1 Grammars

Suppose we have the following grammar.

\[
\begin{align*}
\text{integers} & \quad n \\
\text{variables} & \quad x, y, z \\
\text{statements} & \quad s ::= \text{new } x \text{ init } n \text{ in } s \\
& \quad | \text{print } x \\
& \quad | \text{set } x \text{ to } n \\
& \quad | s ; s
\end{align*}
\]

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

1. print 3
2. print x
3. print z
4. 4
5. new z init 3 in print z
6. new z init 3 in print z
7. new z init 3 in print y
8. new y, z init 3, 4 in print y
9. set y to 5 ; print z ; print z

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

\[
\text{print } x ; \text{print } y ; \text{print } z
\]

Draw the other possible parse tree.

```
;  \\
/ \  \\
/  \  \\
print  ;  \\
|   / \  \\
x   / \  \\
print print \\
|   |  \\
y   z
```
**§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 $s$ to use Lisp-style syntax: every production should look like (word ...), where word is unique to that production (e.g. don’t write two productions that both start with `set`) and ... contains only meta-variables ($n$, $x$, $s$).

I have given you the first production; for the last production, my intent was that $s; s$ represents a sequence of statements, so the word `seq` would be a reasonable choice.

```
  integers    n
  variables  x, y, z
  statements
  s ::= (new x n s)
```

**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 statements (not your grammar with Lisp-like syntax). I have given you the first part.

(a) If $x$ is a variable and $n$ is an integer and $s$ is a statement, then `new x init n in s` is a statement.

(b) 

(c) 

(d) 

Now, for the Lisp-style grammar, write the part of the inductive definition that corresponds to the first production, `(new x n s)`.

(a)
“In logic, there are no morals.”

Since recognizing when things are wrong is just as important as recognizing when things are right, let’s consider some “joke” semantics. We define a judgment, \( e \Downarrow_0 v \), that resembles our big-step semantics but “likes zero”.

\[
\begin{align*}
x \Downarrow_0 v : & \text{expression } e \text{ evaluates to value } v, \text{ but likes zero} \\
\frac{\text{evalzero-const}}{n \Downarrow_0 0} & \quad \frac{e_1 \Downarrow_0 n_1 \quad e_2 \Downarrow_0 n_2}{(e_1 + e_2) \Downarrow_0 n_1 + n_2} \quad \text{evalzero-add}
\end{align*}
\]

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 \( e \Downarrow_0 v \), it is the case that \( v = 0 \).*

**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.

\[ e \downarrow_{0} v \text{ expression } e \text{ evaluates to value } v, \text{ but only works for zero} \]

\[ 0 \downarrow_{0} 0 \text{ zeroevalzero-const} \]
\[ e_1 \downarrow_{0} n_1 \quad e_2 \downarrow_{0} n_2 \text{ zeroevalzero-add} \]

We now have three different definitions of big-step evaluation: the “real” definition \( e \downarrow v \), and two “joke” definitions, \( e \downarrow_{0} v \) and \( e \downarrow_{0} 0 \). 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 \( e \downarrow v \).

\[ e \downarrow v \text{ expression } e \text{ evaluates to value } v \]
\[ n \downarrow n \text{ eval-const} \]
\[ e_1 \downarrow n_1 \quad e_2 \downarrow n_2 \text{ eval-add} \]

(a) The theory \( e \downarrow_{0} v \) is not sound with respect to \( e \downarrow v \). Give a counterexample:

\[ e = \quad v = \]

(b) The theory \( e \downarrow_{0} v \) is not complete with respect to \( e \downarrow v \). Again, give a counterexample.

\[ e = \quad v = \]

(c) Is \( e \downarrow_{0} v \) 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 \( 0 \downarrow_{0} \) rules work. If you don’t find a counterexample, state and prove that \( e \downarrow_{0} 0 \) is sound with respect to \( e \downarrow v \)! There’s room on the next page.

(d) The theory \( e \downarrow_{0} v \) is not complete with respect to \( e \downarrow v \). Give a counterexample.

\[ e = \quad v = \]

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 \( e \downarrow_{0} v \) sound \textit{with respect to} \( e \downarrow_{0} v \)? \textbf{(Note: not with respect to} \( e \downarrow v \text{!)} \) 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 \textit{why} you think it is sound (or unsound).

(f) Is \( e \downarrow_{0} v \) complete \textit{with respect to} \( e \downarrow_{0} v \)? Give a counterexample.

(g) After completing part (f), argue \textit{in one sentence} why \( e \downarrow_{0} v \) is not sound \textit{with respect to} \( e \downarrow_{0} v \).
§3 “In logic, there are no morals.”

Finally, consider the following rule, which is exactly the same as zeroevalzero-const, except that it derives the “good” big-step judgment rather than the joke judgment:

\[
\begin{array}{c}
\hline
0 \Downarrow 0 \\
\hline
\end{array}
\]

proposed-eval-const-zero

(h) 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