An Application of Stochastic Context Sensitive Grammar Induction to Transfer Learning: Appendix

Eray Özkural

May 26, 2014

Appendix

Derivation Lattice

A derivation lattice that shows the derivation of the expression `(sqr (sqr x)` using a subset of a stochastic grammar for Scheme is provided in Fig. 1. We do not demonstrate the entire grammar here because it is too long for a single example. Instead, we refer to a small fragment which is appropriate for display. Here, we show how how a context-sensitive grammar can encode type information, whereas the grammar can distinguish between number variables and string variables. The derivation we are going to demonstrate is the following:

$$(define(sqrx) < body >) \Rightarrow^* (define(sqrx)(*xx)), \tag{1}$$

which we reduce to the derivation of a single non-terminal body:

$$< body >\Rightarrow^* (define(sqrx)(*xx)). \tag{2}$$

Sample Stochastic Grammar Fragment for Scheme

Following are rules for the stochastic grammar fragment for the example in Scheme. The syntax of the rules follow the typical Baus-Naur Form, where non-terminals are written as a-nonterminal, and terminals are written as a-terminal. A variable* denotes that the non-terminal is repeated zero or more times, while variable$^+$ denotes that the non-terminal is repeated one or more times. The only change is that, under each rule, the probability of the rule is written in addition. This grammar fragment is a subset of the Scheme grammar that we use in our prototype system. It is only given for demonstrating how the derivation lattice may be used to derive a sentence from a given stochastic grammar.

Note that context-sensitive productions have been added for the sake of demonstration of useful derivation compression.

$$\text{body} \rightarrow_{1.0} \text{definition}^*\text{sequence}$$
$$\text{sequence} \rightarrow_{1.0} \text{command}^*\text{expression}$$
$$\text{command} \rightarrow_{1.0} \text{command}^*\text{expression}$$

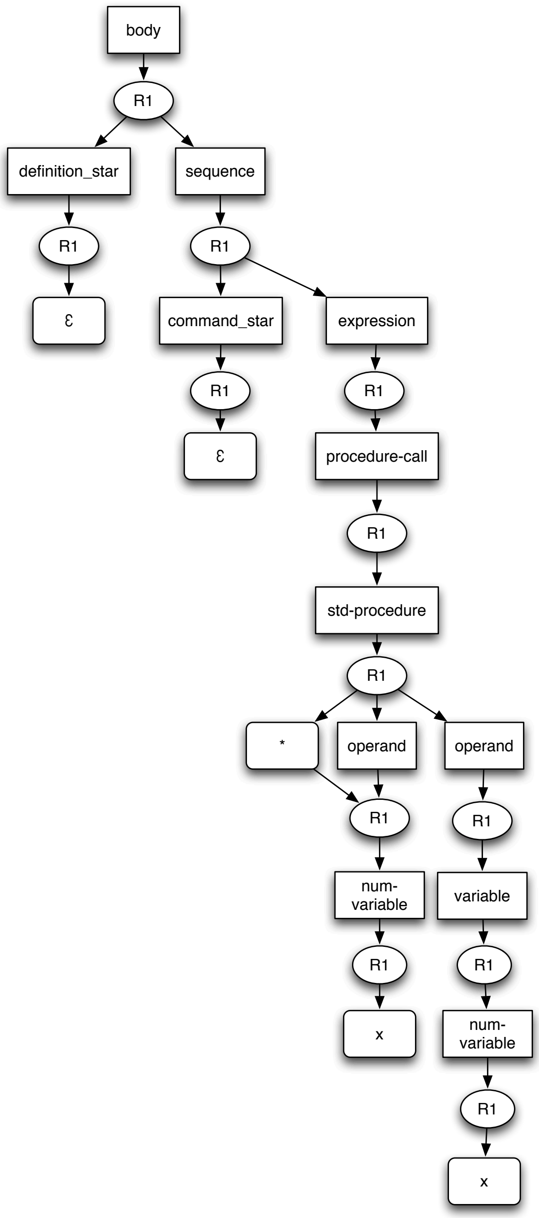

Figure 1: A sample derivation lattice

procedure-call $\rightarrow_{0.2}$ (operator operand*)
procedure-call $\rightarrow_{0.4}$ std-procedure
procedure-call $\rightarrow_{0.4}$ previous-solution

std-procedure $\rightarrow_{0.2}$ * operand$^+$
std-procedure $\rightarrow_{0.2}$ + operand$^+$
std-procedure $\rightarrow_{0.1}$ - operand$^+$
std-procedure $\rightarrow_{0.1}$ / operand$^+$
std-procedure $\rightarrow_{0.1}$ string? str-operand
std-procedure $\rightarrow_{0.1}$ string? make-string
std-procedure $\rightarrow_{0.1}$ string-length str-operand$^+$
std-procedure $\rightarrow_{0.1}$ string-append str-operand$^+$

operand $\rightarrow_{1.0}$ expression
* operand $\rightarrow_{0.9}$ num-operand
* operand $\rightarrow_{0.1}$ 2

expression $\rightarrow_{0.2}$ variable
expression $\rightarrow_{0.1}$ literal
expression $\rightarrow_{0.1}$ procedure-call
expression $\rightarrow_{0.1}$ lambda-expression
expression $\rightarrow_{0.1}$ conditional
expression $\rightarrow_{0.1}$ assignment
expression $\rightarrow_{0.1}$ derived-expression
expression $\rightarrow_{0.1}$ abstract-expression
expression $\rightarrow_{0.1}$ frequent-expression

variable $\rightarrow_{0.5}$ num-variable
variable $\rightarrow_{0.5}$ str-variable
num-variable $\rightarrow_{0.6}$ x
num-variable $\rightarrow_{0.4}$ y
str-variable $\rightarrow_{0.6}$ s
str-variable $\rightarrow_{0.4}$ w