CSE 341: Programming Languages

Dan Grossman

Winter 2008

Lecture 18— DrScheme modules; abstraction with dynamic types

Modularity

Recall from our ML module lecture some good things about modules:

- Namespace management (help keep names short and separate)
- Make some bindings inaccessible (private functions, data)
- Enforce invariants by using abstract types
  - Data is reachable, but outside the module only limited things can be done with it
- In our example:
  - Rationals are always printed in reduced form.
  - Clients can't tell if rationals are *kept* in reduced form.

Scheme vs. DrScheme

"Pure" Scheme (R5RS) has no module system or `define-struct`

- We'll investigate how much of modules' advantages we can get via other means

DrScheme has a module system

- But in a dynamically typed language, there won't be signatures with abstract types
- We can get abstract types using `define-struct` instead
  - Because it makes a new type not equal to any other type
  - Quite different than ML approach but both work

Life without modules

- Can hide private things using `let`
  - Workable but awkward
  - Making the `define-struct` "private" is a huge help

The key to define-struct

It is essential to hiding parts of a `define-struct` that it is a *fresh, different type* than any other type.

- In our example, hid the accessors, mutators, and constructor.
- Sometimes exposing some accessors makes sense.

Otherwise, someone could use other features (e.g., `cons` or `set-car!`) to violate invariants.

It is still the case that any Scheme function can be called with any argument, but we can control invariants on rationals.

DrScheme modules

- `provide` for explicit list of what is available outside
  - Can be "part" of `define-struct`
  - Kind of like "part" of an ML `datatype` (kind of)
- `require` for using another module
  - With optional prefixing of names for namespace management