UMass Boston Computer Science

CS450 High Level Languages

Type Conversions, “Truthiness”

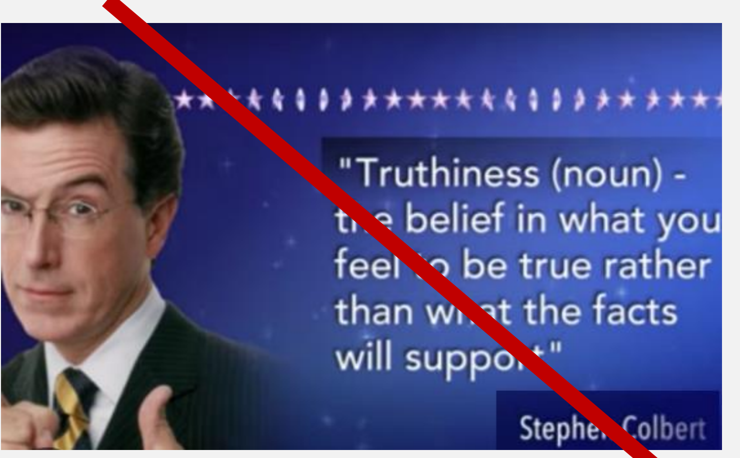
Tuesday, April 7, 2026

lecture17

Logistics

- HW 8 in
  - ~~due: Tues 4/7, 11am EST~~

- HW 9 out
  - <u>due:</u> Tues 4/14, 11am EST



See also: Implicit type conversion

In some programming languages, any expression can be evaluated in a context that expects a Boolean data type. Typically (though this varies by programming language) expressions like the number zero, the empty string, empty lists, and null are treated as false, and strings with content (like "abc"), other numbers, and objects evaluate to true. Sometimes these classes of expressions are called **falsy** and **truthy**. For example, in Lisp, nil, the empty list, is treated as false, and all other values are treated as true. In C, the number 0 or 0.0 is false, and all other values are treated as true.

In JavaScript, the empty string (`""`), `null`, `undefined`, `NaN`, `+0`, `-0` and `false`[3] are sometimes called *falsy* (of which the complement is *truthy*) to distinguish between strictly type-checked and coerced Booleans (see also: JavaScript syntax#Type conversion).[4] As opposed to Python, empty containers (arrays, maps, sets) are considered truthy. Languages such as PHP also use this approach.

"CS450 Lang"

Programmer writes:

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; An Atom is one of:
;; - Number
;; - 450Bool
```

```
;; A 450Bool is either:
;; - '450true
;; - '450false
```

“CS450 LANG”

Programmer writes:

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

“eval450”

```
;; A Result is one of:
;; - Number
;; - Boolean
```

parse

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-boo Boolean)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)

(struct num AST [val])
(struct boo AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

run

(following **JS** semantics?)

“CS450 LANG” Examples

Programmer writes:

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

Valid Program?

“eval450”

```
;; A Result is one of:
;; - Number
;; - Boolean
```

```
(check-equal? (eval450 100) 100)
(check-equal? (eval450 '450true) #t)
(check-equal? (eval450 '(+ 100 200)) 300)
(check-equal? (eval450 '(+ 450true 450)) 451)
(check-equal? (eval450 '(+ 1 2 3 4)) ??? )
```

```
match: no matching clause for '(+ 1 2 3 4)
```

Dynamic Errors (e.g, Exceptions)

When **a function argument:**

1. **Comes from arbitrary users**
2. **Has a sufficiently complex data definition**
   - So that contracts are not enough to **enforce the signature**
   - (Typically involves recursive data)

Then **dynamic errors** may be needed

Parsing: "CS450 Lang" Programs

```
;; parse: Program -> AST
;; Converts a CS450 Lang surface program to its AST
```

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
(define (Program? p)
  (or (atom? p)
      (cons? p)))
```

The best (shallow check) we can do

function argument:

1. Comes from arbitrary users
2. Has sufficiently complex data definition where <u>contracts are insufficient</u>

```
(define/contract (parse p)
  (-> Program? AST?)
  (match p
    [(? atom?) (parse-atom p)]
    [`(+ ,x ,y)  (mk-add (parse x) (parse y))]
    [`(× ,x ,y)  (mk-mul (parse x) (parse y))]))
```

Parsing: “CS450 LANG” Programs

```
;; parse: Program -> AST
;; Converts a CS450 Lang surface program to its AST
```

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
(define/contract (parse p)
  (-> Program? AST?)
  (match p
   [(? atom?) (parse-atom p)]
   [`(+ ,x ,y)  (mk-add (parse x) (parse y))]
   [`(× ,x ,y)  (mk-mul (parse x) (parse y))]
   [_ (error … )]))
```

function argument:

1. Comes from arbitrary users
2. Has sufficiently complex data definition where contracts are insufficient

Interlude: Racket exceptions

Exceptions are just special structs

Super struct (enables using **exception API**)

```
(struct exn:fail:syntax:cs450 exn:fail:syntax [])
```

```
(define/contract (parse p)
  (-> Program? AST?)
  (match p
   [(? atom?) (parse-atom p)]
   [`(+ ,x ,y)  (mk-add (parse x) (parse y))]
   [`(× ,x ,y)  (mk-mul (parse x) (parse y))]
   [_ (error … )]))
```

Interlude: Racket exceptions

Exceptions are just special structs

Super struct (enables using exception API)

```
(struct exn:fail:syntax:cs450 exn:fail:syntax [])
```

```
(define/contract (parse p)
  (-> Program? AST?)
  (match p
   [(? atom?) (parse-atom p)]
   [`(+ ,x ,y)  (mk-add (parse x) (parse y))]
   [`(× ,x ,y)  (mk-mul (parse x) (parse y))]
   [_ (raise-syntax-error
        'parse "not a valid CS450 Lang program" p
        #:exn exn:fail:syntax:cs450)]))
```

“CS450 Lang” Invalid Syntax Example

Programmer writes:

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

“eval450”

```
;; A Result is one of:
;; - Number
;; - Boolean
```

```
(check-equal? (eval450 '(+ 1 2 3 4)) ??? )
```

~~match: no matching clause for '(+ 1 2 3 4)~~

```
parse: not a valid CS450 Lang program in: (+ 1 2 3 4)
```

Can write tests with exceptions!

```
(check-exn exn:fail:syntax:cs450?
    (λ () (eval450 '(+ 1 2 3 4))))
```

Interlude: Racket exceptions (low-level) alternative

```
(struct exn:fail:syntax:cs450 exn:fail:syntax [])
```

```
(define/contract (parse p)
  (-> Program? AST?)
  (match p
    [(? atom?) (parse-atom p)]
    [`(+ ,x ,y)  (mk-add (parse x) (parse y))]
    [`(× ,x ,y)  (mk-mul (parse x) (parse y))]
    [_ (raise
         (exn:fail:syntax:cs450
          (format "not a valid CS450 Lang program ~a" p)
          (current-continuation-marks)))]))
```

“low-level” **raise**

“printf” format specifier (see also ~s ~v)

format is like “printf” that makes a string

Stack Context of the exn

HW9

- Add another data type

  - Strings

Supporting CS450 LANG String Programs

Programmer writes:

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; An Atom is one of:
;; - Number
;; - 450Bool
;; - String
```

Supporting CS450 LANG String Programs

Programmer writes:

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

parse

```
;; An AST is one of:
;; ...
;; - (mk-str String)
;; ...

;; ...

(struct str AST [val])
```

Parsing "CS450 LANG + STRINGS" Programs

```
;; parse: Program -> AST
;; Converts a 450Lang Program to AST

(define (parse p)
  (match p
    ...
   [(? string?) (mk-str p)]
    ...))
```

```
;; An AST is one of:
;; ...
;; (mk-str String)
;; ...

;; ...

(struct str AST [val])
```

Running "CS450 Lang + Strings" Programs

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is a:
;; - …
;; - String
```

```
;; An AST is one of:
;; ...
;; - (mk-str String)
;; ...

;; ...

(struct str AST [val])
```

```
(define (run p)
  (match p
    [(num n) n]
    [(str s) s]
    [(add x y) (???? (run x) (run y))]
    [(mul x y) (???? (run x) (run y))]))
```

Running “CS450 Lang + Strings” Programs

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is a:
;; - Number
;; - String
```

What should happen when **two strings** are **added**???

e.g., What is the “**meaning**” of `(+ “hello” “world!”)`

```
(define (run p)
  (match p
    [(num n) n]
    [(str s) s]
    [(add x y) (450+ (run x) (run y))]
    [(mul x y) (???? (run x) (run y))]))
```

Last Time

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values together
;; (following js semantics)
(define (450+ x y)
  (+ (res->num x) (res->num y)))
```

```
;; A Result is a:
;; - Number
;; - Boolean
```

```
;; res->num: Result -> Number
(define (res->num x)
  (cond
   [(number? x)  ... ]
   [(boolean? x) ... ]))
```

TEMPLATE

Last Time

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values together
;; (following js semantics)
(define (450+ x y)
  (+ (res->num x) (res->num y)))
```

```
;; res->num: Result -> Number
(define (res->num x)
  (cond
   [(number? x) x]
   [(boolean? x) (boo->num x)]))
```

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values together
;; (following js semantics)
(define (450+ x y)
  (+ (res->num x) (res->num y)))
```

```
;; A Result is a:
;; - Number
;; - Boolean
;; - String
```

```
;; res->num: Result -> Number
(define (res->num x)
  (cond
   [(number? x) x]
   [(boolean? x) (boo->num x)]
   [(string? x) ??? (str->num x) ???]))
```

JavaScript Examples Exploration (3 min)

- `repljs.com`
  - **“plus”**
  - and **“times”**
  - with all different **combinations of data types**

```
;; A Result is a:
;; - Number
;; - Boolean
;; - String
```

e.g., What is the "**meaning**" of `(+ "hello" "world!")` → `"helloworld!"`

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
     ...

    [(and (string? x) (string? y)) ???)]
```

`res->num` is not enough!

`+` is sometimes a string operation!

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
     ...

    [(and (string? x) (string? y)) (string-append x y)]
```

`res->num` is not enough!

`+` is sometimes a string operation!

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(string? y) ??? ]

    [(and (string? x) (string? y)) (string-append x y)]
```

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(string? y)                   (string-append (??? x) y)]

    [(and (string? x) (string? y)) (string-append x y)]
```

```
;; A Result is a:
;; - Number
;; - Boolean
;; - String
```

Check if Bool behavior correct? (TODO!)

```
;; res->str: Result -> String
(define (res->str x)
  (cond
   [(number? x) (num->str x)]
   [(boolean? x) (boo->str x)]
   [(string? x) x]))
```

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values tog
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(string? y)                   (string-append (res->str x) y)]

    [(and (string? x) (string? y)) (string-append x y)]
```

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(string? y)                   (string-append (res->str x) y)]
    [(string? x)                   (string-append x (res->str y))]
    [(and (string? x) (string? y)) (string-append x y)]
```

(can any cond clauses be combined?)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(or (string? x) (string? y))
     (string-append (res->str x) (res->str y))]
    [else (+ x y)]))
```

Does this cover bools??
(TODO!)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(or (string? x) (string? y))
     (string-append (res->str x) (res->str y))]
    [else (+ (res->num x) ??? (res->num y))]))
```

Does this cover bools??
(TODO!)

Running: "CS450 LANG" Programs: "times"

```
;; 450times: Result Result -> Result
;; "multiplies" 450Lang Results
;; (following js semantics)
```

```
(define (450times x y)
  (cond
    [(and (number? x) (number? y)) (* x y)]
    [(and (number? x) (string? y)) … ]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) … ]))
```

JS Semantics Exploration: “times” strings

Not always the same as “plus”!

e.g., What is the **“meaning”** of `(× “hello” “world!”)` → `NaN ???`

“Not a Number”

NaN

From Wikipedia, the free encyclopedia

In computing, **NaN** (/næn/), standing for **Not a Number**, is a particular value of a numeric data type (often a floating-point number) which is undefined or unrepresentable, such as the result of 0/0. Systematic use of NaNs was introduced by the IEEE 754 floating-point standard in 1985, along with the representation of other non-finite quantities such as infinities.

mdn web docs_

NaN

The `NaN` global property is a value representing Not-A-Number.

Running: “CS450 LANG” Programs

```
;; run: AST -> Result
;; computes the result of running a CS450Lang program AST
```

```
;; An AST is one of:
;; - (num Number)
;; - (boo Boolean)
;; - (str String)
;; - (add AST AST)
;; - (mul AST AST)
```

```
;; A Result is either:
;; - Number
;; - Boolean
;; - String
```

Running: “CS450 LANG” Programs – with NaN!

```
;; run: AST -> Result
;; computes the result of running a CS450Lang program AST
```

```
;; An AST is one of:
;; - (num Number)
;; - (boo Boolean)
;; - (str String)
;; - (add AST AST)
;;
```

```
;; A Result is either:
;; - Number
;; - Boolean
;; - String
;; - NaN
(struct nan [])
(define NaN (nan)) ; “singleton”
```

Don’t forget to update all “Result” functions!

```
;; res->str: Result -> String
(define (res->str x)
  (cond
   [(string? x) x]
   [(boolean? x) (bool->str x)]
   [(number? x) (num->str x)]
   [(NaN? x) “NaN”]))
```

Predicate? (TODO!)

Running: “CS450 LANG” Programs: “times”

```
;; A Result is either:
;; - Number
;; - Boolean
;; - String
;; - NaN
(struct nan [])
(define NaN (nan))
```

```
;; 450times: Result Result -> Result
;; “multiplies” two 450 lang Results
;; (following js semantics)
```

```
(define (450times x y)     ???
  (cond
    [(and (number? x) (number? y)) (* x y)]
    [else NaN]))
```

Does this cover all corner cases?

Try: `'(× “two” 4)`

CS450Lang - Using Other Data Types

- **Need boolean constructs** (also follow JS semantics?)**:**
  - **“equality”** (JS “loose” == vs “strict” === ?)
  - **conditional** (JS “truthy”?)

A new 450 LANG Boolean construct

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
;; - `(iffy ,Program ,Program ,Program)
```

(sort of like **"if"**)

Truthiness – more than Booleans

In JavaScript, a **truthy** value is a value that is considered `true` when encountered in a Boolean context. All values are truthy unless they are defined as falsy. That is, all values are *truthy* except `false`, `0`, `-0`, `0n`, `""`, `null`, `undefined`, `NaN`, and `document.all`.

https://developer.mozilla.org/en-US/docs/Glossary/Truthy

A new 450 LANG Boolean construct

Truthy

```
;; js “truthy true”
(check-equal? (eval450 '(iffy 10 100 200)) 100)
;; js “truthy false”
(check-equal? (eval450 '(iffy (- 100 100) “a” “b”)) “b”)
```

Falsy

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
;; - `(iffy ,Program ,Program ,Program)
```

Truthy

CS450Lang - Using Other Data Types

- **Need boolean constructs** (also follow JS semantics?)**:**
  - **“equality”** (JS “loose” == vs “strict” === ?)
  - **conditional** (JS “truthy”?)

A 450 Lang "Equality" Operator

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
;; - `(iffy ,Program ,Program ,Program)
```

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
;; - `(iffy ,Program ,Program ,Program)
;; - `(~= ,Program ,Program)
```

A "Loose" Equality Operator

- **Follow JS == "loose" equality operator** (look it up)

Double equals (==) will perform a type conversion when comparing two things,

```
const num = 0;
const str = "0";
console.log(num == str); // true
```

Is this right???

```
;; weird "loose" ~= test case
(check-true (eval450 '(~= (+ 10 90) (+ "10" "0"))))
```

```
;; A Program is one of:
;; - Atom
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
;; - `(iffy ,Program ,Program ,Program)
;; - `(~= ,Program ,Program)
```

As language designers, we give the meaning to these symbols