A prototype manufacturing knowledge base in Syllog

by Craig Fellenstein
Charles O. Green
Lucinda M. Palmer
Adrian Walker
David J. Wyler

This paper describes a prototype knowledge base for manufacturing planning, which we have built using a knowledge system shell called Syllog. We describe a Tester Capacity Planning and Yield Analysis task, knowledge needed for a part of the task, and the use of the knowledge in the Syllog system. We report that the sometimes difficult process of knowledge acquisition turned out, in this case, to be straightforward. Knowledge acquisition and knowledge use are done in the same language in Syllog.

1. Introduction

This paper is about building a manufacturing knowledge base and using the knowledge in a system shell called Syllog [1–3]. For present purposes, we can think of a knowledge base as a combination of a relational database and an expert system, the combination being achieved by some form of logic programming [4], possibly with access to a conventional database [5]. In a relational database [6], facts are stored in tables (so far as the user is concerned), and compound facts can be retrieved from more than one table by noting that the tables have common entries. An expert system, on the other hand, is mainly concerned with knowledge about how to use facts. It provides expert-level solutions to important problems; it is flexible in integrating new knowledge incrementally into its existing store of knowledge; it can show its knowledge in a comprehensible form; it answers questions by using its knowledge; and it provides explanations of its answers. An expert system is often heuristic, in the sense that it can reason with judgmental knowledge as well as with formal knowledge of established theories.

Our manufacturing task concerns the testing of electronic items in a production line before they are plugged into larger assemblies. We wish to plan the numbers of various types of testing machines that are needed to meet a production target. Accurate estimates of the numbers of machines are important, since too little capacity can cause production delays, while too much is wasteful. The numbers of machines needed, the times when they should be available, and their locations depend in a complex way on the bill of materials for the items to be manufactured, on previous experience of the yields of good items from testing, on previous experience of the rates at which items can be tested, on the space available, and on several other considerations.

The information needed for planning consists of facts, and of equations and rules for using the facts. The facts are held in some form of database, and, increasingly, are collected automatically by manufacturing instrumentation. An equation or rule typically reads as a piece of common-sense knowledge about the planning task. However, the ways in which the facts and rules combine to give answers are complicated, and our knowledge base allows us to keep a clear view of the process.

One might consider writing conventional database application programs for our task. However, as rules change, or as new rules are added, such application programs tend to get out of step with reality and to become hard to maintain. In our tester capacity planning task, timely expert advice is

needed. Since the users are not programmers, the expertise is best captured and maintained without programming. In addition, while the facts and rules may be correct, the answers that they produce may sometimes not be obvious. Since responsibility for smooth and timely testing lies with the planning team, it is useful to have explanations of the answers provided by a planning tool. If an answer provided by such a planning tool is positive, an explanation provides a means of checking the correctness of the rules that produced the answer. If an answer provided by a planning tool is negative, an explanation can help in determining what additional facts or rules are needed.

We write knowledge for the Syllog shell in the form of facts and syllogisms. Facts are like tuples in a relational table, but in Syllog a table has an English-like heading. Syllogisms are English-like rules about how to use the facts. From the user's point of view, the syllogisms can be regarded as declarative knowledge, in the sense that the order in which a set of syllogisms is written down does not matter. Since new facts and syllogisms can be added at will (subject to some consistency checking by the system), new knowledge can be added incrementally. Because the knowledge consists of tables with English-like headings, together with syllogisms consisting of simple English-like sentences, the system can show its current knowledge in human-understandable form. Syllog answers questions by reasoning logically and exactly with the knowledge provided. However, the knowledge itself may contain judgmental phrases (e.g., the expected yield of . . .) which can combine to give a judgmental answer. Whenever Syllog answers a specific question, either with a "yes" or with a "no," it can provide an explanation if needed. The explanation is given in terms of the English-like sentences in the knowledge base.

Although Syllog works with English-like sentences, there is no need to build a dictionary or a grammar when new parts of English (e.g., for law or manufacturing) are needed. One simply types in sentences, and the system is immediately ready to use them. In fact, the language used can equally well be French, German, most other natural languages, or even an artificial language. Syllog makes logically correct inferences based on what it is told. However, it "knows" little of the language concerned, when compared with normal natural language systems, such as [7]. This lack of knowledge has both advantages and disadvantages. The advantages are that one can easily add knowledge to the system in the language of one's choice, and that one need not instruct the system about the details of the language one is using. The disadvantages are that one is limited to simple declarative sentences, and these use special "example" words [8]. Also, one must either consistently use the same sentence to mean the same thing, or provide syllogisms to say that different sentences have the same meaning. Syllog supports this approach by prompting with the sentences that are in the knowledge base.

Some simple sets of syllogisms can be thought of as English-like forms of expressions in relational algebra [6], with some arithmetic processing allowed. However, syllogisms can be recursive, and they sometimes need to be. In our manufacturing knowledge base, we define a parts explosion hierarchy, and the definition is recursive. Recursion allows us to express concepts that cannot be written down in the nonrecursive relational algebra [9]. Although recursive syllogisms are quite readable and have a simple common-sense meaning (corresponding to their models in mathematical logic), it is not entirely straightforward to interpret them both declaratively (i.e., order does not matter) and efficiently. For example, it is not enough to compile recursive syllogisms into the language Prolog [10] and then execute them using Prolog's built-in inference engine, for the reasons set out in [1, 11].

Therefore, the Syllog system, which is written in Prolog, contains its own inference engine. This engine computes correctly with many sets of recursive syllogisms that are outside the scope of Prolog itself. Some related ways of computing with recursive logic statements are given a formal treatment in [12]. Although the present paper is self-contained, more detail about Syllog is to be found in [3].

The next section describes our manufacturing planning task. In Section 3, we show the process of building a part of a knowledge base for the task, in Syllog. Then we describe how the knowledge can be used for manufacturing planning by asking questions, by trying "what-if" questions, and by getting explanations.

2. The tester capacity planning and yield analysis task

In this section, we describe the task for which we have built part of a knowledge base. The two subsections describe a simplified version of the task and outline the actual task. Then, in Section 3, we give a Syllog knowledge base for the simplified task, and we show how it is used to plan the number of test machines that are needed to meet a production target.

• *A simplified tester capacity planning task*

This section describes a simplified version of a planning task that is of importance to manufacturing test engineers.

A major step in electronics manufacturing is the testing of individual cards before they are plugged into a larger assembly (e.g., a controller), generally called a product, or box. Typically, a flow of cards must be checked on a battery of specialized test machines. Another important step is to test the components, such as resistors and chips, that will be placed on a card. For this step, test machines are also needed. In each step, some of the items (cards or components) that are tested fail.

So, if there is a production goal to manufacture, say, 1000 boxes during the third quarter of the year, we must plan to

allocate enough test machines of the right kinds. This is the tester capacity planning problem. To solve the problem, we need three kinds of knowledge.

First, we need a bill of materials for a box, showing the number of each type of card needed and the number of each type of component needed for each card. Second, we need an estimate of the percentage of each batch of items that will pass testing. Third, we need to know the capacity of each kind of tester on each kind of item, that is, the rates at which items can be tested.

The bill of materials for a box can be supplied as a table showing, for each item, how many subitems of a given type there are. From this, we need to find how many components of eacn kind are in a box, using the information about how many components are on a card and how many cards of each kind are in a box. This is the well-known "parts explosion" hierarchy. In addition, we need to estimate the larger numbers of each component and card that are actually needed for a batch of boxes, on the basis of expected yields of good items (cards and components) during the two levels of testing.

The expected yield of each of the items, based on previous testing, can be extracted from detailed reports of test batches in earlier production runs.

Finally, in order to estimate the number of test machines needed to meet a production goal, we need to know the rate at which each tester can test each type of item. We usually have an estimate of the capacity of a tester from earlier experience, and this estimate improves with time. However, the capacity is not fixed for a given item, but depends on how we choose to test. For example, we may choose to test some percentage of items with a "burn in" period, which takes longer but gives better results. We get this knowledge by summarizing previous test records for the same items where available.

What we have described so far is a simplified form of the actual task. This form is convenient for illustrating the kinds of knowledge and reasoning that are needed. In Section 3, we describe this simplified task knowledge as a set of syllogisms and tables, and we show how "what if" tester capacity estimates can be made by using the Syllog system to apply the knowledge. We also show how the knowledge and estimates can be checked using Syllog's built-in explanation mechanisms.

The actual task for which we have built part of a knowledge base is more complicated.

- *The actual tester capacity planning task*

The tester capacity problem occurs throughout the life cycle of any product. Capacities are a factor in capital planning—planning for the quantities of test machines needed to meet the future production schedules of some new product. Later on in the life cycle of the product, tester capacities are a factor in the planning of daily workload and machine loading. As the product nears the end of its life cycle, tester capacities are a factor in the problem of reallocating tester resources to new products. The test machines may be switched from one task to another, but a given machine does not cover all testing tasks, and switching from one task to another takes a certain amount of setup time.

For example, suppose that a new product is under development. The circuit card test engineers in manufacturing begin planning for test equipment to meet the production schedules for the cards. They know that the new product has 20 cards which must be tested on a particular tester that will also be used for testing other products in production at the same time. Each of the 20 cards requires different times for various kinds of testing, e.g., for successful testing and for test-analysis-repair and retesting. To plan for suitable tester capacities, we must be able to answer questions such as the following:

1. How many testers will be required for the new and concurrent production of other products?
2. When will these quantities of testers be required?
3. How many more testers will be required, and when, just for the new product?

Answers to these question can lead to further questions, such as

4. When are the new testers to be acquired, considering the time required to install and program them?
5. How much floor space and rearrangement will be required for the new testers?
6. Is there sufficient floor space and sufficient time to acquire the new testers to meet the production schedules?
7. Should the new testers be placed at a vendor shop, or should some of the test workload be placed with a vendor who already has testers?

The answers become part of a final manufacturing plan for the new product.

3. Acquiring and using tester and yield knowledge in Syllog

In Section 2, we described a simplified tester capacity and yield task, and we outlined the corresponding real task. In this section, we show how knowledge for the simplified task of Section 2 is written down for Syllog. Syllog is then used to answer the question "How many test machines of each type will be needed to support the manufacture of a certain number of boxes over a given time period?" We also show how "what-if" questions about tester capacity are posed and answered in Syllog.

To answer these kinds of questions, facts are needed, together with knowledge about how to use the facts. We express these in the language of the Syllog system shell.

Syllog is task-independent; that is, different tasks within a certain range can be performed just by loading different sets of facts and rules. So we do not need to change the Syllog system itself in order to work with facts and knowledge about tester capacities.

We mentioned that for our simplified tester capacity and yield analysis task, we need knowledge of three kinds. We need to know about the parts hierarchy (bill of materials) for the box to be manufactured, about the expected yield of each item from testing, and about test machines and their individual capacities.

• *The parts hierarchy*

First, we write the facts and syllogisms for the parts hierarchy, using the Syllog language of facts and rules.

A group of related facts is written for Syllog in the form of a table. The table is like a table in a relational database, except that it is headed by a sentence in English (or French, German, etc.), containing one or more "example elements." Thus, we can enter (or load) the table

```
eg_item has eg_number of the immediate part eg_subitem
-------------------------------------------------------
box1        2                                 card1
box1        3                                 card2
card1       5                                 resistor1
card1       7                                 chip2
card1       6                                 capacitor1
card2       8                                 chip1
card2       5                                 resistor7
card2       6                                 capacitor2
```

The English sentence has an example element eg_item, which we can think of as "an item" or "some item"; likewise for eg_number and eg_subitem. So the whole sentence above the line can be read as "an item has a number of the immediate part some subitem." The line separates the sentence, which serves as a heading, from the body of the table. The first row of the table can be read as the fact that "box1 has a 2 of the immediate part card1," and the other rows similarly. So a row in a Syllog table corresponds to a fact in the ordinary sense.

We write down knowledge for Syllog in the form of syllogisms. A syllogism consists of one or more sentences, a line, then a single sentence, as in

```
eg_item has eg_4 of the part eg_subitem
eg_subitem has eg_6 of the part eg_subsubitem
eg_4 * eg_6 = eg_24
-------------------------------------------
eg_item has eg_24 of the part eg_subsubitem
```

As for a classical syllogism, the meaning is this: IF each of the sentences above the line is true, THEN the sentence below the line is true. In each sentence the example elements serve as place holders. Here eg_4 stands for some number, and it stands for the same number in two places in the syllogism. (We could equally well have written eg_x for eg_4, eg_y for eg_6, and eg_z for eg_24; the choice is a matter of style.) We refer to a syllogism as a rule, for short. We can read the above rule as "if an item has x of some subitem, each such subitem has y of some subsubitem, and x times y is z, then the item has z of the subsubitems." (We are assuming for simplicity that each kind of item appears in just one place in the parts hierarchy.)

We need another similar rule. If an item has a number x of the immediate subitems of some kind, it has x subitems of that kind. In Syllog, we write this as the rule

```
eg_item has eg_5 of the immediate part eg_subitem
-------------------------------------------------
eg_item has eg_5 of the part eg_subitem
```

Once we have typed the table of facts and the two rules into Syllog, the system responds with a prompt

```
eg_item has eg_5 of the immediate part eg_subitem
eg_item has eg_4 of the part eg_subitem
```

consisting of the sentences it has seen so far. (There are just two sentences in the prompt, since the other important sentences in the knowledge can be obtained from these by renaming the "eg_" example elements.) If we want to see how many of each card and component are in box1, we select the second sentence

```
eg_item has eg_4 of the part eg_subitem
```

of the prompt, change eg_item to box1, and underline the sentence thus:

```
box1 has eg_4 of the part eg_subitem
------------------------------------
```

Syllog understands this as a request to produce a table below the line. If we press the appropriate key, Syllog answers by changing the screen to

```
box1 has eg_4 of the part eg_subitem
------------------------------------
        2                 card1
        3                 card2
        10                resistor1
        14                chip2
        12                capacitor1
        24                chip1
        15                resistor7
        18                capacitor2
```

Thus, our question to Syllog is the sentence that we chose from a menu of sentences and specialized by typing box1, plus the line that indicates that we want a table to be produced. Syllog then fills in the table below the line.

The system has used the table and the two rules to reason that, among other things, box1 contains 10 of resistor1. To get an explanation of the reasoning, we can change the screen to

```
box1 has 10 of the part resistor1
---------------------------------
```

The system understands this as the yes-no question "does box1 have 10 of the part resistor1?" and answers by changing the screen to

```
box1 has 10 of the part resistor1
---------------------------------

   Yes, that's true

   Because...

box1 has 2 of the part card1
card1 has 5 of the part resistor1
2 * 5 = 10
---------------------------------
box1 has 10 of the part resistor1


box1 has 2 of the immediate part card1
---  ---------------------------------
box1 has 2 of the part card1


card1 has 5 of the immediate part resistor1
-------------------------------------------
card1 has 5 of the part resistor1
```

The explanation which follows "Because . . ." consists of the instances of the rules that have been used to establish the answer. The first rule instance gives the main conclusion, while the last two rule instances show why the premises of the first rule instance hold. In general, Syllog explanations may be longer than one screen, so it is useful to see the main reasoning first, and then to scroll to subsidiary justifications as needed.

Now that the system can reason about the numbers of cards and components in a box, we need to add knowledge about the numbers actually needed, given that not all items will pass their tests. This depends on the number of boxes we plan to ship, the length of time we are given to manufacture the boxes, and the expected yield of each item at each test. We can write the knowledge in Syllog as

```
we plan to ship eg_100 of eg_product in quarter eg_q
eg_product has eg_2 of the immediate part eg_card
the expected yield of eg_card is eg_75 %, based on past experience
eg_100 * eg_2 = eg_200
eg_200 divided by eg_75 (normalized and rounded up) is eg_267
-------------------------------------------------------------
we shall set up testers for eg_267 of eg_card in quarter eg_q


we shall set up testers for eg_10 of eg_card in quarter eg_q
eg_card has eg_2 of the immediate part eg_comp
eg_10 * eg_2 = eg_20
the expected yield of eg_comp is eg_50 %, based on past experience
eg_20 divided by eg_50 (normalized and rounded up) is eg_40
-----------------------------------------------------------
we shall set up testers for eg_40 of eg_comp in quarter eg_q


eg_y - 1 = eg_y1
eg_x * 100 = eg_x100
eg_x100 + eg_y1 = eg_x100_plus_y1
eg_x100_pius_y1 / eg_y = eg_z
-------------------------------------------------------
eg_x divided by eg_y (normalized and rounded up) is eg_z
```

The first rule tells us how many cards to plan tester capacity for, given the number of boxes we want to ship and the expected yield of good cards from testing. The second rule does the same for each component, making use of the conclusion of the first rule. The third rule just does normalized, rounded-up division. In order to use the first two rules, we need know the expected yield of each item during testing, based on past experience. For the moment, we shall assume that this is given as

```
the expected yield of eg_item is eg_y %, based on past experience
-----------------------------------------------------------------
                    card1       88
                    card2       95
                    resistor1   90
                    chip2       85
                    capacitor1  90
                    chip1       93
                    resistor7   94
                    capacitor2  95
```

Later, we shall show how the yield information is extracted from more detailed shop floor reports.

If we now select the conclusion of our first rule and ask for a table, we get

```
we shall set up testers for eg_267 of eg_card in quarter eg_q
-------------------------------------------------------------
                    2273    card1       3
                    3158    card2       3
                    12628   resistor1   3
                    18719   chip2       3
                    15154   capacitor1  3
                    27166   chip1       3
                    16798   resistor7   3
                    19946   capacitor2  3
```

If we wish, Syllog will give us an explanation of any row of the table. If we pick the fourth row, we get

```
we shall set up testers for 18719 of chip2 in quarter 3
-------------------------------------------------------

   Yes, that's true

   Because...

we shall set up testers for 2273 of card1 in quarter 3
card1 has 7 of the immediate part chip2
2273 * 7 = 15911
the expected yield of chip2 is 85 %, based on past experience
15911 divided by 85 (normalized and rounded up) is 18719
---------------------------------------------------------
we shall set up testers for 18719 of chip2 in quarter 3


we plan to ship 1000 of box1 in quarter 3
box1 has 2 of the immediate part card1
the expected yield of card1 is 88 %, based on past experience
1000 * 2 = 2000
2000 divided by 88 (normalized and rounded up) is 2273
-------------------------------------------------------
we shall set up testers for 2273 of card1 in quarter 3
```

The bottom of the explanation also contains the reasons for the rounded division result; we omit such details from explanations from now on. They are always available in Syllog by scrolling down through the screens of the full explanation.

At this point, we can predict the numbers of items that we shall need to test, based on some assumed yield figures. Next, we show how the yield figures are found. Then, we add knowledge about testers, so that we can find out how many testers of each type are needed.

- *Yield analysis*

In the previous section, we used the assumed yield figures

```
the expected yield of eg_item is eg_y %, based on past experience
-----------------------------------------------------------------
                    card1       88
                    card2       95
                    resistor1   90
                    chip2       85
                    capacitor1  90
                    chip1       93
                    resistor7   94
                    capacitor2  95
```

These are actually found from shop floor reports of earlier test runs. The kinds of facts that are available are about test jobs. A job has some number of items (cards or components). Some of these items fail test, and the number of failures is noted. Also, the day on which each job is started and the day on which it is finished are tabulated. So the kinds of facts that are available are

```
job eg_j has eg_number of eg_item
-------------------------------------
     1        100          card1
     2        200          card2
     3        500          resistor1
     4        700          chip2
     5        600          capacitor1
     6        800          chip1
     7        500          resistor7
     8        600          capacitor2

in job eg_j the fallout was eg_f items
-------------------------------------
     1                     12
     2                     10
     3                     50
     4                     100
     5                     55
     6                     50
     7                     28
     8                     30

job eg_j was completed during the days eg_d1 through eg_d2
-------------------------------------------------------------
     1                                  1          31
     2                                  1          3
     3                                  3          20
     4                                  1          1
     5                                  3          12
     6                                  51         70
     7                                  1          365
     8                                  1          365
```

These facts are summarized by using the two rules

```
in days eg_d1 thru eg_d2 yield of eg_item was eg_y % in job eg_j
-----------------------------------------------------------------
the expected yield of eg_item is eg_y %, based on past experience


job eg_j was completed during the days eg_d1 through eg_d2
job eg_j has eg_number of eg_item
in job eg_j the fallout was eg_f items
eg_number - eg_f = eg_net
eg_net is eg_y as a percent of eg_number
-----------------------------------------------------------------
in days eg_d1 thru eg_d2 yield of eg_item was eg_y % in job eg_j
```

Once we have entered these three tables and two rules into the system, we can ask for an explanation of how the first row of the expected yield table was found:

```
the expected yield of card1 is 88 %, based on past experience
-------------------------------------------------------------

   Yes, that's true

   Because...

in days 1 thru 31 yield of card1 was 88 % in job 1
--------------------------------------------------------
the expected yield of card1 is 88 %, based on past experience


job 1 was completed during the days 1 through 31
job 1 has 100 of card1
in job 1 the fallout was 12 items
100 - 12 = 88
88 is 88 as a percent of 100
-----------------------------------------------
in days 1 thru 31 yield of card1 was 88 % in job 1
```

So far, we have knowledge about the parts hierarchy, about the number of items we shall need to test, about the number of components needed, and about the expected yield of each item that we shall test. It remains to plan the number of test machines.

• *Planning the number of test machines*

We can now predict the number of items that we shall need to test in quarter 3 on the basis of shop floor reports of yields in earlier production runs. We still need to add knowledge about the test machines themselves, and facts about how many items will be burnt in during testing, so that we can plan the actual number of each kind of test machine.

We start with the facts. We shall plan to burn in items at the rates

```
eg_p % of eg_item are burnt in
------------------------------
50        card1
60        card2
50        resistor1
70        chip2
60        capacitor1
40        chip1
50        resistor7
60        capacitor2
```

and later we shall try some "what if" questions about the effects of these rates on the testers needed. Testing an item on a machine takes different times, depending on whether or not we burn in the item:

```
test of eg_item on eg_m machine takes times eg_t1 (nb) and eg_t2 (b)
--------------------------------------------------------------------
       card1      a50                20          40
       card2      t20                36          62
       resistor1  cr10               2           10
       chip2      nb40               3           20
       capacitor1 cc20               2           15
       chip1      nb50               5           50
       resistor7  cr20               2           9
       capacitor2 cc30               3           20
```

Here, "eg_t1 (nb)" and "eg_t2 (b)" are the times for testing without and with burn in, respectively. The table consists of estimates that we normally obtain by using rules to summarize more detailed reports. For example, with a little more detail on the time taken to complete each job (see the section on yield analysis), the times given in the above table could be found from shop floor reports about individual jobs.

We use the facts with some rules that tell us how many test machines we shall need:

```
we shall set up testers for eg_num of eg_item in quarter eg_q
eg_num divided by 60 (and rounded up) is eg_rate
-------------------------------------------------------------
the daily going rate for eg_item for quarter eg_q is eg_rate
```

Here, we are assuming that there are 60 working days per quarter.

```
test of eg_item on eg_m machine takes times eg_t1 (nb) and eg_t2 (b)
eg_p % of eg_item are burnt in
the rounded eg_p % weighting of eg_t1 and eg_t2 is eg_t
480 divided by eg_t (and rounded up) is eg_c
-----------------------------------------------------------------
can handle eg_c of eg_item pd, with eg_p % burnin, on eg_m machine
```

The 480 in this rule is the number of minutes in a working day.

```
the daily going rate for eg_item for quarter eg_q is eg_rate
can handle eg_c of eg_item pd, with eg_p % burnin, on eg_testm machine
eg_rate divided by eg_c (and rounded up) is eg_number
-------------------------------------------------------------
we need eg_number of the eg_testm test machine in quarter eg_q
```

In the first rule, the sentence

```
we shall set up testers for eg_num of eg_item in quarter eg_q
```

refers to the rules, described above, for the volume of items for which test machines are needed. In the second rule, the first and second sentences refer to tables. The last rule makes use of the first two to tell us how many test machines of each type we need.

If we now select the conclusion of the last rule, Syllog fills in a table

```
we need eg_number of the eg_testm test machine in quarter eg_q
-------------------------------------------------------------
        3               a50                             3
        6               t20                             3
        3               cr10                            3
        10              nb40                            3
        6               cc20                            3
        22              nb50                            3
        4               cr20                            3
        9               cc30                            3
```

showing the numbers of each test machine needed. To see how the knowledge in the system has contributed to the table, we can ask for an explanation of the sixth row:

```
we need 22 of the nb50 test machine in quarter 3
-------------------------------------------------

   Yes, that's true

   Because...

the daily going rate for chip1 for quarter 3 is 453
can handle 21 of chip1 pd, with 40 % burnin, on nb50 machine
453 divided by 21 (and rounded up) is 22
-------------------------------------------------------------
we need 22 of the nb50 test machine in quarter 3


we shall set up testers for 27166 of chip1 in quarter 3
27166 divided by 60 (and rounded up) is 453
-------------------------------------------------------------
the daily going rate for chip1 for quarter 3 is 453


test of chip1 on nb50 machine takes times 5 (nb) and 50 (b)
40 % of chip1 are burnt in
the rounded 40 % weighting of 5 and 50 is 23
480 divided by 23 (and rounded up) is 21
-------------------------------------------------------------
can handle 21 of chip1 pd, with 40 % burnin, on nb50 machine


we shall set up testers for 3158 of card2 in quarter 3
card2 has 8 of the immediate part chip1
3158 * 8 = 25264
the expected yield of chip1 is 93 %, based on past experience
25264 divided by 93 (normalized and rounded up) is 27166
-------------------------------------------------------------
we shall set up testers for 27166 of chip1 in quarter 3


we plan to ship 1000 of box1 in quarter 3
box1 has 3 of the immediate part card2
the expected yield of card2 is 95 %, based on past experience
1000 * 3 = 3000
3000 divided by 95 (normalized and rounded up) is 3158
-------------------------------------------------------------
we shall set up testers for 3158 of card2 in quarter 3


in days 51 thru 70 yield of chip1 was 93 % in job 6
-------------------------------------------------------------
the expected yield of chip1 is 93 %, based on past experience


job 6 was completed during the days 51 through 70
job 6 has 800 of chip1
in job 6 the fallout was 50 items
800 - 50 = 750
750 is 93 as a percent of 800
-------------------------------------------------------------
in days 51 thru 70 yield of chip1 was 93 % in job 6


in days 1 thru 3 yield of card2 was 95 % in job 2
-------------------------------------------------------------
the expected yield of card2 is 95. %, based on past experience


job 2 was completed during the days 1 through 3
job 2 has 200 of card2
in job 2 the fallout was 10 items
200 - 10 = 190
190 is 95 as a percent of 200
-------------------------------------------------------------
in days 1 thru 3 yield of card2 was 95 % in job 2
```

Once we have checked the knowledge in the rules by looking at several such explanations, we can start to ask "what-if" questions. Recall that the present answers are based on the burn in percentages:

```
eg_p % of eg_item are burnt in
------------------------------
50      card1
60      card2
50      resistor1
70      chip2
60      capacitor1
40      chip1
50      resistor7
60      capacitor2
```

If we have this table on the screen, we can increase the burn in percentages for the cards to

```
eg_p % of eg_item are burnt in
------------------------------
70      card1
80      card2
70      resistor1
90      chip2
80      capacitor1
60      chip1
70      resistor7
80      capacitor2
```

and ask again how many test machines are needed. For the new burn in percentages:

```
we need eg_number of the eg_testm test machine in quarter eg_q
-------------------------------------------------------------
        3               a50                             3
        6               t20                             3
        4               cr10                            3
        12              nb40                            3
        7               cc20                            3
        31              nb50                            3
        5               cr20                            3
        12              cc30                            3
```

Once again, we can ask for an explanation, if we so wish.

We have written down some knowledge in Syllog. For the simplified task described in Section 2, we have written a bill of materials and parts hierarchy, an estimate of the yield of items from each phase of testing, and an estimate of the rate at which each item can be tested on a given test machine. The Syllog system has applied the knowledge to plan the number of test machines of each type needed, and also to plan an inventory requirements list of components to be purchased to meet the production goal.

4. Conclusions

Without knowledge, a shell system such as Syllog is just that—a shell; the development of such a system raises two related questions. The first question is "How easy is it to acquire knowledge for the shell?" The second is "What are the mathematical and engineering characteristics of the internals of the shell?" Clearly, the second question can be answered more precisely than the first.

Knowledge acquisition is an extensive topic in its own right. The main concerns are how to inform a system of the concepts it needs, and then how to instruct it in the use of the concepts. Knowledge acquisition can take the form of being told, of computer-assisted debugging, of conversation, or of inductive inference. Indeed, these methods can be combined.

For the present Syllog system, the approach is acquisition by being told, with two techniques for assisting the person who is providing the knowledge. A measure of the success of the first technique is that the person should not notice it! Syllog supports the acquisition of *declarative* knowledge. This means that the person putting in knowledge is largely freed from considerations about how the knowledge will be used in a computation. By contrast, programming in a conventional language (and even, to some extent, in Prolog) is a *procedural* activity, in which we tell the computer a sequence of steps it is to take. The second technique is the automatic provision of explanations. Since knowledge may be supplied in any order, perhaps by different people, the answers produced from the knowledge can be unexpected. So explanations are useful both for checking the answers and for checking the knowledge on which the answers are based.

We have noticed that, in Syllog, the combination of declarative English-like language with explanations tends to encourage direct interactive experimentation with sample knowledge bases relatively early in the process of knowledge acquisition. Our prototype manufacturing knowledge base has been written essentially as a spare time activity by the authors, and knowledge acquisition has been straightforward.

The internals of the Syllog shell consist of a simple translation of the language seen on the screen to and from logic; of an inference engine; and of an explanation generation component. The inference engine allows one to write recursive syllogisms declaratively, as in the parts explosion hierarchy in our manufacturing knowledge base. The engine answers a question by combining a form of on-the-fly compilation (essentially a top-down symbolic execution of the rules relevant to a question) with a form of forward-chaining evaluation [3]. The explanation component contains some task-independent heuristics to select a helpful explanation from several alternatives [11].

Even for the simplified manufacturing knowledge base in the last section, it is clear that the facts and knowledge are complicated enough to make Syllog an attractive improvement over the existing planning methods known to us, including treatment by pencil plus spreadsheet program, or by standard application programming over a database. In Syllog, the knowledge is modular and self-documenting. Thus it is relatively easy to change the knowledge for what-if studies, and to extend the knowledge.

We mentioned in Section 2 that our main task is to answer some more complicated questions. Of these, the questions

1. How many testers will be required for the new and concurrent production of other products?
2. When will these quantities of testers be required?
3. How many more testers will be required, and when, just for the new product?
4. When are the new testers to be acquired, considering the time required to install and program them?

can be answered by straightforward extensions of the simplified knowledge base. The questions

5. How much floor space and rearrangement will be required for the new testers?
6. Is there sufficient floor space and sufficient time to acquire the new testers to meet the production schedules?
7. Should the new testers be placed at a vendor shop, or should some of the test workload be placed with a vendor who already has testers?

are partly about constraint satisfaction (a kind of knowledge that works well in a logic-based system such as Syllog), and partly about the kind of knowledge we have dealt with in the simplified knowledge base.

Our conclusion is that it is reasonably straightforward to acquire and use knowledge of our manufacturing task in Syllog, and that the resulting knowledge base shows promise of practical usefulness.

References

1. A. Walker, "Syllog: A Knowledge Based Data Management System," *Report No. 34,* Department of Computer Science, New York University, New York, 1981.
2. A. Walker, "Data Bases, Expert Systems, and Prolog," *Artificial Intelligence Applications for Business,* W. Reitman, Ed., Ablex, Norwood, NJ, 1984.
3. A. Walker, "Syllog: An Approach to Prolog for Non-Programmers," *Logic Programming and its Applications,* M. van Caneghem and D. H. D. Warren, Eds., Ablex, Norwood, NJ, 1985.
4. D. S. Parker, M. Carey, F. Golshani, M. Jarke, E. Sciore, and A. Walker, "Logic Programming and Databases," to appear as a chapter in the book *Expert Database Systems,* L. Kerschberg, Ed., Springer-Verlag New York, 1985.
5. C. L. Chang and A. Walker, "Prosql: A Prolog Programming Interface with SQL/DS," *Research Report RJ-4314,* IBM Research Laboratory, San Jose, CA, 1984. To appear as a chapter in the book *Expert Database Systems,* L. Kerschberg, Ed., Springer-Verlag New York, 1985.
6. E. F. Codd, "Relational Completeness of Data Base Sublanguages," *Courant Computer Science Symposium 6: Data Base Systems,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1971, pp. 65–98.

7. A. Walker and A. Porto, "KBO1: A Knowledge Based Garden Store Assistant," *Research Report RJ-3928,* IBM Research Laboratory, San Jose, CA, 1983. Also appears in *Proceedings, Logic Programming Workshop,* Portugal, June 1983.
8. M. M. Zloof, "Query-by-Example: A Data Base Language," *IBM Syst. J.* **16**, 324–343 (1977).
9. A. V. Aho and J. D. Ullman, "Universality of Data Retrieval Languages," *Proceedings of the 6th Annual ACM Symposium on Principles of Programming Languages,* 1979, pp. 110–119.
10. W. F. Clocksin and C. S. Mellish, *Programming in Prolog,* Springer-Verlag New York, 1981.
11. A. Walker, "Prolog/Ex1, An Inference Engine Which Explains Both Yes and No Answers," *Proceedings, 8th International Joint Conference on Artificial Intelligence,* Karlsruhe, W. Germany, 1983, pp. 526–528.
12. D. Brough and A. Walker, "Some Practical Properties of Logic Programming Interpreters," *Proceedings, Japan FGCS84 Conference,* Tokyo, 1984, pp. 149–156.

Received February 12, 1985; revised March 14, 1985

Craig Fellenstein *IBM General Products Division, Tucson, Arizona 85744.* Mr. Fellenstein joined IBM in 1980 and is an associate programmer in manufacturing systems. He is currently involved with database systems and advanced information processing tools development. He is interested in intelligent systems in the industrial environment. Before joining IBM, Mr. Fellenstein was a member of the United States Air Force Tactical Air Command.

Charles O. Green *IBM General Products Division, Tucson, Arizona 85744.* Mr. Green, who joined IBM in 1980, is a manufacturing engineering specialist. He is currently working on the planning and development of advanced test systems strategies. Mr. Green is interested in advanced methodologies for automated test equipment and in artificial intelligence approaches to information processing. Prior to joining IBM, he graduated from the Technical Vocational Institute, Albuquerque, New Mexico, where he studied electronic technologies.

Lucinda M. Palmer *IBM General Products Division, Tucson, Arizona 85744.* Ms. Palmer joined IBM in 1979 as a unit test technician following completion of the digital electronics program at the Technical Vocational Institute, Albuquerque, New Mexico. She is currently a programmer in the Printed Circuit Business Operations Department in Tucson.

Adrian Walker *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Walker is manager of principles and applications of logic programming at the Thomas J. Watson Research Laboratory. He joined IBM at the San Jose Research Laboratory in 1981, working on the R* distributed database system and on logic programming and expert systems. He moved to Yorktown in 1984. Dr. Walker received his Ph.D. in computer science from the State University of New York in 1974; he held the posts of assistant professor at Rutgers University and member of the technical staff at Bell Laboratories, Murray Hill, New Jersey, before joining IBM.

David J. Wyler *IBM General Products Division, Tucson, Arizona 85744.* Mr. Wyler joined IBM in 1979; he is a staff engineer in manufacturing systems. He is currently involved with database systems and manufacturing logistics systems in circuit card manufacturing. Before joining IBM, he was a manufacturing engineer for ten years at Hughes Aircraft Company. He received an M.S. in operations management from the University of Arizona, Tempe, in 1976.