S34-61

123

On A Production System Using Default Reasoning For Pattern Classification

Matthew R. Barry
Carlyle M. Lowe

Rockwell Space Operations Company
NASA/Johnson Space Center DF63
Houston. TX 77058
mbarry@nasamail.nasa.gov

1 May 1990

1 Introduction

This paper addresses an unconventional application of a production system to a problem involving belief specialization. The production system reduces a large quantity of low-level descriptions into just a few higher-level descriptions that encompass the problem space in a more tractable fashion. This *classification* process utilizes a set of descriptions generated by combining the component hierarchy of a physical system with the semantics of the terminology employed in its operation. The paper describes an application of this process in a program. constructed in C and CLIPS. that classifies signatures of electromechanical system configurations. The program compares two independent classifications. describing the actual and expected system configurations, in order to generate a set of contradictions between the two.

1.1 Background

The problem application considered herein involves the operational evaluation of NASA's Space Shuttle hardware configurations by flight controllers in the Mission Control Center (MCC). Specifically, the technique has been applied to one of the tasks involved in monitoring the two Shuttle propulsion systems: the *Orbital Maneuvering System* (OMS) and the *Reaction Control System* (RCS).

Shuttle astronauts operate the propulsion systems by manipulating a collection of switches and valves that control fluid flows throughout the plumbing network. Many of the switches control two propellant line valves simultaneously: an oxidizer valve and the corresponding fuel valve. Position indicators within the valves and switches provide insight into their mechanical position. Flight controllers in the MCC help the astronauts to manage these systems by monitoring the on-board configuration. Valve and switch positions appear to the flight controllers as binary values noting presence of (or lack of) an open indication, closed indication, or both. A set of 16-bit *configuration words* relay all of the available measurements through the orbiter computers to the flight controllers.

The MCC computers help the flight controllers to monitor the on- board valve and switch configuration by executing a program that compares *actual* and *expected* configurations. Since only some of the bits in a given configuration word apply to the propulsion systems, the comparison procedure includes a set of masking words. When the bit patterns that are not subject to the mask do not match, the program indicates a problem by displaying a certain status character next to that word. Since the contents of those words are displayed in hexadecimal, flight controllers are made aware of a discrepancy condition through this status character, but are not informed of the actual discrepancy. Furthermore, several discrepancies may occur in the same word.

1.2 Problem

The process of manually decoding this information is time consuming and prone to error. A decoding program is available that will prompt the user for hexadecimal input values, apply the mask values, then display the descriptions of bits that do not match the expected pattern. It is up to the user to remember the patterns from each individual decoding, and to construct a complete signature from the many hexadecimal words. This process actually must be performed twice, once for the *actual* signature and once for the *expected* signature. Comparison of the two signatures relates the changes that have occurred in the configuration since the last state update.

2 Description

A classifier can perform this decoding task easily through deductive and default reasoning. The decoding program can be extended to isolate each bit in the configuration words and to generate a *proposition*[1] for a database stating the observed position of each valve or switch. The classifier can then attempt to generate a state description for these indications. The state descriptions offer an explanation in high-level, intuitive, terminology. For example, instead of being offered the propositions

p_1 = *The manifold 1 ox open indication is present*
p_2 = *The manifold 1 fu open indication is present*
p_3 = *The manifold 1 ox close indication is not present*
p_4 = *The manifold 1 fu close indication is not present*

the flight controller should be informed

p_5 = *The manifold 1 valves are open*

[1]The term *proposition* is used here instead of the expected *fact* in order to provide consistent terminology with the deductive reasoning systems discussed throughout the paper.

due to the application of a typical rule r_1:

$r_1 =$ if $p_1 \wedge p_2 \wedge p_3 \wedge p_4$ then
assert $p_5 =$ *The manifold 1 valves are open*,
and retract p_1, p_2, p_3, and p_4.

Better still, if the following propositions are available,

$p_5 =$ *The manifold 1 valves are open*
$p_6 =$ *The manifold 2 valves are open*
$p_7 =$ *The manifold 3 valves are open*
$p_8 =$ *The manifold 4 valves are open*
$p_9 =$ *The manifold 5 valves are open*

then the best description is

$p_{10} =$ *All manifolds are open*

from the rule r_2:

$r_2 =$ if $p_5 \wedge p_6 \wedge p_7 \wedge p_8 \wedge p_9$ then
assert $p_{10} =$ *All manifolds are open*,
and retract p_5, p_6, p_7, p_8 and p_9.

Carrying on to "meta-level" statements regarding a "configuration of configurations," one might make the specialization of the propositions

$p_{10} =$ *All manifolds are open*
$p_{11} =$ *Both regulators are open*
$p_{12} =$ *Both crossfeed valves are closed*
$p_{13} =$ *All tank isolation valves are open*
$p_{14} =$ *All thruster heaters are off*

resolve to the implicit description

p_{15} =*Prelaunch configuration*

Such descriptions explain implicitly the underlying meaning. In this sense, *the output of the production system is itself the explanation of the reasoning process.*

2.1 Specialization

The sort of classifier described above has been implemented through the use of a production system shell. Statements providing a *specialization of beliefs* are represented conveniently with conventional production rules. The left-hand side of the rule consists of one or more predicate propositions which, when considered together, imply a more specialized statement having equivalent meaning. The right-hand side of the rule asserts the new statement and retracts all of the propositions that were held true in order to activate the rule. This assertion/retraction process decreases the number of propositions in the database, while maintaining equivalent knowledge of the reasoning world. Since the system can retract its own assumptions later in the deduction process, the process is a manifestation of *nonmonotonic reasoning.*

The classifier employs a combination of procedural and declarative programming techniques. NASA's C Language Integrated Production System (CLIPS) provides the rule processing capabilities. The host program, written in C, acquires the necessary data and applies a valuation algorithm to generate database propositions. This algorithm assigns to each positive component position indication a description of the component, a description of the position indication (e.g. *Open*, *Close*, *On*, or *Off*), and a qualifier as to whether that position belongs to the *actual* or *expected* configuration. When all necessary propositions have been generated, the production system evaluates them and builds the state description. The contents of the database after all possible specializations have been applied (i.e. when no more rules fire) represent the state description. The host program expands these remaining propositions into English sentences for display to the users.

2.2 Default Reasoning

Since the independence of valve or switch state indications is not guaranteed by the physical system, the design-intended independence is not considered important by this production system. That is to say, though the valves are intended to reside in either the opened or closed states, the indications may not provide conclusive evidence and perhaps no default assumptions are available. For these situations none of the statements that consider the guilty valve will be applied, thus leaving the lowest level propositions in the database and resulting in a very specific state description. Detection of these situations sometimes leads to further detailed observations of hardware performance in order to obtain alternative cues that support one or more of the indications. Moreover, facts are held based on *observed* states rather than *assumed* states[2].

One important consideration in the solution is that *lack of evidence regarding a position indication is useful information.* That is, missing information may imply a certain position indication. For the OMS and RCS, this happens with the switch positions: lack of an OPEN or CLOSED indication means that the switch is assumed to be in the GPC (General Purpose Computer) position for automatic valve control. Missing information is also important in OMS and RCS valve positions: many valves lack a CLOSED indication, so that if the OPEN indication is not present, then the flight controllers must assume that the valve is closed. For these reasons, the classification process must allow for *default* values for certain propositions.

Recent research efforts attempting to solve default logic problems have centered around extending classical mathematical logics to account for implicit information in the database. This typically is done by making assumptions about missing information by providing default values. In some cases, providing default values is in itself another problem that must be handled in the reasoning system. Etherington [1988] provides a summary of current techniques for handling missing information. Besnard [1989] provides a formal introduction to default logic.

[2]There remains the underlying assumption, however, that the observed state represents the actual state.

In an attempt to restrict the reasoning assumptions to information that is available, the *Closed-World Assumption* (CWA) has been developed [Reiter 1978]. The CWA is the assumption of complete knowledge about which positive facts are true in the world. Under the CWA, it is not necessary to explicity represent negative information. Negative facts may be inferred from the absense of the same positive fact. The CWA corresponds to the knowledge base:

$$\text{if } KB \nvdash P \text{ then infer } \neg P,$$

which states that if the proposition P cannot be derived from the knowledge base KB, then it is reasonable to assume that P is false. Furthermore, one can imagine collecting the set of all false propositions derivable from KB into another knowledge base. Reiter calls this set the *negative extension* of KB, or $\overline{EKB}$.

Traditional logics do not possess means for considering the absence of knowledge. Research has considered two sorts of information types whose implementation can extend the capabilities of traditional logics to cover this shortcoming. In the *positive information* category, one assumes that relevant information is known, therefore anything that is not known must be false. In the *default information* category, one has default values available to fill gaps in the absence of specific evidence. The *default information* category describes the reasoning process embodied by the classifier.

A *default logic* may be constructed from a standard *first-order logic* by permitting addition of new inference rules [Reiter 1980]. These new rules allow *known* and *unknown* premises, making possible conclusions based on missing information. A *default theory*, Δ, is an ordered- pair (D, W) consisting of a set of *defaults*, D, and a set of *first-order formulae*, W. The fundamental statements in Δ are *defaults*, defined by the expression:

$$\frac{\alpha(\bar{x}):\beta_1(\bar{x})\ldots\beta_m(\bar{x})}{\gamma(\bar{x})}\cdot$$

where $\alpha(\bar{x})$, $\beta_i(\bar{x})$, and $\gamma(\bar{x})$ are formulae whose free variables are contained in $\bar{x} = x_1, \ldots, x_n$. This expression states that if certain *prerequisites* α are believed, and it is consistent to belive that certain *justifications* β are true, then it is reasonable to sanction the *consequent* γ. Stated another way, if the prerequisites are known and their justifications are not disbelived, then their consequents can be assumed. Conventionally, if $\beta(\bar{x}) = \gamma(\bar{x})$, then the default is *normal*, and if $\beta(\bar{x}) = \gamma(\bar{x}) \wedge \omega(\bar{x})$, for some $\omega(\bar{x})$, then the default is *semi-normal*. The sets of conclusions sanctioned by Δ are the knowledge base *extensions*.

As a simple demonstration, consider the typical AI example

$$W = \{BLOCK(A) \vee BLOCK(B)\}.$$

If we assume the closed-world defaults

$$D = \{\frac{:\neg BLOCK(A)}{\neg BLOCK(A)}, \frac{:\neg BLOCK(B)}{\neg BLOCK(B)}\},$$

then the theory Δ has the two extensions E_1 and E_2.

$$E_1 = Th(\{\neg BLOCK(A), BLOCK(B)\})$$
$$E_2 = Th(\{BLOCK(A), \neg BLOCK(B)\})$$

This example shows that the system has concluded that either A is a block or B is a block, but not both. The system adds these conclusions to the database as extensions. In elaborate situations it is likely that interactions between defaults may raise conflicts. Semi-normal defaults provide a means for resolving ambiguities between interacting defaults, so long as the interactions are known *a priori* [Reiter and Criscuolo 1981].

Conventional deductive inference involves the *monotonicity* property: as the set of beliefs grows, so does the set of conclusions that can be draw from those beliefs [Ginsberg 1987]. However, if one now adds new information to the set of beliefs, then some of the original conclusions may now be invalidated. The ability to withdraw a previous assumption and reconstruct a new set of conclusions is known as *nonmonotonic reasoning*.

3 Implementation

The pattern classifier presented herein performs default reasoning in a manner analogous to the approach formulated by Reiter. The production system inference engine controls application of the specializations and manages the database. The host program and *deffacts* blocks initialize the database. The host program then calls CLIPS to execute the inference process. After completing the classification, the host program unloads the interesting propositions remaining in the database and displays them to the user.

3.1 Input Processing

Input data can be provided by the user or can be acquired from the telemetry stream via local area network (LAN). If the user provides the data, he is prompted by the host program to enter the configuration word identification tag (or "measurement stimulus identification") and the actual and expected bit patterns (in hexadecimal). When all desired input has been provided, the evalution process begins. The host program unloads the resulting database and parses the remaining propositions into English sentences for display. When the user is satisfied that he understands any configuration descrepancies, he can issue a request to reset the *expected* configuration words to the *actual* configuration words, thus updating the comparison pattern to the known state.

Since there are 90 configuration words recognized by the host program, it is unlikely that the user will provide all possible input. This is of no significance to the classifier, as it will work on whatever propositions are provided, no matter how limited. If very little information can be provided from the configuration words provided. then one should expect low-level results. The more information that is provided, the better the classification. To assist in the data acquisition process, the host program was modified to accept data from a LAN. The network interface requests 24 valve configuration words and 66 switch configuration words from the telemetry stream. These 90 words contain all of the discrete information that pertains directly to OMS and RCS operations[3]. With all of this data, the classifier is able to make the most specific statements possible.

3.2 Providing Defaults

In order to perform reasoning about the default values, a group of special rules were developed. These rules process the *deffacts* statements that are labelled with the *default* token by attempting to match on any overriding fact from the *actual* or *expect* environments. Stated differently, if the default fact is the only one available for a particular valve or switch, then the value provided as the default indication for that component becomes the value of the missing fact. If any evidence other than the default value is available, that evidence is used in the classification process. The rules performing these operations are described in more detail in the following section.

3.3 Production System

The CLIPS inference engine performs all of the deductive reasoning. It is allowed to run through exhaustion, eliminating as many propositions as possible by applying the specialization rules. These rules heavily exploit the pattern matching capabilities provided by CLIPS, due to the symmetric nature of the physical domain. Moreover, the rules work for either of the two configuration states, matching (with restrictions) on the pattern predicate.

The knowledge base construction is rather simple. It consists of *default processing procedures*, *classification schemas*, *configuration comparators*, and *physical system information*. The expertise is explicit in the classification reductions; knowing how to represent a configuration through its operational semantics, and knowing how to manage the associated default assumptions.

The *default processing procedures* are probably the most interesting. These rules fire first so as to build all of the lowest-level indications before starting

[3]Discrete information from other subsystems, such as data processing, indirectly affect OMS and RCS operations, but have not yet been included.

specializations. In order to reason about defaults one must be able to decide when information is missing. This application uses the CLIPS **not** operation for this purpose. This operation returns TRUE if a match is *not* available for the pattern, thus allowing us to determine that default-overriding evidence is not present in the database. Operation of these rules may be described as follows: Given a set of default values in a **deffacts** block,

```
(deffacts default-values
     (default lrcs he-press-a sp-gp)
     (default lrcs he-press-b sp-gp)
     (default lrcs tank-isol-12 sp-gp)
     ...
)
```

we are able to provide a default value for any particular component in the physical system, including those that may be "exceptions."[4] The first entry in the abbreviated table above states that the default position for the Left RCS Helium Pressurization A switch is the GPC position (**sp-gp**). Now, consider the default assertion rule for the expected switch indications,

```
(defrule expect-switch-defaults
     (declare (salience 100))
     (default ?domain ?component ?d&sp-op|sp-cl|sp-gp)
     (not (expect ?domain ?component sp-op))
     (not (expect ?domain ?component sp-cl))
     (not (expect ?domain ?component sp-dm))
     (not (expect ?domain ?component sp-gp))
=>
     (assert (expect ?domain ?componet ?d))
)
```

This rule binds a default indication from the default table (described below), specifying that it handles only switches by restricting the default value

[4]Explicit statement of the default facts is required because the not operator is unable to bind variables for use outside of the not scope.

C-5

to one of the three reasonable switch values (the value of *dilemma* (**sp-dm**), though a possible observed state, is not a reasonable default value). It then proceeds to search for an overriding indication by looking for all possible switch values in the **expect** indications. If a match is found, then an **expect** indication is available and the rule fails. If no match is found, then the default value is assumed appropriate, the rule fires, and the default value is asserted as the **expect** value on the right-hand side. Similar rules exist for reasoning about the **actual** indications and for valves.

Most of the production rules represent the *pattern classification schemas*. As described, these rules assemble collections of facts into a more specialized fact implying the same information. The right-hand side of the rule retracts the premises and asserts the conclusion. Each of these rules works for either of the two comparison states. Recalling the manifold example provided above, the classification schema for this specialization appears as the rule:

```
(defrule specialize-group-manifolds
  ?m1 <- (?mode&actual|expect ?domain manifold-1 ?s ?v)
  ?m2 <- (?mode               ?domain manifold-2 ?s ?v)
  ?m3 <- (?mode               ?domain manifold-3 ?s ?v)
  ?m4 <- (?mode               ?domain manifold-4 ?s ?v)
  ?m5 <- (?mode               ?domain manifold-5 ?s ?v)
=>
  (retract ?m1 ?m2 ?m3 ?m4 ?m5)

  (assert (?mode ?domain manifolds ?s ?v))
)
```

This rule collects all five of the named manifolds for an arbitrary domain (Left RCS, Right RCS or Forward RCS) and either environment (**actual** or **expect**). Provided that the switch and valve positions (**?s** and **?v**) for each manifold are the same, the special conclusion **?domain manifolds** is asserted. Prior to the special assertion, however, the antecedants are retracted from the database[5]. If not all of the five manifolds indicate the

[5]The retraction is performed before the assertion in order to reduce the complexity of driving patterns through the network.

same valve and switch positions, this rule will fail for that domain. This will leave the individual (lower-level) facts in the database for the display utility, thus maintaining the highest level of specialization possible without introducing ambiguity.

Two *configuration comparison* procedures perform the comparison between the *actual* and *expected* configurations. These rules fire last, allowing all possible specialization to take place before evaluating the differences between the two configurations. Simply put, if the **actual** and **expect** equivalents for any one component or configuration are not the same, then the configuration is declared a **mismatch**. This simple rule performs those actions:

```
(defrule config-mismatch
     (declare (salience -100))
     ?ce <- (expect ?domain ?set $?des)
     ?ca <- (actual ?domain ?set $?ind)
     (test (neq $?des $?ind))
=>
     (retract ?ce ?ca)

     (assert (mismatch ?domain ?set $?des $?ind))
)
```

The **des** and **ind** variables are multifield variables because they can bind to either one or two fields, depending on the degree of specialization achieved for any one component. Through the test operation, we see that if the multified variables are not the same, then the mismatch is declared. A similar rule, **config-valid**. is used to assert **confirmed** configurations.

There are only a few facts that remain fixed in the application. These are the *physical system information* facts. All of these facts were installed in order to reduce the number of rules required to manage only slightly different configurations. These facts relate the interdependence among various components in the physical system, and enforce some degree of control over variable binding when a model requires information about a component and another "corresponding" or "associated" component. For example, the **deffacts** block:

```
(deffacts relationships
        (corresponding loms roms)
        (corresponding roms loms)
        (corresponding lrcs rrcs)
        (corresponding rrcs lrcs)
)
```

is used to associate the name of the system related to (but not identical to) the system under consideration. Using the first fact, **(corresponding loms roms)**, the token **roms** becomes available when reasoning about the **loms**. This is handy when trying to determine special hardware configurations where one system is connected to another.

3.4 Post-Processing

The existing hexadecimal decoding program was modified slightly so as to accomodate CLIPS fact processing. For each of the bit descriptions, a fact-like sentence was attached to the corresponding data structure. When this bit is given a value and the classifier is subsequently invoked, the associated sentence is *string-asserted* into the fact list. The program was modified to search the fact list for any **mismatch**, **confirmed**, **actual** and **expect** facts upon return from the classifier. Since the first two fields completely define the structure of the English sentence used to describe the fact, the parse tree is rather simple. The fact fields are assembled into a string using **sprintf()**, then sent to the display processor.

The host program "knows" a few things about CLIPS data structures. Since the output is required to be processed on a graphics terminal running under a window manager, display management has been delegated to the host program instead of the production system. Therefore, in order to parse the facts that remain in the database, a simple procedure for processing the facts list was developed. This procedure steps through the linked fact list, searching for facts whose first token identifies an item of interest to the user, i.e. those with a **mismatch** or **confirmed** token. Once it finds a match, the remaining tokens in that fact are assembled into a text string.

with a prespecified format, then passed to the graphics processor for display. A typical output may appear as follows:

```
Configuration Evaluation:

1] Difference in rrcs manifold-1 indication:
   expected open, actual closed.
2] Difference in rrcs manifold-2 indication:
   expected open, actual closed.
3] Difference in rrcs he-press-a indication:
   expected closed, actual open.
4] Difference in rrcs he-press-b indication:
   expected open, actual closed.
```

4 Examples

This section presents a number of examples stressing the various levels of specialization involved in the classifier. Though the real application of the classifier appears in a workstation environment requiring 1440 bit-description inputs, this sequence of cases demonstrates the reasoning capabilities of the system without requiring the normal input or interpreted output. This sequence shows each level of specialization available for full-input classifications.

Default Assumption Given the default fact

```
(default lrcs he-press-a sp-gp)
```

in the `default-values` construct, the `actual switch defaults` rule checks for existence of the facts

```
(actual lrcs he-press-a sp-gp),
(actual lrcs he-press-a sp-op), and
(actual lrcs he-press-a sp-cl).
```

If we say that none of these facts exist, then this rule will fire and assert the fact

```
(actual lrcs he-press-a sp-gp)
```

per the default value.

Discrete Specialization Given the input statements

```
(actual lrcs he-press-a ox-op)
(actual lrcs he-press-a fu-op)
```

the discrete specialization rule matches a combination pattern from the valve discrete summary facts

```
(combine ox-op fu-op vp-op)
```

reducing the two discrete position statements to the one statement

```
(actual lrcs he-press-a vp-op)
```

This process reduces the lowest-level discretes for this valve, *oxidizer valve open* and *fuel valve open*, into the summary statement *valve position open*.

Valve and Switch Assembly Now that the switch and valve positions are available, they can be assembled into one statement that describes the situation about each component. This operation takes two four-field facts, representing *almost* identical information, and creates a five-field fact. Drawing from the examples above, this operation will take the two facts

```
(actual lrcs he-press-a vp-op)
(actual lrcs he-press-a sp-gp)
```

and create the specialized fact

```
(actual lrcs he-press-a sp-gp vp-op).
```

This might seem unusual, but it is actually quite effective. The process of constructing the classification through this point has been one of determining the *appropriate* low-level signatures. By allowing each indication to exist as a single proposition in the early stages, the system has provided a consistent mechanism for managing default values.

Actual/Expected Comparison Each of the steps outlined above is performed for both the actual and expected signatures. The *actual* and *expect* keywords define the environment in which the associated signature applies. In the examples above, the classifier would eventually determine the *expect* fact corresponding to the *actual* fact that was demonstrated:

```
(expect lrcs he-press-a sp-gp vp-op).
```

So far there are no differences between the two modes. But the purpose of the two different signatures is to provide a mechanism for determining the differences between the two. This is performed by the `config mismatch` and `config valid` rules. The `config valid` rule determines whether both states indicate the same values. If they do, then the statement

```
(confirmed lrcs he-press-a sp-gp vp-op)
```

might be asserted, for example. If the two states do not agree, then the `config mismatch` rule takes affect. Suppose the expected state for the `lrcs he press a` valve is something different:

```
(expect lrcs he-press-a sp-cl vp-cl).
```

Then the `config mismatch` rule would fire because the two states for the same component are different, asserting:

```
(mismatch lrcs he-press-a sp-cl vp-cl sp-gp vp-op).
```

This has detected that the valve, expected to be closed, is now open. These two rules possess low salience so that they are not fired until all of the specializations are complete. These rules operate upon components as well as *configurations*, which are described below.

Valve Group Specialization Now that the individual component descriptions have been assembled into the composite facts. collections of these component facts can be specialized into configuration facts. The **valve groups** structure provides the unifying information. For example, assume that the fact

```
(actual frcs tank-isol-12 sp-op vp-op)
(actual frcs tank-isol-345 sp-op vp-op)
```

were generated by the reasoning sequence described above. Given the **valve groups** fact

```
(valve-group tank-isols tank-isol-12 tank-isol-345)
```

then the **specialize group** rule can make the specialization

```
(actual frcs tank-isols sp-op vp-op).
```

Regulator Operation Specialization The most unusual configuration specialization is that of describing the regulator configurations. The propellant tanks have two pairs of regulators each, and can be operated from both, one or none of the individual pairs. Moreover, the switches controlling the plumbing path to these regulators can be in manual or automatic positions. The approach to solving this problem involves the regulator descriptions from **reg desc table**. and steps analogous to those used for other valve components. The rule **reg check** attempts to match associated regulators, A and B, with an entry in this table. If we add the fact

```
(expect lrcs he-press-b sp-cl vp-cl)
```

to the facts considered above, then this fact and the associated one for the A regulator will be matched with the table entry

```
(reg-config sp-cl vp-cl sp-cl vp-cl man regs-0)
```

to create the specialization

```
(expect lrcs reg-config man regs-0)
```

which contains a lot of meaningful intuitive information[6].

Configuration Specialization Now that pieces of each system have been assembled into configurations, the configurations themselves can be collected into even higher-level statements describing each individual system. These specializations are rules only (due to the idiosyncracies of each system), such as **`rcs feeding manual`**, **`active frcs auto`**, etc. For example, the **`rcs feeding manual`** rule states that if the RCS tank isolation and crossfeed valves are all open, then one can conclude that that RCS is providing crossfeed propellant to another system. This terminology is derived from the actual operations lingo, and is quite meaningful to OMS/RCS console operators. The facts generated by this level of configuration specialization contain the keyword `config` within the fact.

Meta-Configuration Specialization Once the individual system configurations have been determined, it might be possible to assert a more general statement about the "big picture." The *meta-configurations* are essentially *configurations of configurations*. They describe, in one statement, the operational evaluation of all five propellant systems. For input values representing no "problems," the classifier is able to specialize all the way up to this level, deducing a statement such as "Prelaunch configuration." This statement says something about the whole orbiter, and from training flight controllers know that this means the LOMS is feeding crossfeed, the ROMS is active, the RCS systems are in their launch configurations, the OMS regulators are in the auto-closed position, and the RCS regulators are in the manual-open position. Pattern groups representing each of these configurations appear in the rule *prelaunch config*. The effect of this process is to reduce over 100 low-level facts into the one statement

`(actual prelaunch config nominal nominal)`.

Furthermore, the host program interpreter parses this statement to the declaration:

[6]At least to a flight controller.

```
Actual configuration:  PRELAUNCH.
```

5 Enhancements

There are a number of areas for enhancement in the present system. A few of the reasoning extensions are identified below. One obvious quality extension is to change the configuration descriptions to reason about the other orbiter subsystems, such as Data Processing, Life Support, or Electrical Power. Flight controllers responsible for each of these subsystems must monitor telemetry information similar to that monitored for OMS and RCS operations.

5.1 Dynamic Reasoning

Comparing an *actual* signature with an *expected* signature can be interpreted as a matter of temporal persistence. If we can make assumptions about the dynamic behavior of the measured system, then we can draw from knowledge of the *expected* state to help make assumptions about the *actual* state. Often the behavioral assumptions refer to the deduction process, where one might assume *minimum inferential distance* [Touretzky 1986]. Temporal considerations are typically categorized under the *Frame Problem*, as described by Minsky [1975], Hayes [1979], Shoham [1987]. Hanks and McDermott [1986], and many others. An interesting enhancement to this system might be found in *predicting the next configuration signature* by incorporating knowledge of procedures and time [Georgeff and Lansky 1987].

5.2 Analog Information

Though the information provided as input to the classifier currently is discrete (binary), there is no reason why analog information may not be added. For instance, some valves on the orbiter do not have discrete position indications, but rather "percentage open" indications. There are published guidelines for interpreting "percentage flow" through these valves that could be implemented as rules with thresholds on their left-hand sides. If a valve is indicating 2 percent open, for example, the interpretation will probably lead to considering this valve closed.

5.3 Instrumentation Failures

A variety of problems may be introduced into the classification process by supplying nonrepresentative signatures as input. There are many orbiter component failures that will cause an invalid signature to be relayed to Mission Control. For example, failure of a computer, demultiplexer, signal conditioner or transducer will cause all of the telemetry measurements associated with those components to be incorrect, without affecting operation of the measured device. These conditions are detectable, however, and can be provided as input to the classifier. When the classifier made aware an instrumentation component failure, and it "knows" the measurements that come from that component, then it can take this invalid information into account when performing the classification. The heuristics for interpreting the actual signature will likely involve *minimum entropy*, *persistence* and *default reasoning*.

6 Evaluation

This classifier performs extremely well for its intended purpose. There is no apparent hindrance to extending the system to incorporate more input or accomodate more configuration models. Adding this configuration evaluator to an existing program shows the capabilities of an add-on expert system. This application derives most of the benefits for developing an expert system outlined by Giarratano and Riley [1989] (the other benefits are not applicable). For example, due to the declarative construction, the system is able to accomodate changes in orbiter procedures without restructuring the inference process. The application performs a complete task, allowing flight controllers to address their attention to other problems. Most importantly, the expert system is able to perform a mundane task frequently, consistently, and cheaply, and considering the quantity of input, at the level of an expert.

The certified program will be used during all phases of the Shuttle mission to interpret hexadecimal and binary information and to provide a description of the onboard valve and switch configuration. All of the classifications performed thus far in the development process have taken under 6 seconds to complete. This is a highly acceptable amount of time for this activity.

As familiarity with this classifier increases, the users will likely conclude that there are more statements that can be made about spacecraft configurations than have been included in the rule base. There are many subtle descriptions about off-nominal configurations that may prove to be worthwhile in a robust system. The extensibility of the production system will allow such additions to be made without changing the inferencing mechanism or worrying about rule ordering.

References

[Besnard 89] Besnard, *An Introduction to Default Logic*, Springer-Verlag, Berlin, 1989.

[Etherington 88] Etherington, *Reasoning with Incomplete Information*, Morgan Kaufmann Publishers, Inc., Los Altos, CA. 1988.

[Georgeff and Lansky 87] Georgeff and Lansky, "Procedural Knowledge," SRI International Technical Note 411, Menlo Park. CA. 1987.

[Giarratano and Riley 89] Giarratano and Riley. *Expert Systems: Principles and Programming*, PWS-Kent Publishing Company. 1989.

[Ginsberg 87] Ginsberg, *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann Publishers. Inc., Los Altos, CA. 1987.

[Hanks and McDermott 86] Hanks and McDermott, "Default Reasoning, Nonmonotonic Logics, and the Frame Problem," in *Proceedings* *of the Fifth National Conference on Artificial Intelligence*, AAAI, 1986.

[Hayes 79] Hayes, "The Logic of Frames," in *Frame Conceptions and Text Understanding*, Metzing (ed.), McGraw Hill, New York, 1979.

[Minsky 75] Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, Winston (ed.), McGraw Hill, New York, 1975.

[Reiter 78] Reiter, "On Closed-World Data Bases," in *Logic and Data Bases*, Gallaire and Minker (eds.). Plenum Press, New York, 1978.

[Reiter 80] Reiter, "A Logic for Default Reasoning," *Artificial Intelligence 13*, North-Holland, 1980.

[Reiter and Criscuolo 81] Reiter and Criscuolo, "On Interacting Defaults," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981.

[Shoham 87] Shoham, "What is the Frame Problem?", in *Reasoning About Actions and Plans: Proceedings of the 1987 Workshop*, Georgeff and Lansky (eds.), 1987.

[Touretzky 86] Touretzky, *The Mathematics of Inheritance Systems*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.