Syntax, Semantics, Interpreters , & Compilers

Jim Royer

CIS 352

January 29, 2019

THE FOLLOWING PREVIEW HAS BEEN APPROVED FOR

ALL AUDIENCES

BY THE EECS DEPARTMENT OF SYRACUSE UNIVERSITY

www.cis.syr.edu/courses/cis352/ http://eng-cs.syr.edu

Aexp: A very simple language

Syntactic categories

$$
\begin{array}{lll}
n \in \textbf{Num} & & \text{Numerals} \\
a \in \textbf{Aexp} & & \text{Arithmetic expressions}
\end{array}
$$

Grammar

$$a \quad ::= \quad n \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 * a_2)$$

Conventions

- Language vs. Metalanguage
  E.g., n, a, b, S, x, etc. are metavariables.
- We write $\ulcorner 35 \urcorner$ for the numeral 35.

Examples

- $\ulcorner 2 \urcorner$
- $(\ulcorner 2 \urcorner + \ulcorner 5 \urcorner)$
- $(((\ulcorner 2 \urcorner + \ulcorner 5 \urcorner) * \ulcorner 13 \urcorner) - \ulcorner 9 \urcorner)$

Aside: Grammars for Natural Languages, 1

$$
\begin{aligned}
\langle \mathit{sentence} \rangle &::= \langle \mathit{subject} \rangle \langle \mathit{verb1} \rangle \mid \langle \mathit{subject} \rangle \langle \mathit{verb2} \rangle \langle \mathit{object} \rangle \\
\langle \mathit{subject} \rangle &::= \langle \mathit{article} \rangle \langle \mathit{noun} \rangle \mid \langle \mathit{pronoun} \rangle \\
\langle \mathit{object} \rangle &::= \text{that } \langle \mathit{sentence} \rangle \\
\langle \mathit{verb1} \rangle &::= \text{swims} \mid \text{pauses} \mid \text{exists} \\
\langle \mathit{verb2} \rangle &::= \text{believes} \mid \text{hopes} \mid \text{imagines} \\
\langle \mathit{article} \rangle &::= \text{a} \mid \text{some} \mid \text{the} \\
\langle \mathit{noun} \rangle &::= \text{lizard} \mid \text{truth} \mid \text{man} \\
\langle \mathit{pronoun} \rangle &::= \text{he} \mid \text{she} \mid \text{it}
\end{aligned}
$$

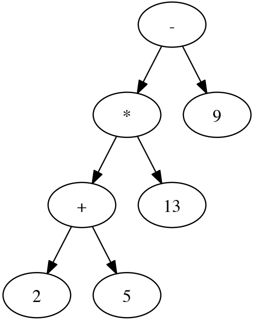
Aside: Grammars for Natural Languages, 2

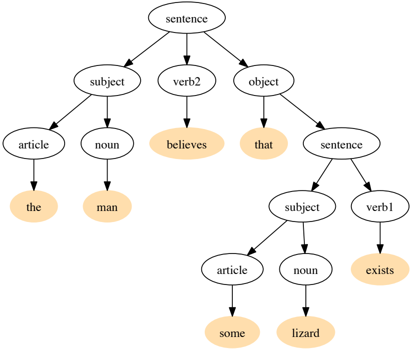
$$
\begin{aligned}
&\underline{\langle \textit{sentence} \rangle}\\
&\qquad \to \underline{\langle \textit{subject} \rangle}\ \langle \textit{verb2} \rangle\ \langle \textit{object} \rangle\\
&\qquad \to \underline{\langle \textit{article} \rangle}\ \langle \textit{noun} \rangle\ \langle \textit{verb2} \rangle\ \langle \textit{object} \rangle\\
&\qquad \to \text{the}\ \underline{\langle \textit{noun} \rangle}\ \langle \textit{verb2} \rangle\ \langle \textit{object} \rangle\\
&\qquad \to \text{the man}\ \underline{\langle \textit{verb2} \rangle}\ \langle \textit{object} \rangle\\
&\qquad \to \text{the man believes}\ \underline{\langle \textit{object} \rangle}\\
&\qquad \to \text{the man believes that}\ \underline{\langle \textit{sentence} \rangle}\\
&\qquad \to \text{the man believes that}\ \underline{\langle \textit{subject} \rangle}\ \langle \textit{verb1} \rangle\\
&\qquad \to \text{the man believes that}\ \underline{\langle \textit{article} \rangle}\ \langle \textit{noun} \rangle\ \langle \textit{verb1} \rangle\\
&\qquad \to \text{the man believes that some}\ \underline{\langle \textit{noun} \rangle}\ \langle \textit{verb1} \rangle\\
&\qquad \to \text{the man believes that some lizard}\ \underline{\langle \textit{verb1} \rangle}\\
&\qquad \to \text{the man believes that some lizard exists}
\end{aligned}
$$

Abstract Syntax

Abstract syntax

$\approx$ the grammatical structure/representation of an expression





$(((\ulcorner 2 \urcorner + \ulcorner 5 \urcorner) * \ulcorner 13 \urcorner) - \ulcorner 9 \urcorner)$

the man believes that some lizard exists

$$a \;::=\; n \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 * a_2)$$

$$\mathit{PLUS}_{BSS}: \quad \frac{a_1 \to v_1 \qquad a_2 \to v_2}{(a_1 + a_2) \to v} \; (v = v_1 + v_2)$$

$$\mathit{MINUS}_{BSS}: \quad \frac{a_1 \to v_1 \qquad a_2 \to v_2}{(a_1 - a_2) \to v} \; (v = v_1 - v_2)$$

$$\mathit{MULT}_{BSS}: \quad \frac{a_1 \to v_1 \qquad a_2 \to v_2}{(a_1 * a_2) \to v} \; (v = v_1 * v_2)$$

$$\mathit{NUM}_{BSS}: \quad \frac{}{n \to v} \; (\mathcal{N}[\![n]\!] = v)$$

Notes

- $a \to v$ is a transition.
- $a \to v \equiv$ expression a evaluates to value v.
- Upstairs transitions are called premises.
- Downstairs transitions are called conclusions.
- Parenthetical equations on the side are called side conditions.
- $\mathcal{N}$: numerals $\to \mathbb{Z}$. I.e., $\mathcal{N}[\![\ulcorner -43 \urcorner]\!] = -43$.
- The NUM_{BSS} rule is an example of an axiom.

Aside: Rules

General Format

$$\textit{rule name:}\ \frac{\textit{premises}}{\textit{conclusions}}\ (\textit{side conditions})$$

The big-step semantics in Haskell

A Haskell data structure for the abstract syntax

```
data Aexp  = Num Integer
           | Add Aexp Aexp
           | Sub Aexp Aexp
           | Mult Aexp Aexp
```

$$
\begin{aligned}
a \; ::= \; & n \\
| \; & (a_1 + a_2) \\
| \; & (a_1 - a_2) \\
| \; & (a_1 * a_2)
\end{aligned}
$$

The big-step semantics as an evaluator function

```
aBig (Num n)       = n
aBig (Add a1 a2)   = (aBig a1) + (aBig a2)
aBig (Sub a1 a2)   = (aBig a1) - (aBig a2)
aBig (Mult a1 a2)  = (aBig a1) * (aBig a2)
```

See AS0.hs, Parser0.hs, and eval0.hs.

Derivation trees

A derivation is a tree of rule applications with "axioms" (rules with empty premises) at the leaves. E.g.,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\ulcorner 2 \urcorner \to 2} \qquad \dfrac{}{\ulcorner 5 \urcorner \to 5}}{(\ulcorner 2 \urcorner + \ulcorner 5 \urcorner) \to 7} \qquad \dfrac{}{\ulcorner 13 \urcorner \to 13}}{((\ulcorner 2 \urcorner + \ulcorner 5 \urcorner) * \ulcorner 13 \urcorner) \to 91} \qquad \dfrac{}{\ulcorner 9 \urcorner \to 9}}{(((\ulcorner 2 \urcorner + \ulcorner 5 \urcorner) * \ulcorner 13 \urcorner) - \ulcorner 9 \urcorner) \to 82}$$

Rule names & side-conditions omitted, to reduce clutter.

Class exercise:

Grow a derivation tree for $(\ulcorner 2 \urcorner + \ulcorner 3 \urcorner) * (\ulcorner 4 \urcorner + \ulcorner 9 \urcorner) \to 65$.

What do **Aexp** expression mean? Small-step rules

$$a \ ::= \ n \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 * a_2) \mid v$$

$$\textit{PLUS-1}_{SSS}: \ \frac{a_1 \Rightarrow a_1'}{(a_1 + a_2) \Rightarrow (a_1' + a_2)}$$

$$\textit{PLUS-2}_{SSS}: \ \frac{a_2 \Rightarrow a_2'}{(a_1 + a_2) \Rightarrow (a_1 + a_2')}$$

$$\textit{PLUS-3}_{SSS}: \ \frac{}{(v_1 + v_2) \Rightarrow v} \ (v = v_1 + v_2)$$

$$\vdots$$

$$\textit{NUM}_{SSS}: \ \frac{}{n \Rightarrow v} \ (\mathcal{N}[\![n]\!] = v)$$

Notes

- These are rewrite rules.
- We now allow values in expressions.
- $a \Rightarrow a'$ is a transition.
- $a \Rightarrow a'$ $\equiv$ expression a evaluates (or rewrites) to a' ***in one-step***.
- v is a terminal expression.
- The rules for $-$ and $*$ follow the same pattern as the rules for $+$.

Class exercise

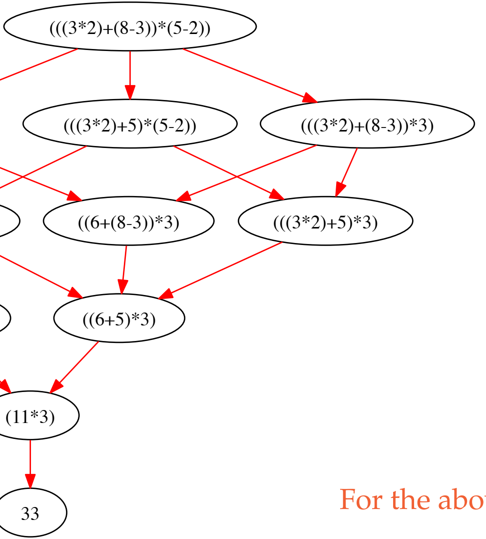
Show:

$$(((3 * 2) + (8 - 3)) * (5 - 2))$$

$$\Rightarrow \begin{cases} ((6 + (8 - 3)) * (5 - 2)) \\ (((3 * 2) + 5) * (5 - 2)) \\ (((3 * 2) + (8 - 3)) * 3) \end{cases}$$

Note: Above, we've already done the boring translation of numerals (e.g., $\ulcorner 3 \urcorner$) to numbers (e.g., 3).

There is a lattice of transitions



For the above read $\Rightarrow$ for $\rightarrow$.

Properties of operational semantics

Definition

A transition system $(\Gamma, \rightsquigarrow, T)$ is deterministic when for all a, a_1, and a_2:

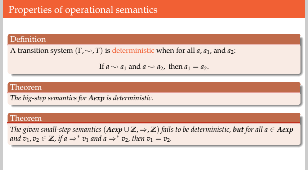
$$\text{If } a \rightsquigarrow a_1 \text{ and } a \rightsquigarrow a_2, \text{ then } a_1 = a_2.$$

Theorem

*The big-step semantics for **Aexp** is deterministic.*

Theorem

*The given small-step semantics $(\textbf{Aexp} \cup \mathbb{Z}, \Rightarrow, \mathbb{Z})$ fails to be deterministic, **but** for all $a \in \textbf{Aexp}$ and $v_1, v_2 \in \mathbb{Z}$, if $a \Rightarrow^* v_1$ and $a \Rightarrow^* v_2$, then $v_1 = v_2$.*



Theorem

*The big-step semantics for **Aexp** is deterministic.*

Proof by structural induction.

- $e = n$, a numeral. Then just one rule NUM_{BSS} applies.
- $e = e_1 + e_2$ and suppose by induction that e_1 and e_2 have unique derivations of values v_1 and v_2. Let $v = v_1 + v_2$. Then $PLUS_{BSS}$ is the one rule that applies to build the (unique) derivation for e (with value v) from the e_1 and e_2 derivations.
- $e = e_1 - e_2$ and $e = e_1 * e_2$ are essentially a repeat of the $+$ case.

!!! The other theorem is a bit trickier to proof. More on that sort of argument later.

A deterministic small-step semantics for **Aexp**

$$a \ ::= \ n \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 * a_2) \mid v$$

$$\textit{PLUS-1}'_{SSS}: \quad \frac{a_1 \Rightarrow a_1'}{a_1 + a_2 \Rightarrow a_1' + a_2}$$

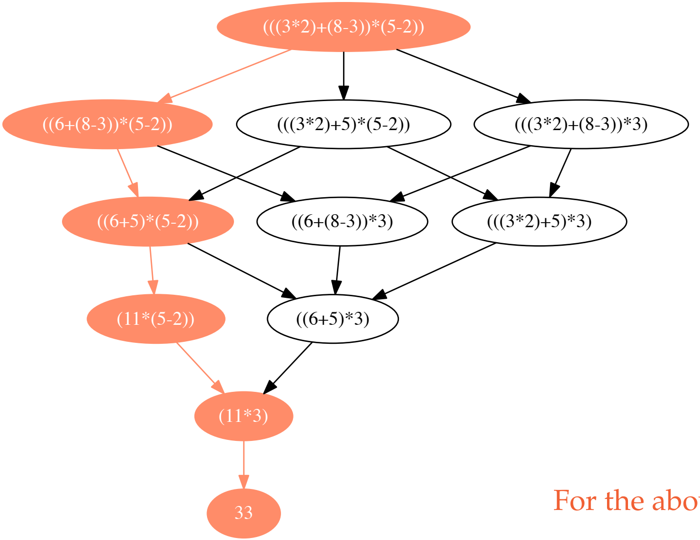
$$\textit{PLUS-2}'_{SSS}: \quad \frac{a_2 \Rightarrow a_2'}{v_1 + a_2 \Rightarrow v_1 + a_2'}$$

$$\textit{PLUS-3}'_{SSS}: \quad \frac{}{v_1 + v_2 \Rightarrow v} \ (v = v_1 + v_2)$$

$$\vdots$$

$$\textit{NUM}_{SSS}: \quad \frac{}{n \Rightarrow v} \ (\mathcal{N}[\![n]\!] = v)$$

The leftmost path through the lattice of transitions



For the above read $\Rightarrow$ for $\rightarrow$.

Interpreters and compilers

interpreter

$$\text{source code} \xrightarrow[\text{and parser}]{\text{via lexer}} \text{abstract syntax} \xrightarrow[\text{interpreter}]{\text{via evaluator/}} \text{value}$$

compiler

$$\text{source code} \xrightarrow[\text{and parser}]{\text{via lexer}} \text{abstract syntax} \xrightarrow{\text{via compiler}} \text{object code}$$

$$\xrightarrow{\text{via linker}} \text{executable} \xrightarrow[\text{interpreter}]{\text{via hardware}} \text{value}$$

There are lots of things in-between and lots of variations on the above.

Problem: Compile Aexp to a stack-based VM

Aexp

$n \in$ **Num** (Numerals) $\qquad$ $a \in$ **Aexp** (Arithmetic expressions)

$$a \;::=\; n \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 * a_2)$$

What is our target virtual machine?*

*See http://en.wikipedia.org/wiki/Virtual_machine for a discussion of virtual machines. In this course we are interested in *process (or language) virtual machines*.

Our target VM, 1

Memory banks

- 256 many 8 bit words
- so 8-bit addresses and 8-bit contents

used to store the stack, object code, and (later) registers.

Registers (internal)

PC = program counter (points to the current instruction)

SP = stack pointer (points to the top of the stack + 1)

Arithmetic

- mod 256 many 8 bit words
- So 255+1 = 0. (IMPORTANT!!!!)

Our target VM, 2

instructions

- Halt
- Push n
- Pop
- Add
- Sub
- Mult

What do they do?

Define a transition system given by a small-step operational semantics:

$$(pc, sp, stk) \Rightarrow (pc', sp', stk')$$

where:

obj	=	the object code	(≈ an array)
stk	=	the stack	(≈ an array)
pc	=	the program counter	(≈ an index into obj)
sp	=	the stack pointer	(≈ an index into stk)

Rule format

$$\textit{name:}\ \frac{\ldots premises \ldots}{obj \vdash (pc, sp, stk) \Rightarrow (pc', sp', stk')}\ \text{(side conditions)}$$

Our target VM, 3

Push: $$\frac{}{obj \vdash (pc, sp, stk) \Rightarrow (pc+2, sp+1, stk[sp \mapsto n])} \; (*)$$

Pop: $$\frac{}{obj \vdash (pc, sp, stk) \Rightarrow (pc+1, sp-1, stk)} \; (\dagger)$$

Add: $$\frac{}{obj \vdash (pc, sp, stk) \Rightarrow (pc+1, sp-1, stk[(sp-2) \mapsto n])} \; (\S)$$

$$\vdots$$

$(*)$ $obj[pc] = push$ and $obj[pc+1] = n$

$(\dagger)$ $obj[pc] = pop$

$(\S)$ $obj[pc] = add$ and $n = stk[sp-2] + stk[sp-1]$

Notes

- Since pointer arithmetic is mod 256, underflow and overflow are wrap-arounds.
- Push is a two byte instruction, all others are one byte.

Evaluation/Compilation rules for Aexp

Num:	$\dfrac{}{n \rightarrow_a v}$	$(\mathcal{N}[\![n]\!] = v)$	BSS rule
Num$_{trans}$:	$\dfrac{}{n \rightarrow [Push\ v]}$	$(\mathcal{N}[\![n]\!] = v)$	**compilation rule**
Plus:	$\dfrac{a_1 \rightarrow_a v_1 \qquad a_2 \rightarrow_a v_2}{(a_1 + a_2) \rightarrow_a v}$	$(v = v_1 + v_2)$	BSS rule
Plus$_{trans}$:	$\dfrac{a_1 \rightarrow I_1 \qquad a_2 \rightarrow I_2}{(a_1 + a_2) \rightarrow I_1 +\!\!+ I_2 +\!\!+ [Add]}$		**compilation rule**
	$\vdots$		

Haskell implementation in `vm0.hs`.

Questions

- Is the translation well-behaved?
  (E.g., In what condition does each expression leave the stack?)
- Is the translation correct?
  (No, we could easily overflow the stack.)
  (Yes, *provided* we stay within size bounds. How to prove this?)

Proposition

Suppose

- a is an *Aexp* expression
- I_a is the sequence of instructions the compiler generates for a
- I_a is loaded into code bank from address ℓ_0 to address ℓ_1.

Then $(\ell_0, sp, stk) \Rightarrow^* (\ell_1 + 1, sp + 1, stk[sp \mapsto v])$, where $a \rightarrow_a v$, *provided* there is no stack overflow.

Proof: By structural induction on a.