CS4400 Lecture 14: Control Stacks, Abstract Machines
Feburary 6, 2026

1 Lecture outline

- Motivating examples: effects
- State transition systems
- CK: a stack machine for CalcLang
- Implementation

2 Motivating examples

We are interested in describing *effects* in programming, which include basically all of the interesting phenomena: file i/o, mutable state, exceptions, and so on. By restricting our attention so far to internal representations of data and computation, we have laid the groundwork for reasoning, but now we'd like to see how these ideas scale to effects.

The presence of effects requires that we reason more explicitly about the order of evaluation, since it will *change the meaning of a program*!

Example:

```
  (\x.\y.(x,x)) (print "hello"; 5) (print "world"; 7)

lazy:   (5,5)   | "hello" "hello"
eager:  (5,5)   | "hello" "world"
```

So we move to a form of dynamic semantics that both gets us closer to an implementation and also makes for nicer proofs in the presence of effects: control stacks.

3 Main idea: state transition systems

We will represent our interpreter as a series of "machine states" and rules for transitioning between them. A *state transition system* has a set of rules for when one state can be rewritten to another. We will use this as our judgment:

$\boxed{S \mapsto S'}$

A machine state S will track the current expression (program) we want to evaluate, as well as a *stack* of computations to do next when it is done computing. Throughout execution, we can push things onto the stack, pop things off of the stack, and modify the current expression. We will also have an indicator to say whether we are *evaluating* or *returning* the current expression.

Here's an example program state:

$$\epsilon \rhd ((1 + 3) + 12))$$

The ϵ represents the empty stack, and the $\rhd$ symbol says to evaluate the expression to the right of it. Here's what it will transition to in our system:

$$\epsilon; (\square + 12) \rhd (1 + 3)$$

In this state, the stack has one *frame*, or expression with a hole, saying "when done evaluating, plug the value in for this hole and continue." The state also contains the expression we need to evaluate next, $1 + 3$.

In general, our states S are

$k \rhd e$: evaluating e with stack k

$k \lhd v$: returning value v to stack k

Judgment forms:

$S \mapsto S'$: stack state S steps to state S'.

4 A stack machine for CalcLang

S terminal: state S is *terminal* (cannot transition).

Stack syntax:

$$\begin{array}{lll} k & ::= & \epsilon \mid k; f \\ f & ::= & (\square\ +e) \mid (\mathsf{num}\ n\ +\square) \mid (\mathsf{if}\ \square\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \end{array}$$

Terminal states have an empty stack and a value:

$$\frac{}{\epsilon \lhd v\ \mathsf{terminal}}$$

TODO: state transition rules for IfLang.

4.1 Examples

1. $(4 + 3) + 2$
2. `(if false then 2 else 1) + (4 + 3)`