1 A Production System Model of Subtraction

The purpose of this exercise is for you to examine in detail how a production system can be used to model children's subtraction behaviour by trying out the system with different collections of rules on different subtraction sums.

The system is based on the article by [YO81]. The program is a revamped version of TEACH *
SUBTRACT with a "point and click" Control Panel.

By the end of this exercise you should understand the basic cycle of operation of a production system and how this Young and O'Shea's system can be used to model correct and incorrect subtraction behaviour.

1.1 Getting Started

At the Unix prompt type
`pop11` $< RETURN >$
`load ~bend/gsubtract.p` $< RETURN >$
`go();` $< RETURN >$

This will bring up a Control Panel with which you can choose which production rules to include in a run as well as choosing the sum the system is to work on.

1.2 Finishing and Keeping a Record

The system automatically makes a record of your interaction in a file in your area called `buggylogfile`.
If you start a new session it adds the new interaction *onto the end of the existing file.*

To finish a session: Press the *Exit_Program* key on the control panel
Press the $< RETURN >$ key
type `bye`
Press the $< RETURN >$ key

To see the names of your files type the Unix command `ls`. To show your log file on the screen type the Unix command `more buggylogfile`. To print out that file on the dot matrix printer named "cia" in 3C2 type the Unix command `lp -d cia buggylogfile`.

1.3 The Control Panel

The Control Panel allows you to choose which sets of rules you want to work with, based on the Figures in the Young and O'Shea article. You should start with the (default) Figure 2.

You can choose whether to work with small or large sums. You should start with the (default) small sums. **To make the system run**: Move the minuend and subtrahend sliders to set the sum.

Click on the individual rules to include or exclude the ones you want (several of which are chosen by default). Press *Start_Restart_Sum*. Keep pressing *Next_Cycle* to make the system go through a cycle. When no more rules can fire, pressing *Next_Cycle* will have no more effect.

Certain sets of rules working on certain sums will make the production system give an error message. Don't worry about it (but make a note of it). Just go through the above actions again.

1.4 Activities

You should first read the Young and O'Shea article carefully. The article explains much of what you need to know about production systems in order to do the exercises in this teach file. Figure 2 of this article describes a production system that will cope successfully with subtraction sums so long as the top number (minuend) has the same number of digits as the bottom number (subtrahend) and is larger than it.

Try out the system on the sum 74 - 28 and selecting the rules
`fd b2a bs1 bs2 cm in nxt ts wa done b2c ac.`

The following is what the system outputs when you execute the line above. My comments on this output are surrounded in $$ signs (they are not part of the output!).

```
The production rules available are:
[fd b2a bs1 bs2 cm in nxt ts wa done b2c ac]
PAPER:                              $$ The initial state of the paper $$
    *
  7  4
  2  8
------
WORKING MEMORY:
[processcolumn]                     $$ The initial state of the working
USING RULE:in                          memory $$
[processcolumn]=>readmands ( ) ;    $$ The rule which is about to fire $$

PAPER:
    *
  7  4
  2  8
------
WORKING MEMORY:
[s 8][m 4][processcolumn]
USING RULE:cm
[m ? m][s ? s]=>compare ( ) ;       $$ Young and O'Shea indicate
                                       undecomposable functions with a
PAPER:                                 * e.g. *Compare.  I have used a
    *                                  a pop11 function compare() $$
  7  4
  2  8
------
WORKING MEMORY:
[s greater m][s 8][m 4][processcolumn]
USING RULE:b2a
[s greater m]=>assert ( [ borrow ] ) ;   $$ Young and O'Shea indicate things
                                            to be inserted in the database by
PAPER:                                      terms such as Borrow, here they
    *                                       are shown as assert([borrow]) $$
  7  4
```

```
    2  8
------
WORKING MEMORY:
[borrow][s greater m][s 8][m 4][processcolumn]
USING RULE:bs2
[borrow]=>decrement ( ) ;


PAPER:
      *
  6  4                                          $$ notice the decrementing $$
  2  8
------
WORKING MEMORY:
[borrow][s greater m][s 8][m 4][processcolumn]
USING RULE:bs1
[borrow]=>addtentom ( ) ;


PAPER:
      *
  6 14                                          $$ notice the adding 10 $$
  2  8
------
WORKING MEMORY:
[borrow][s greater m][s 8][m 4][processcolumn]
USING RULE:fd
[m ? m][s ? s]=>assert ( [ nextcolumn ] ) ; assert ( [ finddiff ] ) ;
```

Some of the output has been omitted.

```
PAPER:
  *                                       $$ the * indicates which column is
  6 14                                         being worked on $$
  2  8
------
      6
WORKING MEMORY:
[processcolumn][result 6][finddiff][nextcolumn][borrow][s greater m][s
    8][m 4][processcolumn]
USING RULE:in                             $$ In many psychological models the
[processcolumn]=>readmands ( ) ;             working memory is limited to 7 or so
                                            items.  There is no limit here $$
```

More of the output has been omitted

```
PAPER:
  6 14                                     $$ shifting left in the last column forces
  2  8                                        [nomore] to be asserted into the WM $$
------
  4  6
WORKING MEMORY:
[nomore][processcolumn][result 4][finddiff][nextcolumn][s less m][s 2][m
    6][processcolumn][result 6][finddiff][nextcolumn][borrow][s greater
    m][s 8][m 4][processcolumn]
USING RULE:done
[nomore]=>halt ( ) ;
```

By running the system with different rules it is possible to mimic various sorts of children's errors. Your main task is to try out various lists of production rules on a variety of sums and see whether

the system conforms to your own expectations of what children do and to the article by Young and O'Shea.

(i) Try out the system as above and reconcile its output with the description of the system on pages 160-161.

(ii) Try out the production system above with a variety of sums. It ought to be able to cope with more than two columns. However if you are awkward you can probably make the system break. Keep a record of all the ways are able to make the system come to a halt with a mishap message.

(iii) You can make the system do subtraction by equal additions rather than by decrementing by using rule bs3 in place of bs2, see page 162. Try this out.

(iv) The conflict resolution method used in the production system is not quite the same as that described by Young and O'Shea. What are the differences?

(v) Page 163 describes various kinds of common error which can be exhibited by making changes to the basic production system as shown in Table 2 of the article. Make up sums to show off all the different kinds of error. Keep records of what you have done and also of any unexpected behaviour from the system. Note that production rule nnn (from Figure 4 and Table 2) has not been implemented so you cannot produce the final bug in Table 2.

2 Further (Optional) Experiments

Young and O'Shea tried to explain data from a paper by Brown and Burton. To do this they needed to expand the original production system a little. This was because in the original system the act of "decrementing" is carried out as an action on the right hand side of bs2 and so is always done correctly. In order to model children's errors in "decrementing" it is necessary to expand this primitive action into a set of production rules and add them to the original set. This can easily be achieved by executing the function try with appropriate arguments, for example:

Try the sum 203 - 17 with the rules `fd b2a bs1 bs4 cm in nxt ts wa done b2c ac lhe ten dec dnz dz1 dz2 d10`.

Notice that rules prop, pz and pnz have not been implemented and that bs4 replaces bs2. The rule bs4 was not mentioned in the original article but it is needed to make sure that the original production system makes proper use of the rules for decrementing (rather than just calling up an undecomposable decrementing function).

This expanded system should be able to cope with sums where the minuend has more digits than the subtrahend as well as the special case where the minuend starts with 1 0 ...

Table 4, on page 169, indicates how various kinds of error can be modelled. You should make up sums to show off as many of them as possible.

Make sure to logout from Unix before leaving the terminal by clicking the EXIT icon at the bottom of the screen.

References

[YO81] Richard M. Young and Tim O'Shea. Errors in children's subtraction. *Cognitive Science*, 5(2):153–177, 1981. QZ 10 Cog.