8-2003

An Evaluation of the Run-Time Performance of the Model-Tracing Algorithm of Two Different Production Systems: JESS and TDK

Sanket Choksey
Worcester Polytechnic Institute, sanket@cs.wpi.edu

Neil T. Heffernan iii
Worcester Polytechnic Institute, nth@wpi.edu

Technical Report: Worcester Polytechnic Institute < WPI-CS-TR-03-31 >

An Evaluation of the Run-Time Performance of the Model-Tracing Algorithm of Two Different Production Systems: JESS and TDK

Sanket Choksey, Neil Heffernan
{sanket, nth}@cs.wpi.edu
Computer Science Department
Worcester Polytechnic Institute, Worcester, MA, USA

Abstract

The Cognitive Tutor Authoring Tools are intended to make Tutor development easier and faster. The model-tracing algorithm is used in the Authoring Tools to trace the student actions and provide appropriate feedback to the student. In this paper we describe our implementation of the model-tracing algorithm in JESS[1] and also present an experimental evaluation of the run time performance of the model-tracing algorithm in two different production rule systems, JESS and TDK[2]. According to our study the implementation of the model-tracing algorithm in JESS is slower than that in TDK but is acceptable for most of the tutoring purposes. Also, the implementation of the model-tracing algorithm using JESS's native backward chaining outperforms that in TDK.

Introduction

The Cognitive Tutor Authoring Tools (CTAT) [Koedinger, K. R., Aleven, V., Heffernan, N. T., 2003], are intended to drastically reduce the tutor development time. CTAT uses a production system to create and encode a cognitive model of the student. A production system consists of a set of rules, a set of facts (also called working memory elements in the ACT-R language) the rules operate on, a control strategy and a rule applier. The rules determine the actions to be performed when a set of conditions are met. The facts constitute the knowledge base. The control strategy determines the order in which the rules will be compared and also determines which rule to fire in case multiple rules can be fired. The student model, which is based on the ACT-R theory, is encoded as a set of production rules and the working memory elements or facts. CTAT consists of the following tools [Figure 1]:

1. Interface builder: Used for rapid prototyping of the Tutor interface.
2. Behavior recorder: Records the solution paths in the problem scenario as the author demonstrates them.
3. Working memory editor: A visual interface for editing the working memory elements.
4. Production rule editor: Used for writing the production rules for the problem and creating a cognitive model for the problem.
5. Cognitive Model visualizer: A debugging tool.

Description of the Model Tracing algorithm

Model tracing algorithm as developed by Anderson & Pellietier (1991), Pelletier (1993) [And described in Koedinger, Anderson, Hadley & Mark (1997)] is a plan recognition technique that interprets the student behavior by comparing the student actions with the student model and provides appropriate feedback and advice to the student when necessary. This model-tracing algorithm has been used in several different tutoring projects including the Cognitive Algebra Tutor (Koedinger, Anderson, Hadley & Mark, 1997, now sold by www.CarnegieLearing.com), the LISP Tutor (Corbett & Anderson, 1995) and the Ms. Lindquist Tutor (Heffernan & Koedinger, 2001) among many others.

In model tracing, the cognitive model is used to interpret each student action and follow the student's step-by-step path through the problem space. The primary goal in this process is to provide whatever guidance is needed for the student to reach a successful conclusion to problem solving.

The model-tracing algorithm is given three inputs:

1. The state of working memory: represented by a group of working memory elements.

[1] – JESS – Java Expert system shell developed by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA, USA.
[2] – TDK – Tutor Development Kit developed by Ray Pelletier at Carnegie Mellon University, PA, USA.

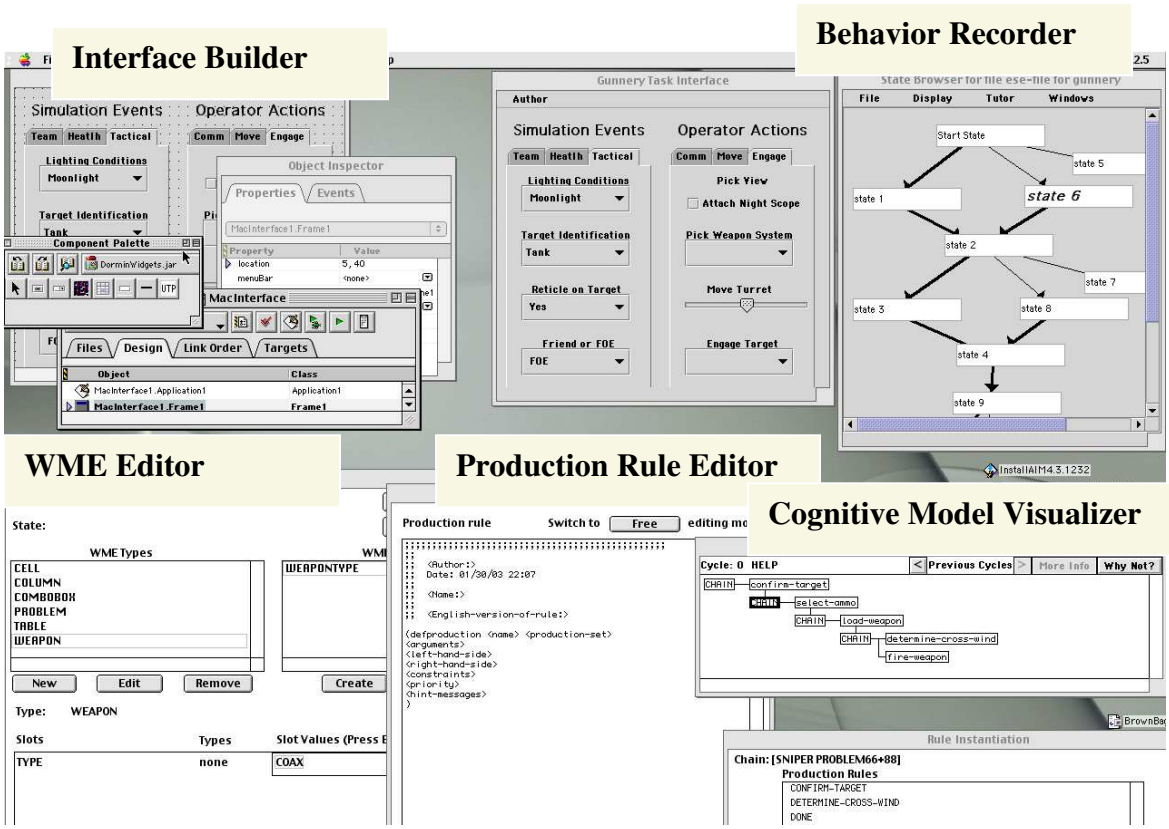

Figure 1: The prototype Cognitive Tutor Authoring Tools

2. A set of productions. Each production represents a cognitive step, which may, or may not, have observable actions.
3. The student's input that we wish to "trace".

The *student model* consists of the working memory elements and the production rules. If the student input can be traced, the algorithms output is a set of interpretations. Each interpretation is a list of productions that are chained together. An interpretation can be though of as the list of mental steps that the student just performed to generate the given student input. Each list of productions represents a different set of steps that could have resulted in the student's action.

TDK

Tutor development kit (TDK) uses Tertl (Anderson & Pellietier, 1991, Pelletier, 1993), which is a production rule system that is written in LISP and is optimized for building cognitive tutors. Pellietier (1993) choose not to use the Rete (Forgy, C. L., 1982) pattern-matching algorithm because he said it had the following drawback:

- In order to reduce the number of comparisons that should be made during rule testing Rete compiles a data structure for the rule base and stores the partial instantiation for the rules. Hence the space usage can increase exponentially over time.

Other production systems do not allow explicit control over the order in which the rules should be compared and fired. Tertl addresses these weaknesses by restricting the expressiveness of the production rule conditions and allowing parameter passing in the rules. Also in the production rules of Tertl, in order for a working memory element to be in a production, it must either have been passed in as a parameter, or referenced already earlier in the production. (i.e., a variable must be bound to the working memory element before it can be used). Thus the bound variable uniquely identifies the working memory element to be tested. This reduces the number of working memory elements that need to be tested in order to check whether the condition on the rule LHS is satisfied or not. Without parameter passing all the instances of that particular wme type have to be tested for a

match. Hence parameter passing increases the efficiency of the Tertl production system considerably.

The need for a different production system

The tutors developed using TDK are hard to deploy on the web and are not portable. Hence we have ported the system to support a more common production system called JESS (Java Expert System Shell), which is based on the Clips production rule system. The tutors thus developed using JESS as the production system can be easily deployed on the web and are also portable across multiple platforms.

The implementation of Model Tracing in JESS

Model tracing can be implemented in JESS in two ways: (a) Using forward chaining or (b) Using native backward chaining in JESS. We first implemented (a) and then later tested both (a) and (b).

(a) Implementation of model tracing algorithm in JESS using forward chaining:

A special wme called selection-action-input has to be created in the working memory. It has three slots: selection – defines the current wme, action – defines the action performed on the current wme and input – the current input. The selection, action and input should be set on the RHS of the rule. The model tracing algorithm starts from the current working memory and fires one rule at a time from the list of activated rules and compares the current selection, action, input with that of the students selection, action, input. If a match is found then the search is terminated or else the working memory is restored back to the previous state and the next possible rule is fired and the search continues. If the student input cannot be traced using the rules in the rule base then one buggy rule is added to the rule base and the search is performed again. Each time one new buggy rule is added and the search is repeated until the student action is traced or 4 buggy rules have been added to the rule base. When the student input is traced the algorithm returns the list of rules i.e. the steps required to model the student behavior.

Pseudo code for the algorithm is:
(i) perform an iterative deepening search until no more successors can be generated or required student selection, action and input is found
(a) For each depth d, do the following:

- Get the list of rules that can be fired for the current working memory state.
- Save current working memory
- For each rule in the list
  - Restore the working memory to the state so that the rule can be fired.
  - Fire the rule.
  - Add the current rule to the list of rules or steps.
  - Compare the selection-action-input wme with the student selection-action-input. If both match then terminate the search and return the list of rules or steps.
  - If the student selection, action and input do not match then make a recursive call to iterative deepening search with depth (d+1).

(b) Implementation of model tracing algorithm in JESS using native backward chaining:

JESS provides support for backward chaining. In order to use backward chaining in JESS, the deftemplates (wme types) have to be declared as backward chaining reactive. Also the LHS of the rules have to be modified so that they now react in a backward chaining manner. Whenever a rule asserts some wme (fact) then a (need-XXX) pattern should be placed on the LHS of that rule. JESS rules for the addition tutor using forward chaining and backward chaining are given in the appendix. A special wme called selection-action-input has to be created in the working memory for model-tracing algorithm. The selection-action-input wme has three slots (i) selection, which defines the current wme the student is working on (ii) action, which defines the action being performed by the student on the selected wme (iii) input, which defines the value entered by the student.

Pseudo code for the model-tracing algorithm using backward chaining:

(i) Assert a "need-selection-action-input" fact with the selection, action and input slot values equal to the students selection, action and input.
(ii) Fire one rule from the list of activated rules and add the rule to the rule sequence.
(iii) If the student action cannot be traced then no rules will be fired.
(iv) When the student selection, action and input are traced return the rule sequence.

Experiments

We knew the run time performance of the model-tracing algorithm would depend upon the average branching factor and depth of the goal node in the search tree. Branching factor is

the average number of rules that can be fired at any working memory state in the search tree. The depth of a goal is the number of rules that need to be chained together to generate the student's input. So we ran a series of experiments where we varied the branching factor and the depth of a solution and measured the time that model tracing took for 1) TDK, 2) forward chaining in Jess and 3) Jess's native backward chaining.

For each of the experiments we took an already existing rule set (happened to be for multi-column addition) and modified it to be able to vary the branching factor and the depth at which the goal node was reached. In order to modify the branching factor we simply duplicated rules (giving them different names), thereby causing the production system to branch on each instance. In order to create a branching factor of 2 we duplicated each rule in the rule set. Similarly to create an example with branching factor 4, we create 4 rules for each rule in the set.

In order to vary the depth, we inserted a counter on the LHS of the productions so that we could set a depth easily. Initially the counter is set to 0 and the first rule fires when the counter value is 0 and it increases the counter by 1.The second rule fires when the counter value is 1 and it sets the counter value to 2 and so on.

The following experiments were run on a Macintosh 867MHz PowerPC G4 machine running OSX operating system. The Jess versions were run within the sun JRE v1.4.

Experiment 1:
Figure 2 shows the runtime evaluation of Model-tracing in TDK:

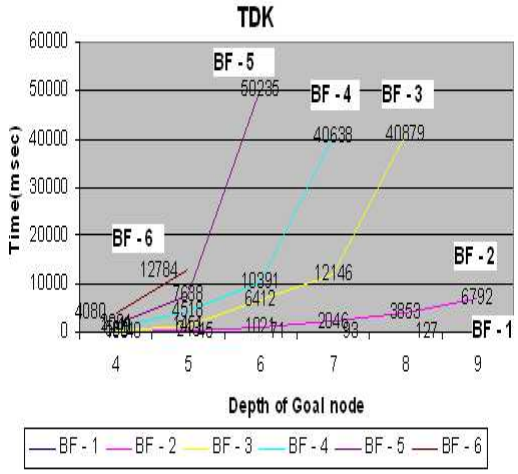

Figure 2: model-tracing using TDK

The x-axis starts at depth 4, because the base rule sets required 4 productions to be chained together. "BF" stands for branching factor. In figure 2 we see the highest point is labeled with "BF-5" and represents when the experiment was run with a branching factor of 5, and a depth of 6, it took 50.235 seconds (or 50235 milliseconds). If we follow the line from that point down and to the left we see that when the depth was 5, it took about 12 seconds (12.784 seconds). You will see that we do not report a value for BF-5 at depth 7. This is because the runtimes for the search were exponential in both the depth and the branching factor, so we could not complete many of the searches within 125 minutes. Due to this it has not been done yet. The run time of the model-tracing algorithm using TDK increases with increase in branching factor and depth of the goal node.

Experiment 2:
Figure 3 shows the run time evaluation of Model-tracing using JESS forward chaining.

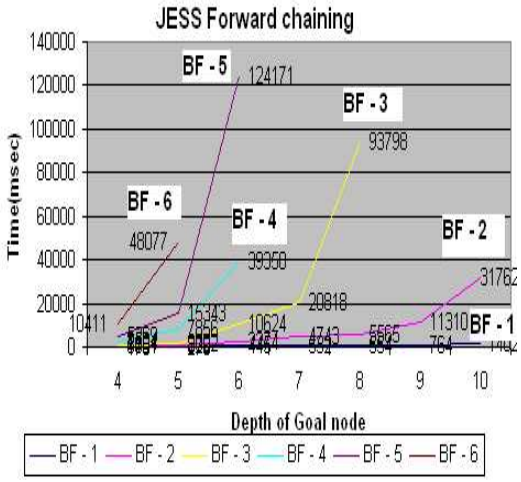

Figure 3: model-tracing using JESS forward chaining

The run time of the model-tracing algorithm increases exponentially as the branching factor and depth of the goal node increases. Hence this implementation is useful in cases where the branching factor and chain length are not large.

Experiment 3:
Run time evaluation of model-tracing using JESS backward chaining:

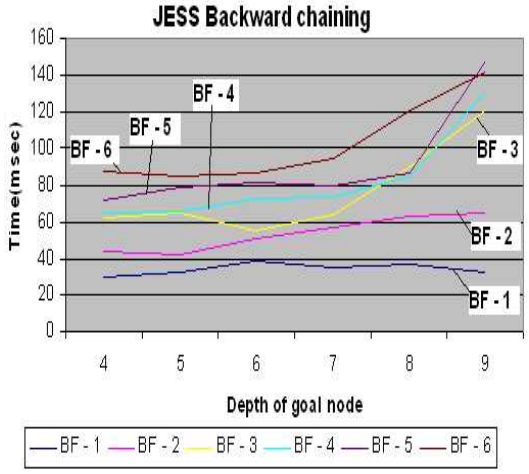

Figure 4: model-tracing using JESS backward chaining

The run time of the model-tracing algorithm using JESS backward chaining does not depend heavily on the branching factor and the depth of the goal node in the search tree. This implementation is suitable for very complex rule sets with large branching factor and depth of goal node.

Following is a graph comparing all three implementations of the model-tracing algorithm. The branching factor is fixed at 3 and the depth of the goal node is varied linearly.

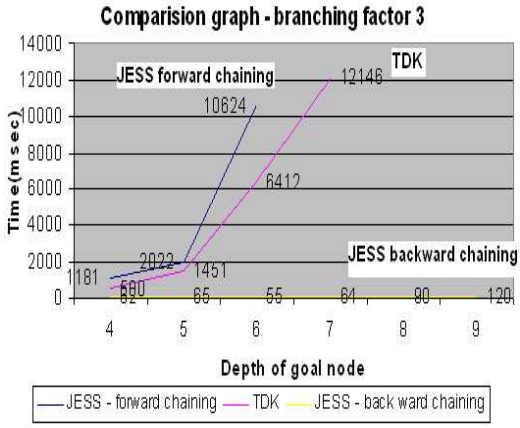

Figure 5: comparison of all three methods

TDK is faster but not by a great deal. The model-tracing algorithm using forward chaining in JESS did good enough for most purposes, but if you wanted something very complicated you would get a faster response from TDK.

Conclusions

Though the Jess implementation of the Rete pattern matching is slower, and we understand why, it is probably fast enough for many tutoring purposes. In the future we will use Jess's native backward chaining in our Authoring Tools project.

References

Ernest Friedman-Hill- *Jess in Action Java Rule-based Systems* Manning Publications Co. http://www.manning.com/friedman-hill/index.html

Forgy, C. L., 1982, Rete: A fast Algorithm for the Many Pattern/Many Object Pattern Matching Problem, Artificial Intelligence 19:17-37.

Heffernan, N. T., & Koedinger, K. R.(2002) *An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor* International Conference on Intelligent Tutoring System 2002. Biarritz, France.

Koedinger, K. R, Anderson, J. R, Hadley, W. H., Mark, A. A., 1997, *Intelligent Tutoring Goes To School in the Big City,* International Journal of Artificial Intelligence in Education (1997), 8, 30-43.

Koedinger, K. R., Aleven, V., & Heffernan, N. T., 2003, *Toward a Rapid Development Environment for Cognitive Tutor,* 12th Annual Conference on Behavior Representation in Modeling and Simulation, Simulation Interoperability Standards Organization.

Pelletier, Ray (1993) *The TDK Production Rule System.* Master Thesis. Carnegie Mellon University. (http://cs.wpi.edu/~sanket/JessAuthoringTools/papers/Production_System.pdf)

Appendix
Addition Rule set:
1. JESS forward chaining rules:

```
(deftemplate selection-action-input
        (slot selection)
        (slot action)
        (slot input))
```

These rules are based upon porting the TDK rules given below, written by Aleven and Koedinger.

Rule 1:

```
(defrule focus-on-first-column
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 0))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-02
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 0))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-1
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 1))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-12
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
```

```
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 1))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-2
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 2))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-22
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 2))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-3
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 3))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-32
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
```

```
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 3))
   =>
   (bind ?new-val (+ ?val 1))
   (modify ?counter-wme (value ?new-val))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-5
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 4))
   =>
   (bind ?current-sub-goal (assert (process-column-goal (name "proc-right-most-col-goal")
(column ?rightmost-column-name) (carry nil) (first-addend ?num1) (second-addend ?num2)
(sum nil) (description nil))))
   (modify ?current-prob (subgoals "proc-right-most-col-goal"))
   (printout t "Focus-on-first-column." crlf))

(defrule focus-on-first-column-52
   (addition (problem ?problem))
   ?current-prob <- (problem (name ?problem) (interface-elements $?_blank_mf8 ?table
$?_blank_mf9) (subgoals nil))
   (table (name ?table) (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column (name ?rightmost-column-name) (cells $?_blank_mf11 ?first-
addend ?second-addend ?result))
   (cell (name ?first-addend) (value ?num1))
   (cell (name ?second-addend) (value ?num2))
   (cell (name ?result) (value nil))
   ?counter-wme <- (counter (value ?val))
   (test (eq ?val 4))
   =>
   (bind ?current-sub-goal (assert (process-column-goal (name "proc-right-most-col-goal")
(column ?rightmost-column-name) (carry nil) (first-addend ?num1) (second-addend ?num2)
(sum nil) (description nil))))
   (modify ?current-prob (subgoals "proc-right-most-col-goal"))
   (printout t "Focus-on-first-column." crlf))

(defrule add-carry
   (addition (problem ?problem))
   (problem (name ?problem) (subgoals $?_blank_mf19 ?subgoal $?_blank_mf20))
   ?current-subgoal <- (process-column-goal (name ?subgoal) (carry ?num0) (first-addend
?num1) (second-addend ?num2) (sum ?sum))
   (test (neq ?num0 nil))
   (test (neq ?sum nil))
   =>
   (bind ?new-sum (+ ?sum 0))
   (modify ?current-subgoal (sum ?new-sum) (carry nil))
   (printout t "Add carry." crlf))

(defrule add-addends
   (addition (problem ?problem))
   (problem (name ?problem) (subgoals $?_blank_mf17 ?subgoals $?_blank_mf18))
   ?current-goal <- (process-column-goal (name ?subgoals) (column ?column) (carry ?carry)
(first-addend ?num1) (second-addend ?num2) (sum nil))
   (test (neq ?num1 nil))
   (test (neq ?num2 nil))
   =>
   (bind ?sum (+ ?num1 ?num2))
   (modify ?current-goal (sum ?sum))
   (printout t "Add addends." crlf))
```

```
(defrule add-addends-2
   (addition (problem ?problem))
   (problem (name ?problem) (subgoals $?_blank_mf17 ?subgoals $?_blank_mf18))
   ?current-goal <- (process-column-goal (name ?subgoals) (column ?column) (carry ?carry)
(first-addend ?num1) (second-addend ?num2) (sum nil))
   (test (neq ?num1 nil))
   (test (neq ?num2 nil))
   =>
   (bind ?sum (+ ?num1 ?num2))
   (modify ?current-goal (sum ?sum))
   (printout t "Add addends." crlf))

(defrule write-sum
   (addition (problem ?problem))
   ?current-problem <- (problem (name ?problem) (subgoals $?sg1 ?subgoal $?sg2))
   ?current-subgoal <- (process-column-goal (name ?subgoal) (column ?column) (carry nil)
(sum ?sum))
   (test (neq ?sum nil))
   (test (< ?sum 10))
   (column (name ?column) (cells $?_blank_mf27 ?result) (position ?pos))
   ?cell <- (cell (name ?result))
   ?selection-action-input <- (selection-action-input)
   =>
   (modify ?cell (value ?sum))
   (modify ?current-problem (subgoals $?sg1 $?sg2))
   (modify ?selection-action-input (selection ?result) (action UpdateTable) (input ?sum))
   (retract ?current-subgoal)
   (printout t "Write sum1." crlf))

(defrule write-sum-2
   (addition (problem ?problem))
   ?current-problem <- (problem (name ?problem) (subgoals $?sg1 ?subgoal $?sg2))
   ?current-subgoal <- (process-column-goal (name ?subgoal) (column ?column) (carry nil)
(sum ?sum))
   (test (neq ?sum nil))
   (test (< ?sum 10))
   (column (name ?column) (cells $?_blank_mf27 ?result) (position ?pos))
   ?cell <- (cell (name ?result))
   ?selection-action-input <- (selection-action-input)
   =>
   (modify ?cell (value ?sum))
   (modify ?current-problem (subgoals $?sg1 $?sg2))
   (modify ?selection-action-input (selection ?result) (action UpdateTable) (input ?sum))
   (retract ?current-subgoal)
   (printout t "Write sum1." crlf))

(defrule write-carry
   (addition (problem ?problem))
   ?current-problem <- (problem (name ?problem) (subgoals $?sg1 ?subgoal $?sg2))
   ?carry-goal <- (write-carry-goal (name ?subgoal) (carry ?num) (column ?column))
   (column (name ?column) (cells ?carry $?_blank_mf30) (position ?pos))
   ?carry-cell <- (cell (name ?carry) (value nil))
   ?selection-action-input <- (selection-action-input)
   =>
   (modify ?carry-cell (value ?num))
   (modify ?current-problem (subgoals ?sg1 ?sg2))
   (modify ?selection-action-input (selection ?carry) (action UpdateTable) (input ?num))
   (retract ?carry-goal)
   (printout t "Write-carry." crlf))

(defrule must-carry
   (addition (problem ?problem))
   ?current-problem <- (problem (name ?problem) (subgoals $?_blank_mf21 ?subgoal
$?_blank_mf22))
   ?current-subgoal <- (process-column-goal (name ?subgoal) (column ?column) (carry nil)
(sum ?sum))
   (test (neq sum nil))
   (test (numberp ?sum))
   (test (> ?sum 9))
```

```
   (problem (name ?problem) (interface-elements $?_blank_mf23 ?table $?_blank_mf24)
(subgoals $?subgoals))
   (table (name ?table) (columns $?_blank_mf25 ?next-column-name ?column $?_blank_mf26))
   ?next-column <- (column (name ?next-column-name) (position ?next-pos))
   =>
   (bind ?new-sum (- ?sum 10))
   (modify ?current-subgoal (sum ?new-sum))
   (assert (write-carry-goal (name "write-carry") (carry 1) (column ?next-column-name)
(description nil)))
   (modify ?current-problem (subgoals "write-carry" ?subgoals))
   (printout t "Must carry1" crlf))

(defrule must-carry-2
   (addition (problem ?problem))
   ?current-problem <- (problem (name ?problem) (subgoals $?_blank_mf21 ?subgoal
$?_blank_mf22))
   ?current-subgoal <- (process-column-goal (name ?subgoal) (column ?column) (carry nil)
(sum ?sum))
   (test (neq sum nil))
   (test (numberp ?sum))
   (test (> ?sum 9))
   (problem (name ?problem) (interface-elements $?_blank_mf23 ?table $?_blank_mf24)
(subgoals $?subgoals))
   (table (name ?table) (columns $?_blank_mf25 ?next-column-name ?column $?_blank_mf26))
   ?next-column <- (column (name ?next-column-name) (position ?next-pos))
   =>
   (bind ?new-sum (- ?sum 10))
   (modify ?current-subgoal (sum ?new-sum))
   (assert (write-carry-goal (name "write-carry") (carry 1) (column ?next-column-name)
(description nil)))
   (modify ?current-problem (subgoals "write-carry" ?subgoals))
   (printout t "Must carry1" crlf))
```

JESS backward chaining rules:

Declaring the wme types as backward chaining reactive.

```
(do-backward-chaining problem)
(do-backward-chaining selection-action-input)
(do-backward-chaining write-carry-goal)
(do-backward-chaining process-column-goal)
```

Rules:

Rule 1:

```
(defrule focus-on-first-column
   (need-problem
        (subgoals ?ss&:(neq ?ss nil)))
   (need-process-column-goal
        (name ?n&:(neq ?n nil)))
   (addition
        (problem ?problem))
   ?current-prob <- (problem
        (name ?problem)
        (interface-elements $?_blank_mf8 ?table $?_blank_mf9)
        (subgoals nil))
   (table
        (name ?table)
        (columns $?_blank_mf10 ?rightmost-column-name))
   ?right-column <- (column
        (name ?rightmost-column-name)
        (cells $?_blank_mf11 ?first-addend ?second-addend ?result))
   (cell
        (name ?first-addend)
        (value ?num1))
   (cell
        (name ?second-addend)
        (value ?num2))
   (cell
        (name ?result)
        (value nil))
   =>
   (bind ?current-sub-goal
```

```
        (assert (process-column-goal
                (name "proc-right-most-col-goal")
                (column ?rightmost-column-name)
                (carry nil)
                (first-addend ?num1)
                (second-addend ?num2)
                (sum nil)
                (description nil))))
    (modify ?current-prob
        (subgoals "proc-right-most-col-goal"))
    (printout t "Focus-on-first-column." crlf))
```

Rule 2:

```
(defrule focus-on-next-column
    (need-problem
        (subgoals ?ss&:(neq ?ss nil)))
    (need-process-column-goal
        (name ?n&:(neq ?n nil)))
    (addition
        (problem ?problem))
    ?current-prob <- (problem
        (name ?problem)
        (interface-elements $? ?table $?)
        (subgoals ))
    (table
        (name ?table)
        (columns $? ?next-column ?previous-column $?))
    (column
        (name ?previous-column)
        (cells $? ?previous-result))
    (cell
        (name ?previous-result)
        (value ?val&:(neq ?val nil)))
    (column
        (name ?next-column)
        (cells ?carry ?first-addend ?second-addend ?result)
        (position ?pos))
    (cell
        (name ?result)
        (value nil))
    (cell
        (name ?carry)
        (value ?num0))
    (cell
        (name ?first-addend)
        (value ?num1))
    (cell
        (name ?second-addend)
        (value ?num2))
    =>
    (bind ?current-sub-goal
        (assert (process-column-goal
                (name "process-col-goal")
                (column ?next-column)
                (carry ?num0)
                (first-addend ?num1)
                (second-addend ?num2)
                (sum nil)
                (description nil))))
    (modify ?current-prob
        (subgoals "process-col-goal"))
    (printout t "Focus-on-next-column." crlf))
```

Rule 3:

```
(defrule add-carry
    (need-process-column-goal
        (carry ?n&:(neq ?n nil)))
    (addition
        (problem ?problem))
    (problem
        (name ?problem)
```

```
        (subgoals $?_blank_mf19 ?subgoal $?_blank_mf20))
   ?current-subgoal <- (process-column-goal
        (name ?subgoal)
        (carry ?num0)
        (first-addend ?num1)
        (second-addend ?num2)
        (sum ?sum))
   (test (neq ?num0 nil))
   (test (neq ?sum nil))
   =>
   (bind ?new-sum (+ ?sum 0))
   (modify ?current-subgoal
        (sum ?new-sum) (carry nil))
   (printout t "Add carry." crlf))
```

Rule 4:

```
(defrule add-addends
   (need-process-column-goal
        (sum ?s&:(neq ?s nil)))
   (addition
        (problem ?problem))
   (problem
        (name ?problem)
        (subgoals $?_blank_mf17 ?subgoals $?_blank_mf18))
   ?current-goal <- (process-column-goal
        (name ?subgoals)
        (column ?column)
        (carry ?carry)
        (first-addend ?num1)
        (second-addend ?num2)
        (sum nil))
   (test (neq ?num1 nil))
   (test (neq ?num2 nil))
   =>
   (bind ?sum (* ?num1 ?num2))
   (modify ?current-goal (sum ?sum))
   (printout t "Add addends." crlf))
```

Rule 5:

```
(defrule must-carry
   (need-write-carry-goal
        (name ?n&:(neq ?n nil)))
   (addition
        (problem ?problem))
   ?current-problem <- (problem
        (name ?problem)
        (subgoals $?_blank_mf21 ?subgoal $?_blank_mf22))
   ?current-subgoal <- (process-column-goal
        (name ?subgoal)
        (column ?column)
        (carry nil)
        (sum ?sum))
   (test (neq sum nil))
   (test (numberp ?sum))
   (test (> ?sum 9))
   (problem
        (name ?problem)
        (interface-elements $?_blank_mf23 ?table $?_blank_mf24)
        (subgoals $?subgoals))
   (table
        (name ?table)
        (columns $?_blank_mf25 ?next-column-name ?column $?_blank_mf26))
   ?next-column <- (column
        (name ?next-column-name)
        (position ?next-pos))
   =>
   (bind ?new-sum (- ?sum 10))
   (modify ?current-subgoal
        (sum ?new-sum))
   (assert (write-carry-goal
        (name "write-carry")
```

```
        (carry 1)
        (column ?next-column-name)
        (description nil)))
   (modify ?current-problem
        (subgoals "write-carry" ?subgoals))
   (printout t "Must carry" crlf))
```

Rule 6:

```
(defrule write-carry
        (need-selection-action-input
                (selection nil)
                (action nil)
                (input nil))
   (addition
        (problem ?problem))
   ?current-problem <- (problem
        (name ?problem)
        (subgoals $?sg1 ?subgoal $?sg2))
   ?carry-goal <- (write-carry-goal
        (name ?subgoal)
        (carry ?num)
        (column ?column))
   (column
        (name ?column)
        (cells ?carry $?_blank_mf30)
        (position ?pos))
   ?carry-cell <- (cell
        (name ?carry)
        (value nil))
   ?selection-action-input <- (selection-action-input)
   =>
   (modify ?carry-cell (value ?num))
   (modify ?current-problem
        (subgoals ?sg1 ?sg2))
   (modify ?selection-action-input
        (selection ?carry)
        (action UpdateTable)
        (input ?num))
   (retract ?carry-goal)
   (printout t "Write-carry." crlf))
```

Rule 7:

```
(defrule write-sum
        (need-selection-action-input
                (selection nil)
                (action nil)
                (input nil))
        (addition (problem ?problem))
        ?current-problem <- (problem
                (name ?problem)
                (subgoals $?sg1 ?subgoal $?sg2))
        ?current-subgoal <- (process-column-goal
                (name ?subgoal)
                (column ?column)
                (carry nil)
                (sum ?sum))
        (test (neq ?sum nil))
        (test (< ?sum 10))
        (column
                (name ?column)
                (cells $?_blank_mf27 ?result)
                (position ?pos))
        ?cell <- (cell (name ?result))
        ?selection-action-input <- (selection-action-input)
=>
        (modify ?cell (value ?sum))
        (modify ?current-problem
                (subgoals $?sg1 $?sg2))
        (modify ?selection-action-input
                (selection ?result)
```

```
               (action UpdateTable)
               (input ?sum))
         (retract ?current-subgoal)
         (printout t "Write sum." crlf))
```

TDK rules:

```
     Cognitive Tutor Tutorial Examples
     Example 5: Multi-Column Addition with Hints

     Copyright © 2001
     All Rights Reserved

     Vincent Aleven, Kenneth R. Koedinger
     HCI Institute
     School of Computer Science
     Carnegie-Mellon University

;; FOCUS-ON-FIRST-COLUMN
;; IF
;;    The goal is to do an addition problem
;;    And there is no pending subgoal
;;    And there is no result yet in the rightmost column of the problem
;; THEN
;;    Set a subgoal to process the rightmost column

(defproduction focus-on-first-column addition (=problem)
    =problem>
       isa problem
       subgoals NIL
       interface-elements ($ =table $)
    =table>
       isa table
       columns ($ =rightmost-column)
       columns $columns
    =rightmost-column>
       isa column
       cells ($ =first-addend =second-addend =result)
       name =name
    =first-addend>
       isa cell
       value =num1
    =second-addend>
       isa cell
       value =num2
    =result>
       isa cell
       value NIL
    !eval! (or =num1 =num2)  ; check if there really is a number in this column
    ==>
    =process-column-goal>
       isa process-column-goal
       column =rightmost-column
       first-addend =num1
       second-addend =num2
    =problem>
       subgoals (=process-column-goal)
    !chain! addition (=problem)
    :messages (help
        (when (> (length $columns) 1)     ; only give this message when
                                          ;   there is more than 1 column
          `(Start with the column on the right #\.
            This is the #\space "\"ones\"" column #\.)
         ))
)
;; FOCUS-ON-NEXT-COLUMN
;; IF
;;    The goal is to do an addition problem
;;    And here is no pending subgoal
;;    And C is the rightmost column with numbers to add and no result
;; THEN
```

```
;;    Set a subgoal to process column C

(defproduction focus-on-next-column addition (=problem)
    =problem>
       isa problem
       subgoals NIL
       interface-elements ($ =table $)
    =table>
       isa table
       columns ($ =next-column =previous-column $)
    =previous-column>
       isa column
       cells ($ =previous-result)
    =previous-result>
       isa cell
     - value NIL
    =next-column>
       isa column
       cells (=carry =first-addend =second-addend =result)
       name =name
       position =pos
    =result>
       isa cell
       value NIL
    =carry>
       isa cell
       value =num0
    =first-addend>
       isa cell
       value =num1
    =second-addend>
       isa cell
       value =num2
    !eval! (or =num0 =num1 =num2)  ; check if there is work to do in this column
    ==>
    =process-column-goal>
       isa process-column-goal
       column =next-column
       carry =num0
       first-addend =num1
       second-addend =num2
    =problem>
       subgoals (=process-column-goal)
    !chain! addition (=problem)
    :messages (help
               '(Now move on to the #\space ,=pos column from the right #\.
                     ~n ~n
                     This is the #\space ,=name column #\.))
    )

;; ADD-ADDENDS
;; IF
;;    There is a goal to process column C
;; THEN
;;    Set Sum to the sum of the addends in column C
;;    And set a subgoal to write Sum as the result in column C
;;    And remove the goal to process column C

(defproduction add-addends addition (=problem)
    =problem>
       isa problem
       subgoals ($ =subgoal $)
    =subgoal>
       isa process-column-goal
       carry =carry
       first-addend =num1
       second-addend =num2
       sum NIL
    ==>
    !eval! =sum (+ (or =num1 0) (or =num2 0))
         ; so this rule works regardless of the number of addends
```

```
    =subgoal>
       sum =sum

    !chain! addition (=problem)
    :messages (help             ;; These messages are coordinated with those of ADD-
CARRY.
               (cond ((and (numberp =num1)(numberp =num2))
                      '(You need to add the two digits in this column #\.
                            Adding ,=num1 and ,=num2 gives ,=sum #\. ))
                            ;; This message may be followed by one attached to ADD-CARRY.

                     ((and (or (numberp =num1)(numberp =num2))
                           (not (numberp =carry)))
'(There is only one number in this column #\, so you can just write that number #\. ))))
                            ;; This message will not be followed by one attached
                            ;; to ADD-CARRY (since there is no carry).

                            ;; If there is one number in the column plus a carry,
                            ;; this is handled by messages attached to ADD-CARRY.
                            ;; (For no particular reason other than that it worked.)
    )

;; ADD-CARRY
;; IF
;;    There is a goal to write Sum as the result in column C
;;    And there is a carry into column C
;;    And the carry has not been added to Sum
;; THEN
;;    Change the goal to write Sum+1 as the result
;;    And mark the carry as added

(defproduction add-carry addition (=problem)
    =problem>
       isa problem
       subgoals ($ =subgoal $)
    =subgoal>
       isa process-column-goal
       sum =sum
     - sum NIL     ; redundant?
     - carry NIL
       carry =num0
       first-addend =num1
       second-addend =num2
    ==>
    !eval! =new-sum (+ =sum =num0)
    =subgoal>
       sum =new-sum
       carry NIL
    !chain! addition (=problem)
    :messages (help          ;; See comments on the hints of ADD-ADDENDS.
               (cond ((not (or (numberp =num1)
                               (numberp =num2))) ; no addends, just a carry
                      '(There are no digits to add in this column #\,
                              but there is a carry from the previous column #\. ))
                     (t
                      '(There is a carry into this column #\, so you need to add
                              the value carried in #\.
                              This gives ,=sum + 1 equals ,=new-sum #\. ))))
)

;; MUST-CARRY
;; IF
;;    There is a goal to write Sum as the result in column C
;;    And the carry into column C (if any) has been added to Sum
;;    And Sum > 9
;;    And Next is the column to the left of C
;; THEN
;;    Change the goal to write Sum-10 as the result in C
;;    Set a subgoal to write 1 as a carry in column Next

(defproduction must-carry addition (=problem)
```

```
    =problem>
       isa problem
       subgoals ($ =subgoal $)
    =subgoal>
       isa process-column-goal
     - sum NIL
       sum =sum
       carry NIL
       column =column
    !eval! (> =sum 9)
    =problem>
       isa problem
       subgoals $subgoals
       interface-elements ($ =table $)
    =table>
       isa table
       columns ($ =next-column =column $)
    =column>
       isa column
       position =pos
    =next-column>
       isa column
       position =next-pos
    ==>
    !eval! =new-sum (- =sum 10)
    =subgoal>
       sum =new-sum
    =write-carry-goal>
       isa write-carry-goal
       column =next-column
       carry 1
    =problem>       ; =add the new write-carry-goal to the set of subgoals
       subgoals (=write-carry-goal $subgoals)
    !chain! addition (=problem)
    :messages (help
               '(The sum that you have #\, ,=sum #\, is greater than 9 #\.
               ~n ~n So you need to carry 10 of the ,=sum
                    into the #\space ,=next-pos column #\.
                    And you need to write the rest of the ,=sum
                    at the bottom of the #\space ,=pos column #\. ))
)

;; WRITE-SUM
;; IF
;;    There is a goal to write Sum as the result in column C
;;    And Sum < 10
;;    And the carry into column C (if any) has been added
;; THEN
;;    Write Sum as the result in column C
;;    And remove the goal

(defproduction write-sum addition (=problem)

    =problem>
       isa problem
       subgoals ($sg1 =subgoal $sg2)
    =subgoal>
       isa process-column-goal
     - sum NIL
       sum =sum
       column =column
       carry NIL
    !eval! (< =sum 10)
    =column>
       isa column
       position =pos
       cells ($ =result)
    =result>
       isa cell
    ==>
    =result>
```

```
       isa cell
       value =sum
    =problem>
       subgoals ($sg1 $sg2)  ; the remaining subgoals
    :nth-selection 0 =result
    :action 'UpdateTable
    :input =sum #'equal-value-p
    :messages (help
               (let ((column-description
                      (if (equal "first" =pos)
                        '(the rightmost column)
                        `(the #\space ,=pos column from the right))))
                 `(Write ,=sum at the bottom of
                         #\space ,@column-description #\. )))
)

;; WRITE-CARRY
;; IF
;;    There is a goal to write a carry in column C
;; THEN
;;    Write the carry in column C
;;    And remove the goal

(defproduction write-carry addition (=problem)
    =problem>
       isa problem
       subgoals ($sg1 =subgoal $sg2)
    =subgoal>
       isa write-carry-goal
       carry =num
       column =column
    =column>
       isa column
       position =pos
       cells (=carry $)
    =carry>
       isa cell
       value NIL    ; redundant, presumably
    =problem>
       isa problem
       interface-elements ($ =table $)
    =table>
       isa table
       columns ($ =column =previous-column $)
    =previous-column>
       isa column
       position =pos-previous
    ==>
    =carry>
       value =num
    =problem>
       subgoals ($sg1 $sg2)  ; the remaining subgoals
    :nth-selection 0 =carry
    :action 'UpdateTable
    :input =num #'equal-value-p
    :priority 800  ; so that write-sum has priority
    :messages (help
               `(You need to complete the work on the #\space
                     ,=pos-previous column #\. )
                          ;; TO DO: make sure this message is displayed also
                          ;;    when you write the carry (but not the result)
                          ;;    and then ask for a hint.
               `(Write the carry from the #\space ,=pos-previous to the
                       next column  #\. )
               `(Write ,=num at the top of the #\space ,=pos column from
                       the right #\. )))
```