The Substitution Model

Prof. Clarkson

Fall 2018

Today's music: *Substitute* by The Who

Attendance question

Are these two the same functions?

```
fun x -> x
fun y -> y
```

A. Yes
B. No

Review

Previously in 3110: simple interpreter

- abstract syntax tree (AST)
- evaluation based on single steps

Today:

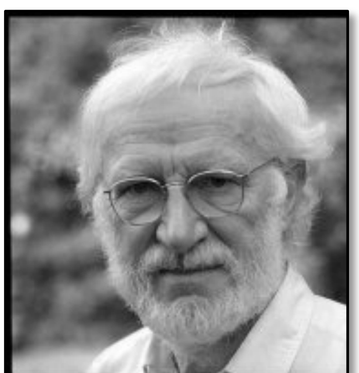
- Formal syntax: BNF
- Formal dynamic semantics: small-step, substitution model
- Formal static semantics

FORMAL SYNTAX

```
e ::= x
    | i
    | e1 + e2
    | let x = e1 in e2
```

Backus-Naur Form (BNF)

John Backus (1924-2007)
ACM Turing Award Winner 1977
"For profound, influential, and lasting contributions to the design of practical high-level programming systems"



Peter Naur (1928-2016)
ACM Turing Award Winner 2005
"For fundamental contributions to programming language design"

BNF

Note resemblance:

```
e ::= x | i | e1 + e2
    | let x = e1 in e2
```

```
type expr =
  | Var of string
  | Int of int
  | Add of expr * expr
  | Let of string * expr * expr
```

FORMAL DYNAMIC SEMANTICS

$$e \longrightarrow e'$$

single-step relation

$$\mathbf{v} \quad \not\rightarrow$$

values never step

$$e \longrightarrow^{*} e'$$

multi-step relation

Question

Which of these is true?

```
A. (5+2)+0 -->* (5+2)+0
B. (5+2)+0 -->* 7+0
C. (5+2)+0 -->* 7
D. All of the above
```

`e1 + e2 --> e1' + e2`

if **`e1 --> e1'`**

`v1 + e2 --> v1 + e2'`

if **`e2 --> e2'`**

`v1 + v2 --> i`

if **`i`** *is the result of primitive operation* **`v1+v2`**

Demo

```
let x = e1 in e2
--> let x = e1' in e2
  if e1 --> e1'

let x = v1 in e2 --> e2{v1/x}
```

Demo

Booleans

```
e ::= x | i | b
    | e1 + e2 | e1 && e2
    | let x = e1 in e2
    | if e1 then e2 else e3

v ::= i | b
```

Demo

Evaluation models

Small-step substitution model:

- Substitute value for variable
- Good mental model for evaluation
- Inefficient: too much work at run time
- Not really what OCaml does

Big-step environment model:

- Maintain data structure binding variables to values
- At the heart of what OCaml really does
- **(next lecture)**

FORMAL STATIC SEMANTICS

Static semantics

We can have nonsensical expressions:

```
5 + false
if 5 then true else 0
```

Need to rule those out...

`if` expressions [from lec 2]

Syntax:

```
if e1 then e2 else e3
```

Type checking:

if **`e1`** has type **`bool`** and **`e2`** has type **`t`** and **`e3`** has type **`t`**
then **`if e1 then e2 else e3`** has type **`t`**

Static semantics

Defined as a ternary relation:

`T |- e : t`

- Read as in typing context `T`, expression `e` has type `t`
- Turnstile `|-` can be read as "proves" or "shows"
- You're already used to `e : t`, because utop uses that notation
- *Typing context* is a dictionary mapping variable names to types

Static semantics

e.g.,

```
x:int |- x + 2 : int
x:int,y:int |- x < y : bool
|- 5 + 2 : int
```

Demo

Purpose of type system

Ensure **type safety:** well-typed programs don't get *stuck:*

- haven't reached a value, and
- unable to evaluate further

Lemmas:

Progress: if `e : t`, then either `e` is a value or `e` can take a step.

Preservation: if `e : t`, and if `e` takes a step to `e'`, then `e' : t`.

Type safety = progress + preservation

Proving type safety is a fun part of CS 4110

Upcoming events

- N/A

This is not a substitute.

THIS IS 3110