arbitrary radixes.

3.1 =digit (Function)

Syntax:

— Function: **=digit** `&optional` *radix*

Arguments and Values:

radix—a *number* of *type* `(integer 2 36)`. The default is `10`.

Description:

`=digit` consumes the next item and succeeds with that item as its result if the next item is a digit *character* in the specified *radix*.

3.2 =integer-number (Function)

Syntax:

— Function: **=integer-number** `&optional` *radix*

Arguments and Values:

radix—a *number* of *type* `(integer 2 36)`. The default is `10`.

Description:

`=integer-number` consumes a signed non-empty sequence of digit *characters* in the specified *radix* and succeeds with the *integer* represented by that sequence. The leading sign is optional and can be `#\+` and `#\-` for positive and negative values respectively. The default is a positive value.

3.3 =natural-number (Function)

Syntax:

— Function: **=natural-number** `&optional` *radix*

Arguments and Values:

radix—a *number* of *type* `(integer 2 36)`. The default is `10`.

Description:

`=natural-number` consumes a non-empty sequence of digit *characters* in the specified *radix* and succeeds with the natural *number* represented by that sequence.