Final Project Checkpoint
Formalized Algebra from Text

Adam Wiemerslage and Rehan Ahmed

April 13, 2018

1 Abstract

In this project, we are proposing to solve algebraic word problems in a way similar to Kushman et al [cite Learning to Automatically Solve Algebra Word Problems]. However, rather than relying on a rule based approach [look at above paper for citations to rule based approaches], or a mapping function to a set of templates (as in Kushman et al), we are proposing a slightly different method. In this method, we will learn a mapping from the input sequence of words, to a set of triples of potential expressions, via a dependency parse of the text. Then, rather than mapping each entity or verb in the text to a template, we will map each element in the triple to a type in our language, based on a probabilistic model.

Finally, our programming language will be implemented to solve the expressions provided by our NLP model, based on a simple operational semantics we have created for solving such problems. The target language will be the simple **IMP-1** language.

2 Example

Example 1. *Pooja has 3 apples. She eats one apple. How many apples does Pooja have now?*

We attempt to translate the sample problem's text into the following commands:

$$\begin{array}{lcc} \textit{Pooja has 3 apples} & -> & a_p := 3 \\ \textit{She eats one apple} & -> & a_p := a_p - 1 \\ \textit{How many apples does Pooja have now?} & -> & \texttt{print}\, a_p \end{array}$$

And since these are sequence of steps we end up with:

$$a_p := 3; a_p := a_p - 1; \texttt{print}\, a_p$$

More formally:

$$\texttt{seq}(\texttt{set}[a_p](3); \texttt{seq}(\texttt{set}[a_p](\texttt{minus}(a_p; 1)); \texttt{print}(a_p)))$$

$\boxed{e \ \mathsf{val}}$

3 Language: IMP-1

$\boxed{e : \tau}$

$$\frac{}{\texttt{addr}[a] : \mathsf{float}} \qquad \frac{}{\mathsf{float}[\mathsf{f}] : \mathsf{float}} \qquad \frac{e_1 : \mathsf{float} \qquad e_2 : \mathsf{float}}{\texttt{plus}(e_1; e_2) : \mathsf{float}} \qquad \frac{e_1 : \mathsf{float}\ e_2 : \mathsf{float}}{\texttt{minus}(e_1; e_2) : \mathsf{float}}$$

$$\frac{e_1 : \mathsf{float}\ e_2 : \mathsf{float}}{\texttt{times}(e_1; e_2) : \mathsf{float}} \qquad \frac{e_1 : \mathsf{float}\ e_2 : \mathsf{float}}{\texttt{divide}(e_1; e_2) : \mathsf{float}}$$

$\boxed{c : ok}$

$$\frac{e : \mathsf{float}}{\texttt{set}[a](e)\ \mathsf{ok}} \qquad \frac{}{\texttt{skip}\ \mathsf{ok}} \qquad \frac{c_1\ \mathsf{ok} \quad c_2\ \mathsf{ok}}{\texttt{seq}(c_1; c_2)\ \mathsf{ok}} \qquad \frac{e : \mathsf{float}}{\texttt{print}(e)\ \mathsf{ok}}$$

Syntax chart:

Typ	τ	::=	`num`	`num`	numbers
			`bool`	`bool`	booleans
Exp	e	::=	`addr`$[a]$	a	addresses (or “assignables”)
			`num`$[n]$	n	numeral
			`bool`$[b]$	b	boolean
			`plus`$(e_1;e_2)$	$e_1 + e_2$	addition
			`times`$(e_1;e_2)$	$e_1 * e_2$	multiplication
			`eq`$(e_1;e_2)$	$e_1 == e_2$	equal
			`le`$(e_1;e_2)$	$e_1 <= e_2$	less-than-or-equal
			`not`(e_1)	$!e_1$	negation
			`and`$(e_1;e_2)$	$e_1\ \&\&\ e_2$	conjunction
			`or`$(e_1;e_2)$	$e_1 \mid\mid e_2$	disjunction
Cmd	c	::=	`set`$[a](e)$	$a := e$	assignment
			`skip`	`skip`	skip
			`seq`$(c_1;c_2)$	$c_1; c_2$	sequencing
			`if`$(e;c_1;c_2)$	`if` e `then` c_1 `else` c_2	conditional
			`while`$(e;c_1)$	`while` e `do` c_1	looping
			`print`(e)	`print` e	printing
Addr	a				