Arithmetic Project for Soar

John E. Laird

3/1/2012

This program supports arithmetic and subtraction between two multi-digit numbers. It formulates the problem in multiple columns. It does not use any math functions and has a table of all single digit addition facts (for addition and one subtraction strategy) and tables of simple subtraction facts and addition by ten to single digits (for a second subtraction strategy). These facts can be converted to a semantic memory access (in the application of computer-result).

Each primitive operator is relatively simple; without complex proposal conditions, control rules, lots of control flags or complex conditional operator applications. The actual execution trace is sometimes a bit tricky – especially for subtraction.

The project supports the automatic generation of random 3 column addition and subtraction problems which are created in generate-problem. The project will execute N of these (set as the value of ^count in initialize-arithmetic).

The project checks that all answers are computed correctly by using Soar's math functions (computed in elaborations/verify and finish-problem) if an incorrect answer is computed, it is printed out and Soar halts (but this should never happen).

The two subtraction strategies differ in what initial facts they assume. One of the subtraction strategies assumes the same knowledge as addition (the sum of two single digit numbers and the resulting carry), but involves remapping that knowledge so that it is appropriate for subtraction. For example it knows that if 7 is subtracted from 6 that the answer is 9 and there must be a borrow from the column to the left.

The second subtraction strategy assumes that the system knows how to subtract any single digit (0-9) from the numbers 0-18, and that it has facts to add ten to any single digit (0-9).

The actual trace of a strategy arises from the available operator applications and impasses that arise. For example, in the second strategy, if a larger number is being subtracted from a smaller number, there is an operator no-change impasse because no fact is available for that situation. This is the standard American approach to subtraction. The key rules for this are in process-column/compute-result.soar

The only differences between the two strategies are the available facts and a single rule in process-column that applies the process-column operator by accessing the facts (process-column\*apply\*compute-result\*subtraction). There are rules that only are used by the second strategy (in the compute-result substate), but there is no explicit control to invoke them and they do not have to be disabled during addition or the other subtraction strategy.

Works with chunking (learn --on).

Key data structures:

arithmetic

add10-facts - all facts for adding 10 to 0-9

digit1 - 0-9

digit-10 - digit + 10

facts - all of the facts about single digit arithmetic

digit1

digit2

sum - 0-9 - the single digit result

carry-borrow - 0/1 if the result is 10 or greater

operation addition/subtraction

subtraction-facts - all facts for subtracting a digit from 0-18

same structure as facts above

arithmetic-problem - holds the complete definition of the problem

one-column - the right-most columns where the ones are held

linked-list to rest of columns

column t - used to test if column exists - makes chunking happy

digit1 0-9

digit2 0-9

carry-borrow - 0/1 - based on the computation on the prior column

next-column - the column to the left of the current - 10x

(nil if no next column)

result - the result of the digits and carry-borrow

count - number of problems to solve

digits - all digits 0-9

All of the operators in this system:

Initialize-arithmetic

Names the problem (^name arithmetic)

Creates the digits 0-9 that are used in generating problems

Initialize the count for the number of problems to solve

Can also define a specific problem to solve (example rule commented out)

If specific problem defined, it will be solved <count> number of times

Generate-facts

Preloads working memory with all of the arithmetic facts (should not be

necessary with semantic memory)

Generate-problem

Creates the arithmetic problem (<s> ^arithmetic-problem)

Generates individual digits, the operation, column by column.

Right now it only does addition problems

Process-column - compute the result for a column

get-digit1 - retrieve digit1 from column and move it onto state

if there is a carry-borrow, recursively add/subtract it to column digit1

to compute final digit1

write-digit1 - return the newly computed digit1 and possible

carry-borrow(if digit1 is 9 for + or 0 for -)

get-digit2 - retrieve digit2 and move it onto the state

compute-result - compute result and carry-borrow from digit1 and digit2 by

using the facts - will replace with semantic memory lookup

carry-borrow - transfer carry-borrow to next column

new-column - creates a new column if there is a carry-borrow at the

left-most column for an addition problem

write-result - move result to the current-column

Next-column - when a result has been computed for a column, go to the next column

Finish-problem - when there is a result for a column with no next-column

(nil), print out result, decrement count

Stop-arithmetic - if count =0 the halt