The Substitution Model

Nate Foster
Spring 2019

Today’s music: Substitute by The Who
Attendance question

Are these two the same functions?

\[
\text{fun } x \rightarrow x \\
\text{fun } y \rightarrow y
\]

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

define type expr =
    | Var of string
    | Int of int
    | Add of expr * expr
    | Let of string * expr * expr
FORMAL DYNAMIC SEMANTICS
single-step relation
values never step
\[ e \rightarrow^* e' \]

multi-step relation
Question

Which of these is true?

A. \((5+2)+0 \implies * \ (5+2)+0\)
B. \((5+2)+0 \implies * \ 7+0\)
C. \((5+2)+0 \implies * \ 7\)
D. All of the above
\[ e_1 + e_2 \rightarrow e_1' + e_2 \]

\[ \text{if } e_1 \rightarrow e_1' \]

\[ v_1 + e_2 \rightarrow v_1 + e_2' \]

\[ \text{if } e_2 \rightarrow e_2' \]

\[ v_1 + v_2 \rightarrow i \]

\[ \text{if } i \text{ is the result of primitive operation } v_1 + v_2 \]
let \( x = e_1 \) in \( e_2 \)

\[\rightarrow \] let \( x = e_1' \) in \( e_2 \)

\( \text{if} \ e_1 \rightarrow e_1' \)

let \( x = v_1 \) in \( e_2 \) \( \rightarrow \) \( e_2\{v_1/x\} \)
Booleans

\[
e ::= x \mid i \mid b \\
| e1 + e2 \mid e1 \&\& e2 \\
| let x = e1 in e2 \\
| if e1 then e2 else e3
\]

\[
v ::= i \mid b
\]
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)
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:

\[ T \vdash e : t \]

• Read as in typing context \( T \), expression \( e \) has type \( t \)
• Turnstile \( \vdash \) 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.,

\[ x : \text{int} \mid - x + 2 : \text{int} \]

\[ x : \text{int}, y : \text{int} \mid - x < y : \text{bool} \]

\[ - 5 + 2 : \text{int} \]
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

• [today] Foster OH @ 1:15pm

This is not a substitute.

THIS IS 3110