Some Details on Scheme

Stefan Wehr

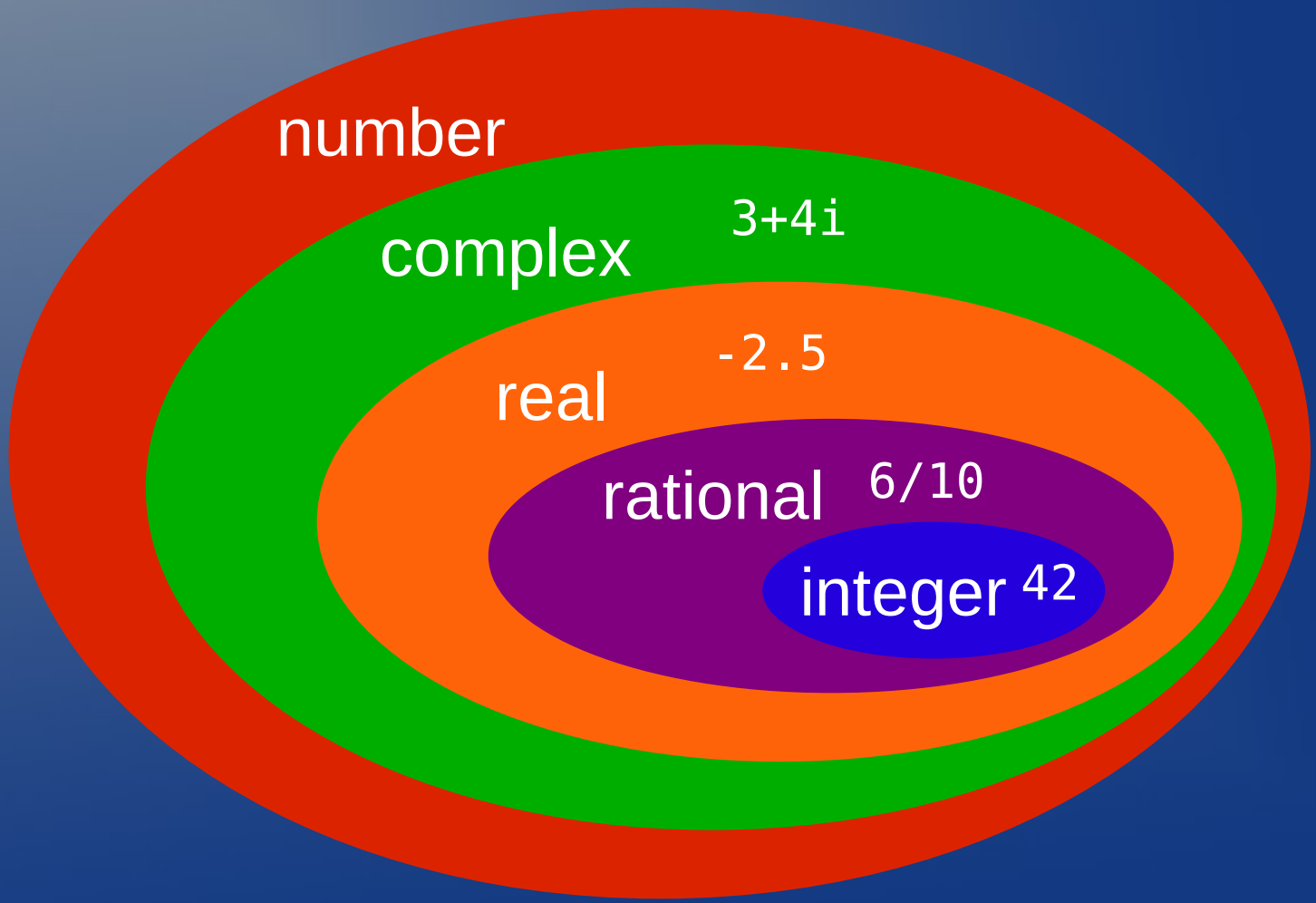
2009-05-07

Numbers

- The term numbers may denote different things:
  - Mathematical numbers
  - Scheme objects used to represent mathematical numbers
  - Machine representation used to implement numbers
  - Notation used to write numbers

Numerical Tower

Hierarchy of scheme objects representing numbers



Exactness

- Exact numbers: correspond exactly to a mathematical number
- Inexact numbers: correspond closely but not exactly to a mathematical number
- Coercion from inexact to exact (except for `+inf.0`, `-inf.0`, and `+nan.0`):

```
(inexact->exact 4.5) → 4 1/2
```

- Orthogonal to the numerical tower:
  - `(integer? 4.0) → #t`
  - `(exact? 4.0) → #f`

Equality

- *Disclaimer: Behavior described here is specific to PLT Scheme, some parts are unspecified in the standard R6RS*
- **`eq?`** The expression `(eq? v1 v2)` returns `#t` if `v1` and `v2` refer to the same object, `#f` otherwise.
- **`eqv?`** Two values are `eqv?` if and only if they are `eq?`, unless otherwise specified for a particular datatype.

  The number and character datatypes are the only ones for which `eqv?` differs from `eq?`.
- **`equal?`** Two values are `equal?` if and only if they are `eqv?`, unless otherwise specified for a particular datatype.

  Datatypes with further specification of `equal?` include strings, byte strings, numbers, pairs, ...

Equality - Examples

- `(eq? "string" (string-append "str" "ing")) → #f`
- `(eqv? "string" (string-append "str" "ing")) → #f`
- `(equal? "string" (string-append "str" "ing")) → #t`
- `(eq? (expt 2 100) (expt 2 100)) → #f`
- `(eqv? (expt 2 100) (expt 2 100)) → #t`
- `(equal? (expt 2 100) (expt 2 100)) → #t`
- `(eq? 1 1.0) → #f`
- `(eqv? 1 1.0) → #f`
- `(equal? 1 1.0) → #f`
- `(= 1 1.0) → #t`

Datum Values

- Datum values can be represented in textual form without loss of information
- Datum values include
  - Booleans
  - Numbers
  - Characters
  - Symbols
  - String
  - Lists of datum values
  - Vectors of datum values

Syntactic Data (1)

- Textual representation of datum values
- Quoting ' turns a syntactic datum into a literal expression (evaluation stops at the quote)
  - `'23 → 23`
  - `'#f → #f`
  - `'foo → foo`
  - `'(1 2 3) → (1 2 3)`
  - `'(+ 1 2) → (+ 1 2)`

Syntactic Data (2)

- Quoting is not needed for numbers, booleans, characters, and strings:
  - `(eq? 1 '1) → #t`
  - `(eq? #t '#t) → #t`
  - `(eq? #\a '#\a) → #t`
  - `(eq? “stefan“ '“stefan“) → #t`
- Omitting quotes for other syntactic data yields undesired results
  - `(eq? (1 2 3) '(1 2 3)) →` *ERROR*
  - `(eq? foo 'foo) →` *ERROR*

Quoting

- Quote operator ' simply abbreviates `quote`, i.e. '<*datum*> abbreviates (`quote` <*datum*>)
  - `(eq? 'foo (quote foo)) → #t`
  - `(eq? '(1 2 3) (quote (1 2 3))) → #t`
  - `(eq? ''1 (quote (quote 1))) → #f`
  - `(eqv? ''1 (quote (quote 1))) → #f`
  - `(equal? ''1 (quote (quote 1))) → #t`
  - `(equal? ''1 '(quote 1)) → #t`
  - `(equal? ''1 (quote '1)) → #t`
  - `(equal? ''1 (quote 1)) → #f`

Symbols

- A symbol is an object representing a string, the symbol's name.
- Unlike strings, two symbols whose names are spelled the same way are never distinguishable
  - `(eq? "string" (string-append "str" "ing")) → #f`
  - `(eq? (string->symbol "string")`
    `(string->symbol (string-append "str" "ing")))`
    `→ #t`
- Symbols and quoting are different things!
  - `(symbol? 'foo) → #t`
  - `(symbol? '1) → #f`
  - `(symbol? '(1 2 3)) → #f`

Pairs and Lists

- A pair is written `(e1 . e2)`
  - `(car '(1 . 2))` → `1`
  - `(cdr '(1 . 2))` → `2`
- A list is either the empty list `()` or a pair `(x . l)` where `l` is a list
  - `'()`
  - `'(1 . (2 . (3 . ())))`
- Convention: if a dot is followed by an open parenthesis, the dot, open parenthesis, and matching closing parenthesis can be omitted
  - `(equal? '(1 2 3) '(1 . (2 . (3 . ()))))`
    → `#t`

Improper Lists

- An improper list does not end with the empty list
  - `(cons 1 (cons 2 3))` → `(1 2 . 3)`
  - `'(1 2 . 3)` → `(1 2 . 3)`
- Avoid using improper lists