CS 330 Lecture 14

- Operational Semantics
  - explain how a program behaves by specifying how an arbitrary program is to be executed on a machine whose operation is completely known
  - abstract machine (easy to be understood and simulated by the user or the computer)
- Alternative:
  - definitional interpreters and compilers
- Specification: states + rules for transitioning between states

State in Impcore

- Toplevel t or expression e
- value env (global variables) g
- fundef-env f
- value env (formal parameters) p
- < e, , , > machine evaluating expr
- < t, , > machine evaluating top

Judgments

- Transition rules of the abstract machine
  - evaluating e produces value v
  - <e, g, f, p> => <v, g', f, p'>
  - In the environments g, f, p evaluating e produces value v and also produces new environments g' and p', f is unchanged
  - Eval of expr always produces a value
  - Might change a global variable or formal param
  - Never adds or changes function definition

eval

- Heart of the interpreter
  - eval(e, g, f, p) returns v and has side effects on g and p such that <e, g, f, p> => <e, g', f, p'>
- Recursive implementation
  - match each pattern for expression, recursively eval subexpressions and reduce pattern
  - if no rule can be found machine is stuck
    - compile or run-time error

Top-level

- $<t, g, f> \; -> \; <g', f'>$
- evaluating top-level item t in the environment g and f yields new environments g' and f'

Rules of inference

- Not each judgment is true
- Operational semantics uses rules of inference to tell which judgements are valid
- Form: premises
  ----
  conclusion

IFTRUE rule:

$<e_1,g,f,p> \; => \; <v_1,g',f,p'> \quad v_1 \; != 0 \; <e_2,g',f,p'> \; => \; <v_2, g'', f, p''>$

$<IF(e_1,e_2,e_3),g,f,p> \; => \; <v_2, g'', f, p''>$

Bottom-up over rules

IFTRUE

$<e_1,g,f,p> \; => \; <v_1,g',f,p'> \quad v_1 \; != 0 \; <e_2,g',f,p'> \; => \; <v_2, g'', f, p''>$

$<IF(e_1,e_2,e_3),g,f,p> \; => \; <v_2, g'', f, p''>$

Look at form – match if expression
Recursive call to find v_1, g', p' such that $<e_1,g, f, p> \; => \; <v_1,g',f,p'>$
then if v_1 != 0 another recursive fall to find v_2, g'', p'' such that
$<e_2, g', f, p'> \; => \; < v_2, g'', f, p''>$
Having satisfied all premises of rule IFTRUE return
v_2 and modified environments

Rules

Values

$<LITERAL(v), g, f, p> \; => \; <v,g,f,p>$

Variables

x is in p
-- (formal var)
$<VAR(x), g, f, p> \; => \; <p(x), g, f, p>$

x is not in p   x is in g
-- (global var)
$<VAR(x), g, f, p> \; => \; <g(x), g, f, p>$

Assignments

x is in p $\quad <e,g,f,p> \Rightarrow <v,g',f,p'>$

$<SET(x,e),g,f,p> \Rightarrow <v,g',f,p'\{x\text{->}v\}>$

formal assign

x is not in p $\quad$ x is in g $\quad <e,g,f,p> \Rightarrow <v,g',f,p'>$

$<SET(x,e),g,f,p> \Rightarrow <v,g'\{x\text{->}v\},f,p'>$

global assign

x is not in p $\quad <e,g,f,p> \Rightarrow <v,g',f,p'>$

$<SET(x,e),g,f,p> \Rightarrow <v,g'\{x\text{->}v\},f,p'>$

global assign awk

Control Flow

IFTRUE:

$<e_1,g,f,p> \Rightarrow <v_1,g',f,p'> \quad v_1 \,!= 0 \quad <e_2,g',f,p'> \Rightarrow <v_2,g'',f,p''>$

$<IF(e_1,e_2,e_3),g,f,p> \Rightarrow <v_2,g'',f,p''>$

IFFALSE:

$<e_1,g,f,p> \Rightarrow <v_1,g',f,p'> \quad v_1 = 0 \quad <e_3,g',f,p'> \Rightarrow <v_3,g'',f,p''>$

$<IF(e_1,e_2,e_3),g,f,p> \Rightarrow <v_3,g'',f,p''>$

Loops

WHILE-ITERATE

$<e_1,g,f,p> \Rightarrow <v_1,g',f,p'> \quad v_1 \,!= 0$

$<e_2,g',f,['> \Rightarrow <v_2,g'',f,p''> \quad <WHILE(e_1,e_2),g'',f,p''> \Rightarrow <v_3,g''',f,p'''>$

$<WHILE(e_1,e_2,g,f,p)> \Rightarrow <v_3,g''',f,p'''>$

WHILEEND

$<e_1,g,f,p> \Rightarrow <v_1,g',f,p'> \quad v_1 = 0$

$WHILE(e_1,e_2),g,f,p> \Rightarrow <0,g',f,p'>$

e_2 evaluated only for side effects

Sequential Execution

$<BEGIN(),g,f,p> \Rightarrow <0,g,f,p>$

order of expressions matters
order of premises doesn't

$<e_1,g_0,f,p_0> \Rightarrow <v_1,g_1,f,p_1>$

$<e_2,g_1,f,p_1> \Rightarrow <v_2,g_2,f,p_2>$

....

....

$<e_n,g_{n-1},f,p_{n-1}> \Rightarrow <v_n,g_n,f,p_n>$

$<BEGIN(e_1,e_2,....e_n),g_0,f,p_0)> \Rightarrow <v_n,g_n,f,p_n>$

Function Application

$f(foo) = USER(<x_1,.....x_n>, e)$

$x_1,...., x_n$ all distinct

$<e_1, g_0, f, p_0> \Rightarrow <v_1, g_1, f, p_1>$

...

$<e_n, g_{n-1}, f, p_{n-1}> \Rightarrow <v_n, g_n, f, p_n>$

$<e, g_n, f, \{x_1 \rightarrow v_1, ... x_n \rightarrow v_n\}> \Rightarrow <v, g', f, p'>$

-- APPLY USER

$<APPLY(foo, e_1, ..., e_n), g_0, f, p_0> \Rightarrow <v, g', f, p_n>$

behavior of function doesn't depend on function name, only definition
body of a function can't get the formal parameters of its caller
functions assigns to formal parameters changes are not visible outside
p' can be thrown away after evaluation => VERY IMPORTANT

Primitive Functions

$f(foo) = PRIMITIVE(+)$

$<e_1, g_0, f, p_0> \Rightarrow <v_1, g_1, f, p_1>$

$<e_2, g_1, f, p_1> \Rightarrow <v_2, g_2, f, p_2>$

$<APPLY(foo, e_1, e_2), g_0, f, p_0> \Rightarrow <v_1+v_2, g_2, f, p_2>$

APPLYADD

Top-level items

$<e, g, f, \{\}> \Rightarrow <v, g', f, p'>$

---------------------------------- EVALEXP

$<EXP(e), g, f> \rightarrow <g', f>$

$<e, g, f, \{\}> \Rightarrow <v, g', f, p'>$

--- DEFINEGLOBAL

$<VAL(x,e),g,f> \rightarrow <g'\{x \rightarrow v\}, f>$

x1, x2,... xn all distinct

---------------------------- DEFINE FUNCTION

$<DEFINE(foo, <x_1,..., x_n>, e>, e), g, f> \rightarrow \{g, f \rightarrow USER(<x_1..x_n>, e)\}>$