CSE 341: Programming Languages

Autumn 2006

Oct 13 — `define-struct & misc`

A Few Miscellaneous Scheme Topics

In DrScheme, you can use square brackets as well as parenthesis. (You need to match left parentheses with right parentheses, left square brackets with right square brackets.) Suggestion: use not at all, or sparingly for readability.

`'` is a macro — `'x` and `'(a b c)` are equivalent to `(quote x)` and `(quote (a b c))`.

Scheme functions can take a variable number of arguments.

```
(define (squid a b . c)
   (print a)
   (print b)
   (print c))
```

`squid` requires at least 2 arguments. Any remaining arguments (perhaps 0) are put into a list, which is bound to `c`.

define-struct

MzScheme extends Scheme with `define-struct`, e.g.:

```
(define-struct square (x y))
(define-struct piece (squares))
```

Semantics:

- Binds constructors (`make-square`, `make-piece`) that take arguments and make values.
- Binds predicates (`square?`, `piece?`) that take one argument and return `#t` only for values built from the right constructor.
- Binds accessors (`square-x`, `square-y`, `piece-squares`) that take one argument, return the appropriate field, and call `error` for values not built from the right constructor.
- Binds mutators (`set-square-x!`, `set-square-y!`, `set-piece-squares!`).

`define-struct` is special

Claim: `define-struct` is not a function.

Claim: `define-struct` is not a macro.

It could be a macro except for one key bit of its semantics: Values built from the constructor cause every *other* predicate (including all built-in ones) to return `#f`.

Advantage: abstraction

Disadvantage: Can't write "generic" code that has a case for every possible variant in every Scheme program.