Assignment 1

J. Dunfield

due: Friday, 2020–01–24

You may not work in a group on this assignment.

Name or ID number: Estimated time spent: (does not affect your mark)

1 Grammars

Suppose we have the following grammar.

variables x, y, z
processes p ::= new x as p
| stop
| read x
| write x to y
| p ; p

Part (a). According to the above grammar, which of the following strings can be produced from the nonterminal (meta-variable) p? (List their numbers, or put a checkmark next to those that can be produced.)

1. stop 3
2. stop
3. new z as read z
4. new x, y as stop
5. new z as read y
6. new z as new z as stop
7. write y to x ; stop ; read z

Part (b). Below, I have drawn one of two possible parse trees for the process

read z ; stop ; read x

Draw the other possible parse tree.

; /
| / \
| / \
read ;
| /
| /
z / \stop read
| |
x
§1 Grammars

Part (c). Lisp-style syntax, like (+ 1 (+ 2 3)), eliminates the possibility of multiple parse trees for the same string. Rewrite the above grammar of \( p \) to use Lisp-style syntax: every production should look like \((\text{word} \cdots)\), where \text{word} is unique to that production (e.g. don't write two productions that both start with \text{read}) and “\cdots” contains only meta-variables \((x, y, z, p)\).

I have given you the first production; for the last production, my intent was that \( p ; p \) represents a sequence of processes, so the word \text{seq} would be a reasonable choice.

\[
\begin{align*}
\text{variables} & \quad x, y, z \\
\text{processes} & \quad p ::= (\text{new} \ x \ p) \\
& \quad | \\
& \quad | \\
& \quad | \\
& \quad |
\end{align*}
\]

2 Grammars as inductive definitions

Following the example at the beginning of lec2 §4 for arithmetic expressions, write an inductive definition that corresponds to the original grammar for processes (not your grammar with Lisp-like syntax). I have given you the first part.

(a) If \( x \) is a variable and and \( p \) is a process, then \text{new} \( x \) as \( p \) is a process.

(b)

(c)

(d)

(e)

Now, for the Lisp-style grammar, write the part of the inductive definition that corresponds to the first production, \((\text{new} \ x \ p)\).

(a)
Grammars as inductive definitions

"In logic, there are no morals."

Recognizing when things are wrong is just as important as recognizing when things are right.

We define a judgment, $e \Downarrow_0 n$, that resembles our big-step semantics but “likes zero”.

$\begin{align*}
\Downarrow_0 &\quad \text{expression } e \text{ evaluates to } n, \text{ but likes zero} \\
\Downarrow_0 n &\quad \text{evalzero-const} \\
\Downarrow_0 e_1 n_1 &\quad \Downarrow_0 e_2 n_2 \\
\Downarrow_0 (+ e_1 e_2) n_1 + n_2 &\quad \text{evalzero-add}
\end{align*}$

Prove the following conjecture. You may choose whether to induct on the expression or the derivation, and whether to do case analysis on the expression or the derivation; for this proof, either approach should work.

Write out every step in detail. Use additional pages if necessary.

**Conjecture 3.1 (Always zero).**

For all expressions $e$ and derivations $D$ such that $D$ derives $e \Downarrow_0 n$, it is the case that $n = 0$.

**Proof.** By structural induction on

**Induction hypothesis:**

Consider cases of

- Case
Now consider yet another judgment, which also likes zero: to evaluate an integer expression, it must be zero.

\[ e \downarrow 0 \vdash v \] expression \( e \) evaluates to value \( v \), but only works for zero

\[ \text{zeroevalzero-const} \]

\[ 0 \downarrow 0 \vdash 0 \]

\[ \text{zeroevalzero-add} \]

\[ (+ e_1 e_2) \downarrow 0 \vdash n_1 + n_2 \]

We now have three different definitions of big-step evaluation: the “real” definition \( e \downarrow v \), and two “joke” definitions, \( e \downarrow 0 \vdash v \) and \( e \downarrow 0 \vdash v \). Let’s compare these definitions in the framework of soundness and completeness. Since we have (I hope) some faith in the validity of \( e \downarrow v \), we will consider that to be our “ground truth”, and compare the joke definitions with respect to \( e \downarrow v \).

\[ e \downarrow v \] expression \( e \) evaluates to value \( v \)

\[ \text{eval-const} \]

\[ n \downarrow n \]

\[ \text{eval-add} \]

\[ (+ e_1 e_2) \downarrow n_1 + n_2 \]

(a) Show that the theory \( e \downarrow 0 \vdash v \) is not sound with respect to \( e \downarrow v \).

(b) Show that the theory \( e \downarrow 0 \vdash v \) is not complete with respect to \( e \downarrow v \).

(c) Is \( e \downarrow 0 \vdash v \) sound with respect to \( e \downarrow v \)? To approach this question, you may want to spend a little time looking for a counterexample—if you find one, you’re done; if not, doing so should help you understand how the \( 0 \downarrow \) rules work. If you don’t find a counterexample, state and prove that \( e \downarrow 0 \vdash v \) is sound with respect to \( e \downarrow v \). There’s room on the next page.

(d) Show that \( e \downarrow 0 \vdash v \) is not complete with respect to \( e \downarrow v \).

Your answers should give you a picture of the soundness/completeness connections (if any) that our two joke systems have to \( e \downarrow v \). Now let’s consider their relationships with each other. (Yes, this is a very long question.)

(e) Is \( e \downarrow 0 \vdash v \) sound with respect to \( e \downarrow 0 \vdash v \)? (Note: not with respect to \( e \downarrow v \)!) Start by following the approach of part (c), but if you can’t find a counterexample, explain in 1 or 2 sentences whether you think it is sound (or unsound) and why you think it is sound (or unsound).
Finally, consider the following rule, which is exactly the same as zerovalzero-const, except that it derives the “good” big-step judgment rather than the joke judgment:

\[
0 \Downarrow 0 \quad \text{proposed-eval-const-zero}
\]

(f) Is proposed-eval-const-zero an *admissible* rule? That is, if proposed-eval-const-zero can derive a judgment, could we also derive that judgment using our two existing rules for \(\Downarrow\)? Briefly explain.

**Conjecture 3.2** (Proof of soundness for part (c)—if you don’t find a counterexample).

*For all*

*such that*

*it is the case that*

**Proof.** By structural induction on 
Consider cases of