CSC 530 Lecture Notes Week 7

More on Tennent-Style Denotational Semantics

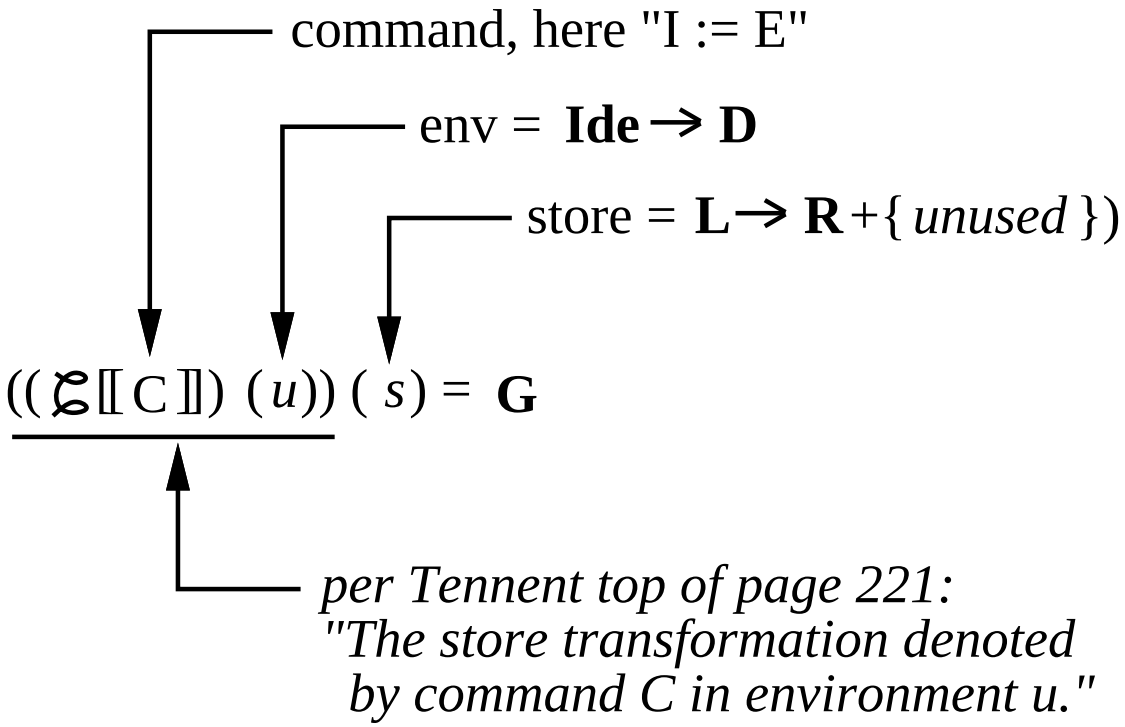
I. Tennent Ch 13.

A. We'll dissect some key definitions.

B. Prepare for pervasive use of functions (of functions (of functions)).

II. Assignment statements

A. Dissection of assmnt from 13.3:



Assignment, cont'd

where **G** expands as follows:

$$
\begin{aligned}
\mathbf{G} &= s[d \mid\!\rightarrow e] \\
&= s[u[\![\mathrm{I}]\!] \mid\!\rightarrow \mathcal{E}[\![\mathrm{E}]\!]\, u\, s\ ] \\
&= s[(\mathbf{Ide} \rightarrow \mathbf{D})\ [\![\mathrm{I}]\!] \mid\!\rightarrow \\
&\qquad \mathcal{E}[\![\mathrm{E}]\!]\ (\mathbf{Ide} \rightarrow \mathbf{D}) \\
&\qquad\quad (\mathbf{L} \rightarrow (\mathbf{R} + \{\mathit{unused}\}))]
\end{aligned}
$$

Assignment, cont'd

B. What this says is that the meaning of an assignment statement of the form "I := E" for some identifier "I" and expression "E" is a modified store, where the location of the modification is the memory location bound to "I" in the environment and the value of the modification is the value produced by evaluating E in the environment and pre-modified store.

C. Cool, huh?

III. Procedures

A. Consider:

Code	**Semantics**
val n = 100 **new** x = 0 t = **true**	$\mathcal{D}$ [[**val** n = 100]] $\mathcal{D}$ [[**new** x = 0]] $\mathcal{D}$ [[**new** t = **true**]]
new Pn = (**procedure** x := x+1)	$\mathcal{D}$ [[**val** Pn = (**proc** ...)]]
val Pv = (**procedure** x := x+1)	$\mathcal{D}$ [[**val** Pv = (**proc** ...)]]
call(Pv)	$\mathcal{C}$ [[Pv]]u s

Procedures, cont'd

B. Resulting env and store

Environment:

Ide	Value x Tag
n	100,**Z**
x	0xff20, **L**
t	0xff24, **L**
Pn	0xff25, **L**
Pv	$\mathcal{C}$[[x:=x+1]]u_v

Store:

Addr	Value
0xff20	0
1	0
2	0
3	0
4	true
5	$\mathcal{C}$[[x:=x+1]]u_n *storage for body of Pv* ...

Procedures, cont'd

C. Notes

1. Depiction of env resembles lookup table, depiction of store resembles memory.

   a. Both depictions are merely suggestions.

   b. Abstractly, env and store are unary functions.

Procedures, cont'd

2. Assoc-style lookup denoted

   $u[\![x]\!]$

   a. This means apply the env function *u* to ident "x".

   b. Think of *entire table* applied as function to "x".

Procedures, cont'd

3. Proc values bound to Pn and Pv are unevaluated lambda bodies, with an *attached env*.

    a. Attachment defines language as *statically scoped*.

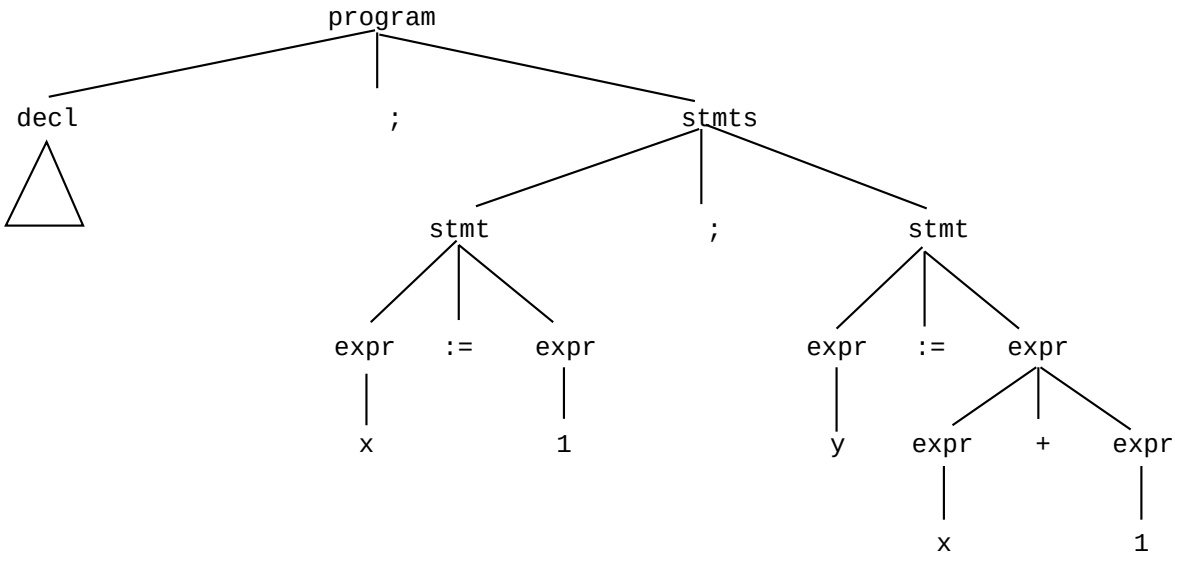
    b. Note different static env in Pn versus Pv.

IV. Tennent versus Knuth eval

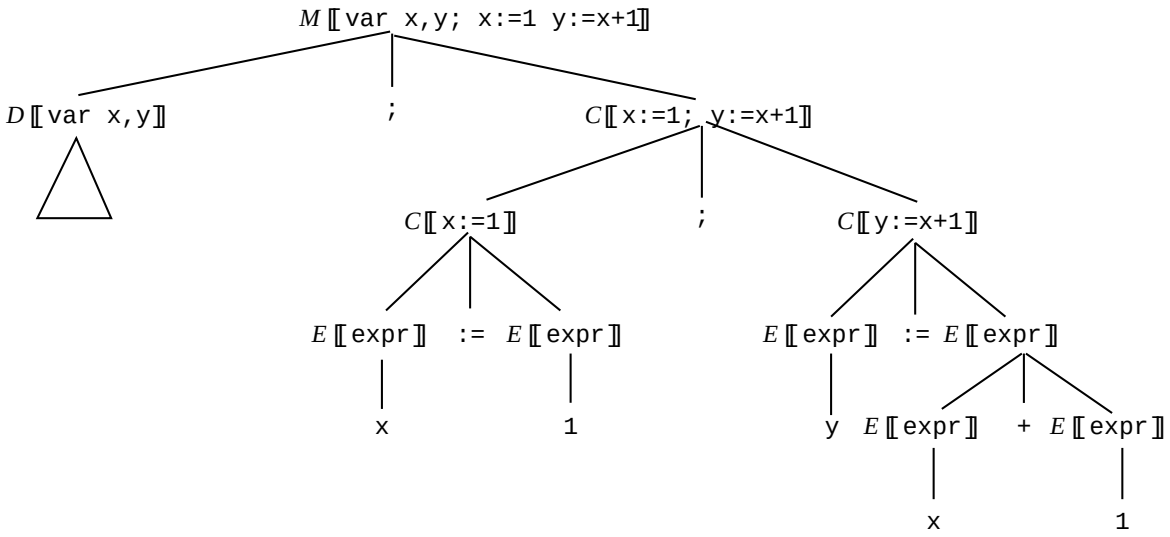
A. Consider

var x,y;
x := 1;
y := x+1;

B. Knuth eval walks the parse tree:



C. Comparable Tennent eval:



V. Infinitary program behavior -- loops

A. Thus far, not directly considered.

B. There are two approaches.

1. "Operational" mathematical approach.

2. Limit/Fixpoint approach.

VI. Fixpoint def of `while`

A. Basic idea

"**while** E **do** C"=
if E **then {** C ; **while** E **do** C **}**

Using a bit of notation:

$\mathcal{C}[\![$ **while** E **do** C $]\!]$ =
if $\mathcal{E}[\![$E$]\!]$ **then begin**
$\mathcal{C}[\![$C$]\!]$ **;** $\mathcal{C}[\![$**while** E **do** C$]\!]$**end**

where let

$f = \mathcal{C}[\![$ **while** E **do** C $]\!]$
F(f) = **if** $\mathcal{E}[\![$E$]\!]$ **then begin** $\mathcal{C}[\![$C$]\!]$ **;**
$\mathcal{C}[\![$**while** E **do** C$]\!]$ **end**

While, cont'd

B. **Question**: What's going on here?

Ans: it's an *equation* to be solved for *f*, i.e., for $f = F(f)$

C. **Question**: Does this form of equation have a solution in general? I.e., for a functional $F: D \rightarrow D$, does there exist

$Y: (D \rightarrow D) \rightarrow D$ such that $Y(F) = F(Y(F))$

While, cont'd

Ans: Yes, if we make the following assumptions:

1. Function domains properly defined.

2. All functions are continuous

D. Solution is called a *fixpoint*

While, cont'd

E. **Question**: What does such a fixpoint look like?

Ans:

1. For non-recursive cases, e.g.,

fix(f(x) = 4) is 4

fix(f(x) = 8 - x) is 4

While, cont'd

2. For recursive cases, consider

   let f(x) = if x=0 then 1
   else if x=1 then f(3) else f(x-2)

   a. A fixpoint of this is

      f(x) =1 if x is is even and x≥0
      undefined otherwise

While, cont'd

b. But also a fixpoint is

$$f(x) = \begin{cases} 1 & \text{if } x \text{ is even and } x \geq 0 \\ a & \text{if } x \text{ is odd and } x > 0 \\ b & \text{otherwise} \end{cases}$$

for any a,b

c. The first is the "*least*" fixpoint.

While, cont'd

3. For while loop, fixpoint function looks like a *limit* of successive approximations.

   a. I.e.,

   (**while** E **do** C) $_0$ =

   "the worst loop approximation"

   (**while** E **do** C) $_{i+1}$ =

   if E **then** C;

   (**while** E **do** C) $_i$

While, cont'd

b. More specifically,

(**while** E **do** C) $_0$ =

while true do null;

(**while** E **do** C) $_1$ =

if E **then** C;

while true do null;

(**while** E **do** C) $_2$ =

if E **then** C;

if E **then** C ;

while true do null;

...

While, cont'd

c. In terms of functions,

let f =

$\mathcal{C}[\![$**while** E **do** C$]\!]$

and we want

$f = \mathcal{E}[\![\text{E}]\!] \rightarrow \mathcal{C}[\![\text{C; }$**while** E **do** C$]\!]$

$= \mathcal{E}[\![\text{E}]\!] \rightarrow (\mathcal{C}[\![$**while** E **do** C$]\!](\mathcal{C}[\![\text{C}]\!]))$

$= \mathcal{E}[\![\text{E}]\!] \rightarrow f(\mathcal{C}[\![\text{C}]\!])$

While, cont'd

and now, we want

$$f = F(f)$$

d. That is, we have arrived at

$$\perp, F(\perp), F(F(\perp)), ..., F^i(\perp)$$

which approximates *f*. I.e.,

While, cont'd

$$\mathbf{Fix_D} : (D \to D) \to D$$

defined as

$$\mathbf{Fix_D} = \lim_{i\to} \; F^i(\mid)$$

is the "*least*" solution of $f = F(f)$.