

Implementing  
Problem Solving Methods  
in CYC

Dimitrios Sklavakis

MSc in Artificial Intelligence  
Department of Artificial Intelligence  
University of Edinburgh  
1998



## Abstract

Although the CYC system is very good in answering questions through backtracking in its large knowledge base (KB), it can be illustrated in the case of systematic fault diagnosis that this is not enough to solve more complex problems that demand dynamic collection of information and KB updating. A solution proposed in this thesis is to provide CYC with a richer set of inference mechanisms and with the means to combine them into more complex problem-solving procedures.

The KADS methodology provides an excellent source both for inference mechanisms, its *inference types*, and ways to combine them, through its *inference structures* and *Generic Task Models*. Therefore, KADS provides the means to develop a rich set of inference mechanisms which will improve the reasoning power of CYC. In addition, KADS provides a systematic, task-oriented approach to knowledge acquisition which naturally complements the task-independent development of CYC's *Upper Ontology*. The combination of these two approaches provides the means to fill in the knowledge gap between these two extremes of the Ontology pyramid; the task-specific knowledge, which forms the base, and the Upper Ontology, which forms the top. A relevant issue discussed, is the problem of "brittleness" of Expert Systems, an issue fundamental for the creation and use of CYC.

The actual implementation of the Systematic Diagnosis problem solving method for faults in PCs and Automobiles, described in this thesis, demonstrates a way to implement KADS problem solving methods in CYC. The implementation maps directly the various layers of KADS *Expertise model* onto CYC's KB and its LISP-like programming language, SubL. The implemented system thus combines the declarative richness, transparency and expressiveness of the CYC KB with the conceptual analysis and structured search represented in the problem solving method.



### **Acknowledgements**

The research reported here was carried out in the Department of Artificial Intelligence (DAI) with the cooperation of the Artificial Intelligence Applications Institute (AIAI). I would like first of all to thank my supervisors, Dr Helen Pain, from DAI, and Dr Stuart Aitken, from AIAI , for their patient and careful guidance during the course of the research reported here. Finally, I thank the Greek Ministry of National Economy, Directorate of International Economic Organizations, who provided financial support during my year of study via NATO Science Fellowship 107559/DOO 1094/25-7-97.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The Problem . . . . .	1
1.2	Motivation . . . . .	2
1.3	Summary of Results . . . . .	3
<b>2</b>	<b>The Component Technologies: KADS and CYC</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	The KADS Methodology . . . . .	5
2.2.1	Expertise Analysis . . . . .	6
2.2.2	Generic Task Models (GTMs) . . . . .	12
2.3	The CYC System . . . . .	14
2.3.1	The CYC Knowledge Base (KB) . . . . .	14
2.3.2	The CycL Representation Language . . . . .	19
2.3.3	The CYC Inference Engine . . . . .	22
2.3.4	Interface Tools . . . . .	23
2.3.5	Interface Tools - The KB Browser . . . . .	23
2.3.6	Interface Tools - The Knowledge Editing Text (KE Text) facilities	24
2.3.7	Interface Tools - The SubLanguage (SubL) Interactor . . . . .	29
2.3.8	Interface Tools - The Functional Interface (FI) . . . . .	29
2.4	Summary . . . . .	29
<b>3</b>	<b>The Analysis Phase</b>	<b>31</b>
3.1	Introduction . . . . .	31
3.2	Knowledge Acquisition for the Domain Layer . . . . .	32
3.3	Selecting a Generic Task Model . . . . .	33
3.4	Domain Roles and Domain Layer . . . . .	34
3.4.1	The System Model . . . . .	36
3.4.2	The Testing Knowledge . . . . .	40
3.4.3	The Diagnosis Context . . . . .	42
3.5	The Inference Layer . . . . .	44
3.6	The Task Layer . . . . .	44
3.7	Summary . . . . .	46

<b>4</b>	<b>The Implementation Phase in CYC</b>	<b>47</b>
4.1	Introduction . . . . .	47
4.2	The Overall Model . . . . .	48
4.3	The Implementation of The Domain Layer . . . . .	49
4.3.1	The PC Diagnosis Microtheory. . . . .	50
4.3.2	The System Model. . . . .	50
4.3.3	The Testing Knowledge . . . . .	53
4.4	The Implementation of the Inference and Task Layers . . . . .	56
4.4.1	Maintaining the Declarative Scheme . . . . .	63
4.5	Extending the system . . . . .	64
4.5.1	Changes of Microtheories . . . . .	64
4.5.2	Changes in the Domain Layer . . . . .	65
4.5.3	Changes in the Inference and Task Layers . . . . .	66
4.6	Overview . . . . .	68
<b>5</b>	<b>Issues and Results</b>	<b>69</b>
5.1	Introduction . . . . .	69
5.2	Problem Solving in CYC . . . . .	70
5.2.1	Inferencing in CYC . . . . .	71
5.3	Problem Solving Methods in CYC . . . . .	72
5.3.1	Implementing the Systematic Diagnosis PSM . . . . .	72
5.3.2	The Implemented System . . . . .	73
5.4	KADS and CYC . . . . .	77
5.5	Further Work . . . . .	79
5.6	Conclusions . . . . .	80
	<b>Appendices</b>	<b>86</b>
<b>A</b>	<b>The CYC FI Function Reference</b>	<b>87</b>
<b>B</b>	<b>The CYC KE-Text for PC Domain</b>	<b>91</b>
<b>C</b>	<b>The CYC SubL Code for PC Domain</b>	<b>121</b>
<b>D</b>	<b>The CYC KE-Text for PC/Automobile Domains</b>	<b>131</b>
<b>E</b>	<b>The CYC SubL Code for PC/Automobile Domains</b>	<b>171</b>



# List of Figures

2.1	A hierarchy of Inference types . . . . .	9
2.2	An Inference Structure for diagnosing faults in an audio system . . . . .	10
2.3	Task Structure for Systematic Diagnosis (pseudo-code) . . . . .	11
2.4	Relationship of Generic Task Model to Expertise Model . . . . .	12
2.5	A hierarchy of Generic Task Models in the library . . . . .	13
2.6	An Example of genls and isa . . . . .	17
3.1	Inference structures for Systematic Diagnosis (left) and Heuristic Classification (right) GTMs . . . . .	34
3.2	The Task Structure for Systematic Diagnosis PSM for PCs . . . . .	45
4.1	The Overall Model (schematic view) . . . . .	49
4.2	The System Model Hierarchy in CYC . . . . .	51
4.3	The Test Knowledge Hierarchy in CYC . . . . .	54
4.4	The Overall Model (detailed view) . . . . .	58
4.5	Starting the diagnosis session . . . . .	61
4.6	Presenting a Test to The User . . . . .	61
4.7	New Hypothesis After a “Normal” Result Type . . . . .	61
4.8	Decomposing a PCSystem After a “Not Normal” Result Type . . . . .	62
4.9	Confirming a PCComponent After a “NotNormal” Result Type . . . . .	62
4.10	New Test After an “Insufficient” Result Type . . . . .	62
4.11	Changes in the Microtheories . . . . .	65
4.12	The Changes in the Domain Layers . . . . .	67
5.1	Filling the Knowledge Gap . . . . .	78



# List of Tables

3.1	The Systematic Diagnosis Inference Types and their Domain Roles . . .	35
3.2	Domain Roles for Systematic Diagnosis and the Localisation equivalents .	35
3.3	The Inference Layer of the Systematic Diagnosis PSM for PCs . . . . .	45
4.1	The SubL Support Functions . . . . .	63



# Chapter 1

## Introduction

### 1.1 The Problem

The principal goal of this thesis is to take a Problem Solving Method (PSM) from the Knowledge Based System (KBS) development methodology KADS ([Schreiber *et al.* 93], [Tansley & Hayball 93]) and implement its component inference steps in Doug Lenat's CYC KBS ([Lenat & Guha 90]).

Problem Solving Methods (PSMs) are conceptual models of problem solving which describe the elementary inference steps that must be performed for the solution of various tasks such as diagnosis, planning and design. They were developed in the context of methodologies for KBS design such as KADS. The task considered in this thesis is diagnosis and especially fault diagnosis in Personal Computers (PC). A small - but conceptually significant - extension of the task in the domain of Automobile fault diagnosis is also considered.

The implementation medium for the diagnosis PSM is CYC, a large Knowledge Base (KB) designed to contain enough knowledge to perform common sense reasoning. It is the product of a ten-year project, started in 1984 and ended in 1995, and is still under development ([Lenat 95]). The motive for building CYC was to overcome the "brittleness" of Expert Systems (ES), their inability to fall back on "first principles" when they encounter novel situations. The key problem was regarded to be analogical reasoning in all its forms. The overwhelming attempt to develop this kind of reasoning was divided in

two sub-goals, as they are quoted from [Lenat & Guha 90]:

1. Breaking down the phenomenon, i.e. analogical reasoning, into its various subtypes and then handling each one.
2. Having a realistically large pool of (millions of) objects, substances, events, sets, ideas, relationships, etc., to which to analogize.

The development of CYC was directed mostly toward the second goal and much less to the first one. As a result, CYC now contains  $10^6$  common sense axioms expressed using a vocabulary of  $10^5$  concepts and just a handful of inferencing methods (backward and forward chaining, modus ponens, modus tollens, equality). The main method is backward chaining with resolution supported by special-purpose heuristic modules. The implementational details of CYC inferencing are discussed in the next chapter.

## 1.2 Motivation

The motive for implementing problem solving methods in CYC is to increase its inferencing power through the implementation of more inference types and methods.

*Inference types*, *inference structures* and *generic tasks*, another name for problem solving methods, are the conceptual products of research for the development of the second-generation expert systems. This research focused on distinguishing the knowledge-level ([Newell 82]) of an ES from its implementation level. These issues are very well illustrated in [Steels 90] and [Chandrasekaran 86].

The KADS methodology incorporates systematically all these concepts through its *Expertise Model*, divided into four layers, the *Domain*, *Inference*, *Task* and *Structure* layers, and its library of *Generic Task Models* which are domain-independent models of problem-solving behaviour. All these concepts are discussed in detail in the next chapter. Therefore, KADS provides two features in the same time:

- A complete methodology for developing ES independently from its implementation medium
- Domain-independent problem-solving methods (Generic Task Models) to be implemented

The domain independence of both CYC and KADS and the lack of any task-specific problem-solving methods in CYC are the motivating factors for this thesis. A secondary motive is that, although CYC was built to support the creation of ES, it does not provide any methodology either for the knowledge acquisition of the expert knowledge or for the common sense knowledge that underlies the expert knowledge. All these issues are discussed in the last chapter, under the light of the implementation of one problem-solving method from KADS for the task of PC fault diagnosis. This latter task was chosen since it is one of the most well studied and developed tasks.

### 1.3 Summary of Results

The main result of this thesis is that, the system developed enhances the problem-solving ability of CYC. A simple argument for this is that now CYC can solve a *whole range* of problems (those lending themselves to Systematic Diagnosis) that it could not solve before. This is because the specific problem-solving method demands the dynamic collection of information from the user, the appropriate update of the KB and the consequent, dynamic choice of the inferencing path. The system itself has features that are quite desirable from an ES, like transparency in its reasoning, ability of some justification, ease of maintenance and extension. The “brittleness” is still there, both in the domain and inference levels, but this was a rather expected feature. Overcoming “brittleness” is a promising direction of research rather than a nuisance. This is because this “brittleness” is due to the “distance” between the expert knowledge needed by the system and the restricted common sense knowledge base that the specific CYC system had<sup>1</sup>. This issue is also discussed in the last chapter.

In Chapter 2, KADS and CYC are described in the level of detail necessary to understand Chapters 3 and 4. Chapter 3 describes the Analysis phase, as defined by KADS, applied to the problem of PC fault diagnosis. This chapter also describes the specific problem-solving method, Systematic Diagnosis, together with how and why it was selected. Chapter 4 describes the design decisions and the actual implementation of the

---

<sup>1</sup>The CYC system used contained just a part of the *Upper Ontology* (see at <http://www.cyc.com/cyc-2-1/intro-public.html>).

problem solving method in CYC. Finally, Chapter 5 discuss the results of the implementation as well as further issues and research directions.



# Chapter 2

## The Component Technologies: KADS and CYC

### 2.1 Introduction

In this chapter the two main components of this thesis will be described, namely the KADS methodology and the CYC system. Of course, the description will cover only the parts of these two components that are important for the understanding of the thesis. A detailed description is beyond the scope of this thesis. For a more detailed description the reader must refer to [Wielinga *et al.* 92] and [Tansley & Hayball 93] for KADS and to [Lenat & Guha 90] and <http://www.cycorp.com>, the WWW cite of CYCORP, for CYC.

### 2.2 The KADS Methodology

The KADS methodology is the result of the corresponding European research project (ESPRIT-I P1098)<sup>1</sup>. It does not seem that a single interpretation of the KADS acronym exists. Two possible interpretations are “**K**nowledge **A**cquisition and **D**omain **S**tructuring” and “**K**nowledge-based systems **A**nalysis and **D**esign **S**upport”. However, the KADS acronym stands as a name by itself. It is a comprehensive methodology for

---

<sup>1</sup>The main reference is [Hesketh & Barrett 90]. It is available from: KADS Information, Touche Ross Management Consultants, Peterborough Court, 133 Fleet Street, London, EC4A 2TR, UK

the development of KBS. The initiative for the development of KADS was the lack of any methodology for the development of KBS. Another reason was the awareness from organisations using KBS that the development of this kind of system had a lot in common with the development of other information systems. Therefore, KADS is a complete method for the *Analysis* and *Design* of an information system which may be a KBS by its own or containing a KBS as one of its parts. This fact is reflected in both its Analysis and Design phases and the stages they contain:

1. Analysis phase

- Process Analysis
- Cooperation Analysis
- **Expertise Analysis**
- Constraints Analysis
- System Overview

2. Design phase

- Global Design
- KBS Design

In the Analysis phase, the Expertise Analysis stage is the one concerned with the KBS part of the information system under development and it is the one that is of interest in this thesis. On the other hand, the Design phase will not be needed since it is not used; the pre-selection of the implementation medium, i.e., the CYC KBS, restricts and guides the design decisions that have to be made. A good overview of KADS can be found in [Wielinga *et al.* 92]. For a detailed description of the KADS methodology the reader may refer to [Tansley & Hayball 93].

### 2.2.1 Expertise Analysis

Before describing the Expertise Analysis stage of the Analysis phase, a major characteristic of KADS, which distinguishes it from the usual methods for developing KBS, must be noted.

The main activity when building a KBS is that of *Knowledge Acquisition*. Traditionally this process was viewed as a process of extracting knowledge from an expert, using various methods, and transferring (cf. encoding) this knowledge into the KBS. In KADS, a different view is adopted, regarding knowledge acquisition as a *modelling activity*. The KBS is not regarded as a container filled with knowledge but rather as a computational implementation of a desired behaviour. This behaviour is described in terms of *models*. Each model describes a particular aspect of the overall behaviour of the KBS, emphasizing certain characteristics and abstracting from others. Under this view, every stage of Analysis and Design in KADS produces a corresponding model of the overall KBS. Therefore, the result of Expertise analysis is the *Expertise model*.

The Expertise Model defines the desired problem solving behaviour (expertise) that the KBS must exhibit. It is the construction of this model that distinguishes KADS from other methodologies for information systems' development, such as Structured Analysis/Structured Design (Yourdon method) or Structured Systems Analysis and Design Method (SSADM). The construction of this model is based on two major assumptions:

1. The problem solving knowledge can be distinguished in *domain knowledge* and *control knowledge*. Furthermore, control knowledge can be distinguished in *inference*, *task* and *strategic* knowledge. These distinctions give four different layers of knowledge.
2. These four different layers have limited interaction between each other.

These four layers of the Expertise Model are described in more detail below.

### **The Domain Layer**

In this layer a definition of the static domain knowledge is made, consisting of domain concepts, structures of concepts, attributes of concepts and relations between concepts. This knowledge is static in the sense that it describes some facts about the domain without specifying how this knowledge is going to be used. Therefore, this kind of description makes the knowledge implementation-independent to a certain degree, i.e., it may be used for different reasoning tasks such as diagnosis, teaching, explanation.

The structures of domain concepts and their relations are known in KADS as *Domain structures*. KADS does not provide a definitive and exhaustive formalism for what a Domain Structure should be as this decision depends strongly on the domain and the use of these structures. However, some general-purpose domain constructs are:

- **Concept:** the basic objects in the domain knowledge. It may correspond to either an individual or a collection. E.g., `component`, `system`, `subsystem`.
- **Attributes:** concepts may have attributes, e.g. `age(man)`, where `age` is an attribute of concept `man`.
- **Structure:** a complex object consisting of other concepts. E.g. `address`.
- **Set:** a collection of other domain constructs. All instances must be of the same type, i.e., concepts, structures, sets.
- **Relation:** they may be relations between concepts, e.g. `component isa subsystem` or relations between proper expressions, e.g., `temperature < 0 IMPLIES frozen (water)`.

### The Inference Layer

This layer describes the basic inference capability of the KBS in terms of *inference types* and *inference roles*. It identifies which basic inferences are supported over the knowledge in the Domain Layer but it does not specify when or in what order these inferences actually happen.

**Inference Types** are primitive inference steps that can be performed on the domain knowledge. They are primitive since they are specified in terms of their input/output and their name which is a general description of what they do. For example, the *decompose* inference type takes a structured arrangement of objects and returns a collection of objects; the *select* inference type takes a collection (structured or unstructured) of objects and returns a filtered collection of objects. Therefore, inference types define ways the static domain knowledge may be used. See figure 2.1 for a detailed classification of inference types.

## 1. Concept Manipulation

Generate Concept

- **Instantiate**
- **Generalise**
- **Classify (Identify)**

Change Concept

- **Abstract**
- **Specialise (Refine, Specify)**
- **Assign\_Value (Change\_Value)**

Distinguish Between Concepts

- **Compare**
- **Confirm**
- **Select**

Associate Concepts

- **Match (Associate, Relate, Map)**

## 2. Structure Manipulation

Build or Destroy Structure

- **Assemble (Aggregate, Compose, Augment)**

Re-arrange Structure

- **Transform**
- **Sort**
- **Parse**

Figure 2.1: A hierarchy of Inference types

**Domain roles** define functions that domain structures may perform in various inference types. For example, in a PC fault diagnosis system, a specific component may be either a *hypothesis* to be selected by a *select* inference type or a *conclusion* to be made by a *confirm* inference type. These are two different roles for the same domain concept. Therefore, domain roles describe the static domain knowledge from a more problem-solving specific point of view. Domain roles may be classified according to the way they are used: they may be Input, Output and Intermediary (both Input and Output) roles.

Inference types and Domain roles are combined in *Inference Structures*. An Inference structure is a network of inference types and domain roles. In figure 2.2 an inference structure for fault diagnosis in an audio system is shown.

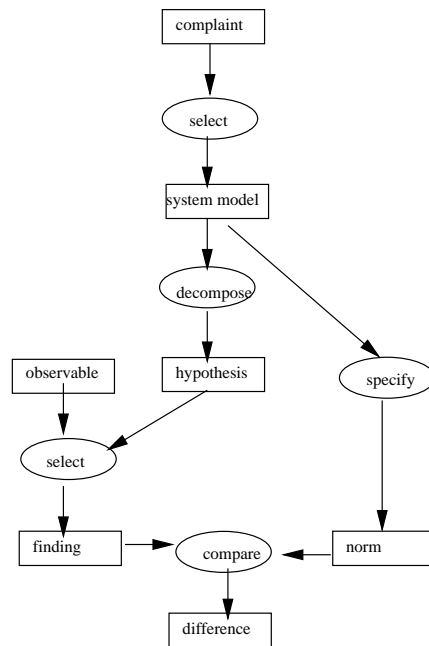


Figure 2.2: An Inference Structure for diagnosing faults in an audio system

### The Task Layer

The Task Layer describes how the individual inferences described in the Inference layer may be sequenced in order to achieve each of the required problem-solving goals. The knowledge in this layer is defined in terms of *Task Structures*. These are usually written

in pseudo-code which comprises simple sequences of inferences combined with some conventional control structures, such as conditionals (IF...THEN...ELSE), repetition (FOR, WHILE, REPEAT), or more complicated, such as pipelining and recursion. In figure 2.3 the Task Structure for performing systematic diagnosis is shown. Variables with a '+' are instantiated (input) while these with a '-' are uninstantiated (output).

```

Systematic Diagnosis(+complaint,+possible observables,-hypothesis) by
  select1(+complaint, -system model)
  REPEAT
    decompose(+system model, -hypothesis)
    WHILE number of hypotheses > 1
      select2(+possible observables, -variable value)
      select3(+hypothesis, -norm)
      compare(+variable value, +norm, -difference)
      system model <- current decomposition level of system model
    UNTIL confirm(+hypothesis), i.e. system model cannot be decomposed
      further

```

Figure 2.3: Task Structure for Systematic Diagnosis (pseudo-code)

### The Strategy Layer

It is the final layer of the Expertise Model. It describes the knowledge for Task Structure selection, sequencing, planning or repairing (when a task fails). This layer will not be described since it is not used here<sup>2</sup>.

---

<sup>2</sup>The main reference used in this thesis for KADS is [Tansley & Hayball 93], which describes a slightly modified version of KADS-I. A newer version, KADS-II or CommonKADS is now developed, but it is compatible with the parts of KADS described in this thesis. The only difference is that the Strategy layer is removed from the Expertise model, which is not important since this layer is not used here.

### 2.2.2 Generic Task Models (GTMs)

One of the goals for the KADS methodology was to overcome the *knowledge acquisition bottleneck*. This is the inherent difficulty in the process of extracting the problem-solving knowledge from an expert and encoding it in a computer system. Although the construction of the Expertise Model as it has been described is a step forward, it still remains a significant effort to construct the Expertise Model from scratch. For this reason KADS provides a library of Generic Task Models. These are Expertise Models (tested and verified) but without a Domain Layer. This relationship is shown in figure 2.4

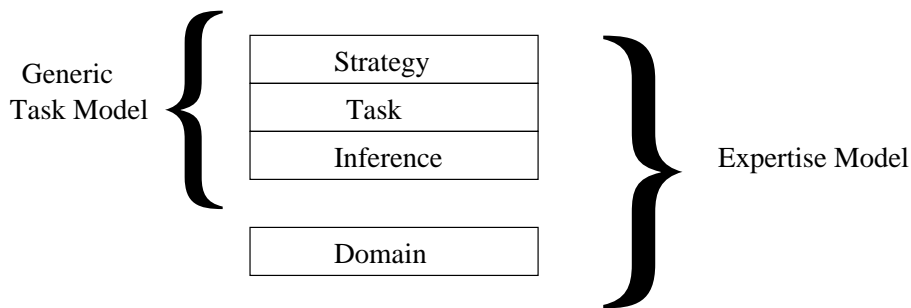


Figure 2.4: Relationship of Generic Task Model to Expertise Model

With GTMs provided, the construction of the Expertise model consists of three activities:

- Analyse Static Knowledge: this is the construction of the Domain Layer.
- Select Initial GTM: the selection of an initial GTM that will guide the construction of the Expertise Model. This selection is guided by the classification of the GTM library into a hierarchy of GTMs according to the specific problem-solving tasks that the KBS has to perform. This hierarchy is shown in figure 2.5. The GTM defines the Inference, Task and Strategy Layer. It also assists in further knowledge acquisition through the description of Domain roles that are needed for the various inference types.
- Construct Expertise Model: completion of the Expertise model by filling in the details of the three upper layers and putting them together with the Domain layer.



## 1. SYSTEM ANALYSIS

- Identification
    - Diagnosis
      - \* Single Model Diagnosis
      - + Systematic Diagnosis
    - \* Multiple Model Diagnosis
    - + Mixed Mode Diagnosis
  - Verification
  - Correlation
    - \* Assessment
  - Monitoring
  - Classification
    - \* Simple Classification
    - \* Heuristic Classification
    - \* Systematic Refinement
- Prediction
    - Prediction of Behaviour
    - Prediction of Values

## 2. SYSTEM MODIFICATION

- Repair
- Remedy
- Control
- Maintenance

## 3. SYSTEM SYNTHESIS

- Design
  - Hierarchical design
  - Incremental design
- Configuration
  - Simple Configuration
  - Incremental Configuration
- Planning
- Scheduling
- Modelling

Figure 2.5: A hierarchy of Generic Task Models in the library

## 2.3 The CYC System

CYC is the implementation system for this thesis. CYC is a very large, multi-contextual knowledge base and inference engine. The project for its development started by the Microelectronics and Computer Technology Corporation (MCC) in the early 1984 and ended in 1995. In that year the CYCORP company was created, to work further on this project. The original idea behind CYC, introduced and developed by Doug Lenat, is that, to perform any kind of reasoning in a consistent and flexible way, any intelligent agent must have a considerable amount of common-sense, pre-scientific knowledge ([Lenat & Guha 90]). This kind of knowledge includes heuristics (rules of thumb) as well as assertions (facts) about the real world that can be known to a mechanical intelligence only if it is told about them. This “teaching” is actually implemented through manual knowledge editing. Using this considerable amount of knowledge - estimated to be at about 10,000,000 rules and facts - the agent can then perform common-sense reasoning and furthermore expert-like reasoning. The CYC system includes the following components<sup>3</sup>:

- The CYC Knowledge Base (KB)
- The CycL Representation Language (CycL)
- The CYC Inference Engine
- Interface Tools

### 2.3.1 The CYC Knowledge Base (KB)

The CYC knowledge base (KB) consists of:

- *constants*, also called *terms* or *units*. Constants form the basic vocabulary of the KB.
- *assertions* about these constants, which include *facts* and *rules*. All the assertions are formally expressed in a representation language, CycL, described below.

---

<sup>3</sup>In the following sections the version of CYC in the Artificial Intelligence Applications Institute (AIAI) is described and only to the extent necessary to understand the thesis.

Each CYC constant is the representation of a concept. In CycL, by convention, the names of constants begin with the prefix '#\$' (read “hash dollar”), e.g. `#$PCComponent`<sup>4</sup>. A constant can represent a *collection* (such as the collection of PC components, `#$PCComponent`), an individual object (such as a particular PC component, `#$VideoCard`), a relation (a predicate, function, e.g., `#$functionalPartOf`) and so on.

All CYC KB constants form a hierarchy of *Collections*, subsets of them and instances of them. These hierarchical relationships are expressed with two special predicates, `#$genls` (meaning “subset of”) and `#$isa` (meaning “element of”). This hierarchy is very important, it forms the *CYC Ontology*, and therefore must be described in more detail.

### The CYC Ontology

CYC constants can either denote sets, like “the set of all PC components”, or individuals, like “the video card”. Every term in CYC is an element of `#$Thing`, the universal collection. `#$Thing` is partitioned into `#$Individual` and `#$SetOrCollection`.

`#$Individual` denotes the set of all things which are not sets. Individuals in the CYC KB include constants such as `#$VideoCard` and `#$Decompose` (inference type).

`#$SetOrCollection` is partitioned into `#$Set-Mathematical` and `#$Collection`. In this thesis we will not use mathematical sets and therefore their properties as far as it concerns CYC are not described. In fact, collections are more important in the CYC ontology and used more often<sup>5</sup>. The important thing to know about collections is that they can have elements. Therefore, they can enter in set-theoretic relations like “element”, “subset” and “superset”.

Membership in a collection is typically expressed as “instance of” or “element of” or “is a”, as in “Monitor is an instance of the collection `#$PCComponent`,” or “Monitor is an element of `#$PCComponent`,” or “Monitor is a `#$PCComponent`.” If the terms

---

<sup>4</sup>All terms in the examples come from the problem domain of PC fault diagnosis as this is analysed in the next chapter. If the meaning of any of the terms prevents the understanding of the examples, the reader can refer to its definition in the next chapter.

<sup>5</sup>The difference between mathematical sets and collections is that the former are defined *extensionally*, i.e. by their members, while the latter are defined *intensionally*, i.e. by their criteria for membership. Therefore, two sets with the same members are equal, while two collections may be still different.

“subset” and “superset” are used with reference to collections, they typically are intended to mean “more specific collection” and “more general collection”, respectively.

The predicate `#$genls` is used to indicate that one collection is more general than another, that it is a “superset”. For example,

```
(#$genls #$PCComponent #$PCSubSystem)
```

indicates that the collection of PC subsystems is a superset of the collection of PC components.

The predicate `#$isa` is used to indicate that a thing is an instance of (element of) a collection, as in

```
(#$isa #$VideoSystem #$PCSubSystem)
($isa #$VideoCard #$PCComponent)
```

where `#$VideoSystem` is stated to be an instance of the collection of PC subsystems, and `#$VideoCard` as an instance of the collection of PC components.

The predicates `#$isa` and `#$genls` are strongly supported by the CYC system code. There are special datastructures and special code routines inside CYC which allow rapid, efficient reasoning about collection-membership and collection-supersethood using `#$isa` and `#$genls`.

To summarise, every CYC constant is an element of at least one collection. In fact, everything that can appear in a CYC expression is an element of some collection. Every collection, with the exception of `#$Thing`, is a subset of at least one other collection. These “subset” and “instance of” relations, expressed with assertions using `#$genls` and `#$isa`, make up the basic framework (ontology) of the CYC KB. Figure 2.6 shows some constants and the `#$genls` and `#$isa` relations between them. In this diagram, the following conventions are introduced:

- CYC constants are represented as text in rectangular boxes.
- A constant with bold text denotes a collection.
- A constant with normal text denotes an individual.
- Assertions involving binary predicates are shown as lines between constants.

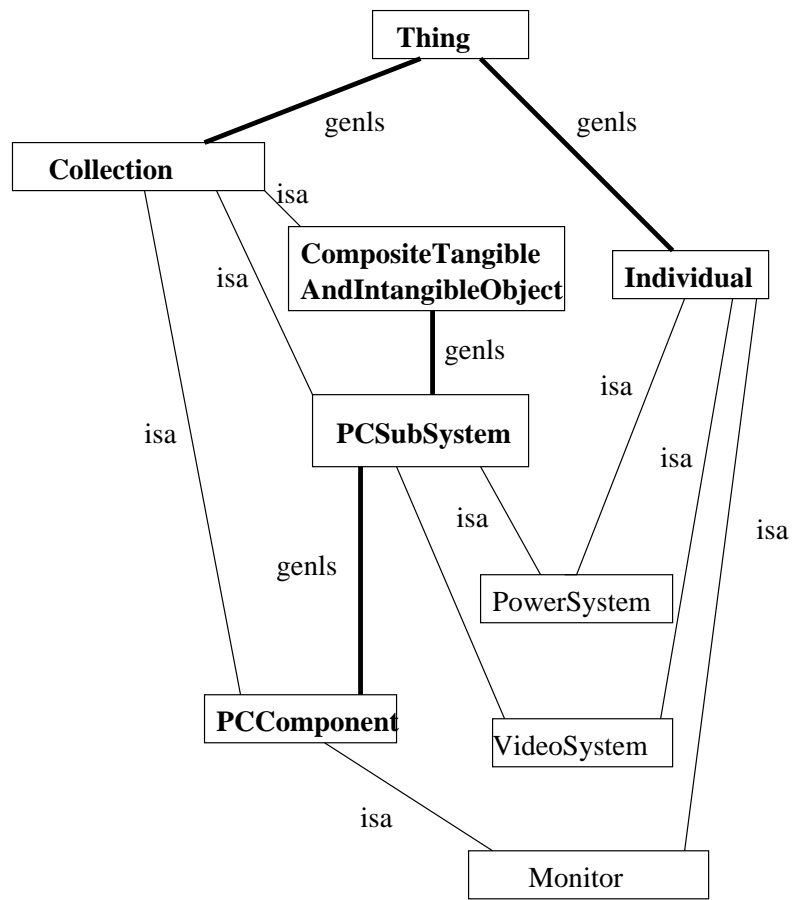


Figure 2.6: An Example of genls and isa

- $\#\$isa$  assertions are shown as thin lines.
- $\#\$genls$  assertions are shown as thick lines.
- More general collections are placed higher on the page than their subsets.
- Collections are placed higher on the page than their instances.

The diagram therefore indicates the following assertions:

```
(#\$isa #\$CompositeTangibleAndIntangibleObject #\$Collection)
(#\$isa #\$PCSubSystem #\$Collection)
(#\$genls #\$PCSubSystem #\$CompositeTangibleAndIntangibleObject)
(#\$isa #\$PCCComponent #\$Collection)
(#\$genls #\$PCCComponent #\$PCSubSystem)
(#\$isa #\$PowerSystem #\$PCSubSystem)
(#\$isa #\$PowerSystem #\$Individual)
(#\$isa #\$VideoSystem #\$PCSubSystem)
(#\$isa #\$VideoSystem #\$Individual)
(#\$isa #\$Monitor #\$PCCComponent)
(#\$isa #\$Monitor Individual)
```

It is important to notice that, even though not directly indicated by the diagram, the following assertions also hold:

```
(#\$isa #\$Monitor #\$PCSubSystem)
(#\$isa #\$VideoSystem #\$CompositeTangibleAndIntangibleObject)
(#\$genls #\$PCCComponent #\$CompositeTangibleAndIntangibleObject)
```

These assertions are implied by two elementary properties:

- If B is a subset of A and X is an element of B, then X is an element of A too. In CYC terms:  $(\#\$genls B A)$  and  $(\#\$isa X B)$  implies  $(\#\$isa X A)$ .
- If B is a subset of A and C is a subset of B, then C is a subset of A too. In CYC terms:  $(\#\$genls B A)$  and  $(\#\$genls C B)$  implies  $(\#\$genls C A)$ .

As mentioned before, CYC has special inference mechanisms for inferring these kind of relationships.

### 2.3.2 The CycL Representation Language

CycL is a formal language whose syntax derives from first-order predicate calculus (the language of formal logic) and from Lisp. The vocabulary of CycL consists of *terms*. The set of terms can be divided into *constants*, *non-atomic terms* (NATs), *variables*, and a few other types of objects. Terms are combined into meaningful CycL *expressions*, which are used to make *assertions* in the CYC knowledge base (KB).

#### Constants

The CycL constants are the same as the KB constants described earlier. They make up the “vocabulary” of CycL. It must be remembered that they (usually) begin with the prefix ‘#\$’ (read “hash-dollar”). These characters may be omitted by certain interface tools, e.g., in the KE text interface described below. Some important naming conventions are:

- All CYC predicate names must begin with a lowercase character.
- All non-predicate constant names must begin with an uppercase character.

#### Variables

They are the common variables of any language. A variable may appear (nearly) anywhere a constant can appear. This gives to CycL some flavour of higher-order predicate calculus but this is not of interest in this thesis. Variable names must begin with a question mark and are ordinarily written in capital letters, e.g., ?TEST. Variables in CycL expressions can be either free or quantified. CycL provides the two main quantifiers of first-order predicate calculus; the universal quantifier `#$forAll`, and the existential quantifier `#$thereExists`. For expressiveness, it also provides three more existential quantifiers: `#$thereExistAtLeast`, `#$thereExistAtMost`, and `#$thereExistExactly`. Free variables are regarded to be universally quantified.

#### Formulas

CycL formulas combine terms into meaningful expressions. Every formula has the structure of a Lisp list: it is enclosed in parentheses, and consists of a list of objects, the

arguments. The first argument may be a predicate, a logical connective, or a quantifier. The remaining arguments may be atomic constants, non-atomic terms, variables, numbers, strings delimited by double quotes (“), or other formulas. The simplest kind of formula is an *atomic formula*, a formula in which the first argument is a predicate, and all the other argument are terms:

```
(#$functionalPartOf #$VideoSystem #$Monitor)
($isa #$PowerSocket #$PCComponent)
($testAfter ?SUBSYSTEM1 ?SUBSYSTEM2)
```

The first two of the atomic formulas above are *ground atomic formulas* (GAFs), since none of the terms are variables.

## Predicates

Every CycL atomic formula must begin with a predicate in order to be well-formed. The number of arguments a predicate takes is determined by its arity. A predicate is described as unary, binary, ternary, quaternary, or quinary, according to whether it takes 1, 2, 3, 4, or 5 arguments. Currently, no CycL predicate takes more than 5 arguments.

The type of each argument must be specified in the definition of the predicate, using the predicates `#$arg1Isa`, `#$arg2Isa`, etc. For example, suppose the predicate `#$resultOfTest` is defined by the following:

```
(#$isa #$resultOfTest #$BinaryPredicate)
($arg1Isa #$resultOfTest #$Test)
($arg2Isa #$resultOfTest #$PossibleObservableValue)
```

To be well-formed, every formula which has `#$resultOfTest` as its first argument must have a term which is an instance of `#$Test` as the second argument, and a term which is an instance of `#$PossibleObservableValue` as its third argument. So,

```
(#$resultOfTest #$Monitor #$BootTime)
```

is probably not well-formed. Though we can never be absolutely certain just from the names, `#$BootTime` could be an instance of `#$PossibleObservableValue`, but `#$Monitor` is probably not an instance of `#$Test`.



### Logical Connectives

Complex formulas can be built up out of atomic formulas or other complex formulas by using logical connectives, which are special constants analogous to the logical operators of formal logic. The most important logical connectives in CycL are `#$not`, `#$and`, `#$or`, and `#$implies`. The three former have the obvious interpretation. The connective `#$implies` takes exactly two formulas as arguments. Like the “if-then” statement of formal logic, it returns true if and only if it is not the case that its first argument is true and its second argument is false. Here’s an example:

```
(#$implies
  ($and
    ($diagnosisContext #$BootTime)
    ($possibleHypotheses ?SUBSYSTEM)
    ($testFirst ?SUBSYSTEM))
  ($hypothesis ?SUBSYSTEM))
```

Assertions involving `#$implies` are very common in the CYC KB. We also call them *conditionals* or *rules*, and we often refer to the first argument as the *antecedent* and the second argument as the *consequent*. In the previous example, the antecedent is

```
(#$and
  ($diagnosisContext #$BootTime)
  ($possibleHypotheses ?SUBSYSTEM)
  ($testFirst ?SUBSYSTEM))
```

and the consequent is

```
(#$hypothesis ?SUBSYSTEM))
```

### Assertions

CycL formulas are used by Knowledge Editors (KEs) to enter assertions in the CYC KB and to ask questions to the KB. However, KB assertions are more than CycL formulas. They consist of many elements of which the most important are:

- a CycL formula: Formulas have been described in the previous section.
- a microtheory: Every assertion is contained in a single microtheory. A particular formula may be asserted into (or concluded in) more than one microtheory; when this is the case, there will be an assertion which has that formula in each of those microtheories. The largest number of assertions are currently in the `#$BaseKB`. All the assertions relative to the PC fault diagnosis are in the `#$PCDiagnosisMt` microtheory.
- a truth value: Attached to every assertion is a truth value that indicates its degree of truth. CycL contains five possible truth values, of which the most common are default true and monotonically true.

By default, GAFs which begin with the predicates `#$isa` and `#$genls` are monotonically true, while all other assertions (including rules) are default true.

- a direction: Direction is a value associated with every assertion that determines when inferencing involving that assertion should be performed. There are three possible values for direction: *forward*, *backward*, and *code*. Inferencing involving assertions with direction forward is performed at assert time (that is, when a new assertion is added to the KB), while inferencing involving assertions with direction backward is postponed until a query occurs and that query allows backward inference. By default, GAFs have direction forward, while rules have direction backward. Changing the direction of rules to *forward* enables forward reasoning.
- a support: CYC uses a Truth Maintenance System (TMS) for its assertions. The support is the known support list of a TMS.

### 2.3.3 The CYC Inference Engine

The CYC inference engine handles modus ponens and modus tollens (contrapositive) inferencing, universal and existential quantification, and mathematical inferencing. It uses contexts called microtheories to optimize inferencing by restricting search domains. CYC also includes several special-purpose inferencing modules for handling a few spe-

cific classes of inference. One set of modules handles reasoning concerning collection membership, subsethood, and disjointness. Another handles equality reasoning.

Inferencing is initiated by an ASK operation. An ASK performed with direction *:forward* will simply do KB lookup; an ASK performed with direction *:backward* will initiate backward inferencing. Backward inferencing can be regarded as a search through a tree of nodes, where each node represents a CycL formula for which bindings are sought, and each link represents a transformation achieved by employing an axiom in the knowledge base.

### 2.3.4 Interface Tools

The Interface Tools that are of interest for this thesis are:

- The CYC Knowledge Base Browser
- The Knowledge Editing Text (KE Text) facilities
- The SubLanguage Interactor (SubL)
- The Functional Interface (FI-interface)

### 2.3.5 Interface Tools - The KB Browser

The CYC KB Browser is the main interface tool for accessing the CYC Knowledge Base (CYC KB). It provides a means for browsing the KB in a number of different ways, a means for querying the KB, and (for registered users) a means for modifying or adding to the KB itself. From the KB Browser, it is possible to reach virtually all other areas of the CYC System simply by following HTML links. Through the KB Browser the following operations may be performed:

- Creating, Viewing, Searching for and Editing Constants
- Adding, Viewing, Searching for and Editing Assertions

### 2.3.6 Interface Tools - The Knowledge Editing Text (KE Text) facilities

#### Introduction

KE Text is an ascii text format for specifying changes to a CYC KB. It uses a mixed “frame and formula” syntax and is batch-processed to add those changes to a CYC Server machine. KE Text (Knowledge Editing Text) is handled by two facilities: KE-File, which loads a file in KE Text format, and the Compose page in the CYC Web Interface.

#### KE Text Syntax

KE Text syntax is just a syntactic/notational variation of CycL. To some extent, it is a holdover from when CYC was a frame-based system and CycL was a frame-based language.

#### KE Text Syntax - Notation:

##### Variables

Variables occurring anywhere in a KE text (e.g., inside rule statements) must begin with a question mark (?).

##### Constants

Known constants (i.e., constants which CYC already knows to exist) may be written with a preceding ‘#\$’ (e.g., #\$Monitor , #\$Decompose), but this is in no case necessary and usually is not desirable. Accepted practice is to write KE text without #\$ characters.

##### Strings

Strings referred to in KE text (such as entries on the #\$comment predicate for a constant) must be delimited by double quotes (e.g., “This is a string.”), as in Common Lisp and C.

##### Expressions

Expressions in KE Text syntax are analogous to expressions in a programming language such as Lisp or C. In KE Text syntax, each expression must end with a period (`.`), and the period must be outside of a comment or a string. The general form of an expression in KE Text syntax is as follows:

```
<directive>: <data-object-or-object-sequence>.
```

A *directive* may be a reserved word (analogous to reserved words in a programming language) or a predicate. Note that reserved word directive names are not case-sensitive. For example, “constant” is the same as “Constant”.

### KE Text Syntax - Reserved Words:

#### Constant

If the reserved word is “Constant”, the data object following the colon delimiter must be the name of a CYC constant (e.g., `PCSubSystem`, or `TestAction`, or some other CYC constant). For example:

```
Constant: PCSubSystem.
```

```
Constant: TestAction.
```

If the data object following the colon delimiter is not already known (by CYC ) to be a CYC constant, then this constant is created. The microtheory is set by default to be `BaseKB`. The only exception to this is if the microtheory has previously been set via the **Default Mt** directive, in which case the use of the `Constant` directive leaves the microtheory unchanged.

#### In Mt

If the reserved word is “In Mt”, the data object following the colon delimiter must be a known (i.e., already existing) microtheory.

Example:

```
In Mt: PCDiagnosisMt.
```

When an expression beginning with an `In Mt` directive is evaluated, it causes the default entry microtheory to be set to the named microtheory. This setting persists until the next occurrence of an `In Mt` directive, `Default Mt` directive or a `Constant` directive.

### Direction

The `Direction` directive sets the default direction for the assertion immediately following. It must be followed either by the constant *forward* or the constant *backward*. Note that, by default, ground atomic formulas have a “forward” direction and rules have “backward” direction. It is most commonly used to assert rules with “forward” direction.

### F

If the reserved word is “F” (for “formula”), the data object following the colon delimiter must be a well-formed CycL Formula.

The constants referred to in the CycL formula must already be known to CYC (i.e., must already exist, perhaps as a result of being created at some previous point in the KE text).

Examples:

```
F: (implies
    (resultOfTest
      (TestFn PCSystem ConfirmSensorily ProblemContext) ?PROBLEM)
    (diagnosisContext ?PROBLEM)).

F: (possibleResultOfTest
    (TestFn PCSystem ConfirmSensorily ProblemContext) BootTime NotNormal).

F: (functionalPartOf VideoSystem PCSystem).
```

### Default Mt

If the reserved word is “Default Mt”, the data object following the colon delimiter must be a known (i.e., already existing) microtheory. For example:

```
Default Mt: PCDiagnosisMt.
```

When an expression beginning with a Default Mt directive is evaluated, it causes the default microtheory to be set to the named microtheory. This setting persists until the next occurrence of a Default Mt or In Mt directive, or the end of the file/text being processed. Note that this directive is stronger than the In Mt directive, since it prevents each occurrence of a Constant directive from resetting the default microtheory to BaseKB. This directive makes it easier to process all (or most) of the expressions in a file/text segment in the same microtheory.

### Predicate Directives

The second type of directive comprises CYC predicates occurring within the scope of a (previously occurring) Constant directive. The Constant directive sets the “current” constant, which then is understood to be the first argument to assertions generated from the following predicate directive expressions. Note that predicate directive names, unlike reserved word directive names, are case-sensitive.

Each predicate directive is followed by a colon delimiter, one or more data objects, and a period. That is, the form of a predicate expression in KE Text syntax is

```
<predicate>: <data-object-1> [<data-object-2>...<data-object-n>].
```

The data objects following the colon delimiter comprise the additional argument(s) to the predicate in the predicate directive.

Example:

```
constant: PCSubSystem.
isa: Collection.
genls: CompositeTangibleAndIntangibleObject.
comment: ‘‘The collection of all PC sub-systems, like the
#$VideoSystem, #$PowerSystem, #$KeyboardSystem.’’.
```

In this example, the Constant directive sets the “current” constant to be PCSubSystem. PCSubSystem is then assumed to be the first argument to assertions formed from the three following predicate directive expressions (the expressions which begin with “isa”, “genls”, and “comment”).

If the predicate directive is the name of a binary predicate (such as `isa` and `comment`), each of the data objects following the colon delimiter is assumed to be part of an assertion in which the predicate directive is the predicate, the default constant is the first argument, and the data object is the second argument. So, when evaluated and processed, the KE text fragment in the example above would result in the addition of the following three assertions to the KB:

```
(#$isa #PCSubSystem #Collection)
($genls #PCSubSystem #CompositeTangibleAndIntangibleObject)
($comment #PCSubSystem ‘‘The collection of all PC sub-systems,
like the #VideoSystem, #PowerSystem, #KeyboardSystem.’’)
```

The same assertions could have been introduced using F: directives

Example:

```
F: (isa PCSubSystem Collection)
F: (genls PCSubSystem CompositeTangibleAndIntangibleObject)
F: (comment PCSubSystem ‘‘The collection of all PC sub-systems, like the
#$VideoSystem, #PowerSystem, #KeyboardSystem, #FloppySystem .’’)
```

Note that this mechanism cannot be used for assertions involving unary predicates. For example, `#$hypothesis` is a such a predicate. Assertions using this predicate could be entered with an expression such as this:

```
F: (hypothesis PowerSystem) .
```

### Comments in KE Text

Comments (text to be read by a human, but not interpreted or entered by a program) are allowed in KE text. The comment indicator is the semi-colon (;), as in Common Lisp. Lines beginning with a semi-colon will be ignored.

### Order of Expressions

Expressions in KE text are evaluated and processed in the order of their occurrence in the text.



### 2.3.7 Interface Tools - The SubLanguage (SubL) Interactor

SubL is a computer language built by members of the CYC team. SubL was written to support the CYC application, allowing it to run both under Lisp environments and as a C application generated by a SubL-to-C translator.

SubL<sup>6</sup> is intended to be somewhat similar to Common Lisp, with features that are complex or rarely-used or difficult to implement in C excised. Also, unlike Common Lisp, SubL is not a purely functional language. Several Common Lisp constructs can only be used procedurally. In order to emphasize this difference, the following constructs have their names preceded either by 'c', 'p' or 'f' in SubL: `pif`, `pwhen`, `punless`, `pcond`, `pcase`, `csetq`, `cinc`, `cdec`, `cpush`, `cpushnew`, `cpop`, `clet`, `cmultiple-value-bind`, `cdo`, `cdotimes`, `cdolist`, `csome`, `cdohash`, `ccatch`, `cunwind-protect`, `cnot`, `cand`, `cor`, `fif`, `fwhen`, `funless`.

The SubL Interactor is an input window for evaluating SubL expressions.

### 2.3.8 Interface Tools - The Functional Interface (FI)

The CYC Functional Interface (FI) is an API (Application Program Interface) that external programs can use to query and update a version of CYC. The CYC FI provides the ability to find, create, kill, and rename constants, assert, unassert, ask, retrieve justifications for, and prove propositions, get and set application parameters, and a few other things. There is also a set of FI extensions for database integration. The commands of the functional interface can be invoked like normal lisp function calls from a lisp interactor such as the SubL Interactor of the CYC Web Interface as well as from other SubL functions. For a list of the FI functions used in this thesis see Appendix A.

## 2.4 Summary

In this chapter, a thorough description of KADS and CYC has been given. Specifically, KADS *Expertise model* was described with its four layers, namely the *Domain*, *Inference*, *Task and Strategy* layers, what they consist of, how they are built and what their inter-

---

<sup>6</sup>For more information see [www.cyc.com/cyc-2-1/toc.html](http://www.cyc.com/cyc-2-1/toc.html)

relationships are. Also *Generic Task Models* were described, what they are and how they are used to build the Expertise model. All these parts of KADS are used in the next chapter for the Analysis phase of the problem solving method implementation.

For CYC, its *Ontology* was described, what this Ontology is, what it consists of and how it is represented through *CycL*, CYC's representation language. Also CYC's inference engine was described as well as the various interfaces provided for maintaining the KB (KB Browser), editing knowledge into the KB (Knowledge Editing Text) and executing Lisp code (the SubL Interactor and the Functional Interface). These parts of CYC will be used in chapter 4 to implement the Systematic Diagnosis problem solving method for PC and Automobile diagnosis.

# Chapter 3

## The Analysis Phase

### 3.1 Introduction

In the previous chapter (§2.2) it was noted that the analysis phase of KADS which is actually concerned with the development of the KBS is the *Expertise* analysis (§2.2.2) and that a library of Generic Task Models (GTMs) is provided. The building of the expertise model consists of three stages:

- the construction of the Domain Layer which contains the static knowledge,
- the selection of an appropriate initial GTM which contains a general description of the Inference, Task and (probably) the Strategic knowledge. More knowledge acquisition according to the *Domain roles* defined by the GTM and
- the filling of the details in the upper three layers and probably some modifications of the GTM.

In the following sections these three stages will be described in detail. But, before this, it must be made clear that the use of KADS as a methodology for analysing the system that will implement a problem solving method (PSM) is *completely independent* from the implementation medium which is CYC. This independence is twofold:

1. The problem solving method that will be selected should not necessarily come from KADS. It could have been developed in another context. For example, the Heuristic

Classification PSM ([Clancey 85]) was developed independently from the KADS methodology, although it can be found in the KADS GTM library (see Figure 2.5).

2. Even if the PSM was taken from KADS GTM library, still another approach for ES development could be used, e.g. rapid prototype development and further refinement. However, most of the PSM's power as part of KADS would have been lost, since there would be neither domain roles to guide the knowledge acquisition nor the Task structure to guide the PSM. If KADS was not used, one could try to implement the PSM in CYC by extending the ontology with new terms and rules using general knowledge acquisition techniques, and then using backward inference to implement the various steps of the PSM.

## 3.2 Knowledge Acquisition for the Domain Layer

There are two main sources of domain expert knowledge: bibliography and the human domain experts. The author has relatively good personal experience of PC troubleshooting, gained in a six month period of assembling and repairing PCs. Therefore, certain domain knowledge was at hand through self-introspection. However, more systematic knowledge was needed and it was found in the WWW site of the PCGuide magazine's Troubleshooting Expert ([www.pcguides.com/ts/x/index.htm](http://www.pcguides.com/ts/x/index.htm)) developed by Charles M. Kozierek. This "Expert" is actually a set of menus containing questions about the PC system status and possible answers in the form of HTML links that guide the user to find any problem related to a PC. From the author's personal experience, and having studied the "Expert", it can be claimed that it is one of the most complete ever seen. The "Expert" has three main categories of troubleshooting *contexts*:

- Troubleshooting Boot Problems
- Troubleshooting The System Overall
- Troubleshooting Specific Components

The knowledge analysed is only that related to the first context, the boot-time problems troubleshooting. This may seem a major limitation but it is not, since the development of the KBS is only to test the feasibility and effectiveness of implementing problem

solving methods in CYC; therefore, the KBS serves just as an experimental model and not as a fully functioning expert system per se.

After a thorough study of the “Expert”, the following domain structures appeared as significant in the diagnosis process:

1. A PC system model, a hierarchy of simple components and composite components, consisting of more simple components.
2. Testing knowledge, consisting of three other, more specific, domain structures:
  - Questions (Tests) that the “Expert” made about the PC system status,
  - Possible results of these Tests and
  - Actions taken according to each of these possible results.

It is quite obvious from the description of the emerging domain structures that they are far from being clearly defined. This is the *Knowledge Acquisition bottleneck* and the author was caught in it. Necessarily, the author used the help of KADS, its Generic Task Models and the *Domain roles* that these provide. Consequently, the second stage had to be entered, that is the selection of a GTM.

### 3.3 Selecting a Generic Task Model

One of the main problems when using KADS is the selection of a GTM. In fact, KADS provides little guidance for this selection and the decision comes back to the knowledge engineer. Criticism about this lack of guidance as well as about the potential dangers of selecting the wrong GTM can be found in [Rademakers & Vanwelkenhuysen 93].

The task for selecting a GTM in the case of PC fault diagnosis was quite simple; in figure 2.5 from [Tansley & Hayball 93], specific GTMs are given for the task of diagnosis. For simplicity reasons, and because of the way the “Expert” was designed, the Systematic Diagnosis GTM was selected. However, it is important to note that other GTMs were applicable, such as the Heuristic Classification GTM, a well established method for diagnosis, first developed by Clancey (see [Clancey 85]). The problem of more than one GTM applying to a specific task can be further explored in

[Rademakers & Vanwelkenhuysen 93]. The inference structures for the two GTMs are shown in figure 3.1.

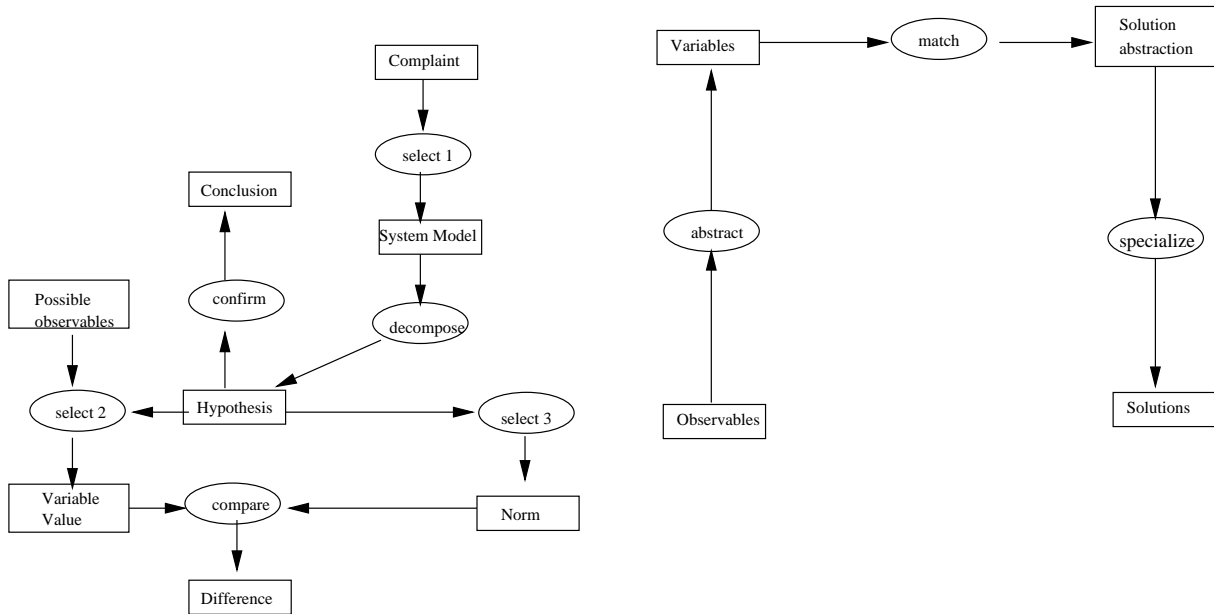


Figure 3.1: Inference structures for Systematic Diagnosis (left) and Heuristic Classification (right) GTMs

The selection of a GTM makes the construction of the Domain layer easier through the specification of *Domain roles* for the inference types included in the inference structure of the GTM. To illustrate this, each inference type and its associated domain roles of the Systematic Diagnosis GTM (Localisation version) are presented in table 3.1, as adapted from [Tansley & Hayball 93].

Note, that the Domain roles are customised for the Localisation version of the Systematic Diagnosis GTM. The actual, generic Domain Roles are given in table 3.2.

### 3.4 Domain Roles and Domain Layer

After having presented the Domain roles described by the GTM, it is much easier to construct the Domain layer of the KBS. The first domain structure that prevails the whole procedure is the *System model*. The Systematic Diagnosis GTM suggests a *consists-of*

<b>Inference</b>	<b>Input Role</b>	<b>Output Role</b>	<b>Method and Knowledge</b>
<b>Select 1</b>	Faulty system description	Consists-of model	Direct association - System behaviour and structure
<b>Decompose</b>	Consists-of model	(Sub)-System containing faulty component	Descending consists-of tree Consists-of structure
<b>Select 2</b>	(Sub)-System containing faulty component. Observable Output Values.	Observed Output Value	Generate and test- Test methods
<b>Select 3</b>	(Sub)-System containing faulty component	Expected Output	Direct association - System behaviour
<b>Compare</b>	Observed Output Value	Decision Class	Compares values Significance of differences
<b>Confirm</b>	(Sub)-System containing faulty component	Yes/No (+fault location)	Primitive part reached System structure

Table 3.1: The Systematic Diagnosis Inference Types and their Domain Roles

<b>Domain Role</b>	<b>Localisation</b>
Complaint	Faulty system description
System model	Consists-of model
Possible observables	Observable output variables
Hypothesis	(Sub)-System containing faulty component
Variable value	Observed output value
Norm	Expected output
Difference	Decision class
Conclusion	Yes/No (+ fault location)

Table 3.2: Domain Roles for Systematic Diagnosis and the Localisation equivalents

system model, however, as we will see below, there is a great variety of system models to choose from and, moreover, a consists-of model is not the best for diagnosis in a PC system. This significant variation from the proposed model does neither come as a surprise nor as something unusual. The domain structures proposed by a GTM are simply general guidelines and are not restricting for the knowledge engineer. They are another point of view for the domain knowledge, one more task-oriented. The same holds even for the inference structures of the GTMs as we will see later. This flexibility of the GTMs is discussed in [Rademakers & Vanwelkenhuysen 93] and [Wielinga *et al.* 92].

The prevalence of the PC system model and the inadequacy of a *consists-of* model are discussed below.

### 3.4.1 The System Model

The first thing that must be understood for the selection of a system model is the overall role that this plays in the Systematic Diagnosis problem solving method and especially in its Localisation version. The outline of the method is:

1. Begin with a general symptom of the system. Select a part of the system that probably contains the faulty component.
2. Decompose the suspected (sub)system into its *parts* which may be simple components or (sub)systems themselves.
3. Take the first/next component/subsystem and check if it is the one that contains the faulty component. If it does and it is a component then stop; if it is a (sub)system, go to 2. If it does not contain the faulty component, then repeat step 3 as far as there are candidate components/(sub)systems.

From the above outline it is clear that the system model should be a hierarchy of (sub)systems and their parts which in turn can be either simple components or further decomposable (sub)systems. But there are a lot of hierarchies. In [Steels 90] various such models are mentioned:



- Structural: these models describe the way the parts of the modelled system contain and form each other, like the model of a subway containing the stations, tracks and so on [Steels 90],
- Topological: these models describe the way the parts of the modelled system are connected to each other, like the model of a heating system containing the connecting pipes between the various parts of the system [Borst *et al.* 97],
- Functional: these models describe which parts are used by each (sub)system in order to carry out its function(s), like the model of a printed circuit board containing the *circuit operations* and the corresponding *hardware modules* that are involved in carrying out these operations [Vanwelkenhuysen 92].

The model selected for the Personal Computer (PC) fault diagnosis is a *functional* model. This decision is based in the fact that a PC is an information processing machine with central control and therefore exhibits the following two characteristics which distinguish it from other mechanical devices:

1. The *same* components take part in *different* functions. This results in *overlapping* (sub)systems and in different functional roles for the components, and therefore different behaviours and different possible faults, according to the function of the component.
2. Because of the central control of the CPU which imposes either predefined or dynamic sequences of component operations, the physical connections of the components do not always define the order of component operations and therefore the order of diagnosis.

The appropriateness of a functional model in the case of PCs is supported by its use in troubleshooting of electronic circuits in [Vanwelkenhuysen 92] and [Hamscher 88]. In contrast, the reader can refer to another, *mereotopological*<sup>1</sup> model, presented in [Borst *et al.* 97]. The two above characteristics mean that the functional model of the PC cannot be expressed in a static hierarchical structure but rather like a set of rules that describe different decompositions, according to which (sub)system is being decomposed and in which

---

<sup>1</sup>A mereotopological model is a combined structural and topological model

function it takes part. In addition, it must be noted that the decomposition process for the Systematic Diagnosis problem solving method requires specification of not only which (sub)parts of the (sub)system being decomposed are candidates for diagnosis but also *in which order* they will be diagnosed. An example will clarify the order's importance:

### **Example**

Suppose that the PC does not produce any video signal on the monitor when it is turned on. This suggests that the video (sub)system of the PC has a fault. This (sub)system contains the following components: the motherboard, the video card and the monitor. It is obvious that the monitor should be checked first, otherwise no test can be performed on either of the other two components due to lack of feedback (control) from the (possibly) faulty monitor.

To summarise about the desired properties of the PC system model:

- It must provide a different decomposition of the PC (sub)systems in different cases,
- It must provide the order of testing for the components of the decomposed (sub)system and
- It must distinguish between decomposable (sub)systems and simple components.

All these properties lead to the following domain structures<sup>2</sup>:

## **CONCEPTS**

---

<sup>2</sup>There are many more instances of PCComponent and PCSubSystem to be defined as concepts. A complete list can be found in the KE-text for the CYC KB in Appendix B.

- PCComponent** A simple, non-decomposable PC component. A PCComponent is the lowest level of the system's analysis.
- PCSubSystem** A decomposable PC (sub)system. A PCSubSystem involves one or more PCComponents and/or PCSubSystems and it is an intermediate level of the system's analysis.
- PowerSystem** The power system of the PC.
- PowerSupply** The power supply device of the PC.

#### ATTRIBUTES

- hypothesis(PCSubSystem)**: The PCSubSystem is the current candidate for diagnosis.
- possibleHypotheses(PCSubSystem)**: The PCSubSystem is one of the next candidates for diagnosis.
- testFirst(PCSubSystem)**: The PCSubSystem is the first to be tested from all the other candidate PC subsystems; it is the next hypothesis.

#### RELATIONS

- subsetOf(PCComponent, PCSubSystem)**: A PC component is the simplest PC subsystem
- isa(PowerSystem, PCSubSystem)**: PowerSystem is a PCSubSystem.
- isa(PowerSupply, PCComponent)**: PowerSupply is a PCComponent.

- functionalPartOf(PCSubSystem-Whole,PCSubSystem-Part)**: The PCSubSystem-Whole involves in its function the PCSubSystem-Part.
- testAfter(PCSubSystem-1, PCSubSystem-2)**: The PCSubSystem-2 must be tested immediately after the PCSubSystem-1.

The decomposition rules are missing. This is because their 'if' part - the antecedent - requires concepts not yet defined. These concepts are the subject of the next section.

### 3.4.2 The Testing Knowledge

The basic tool in the Systematic Diagnosis problem-solving method (PSM), as well as in any other diagnostic PSM, for carrying out the diagnostic procedure, is various Tests that must be done to provide information (knowledge) about the state of the system. This knowledge may concern the actual behaviour of the system's components, e.g. the absence of electric power or control information produced by the system, e.g., beep codes or screen messages. Conceptually, a Test is a question that the user must make to the system under diagnosis to extract knowledge about it. A Test is a structure consisting of three other concepts:

1. The PC subsystem to which it is related, i.e., to which the question is addressed,
2. The Action that the human user must make to perform the Test and
3. The Possible Observable (system variable) that the Test is asking about.

Although not part of a Test structure, there is a fourth concept related to it, the Possible Observable Value (system variable value) which is the result (answer) of the Test (question). A final concept, related to the Test's result, is the Result Type which describes what the result of a Test means for the diagnosis procedure, that is, what further decision it entails. All these lead to the following domain structures<sup>3</sup>:

## CONCEPTS

---

<sup>3</sup>There are many more instances of TestAction, PossibleObservable, PossibleObservableValue and ResultType to be defined as concepts. A complete list can be found in the KE-text for the CYC KB in Appendix B.

<b>TestAction</b>	A physical action that the human user must make to perform a Test.
<b>PossibleObservable</b>	A system variable the values of which give information about the system status.
<b>PossibleObservableValue</b>	A possible value of a system variable.
<b>ResultType</b>	The type of a Test's result. These types are characterised from the kind of conclusions they lead relative to the PCSubSystem being currently diagnosed ( hypothesis(PCSubSystem) ). E.g., such a type can be Normal which denotes that the PCSubSystem currently being diagnosed is not faulty and therefore must be discarded as a hypothesis and a new hypothesis must be selected.
<b>ConfirmSensorially</b>	The action of confirming the existence of a PossibleObservable only by one's senses, e.g., visually, acoustically.
<b>ElectricPower</b>	The electric power that any PCSystem needs to operate.
<b>Yes</b>	Most of the Tests have as a possible result only 'Yes' or 'No'.
<b>NotNormal</b>	This type of result indicates that the result is not normally expected when the PCSubSystem related with it is working properly. Such a kind of result implies that the fault lies in the PCSubSystem which is the current hypothesis.

## STRUCTURE

<b>Test</b>	<b>Constituent concepts</b>
	PCSubSystem
	TestAction
	PossibleObservable

## ATTRIBUTES

**possibleTest(Test)**: Test can be currently performed.

## RELATIONS

**isa(ConfirmSensorially, TestAction)**: ConfirmSensorially is a TestAction

**isa(ElectricPower, PossibleObservable)**: ElectricPower is a PossibleObservable

**isa(Yes, PossibleObservableValue)**: Yes is a PossibleObservableValue

**isa(NotNormal, ResultType)**: NotNormal is a ResultType

**possibleResultOfTest(Test, PossibleObservableValue, ResultType)**: The test Test has as a possible result the PossibleObservableValue which is of type ResultType.

**resultOfTest(Test, PossibleObservableValue)**: The test Test gave as result the PossibleObservableValue when it was performed.

The rules introducing the possible Test are missing. This is because their 'if' part - the antecedent - requires a concept not yet define. This concept is the subject of the next section.

### 3.4.3 The Diagnosis Context

The Systematic Diagnosis GTM inference structure starts with a SELECT inference (select 1, see figure 3.1). A general symptom is entered by the user and an appropriate system model is chosen. This inference is slightly changed for PC diagnosis. What is actually asked of the user is to distinguish three major contexts of diagnosis:

1. Boot-time troubleshooting,
2. Run-time troubleshooting and
3. Component-specific troubleshooting.

This categorisation is significant since completely different rules are applicable in each context. This contextual dependency will be embedded in the antecedent part of the

rules as an extra condition. The necessary domain structures are:

### ATTRIBUTES

**isa(BootTime, PossibleObservableValue)**

**isa(RunTime, PossibleObservableValue)**

**isa(ComponentSpecific, PossibleObservableValue)**

**diagnosisContext(PossibleObservableValue):** The PossibleObservableValue is the current diagnosis context.

and the way they are used in rules is:

`diagnosisContext(BootTime) and ...[more conditions]... implies [consequent].`

To satisfy all these specifications, the rules for decomposition of the PC system model will have the following general form:

`diagnosisContext(PossibleObservableValue) and`

`hypothesis(PCSubSystem) and`

`resultOfTest(Test1, PossibleObservableValue1) and ...`

`....`

`and resultOfTest(TestN, PossibleObservableValueN)`

**IMPLIES**

`testFirst(PCSubSystem1) and`

`testAfter(PCSubSystem1, PCSubSystem2) and...`

`...`

`and testAfter(PCSubSystemN-1, PCSubSystemN).`

while the rules for introducing new Tests and their corresponding results, will have the form:

`diagnosisContext(PossibleObservableValue) and`

`hypothesis(PCSubSystem) and`

```

resultOfTest(Test1, PossibleObservableValue1) and ...
.
.
.
resultOfTest(TestN, PossibleObservableValueN)
IMPLIES
possibleTest(Test) and
possibleResultOfTest(Test, PossibleObservableValue1, ResultType1) and ...
.
.
.
and possibleResultOfTest(Test, PossibleObservableValueN, ResultTypeN).

```

### 3.5 The Inference Layer

As described in the previous chapter (§2.2.1), the Inference layer defines the inference types that must be performed by the problem solving method. After the description of the domain layer in the previous section, the inference layer of the Systematic Diagnosis problem solving method for PCs is given in table 3.3, in terms of the various inference types as well as their input and output domain structures. Note that the input and output roles have now been substituted by actual domain structures, and that inferences *Select 2* and *Select 3* have been combined in one inference, *Select 2-3*. These changes are well situated into the development procedure of the Expertise model as described in [Wielinga *et al.* 92].

### 3.6 The Task Layer

The next layer is the Task layer, where the order of performing the inference types is described. This is done in terms of the pseudo-code given in figure 3.2. A comparison with the Task Structure given in figure 2.3 can show how this last one was modified.

The last layer of the Expertise model in KADS is the Strategy layer, however the



<b>INFERENCE TYPE</b>	<b>INPUT Role</b>	<b>OUTPUT Role</b>
Select 1	PC symptom's general nature (BootTime, RunTime, ComponentSpecific)	Problem solving Context
Select 2-3	Problem Solving Context Hypothesis Results of previous Tests	Possible Test(s) Possible Results of Test(s) Types of Possible Results
Decompose	Problem Solving Context Hypothesis Results of previous Tests	Hypothesis Possible Hypotheses
Compare	Last Test's Result Type	Next inference to be executed
Confirm	Last Test's Result Type	Faulty component

Table 3.3: The Inference Layer of the Systematic Diagnosis PSM for PCs

```

Systematic Diagnosis(+complaint,+possible observables,-hypothesis) by
select1(+complaint nature, -diagnosis context)
REPEAT
  decompose(+hypothesis, +diagnosis context, +previous tests' results,
            -hypothesis, -possible hypotheses)
WHILE number of possible hypotheses > 1
  select2-3(+hypothesis, +diagnosis context, +previous tests' results,
           -possible test(s), -possible observables, -result types)
  compare(+last test's result type, -inference to be performed)
UNTIL confirm(+hypothesis), i.e. system model cannot be decomposed further

```

Figure 3.2: The Task Structure for Systematic Diagnosis PSM for PCs

GTM for Systematic Diagnosis does not provide one and no such layer is needed for the implementation of this problem solving method.

### 3.7 Summary

In this chapter, the KADS Analysis phase was described as it was applied on the domain of PC fault diagnosis. Why the Systematic Diagnosis GTM was selected as the problem solving method and the guidance this provided for knowledge acquisition through the *domain roles* for its constituents inference types. What was the system model and the testing knowledge needed as domain knowledge for this GTM. Finally, the modifications that were made on some of its constituent inference types to fit the specific domain and the final Task structure. All these products will be used in the next chapter to implement the Systematic Diagnosis PSM in CYC. It is very important to remember that this phase is *completely independent* of the implementation medium, which in this thesis is CYC; however, it could be an expert system shell, CLIPS or PROLOG.

# Chapter 4

## The Implementation Phase in CYC

### 4.1 Introduction

The next step to implement the Systematic Diagnosis problem solving method (PSM) for PCs in CYC was to implement each one of the three layers of the Expertise model, namely the Domain, Inference and Task layers. Before presenting the implementational decisions, the main components of CYC, on which this implementation is based, are repeated:

- The CYC Knowledge Base (KB) consisting of terms structured in a Collection - Sub-collection - Instance hierarchy as well as rules connecting them,
- The CycL representation language which defines the KB terms as well as the rules between them,
- An 'Ask' interface for querying the KB, in a Prolog-like fashion, using the CycL language,
- The SubL language, a subset of Common Lisp,
- The Functional Interface (FI), a set of SubL functions that provide the means to perform all the necessary operations to the KB through SubL code and
- The SubL Interactor, an interface to execute SubL code.

All these CYC components are described in chapter 2.

## 4.2 The Overall Model

The common use of CYC is through its Ask interface. The user enters a CYC formula, possibly containing variables, and the CYC inference mechanism tries to find matchings for these variables through unification, using rules and facts in the KB. Under this scheme, solving a problem consists in defining the right knowledge, that is terms, facts and rules, making the right(?) questions and (probably) modifying the KB. This is a declarative way to solve problems. On the other hand, SubL provides a more procedural/functional way to do things. Through CYC's FI, it is possible to perform all the above operations from SubL code. Having all these in mind, the first and crucial problem to solve implementing a PSM is what goes where. In the case of this research, where the KADS method was used, this question is which expertise layer is implemented in which component of CYC. The final decisions that the author made in solving this problem were based in the following considerations:

1. The KADS methodology, through the Expertise model, provides a significant differentiation of problem-solving knowledge in *domain* and *control* knowledge. Since this distinction is at the heart of analysing problems, preserving it in the implementation phase keeps things simple and clear. Moreover, mixing domain and control knowledge was one of the major drawbacks of the early experts systems, like MYCIN ([Clancey & Letsinger 82]).
2. CYC's use is mainly through the declarative scheme described above. Although the goal of implementing PSM's in CYC is to make it more effective, the advantages of solving problems in a declarative way should not be lost in the obscurity of functional descriptions in Lisp (SubL) code. The overall guideline is that the implementation should permit the problem to be solved by the declarative way too. How this can be done will be explained later on in this chapter.

At the implementation level, these guidelines led to the following decisions:

1. The Domain layer was encoded in the CYC KB and
2. The Inference and Task layers were encoded in SubL code

This overall model can be seen in figure 4.1.

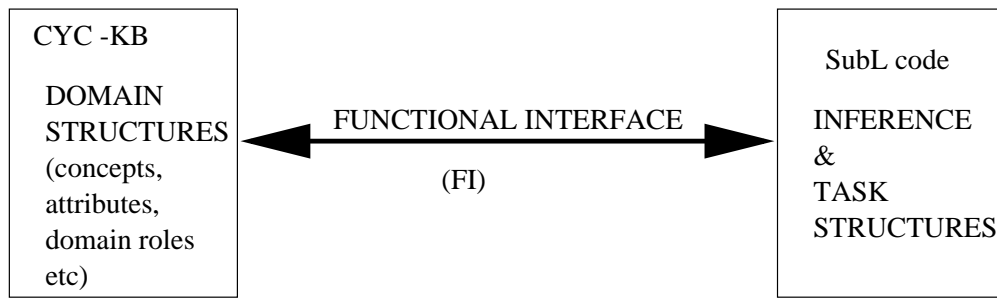


Figure 4.1: The Overall Model (schematic view)

### 4.3 The Implementation of The Domain Layer

Representing the Domain layer in the CYC KB is a significant task by itself. This is mainly for two reasons:

1. Every term in the CYC KB must be an instance of a Collection, except `#$Thing`, the top-most collection. Therefore, every domain structure defined in the analysis phase must be represented as the instance of another collection. Consequently, some domain structures must be fitted somewhere in the CYC *Ontology*<sup>1</sup>. The problem here lies in how to integrate the new knowledge that comes from a specific domain, which usually comprises very specialised concepts of the real world, with the knowledge already in CYC KB, which consists of very general concepts of the world. A *connecting, intermediate ontology* is needed to integrate the two kinds of knowledge in a realistic way.
2. Another dimension of the same problem occurs when different people are developing expert systems in CYC and consequently create new concepts and try to integrate them in the *already existing CYC Ontology*. This raises the issues of consistency, i.e. the same concepts may be defined as parts of contradictory parts of the Ontology, and duplication, i.e. different terms are defined that conceptually are identical. In [Lenat & Guha 90], these issues are discussed in detail.

---

<sup>1</sup>The AIAI version of CYC, on which this project was developed, contained only the *upper ontology* which can also be found at [www.cyc.com/cyc-2-1/intro-public.html](http://www.cyc.com/cyc-2-1/intro-public.html)

The decisions required are quite hard and they primarily demand good knowledge of the given CYC ontology. However, such knowledge comes more by using than by studying the terms of the KB. The author studied the given ontology to the extent permissible in the time available. Therefore, it is not claimed that the solutions are either unique or the best. This is an issue discussed in the next chapter. However, the major points of these decisions are discussed below<sup>2</sup>:

### 4.3.1 The PC Diagnosis Microtheory.

In CYC, *microtheories* is a way of talking about a group of related assertions by representing this grouping explicitly with a Cyc unit ([Blair *et al.* 92]). The default microtheory is the `#$BaseKB` and any other microtheory is almost always a superset of this microtheory in a hierarchy which is constructed using the predicate `#$genlMt`. For all the reasons cited in [Blair *et al.* 92], all the domain knowledge for the Systematic Diagnosis PSM for PCs was grouped in a special microtheory, the `#$PCDiagnosisMt` (B:10).

### 4.3.2 The System Model.

The representation in CYC of the domain structures related to the system model knowledge was quite straightforward. Figure 4.2 illustrates the decisions about this representation. The meaning of all terms can be found in the corresponding part of the KE-text in Appendix B (B:21-207, B:691-717). Recall that in CYC, the `#$isa` predicate means “element of”, while the `#$genls` predicate means “subset of”. The important thing here is that the `#$PCComponent` collection is a sub-collection of the `#$PCSubSystem` collection. This is both for grouping every part of a PC in the latter and for distinguishing the simple, non-decomposable components by the former collection.

Another important aspect of the system model implementation is that, although a description of it is given in terms of the `#$functionalPartOf` assertions (B:691-717) and the general decomposition rule<sup>3</sup> (B:681-690),

---

<sup>2</sup>Numbers in parentheses refer to number lines of the KE-text listing in Appendix B.

<sup>3</sup>Terms preceded by ‘?’ are variables in CYC

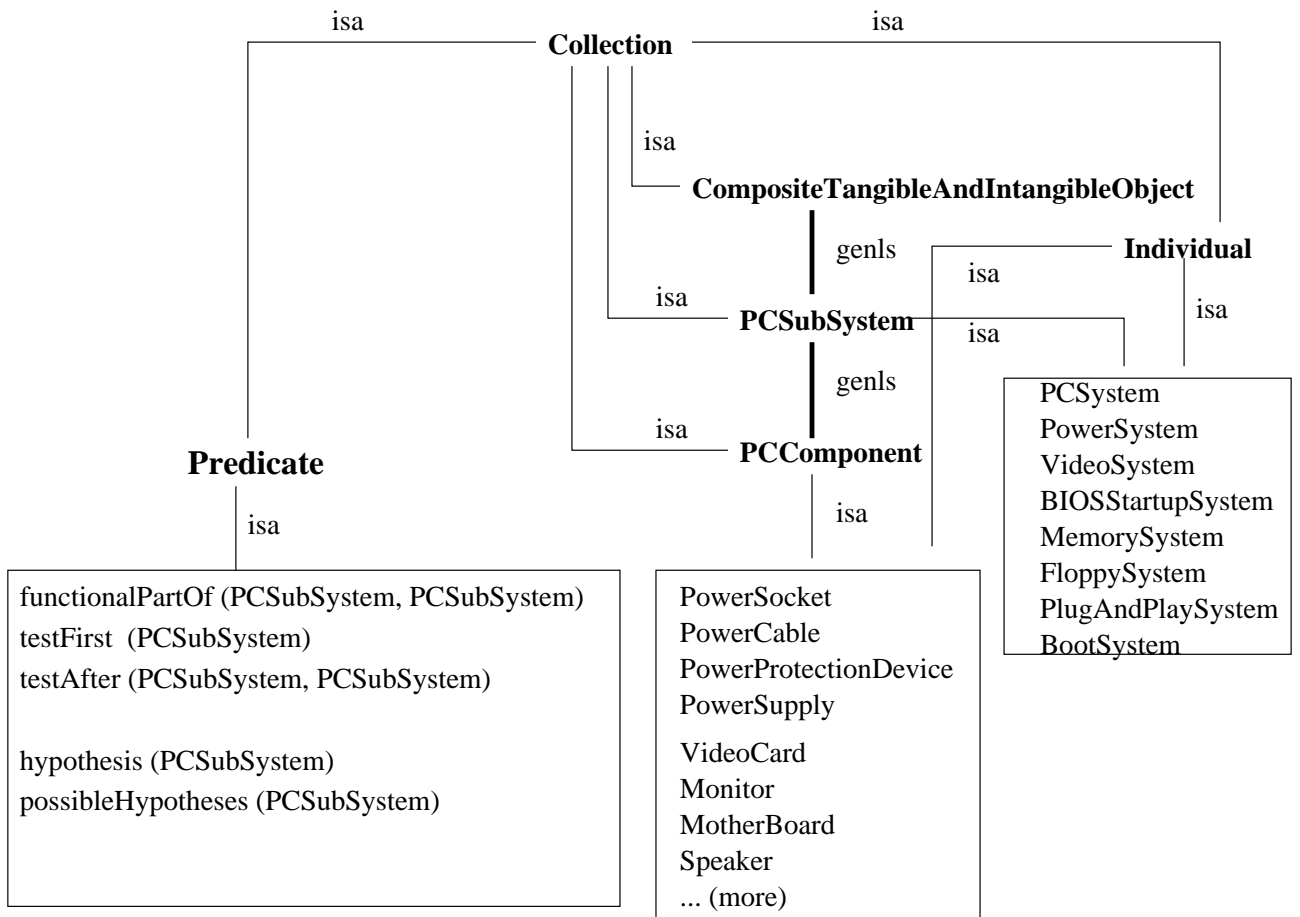


Figure 4.2: The System Model Hierarchy in CYC

```
(implies
  (and
    (diagnosisContext BootTime)
    (hypothesis ?HYPOTHESIS)
    (plausibleInference Decompose)
    (functionalPartOf ?HYPOTHESIS ?PART))
  (possibleHypotheses ?PART)).
```

the order of testing is given through explicit decomposition rules which are context dependent, that is they have the general form:

```
(implies
  (and
    (diagnosisContext BootTime)
    (hypothesis HYPOTHESIS)
    (plausibleInference Decompose)
    (resultOfTest TEST_1 RESULT_1)
    ...
    (resultOfTest TEST_M RESULT_M))
  (and
    (testFirst PC_SUBSYSTEM_1)
    (testAfter PC_SUBSYSTEM_1 PC_SUBSYSTEM_2)
    ...
    (testAfter PC_SUBSYSTEM_N-1 PC_SUBSYSTEM_N))).
```

**Example of Decomposition rule.** The rule

```
(implies
  (and
    (diagnosisContext BootTime)
    (hypothesis VideoSystem)
    (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
    (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No)
    (plausibleInference Decompose))
```



```
(and
  (testFirst MotherBoard)
  (testAfter MotherBoard VideoCard)).
```

means that

IF

1. The problem solving context is Boot-time and
2. The video system is hypothesised as being in fault and
3. Something is written on the screen (VideoSignal=Yes) and
4. There is no video BIOS message and
5. A *Decompose* inference must be performed

THEN

1. Test first the Motherboard and
2. Test the Video card after the Motherboard.

It is the significance in the order of testing that imposes the encoding of explicit decomposition rules. *If the order of testing was not important, the explicit decomposition rules would not be necessary.* This would simplify the implementation of the system model in just the `#$functionalPartOf` assertions (B:691-717) and the general decomposition rule (B:681-690). The explicit decomposition rules are grouped in the KE-text by component/subsystem (B:739-2130).

### 4.3.3 The Testing Knowledge

The representation of the domain structures related to the system testing knowledge was quite straightforward in CYC. Figure 4.3 illustrates the decisions about this representation. The meaning of all terms can be found in the corresponding part of the KE-text in Appendix B (B:214-559). There is a technicality concerning the representation of the Test structure. It is represented via a `#$NonPredicateFunction`, `#$TestFn`

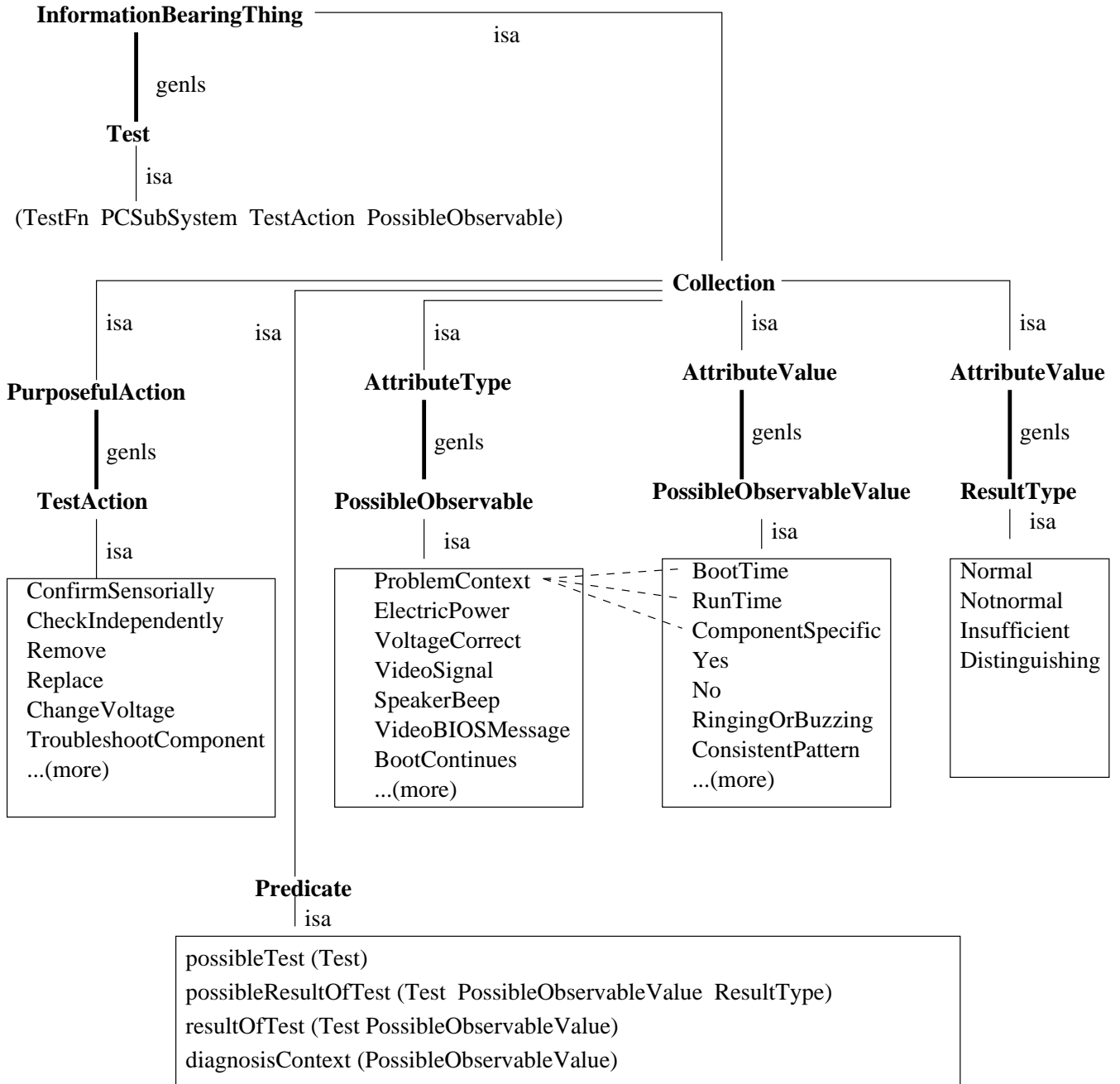


Figure 4.3: The Test Knowledge Hierarchy in CYC

(B:258), which takes as arguments the three constituent concepts, i.e. `#$PCSubSystem`, `#$TestAction` and `#$PossibleObservable`, and returns a `#$Test` instance. Schematically:

```
(TestFN PCSubSystem TestAction PossibleObservable) -> Test
```

Once again, the rules that introduce the possible tests, their possible results and their types, are context dependent:

```
(implies
  (and
    (diagnosisContext BootTime)
    (hypothesis HYPOTHESIS)
    (resultOfTest TEST_1 RESULT_1)
    ...
    (resultOfTest TEST_M RESULT_M))
  (and
    (possibleTest TEST)
    (possibleResultOfTest TEST POSSIBLE_OBSERVABLE_VALUE_1 RESULT_TYPE_1)
    ...
    (possibleResultOfTest TEST POSSIBLE_OBSERVABLE_VALUE_N RESULT_TYPE_N))).
```

where

```
TEST:= (TestFN PC_SUBSYSTEM TESTACTION POSSIBLE_OBSERVABLE).
```

**Example of Test introduction rule.** The rule

```
(implies
  (and
    (diagnosisContext BootTime)
    (hypothesis MotherBoard)
    (resultOfTest (TestFN VideoSystem ConfirmSensorially VideoSignal) Yes)
    (resultOfTest (TestFN VideoSystem ConfirmSensorially VideoBIOSMessage) No))
  (and
    (possibleTest (TestFN MotherBoard ConfirmSensorially SpeakerBeep))
```

```
(possibleResultOfTest
  (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No Insufficient)
(possibleResultOfTest
  (TestFn MotherBoard ConfirmSensorially SpeakerBeep) ConsistentPattern
  Insufficient)))
```

means that

IF

1. The problem solving context is Boot-time and
2. The Motherboard is hypothesised as being in fault and
3. Something is written on the screen (VideoSignal=Yes) and
4. There is no video BIOS message

THEN

- A possible test is to check if the speaker beeps and
- There are two possible results:
  1. No sound comes from the speaker, which is insufficient and necessitates more tests (about the speaker itself).
  2. A beep with a consistent pattern comes from the speaker, which is insufficient and necessitates more tests (about the beep code).

These rules are grouped in the KE-text by component/subsystem (B:763-1678).

## 4.4 The Implementation of the Inference and Task Layers

The guidelines for implementing the *control* knowledge, namely the Inference and Task layers, are:

1. These layers will be implemented in SubL code and

2. The implementation will allow for the problem-solving method, the Systematic Diagnosis PSM, to be executed *declaratively*, i.e. merely in terms of asking and (possibly) changing facts in the CYC KB.

Under these guidelines, the encoding of the Inference and Task layers in SubL code was done in the following way<sup>4</sup>:

1. Each inference type was encoded as a single SubL function-inference.
2. The only thing that these functions do is to ask the CYC KB questions (fi-ask), assert (fi-assert) and/or retract (fi-unassert) facts from the CYC KB (Inference layer) and - according to the new facts - call the appropriate function-inference (Task layer).
3. The Task Structure in figure 3.2 is encoded in a function, `systematic` (C:22), although this is not obvious from the SubL code for technical reasons that will be explained later.

The description of the implementation for the Inference and Task layer is based on figure 4.4. According to this figure, the overall system in CYC works as follows:

1. The function-inference `systematic` is called (Figure 4.5). When the diagnostic session begins, i.e. no `#$PCSystem` is hypothesised as faulty (number of hypotheses = 0), then the `sd-select1` (C:157) function is called. The latter, asserts to the KB the fact that the whole `#$PCSystem` is at fault (C:166). Each time that a new (`hypothesis ...`) assertion is entered into the KB, the appropriate rules concerning possible tests “fire”. This is because, in CYC, a rule can have a direction associated with it. If this direction is *forward*, then, whenever new assertions are asserted in the KB, the inference engine of CYC finds all rules with direction *forward* of which the antecedent part is satisfied by the contents of the KB and immediately asserts all the assertions in the consequent part of it. This enables *forward reasoning* to be implemented, which is the case for the system developed in this research. In fact, forward reasoning is implemented by the Task structure.

---

<sup>4</sup>Numbers in parentheses refer to line numbers of the SubL listing in Appendix C.

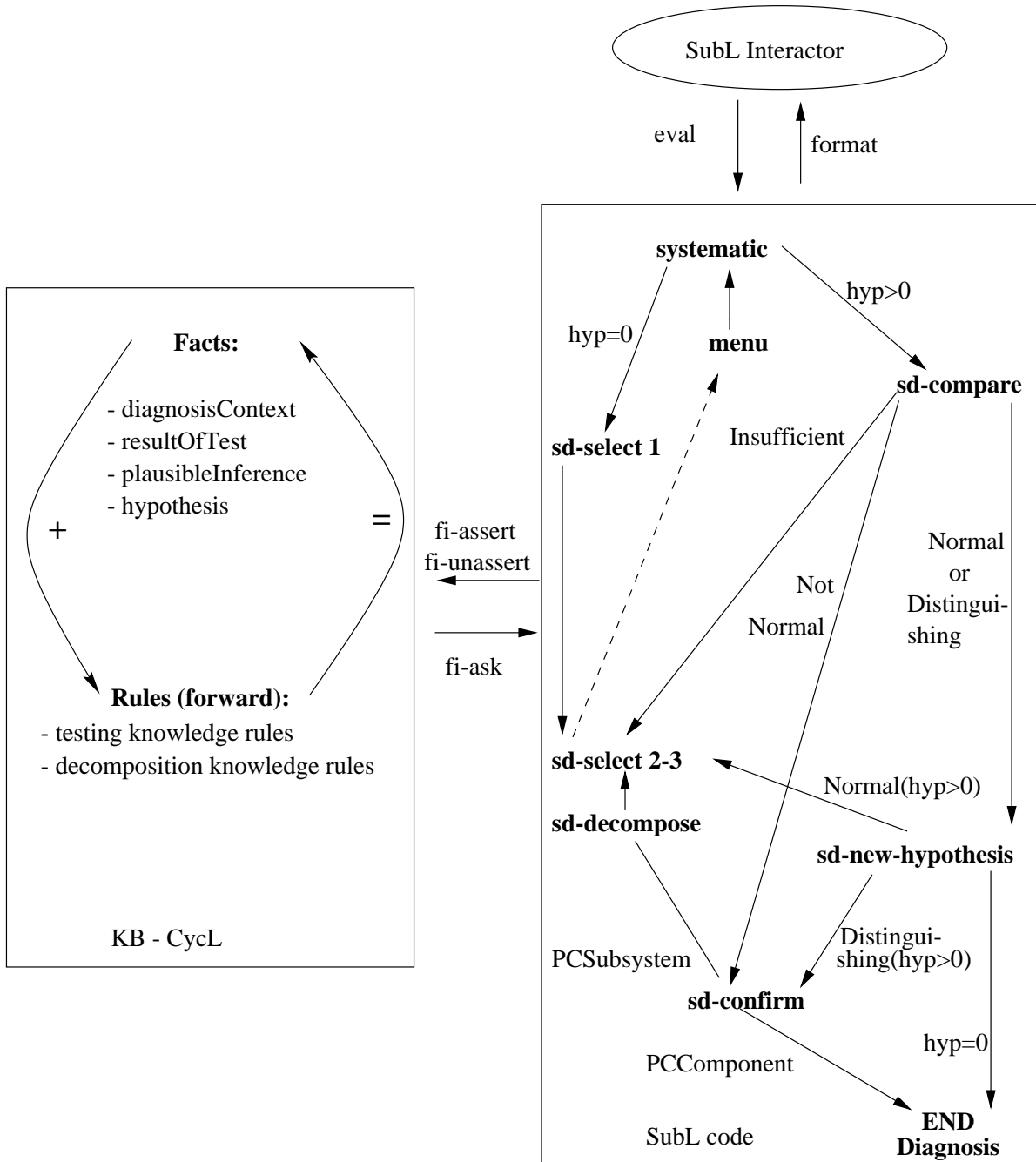


Figure 4.4: The Overall Model (detailed view)

The forward rules are involved only in answering questions to the KB and they can be replaced in the KE-text by the usual, backward rules of CYC, with minimal modifications to the SubL code<sup>5</sup>.

2. The aforementioned rules assert in the KB facts about the possible tests that may be performed, their possible results and the types of these results. Therefore, the *select2-3* inference is actually implemented through these forward rules while the `sd-select2-3` (C:172) function is responsible only for asking the KB which is the next possible test and present it to the user in the SubL Interactor panel (Figure 4.6).
3. This is the place where a technical problem occurred, which had a drawback and a benefit. The problem is that SubL code *cannot ask for direct input from the user* through the Interactor panel. This is a problem caused by the implementation of the CYC Web interface. Therefore, for the user to interact with the SubL code, the code must stop running and the user must enter the input needed by the code as a parameter to the `menu` (C:332) function(Figure 4.6). This is reflected in figure 4.4 by the dashed arrow to the `menu` function. The drawback now is that the code of the `systematic` (C:28) function does not reflect the Task Structure in figure 3.2, although the structure of the SubL code in figure 4.4 reflects the Task Structure when the `menu` function is removed and its calling functions are directly connected to the `systematic` function. The benefit is that, since the SubL code execution is interrupted and the control goes back to the CYC interface, the user can now go to other panels of the interface, like the Ask interface or the KB Browser interface and view the contents of the KB *as they have been modified* by the SubL code, with the following advantages:

- The user can explicitly view which the current state in the diagnosis problem solving method is.

---

<sup>5</sup>A version of the SubL code that works with backward rules was developed but it is not included in this thesis, since the only modifications made were just adding some extra arguments to the calls of `fi-ask` function, that is, adding some parameters to the `fi-ask` function to perform backward chaining (see Appendix A).

- The user can use the justification mechanism of CYC to get a justification of the current state of the diagnosis.
  - The user can even explicitly change the current state of the diagnosis by changing the contents of the KB and experiment with the system, although this will probably lead to system malfunction and would be recommended only to users experienced with the co-operation of the SubL code and the contents of the KB.
4. The `menu` function asserts into the KB the actual result of the last test performed and the result's type. It then calls function `systematic` to continue the diagnosis process. Having performed the *select2-3* inference, the next inference to be performed is *compare* and therefore the corresponding function, `sd-compare` (C:43) is called. This function decides which function-inference to perform next according to the result type:
- If `#$Normal` or `#$Distinguishing`, then the subsystem which is the current hypothesis is not faulty and the next hypothesis, if one exists, must be considered (Figure 4.7). This is done by the `sd-new-hypothesis` (C:123) function which does not correspond to an inference but its a control function added as a result of the technical problem described above.
  - If `#$NotNormal`, then the subsystem which is the current hypothesis is faulty and it must be either decomposed, if it is a `#$PCSubsystem` (Figure 4.8), or located as the faulty component, if it is a `#$PCComponent` (Figure 4.9); that is what the `sd-confirm` (C:94) function does.
  - If `#$Insufficient`, then a new test must be performed and therefore, the `select2-3` function is called (Figure 4.10) .

It must be noted that the *decompose* inference is implemented through *forward* rules in the same way that the *select2-3* inference is implemented, while the `sd-decompose` (C:358) function performs only *managing* work with the KB by asking, asserting and retracting.



```
(systematic '#$PCSystem)I
```

Figure 4.5: Starting the diagnosis session

```
HYPOTHESIS: #$PCSystem
NEW TEST:
PCSubSystem: #$PCSystem
TestAction: #$ConfirmSensorially
Observable: #$ProblemContext
1. #$ComponentSpecific
2. #$RunTime
3. #$BootTime

Please, type '(menu [number_of_result])'
```

((HYPOTHESIS: [PCSystem](#))  
 (NEW TEST [TestFn PCSystem ConfirmSensorially ProblemContext](#))  
 (1. [ComponentSpecific](#))  
 (2. [RunTime](#))  
 (3. [BootTime](#)))

Enter Form :

```
(menu 3)I
```

Figure 4.6: Presenting a Test to The User

```
Results :
LAST TEST: (TestFn PowerSystem ConfirmSensorially ElectricPower),
RESULT: Yes, RESULT TYPE: Normal, INFERENCE: NewHypothesis

HYPOTHESIS: #$VideoSystem
NEW TEST:
PCSubSystem: #$VideoSystem
TestAction: #$ConfirmSensorially
Observable: #$VideoSignal
1. Yes
2. No

Please, type '(menu [number_of_result])'
```

((LAST TEST: [TestFn PowerSystem ConfirmSensorially ElectricPower](#))  
 (RESULT: [Yes](#) RESULT TYPE: [Normal](#) INFERENCE:NEW\_HYPOTHESIS)  
 (HYPOTHESIS: [VideoSystem](#))  
 (NEW TEST [TestFn VideoSystem ConfirmSensorially VideoSignal](#))  
 (1. [Yes](#))  
 (2. [No](#)))

Figure 4.7: New Hypothesis After a “Normal” Result Type

```

LAST TEST: (TestFn PCSystem ConfirmSensorially ProblemContext),
RESULT: BootTime, RESULT TYPE: NotNormal, INFERENCE: Confirm/Decompose Decomposing...

HYPOTHESIS: PowerSystem
NEW TEST:
PCSubSystem: PowerSystem
TestAction: ConfirmSensorially
Observable: ElectricPower
1. No
2. Yes

Please, type '(menu [number_of_result])'

((LAST TEST: TestFn PCSystem ConfirmSensorially ProblemContext)
 (RESULT: BootTime RESULT TYPE: NotNormal INFERENCE:CONFIRM)
 (HYPOTHESIS: PowerSystem)
 (NEW TEST TestFn PowerSystem ConfirmSensorially ElectricPower)
 (1. No)
 (2. Yes))

Enter Form :   

(menu 2)

```

Figure 4.8: Decomposing a PCSystem After a “Not Normal” Result Type

```

Results :

LAST TEST: (TestFn PowerCable Replace ElectricPower),
RESULT: Yes, RESULT TYPE: NotNormal, INFERENCE: Confirm/Decompose

DIAGNOSIS ENDED. FAULTY COMPONENT: PowerCable

((LAST TEST: TestFn PowerCable Replace ElectricPower)
 (RESULT: Yes RESULT TYPE: NotNormal INFERENCE:CONFIRM)
 (DIAGNOSIS ENDED. FAULTY COMPONENT: PowerCable))

```

Figure 4.9: Confirming a PCComponent After a “NotNormal” Result Type

```

Results :

LAST TEST: (TestFn VideoSystem ConfirmSensorially VideoSignal),
RESULT: Yes, RESULT TYPE: Insufficient, INFERENCE: Select2-3

HYPOTHESIS: VideoSystem
NEW TEST:
PCSubSystem: VideoSystem
TestAction: ConfirmSensorially
Observable: VideoBIOSMessage
1. No
2. Yes

Please, type '(menu [number_of_result])'

((LAST TEST: TestFn VideoSystem ConfirmSensorially VideoSignal)
 (RESULT: Yes RESULT TYPE: Insufficient INFERENCE:SELECT2-3)
 (HYPOTHESIS: VideoSystem)
 (NEW TEST TestFn VideoSystem ConfirmSensorially VideoBIOSMessage)
 (1. No)
 (2. Yes))

```

Figure 4.10: New Test After an “Insufficient” Result Type

In the SubL code listing in Appendix C, some more functions can be found. These are support functions and a brief description of them is given in table 4.1.

Function	Description
<code>get-test-result</code>	Presents the test to be performed by calling the appropriate functions. It provides to the user HTML links to every term of the test in the CYC KB (C:214).
<code>present-test-parameters</code>	Presents the test parameters, that is, the <code>#\$PCSubSystem</code> , <code>#\$TestAction</code> and <code>#\$PossibleObservable</code> (C:246).
<code>present-test-results</code>	Presents the test's possible results (C:261).
<code>position-list</code>  <code>get-ask-binding</code>	Provide access to the bindings lists that function <code>fi-ask</code> returns. These lists are the ones returned when a question is made from the Ask interface (C:295, C:314).
<code>sd-reset</code>	Resets all the assertions concerning the systematic diagnosis problem solving method in CYC's KB. Must be called once before a diagnostic session begins (C:400).

Table 4.1: The SubL Support Functions

#### 4.4.1 Maintaining the Declarative Scheme

A close examination of the SubL code in Appendix C for the function-inferences reveals that all they do is *managing* work:

1. they *ask* questions to the KB,
2. according to the results of these questions, they *retract* from the KB knowledge that is no more valid/necessary, *assert* knowledge that is valid/necessary and they *decide* which inference to perform next.

Therefore, *there is not any domain knowledge* hard-wired inside the code and the control knowledge is clearly seen. In turn, this means that a human user could perform the

problem solving method equally well from the Ask interface, provided that he/she knew what question to ask to CYC each time and which assertions are valid/needed in each step of the method. This is the preservation of the declarative scheme in the implementation of the problem-solving method through SubL code.

## 4.5 Extending the system

The original goal of this research was to investigate the potential of using KADS as a methodology for developing problem solving methods (PSMs) in CYC. As far as it concerns KADS itself, the Generic Task Model (GTM) that forms the frame for developing the Expertise model is *independent* of the domain in which it is applied; it is only dependent on the task that must be performed in this domain. This in turn suggests that a significant part of the knowledge developed for a task in a specific domain should be easily re-usable for the same task in another domain. But the issue is, how easily? Moreover, with CYC given as the implementation environment, would this affect the ease of extending the GTM in another domain for the same task?

In order to answer these critical questions, the next step was to implement Systematic Diagnosis in another domain. The domain was Automobile fault diagnosis, restricted just to the Ignition system. The changes that occurred are discussed below<sup>6</sup>.

### 4.5.1 Changes of Microtheories

In the implementation of Systematic Diagnosis only for the PC domain, all the knowledge was grouped in a single microtheory, #PCDiagnosisMt. However, this was done for reasons of simplicity. Despite this simplification, it was obvious, even from the analysis phase, that a great deal of the knowledge did not have to do with the PC domain but the problem solving method itself. For example, the general concepts of the testing knowledge (#Test, #TestAction, #PossibleObservable, #PossibleObservableValue) are concepts related with the Systematic Diagnosis PSM rather than with the specific domain. Therefore, by introducing another domain, this knowledge should be separated from the PC domain knowledge, and it should also be available to the Automobile domain.

---

<sup>6</sup>Numbers in parentheses refer to the line numbers of the KE text listing in Appendix D.

In CYC terms, a different hierarchy of microtheories was needed. Two new microtheories were created, `#$SystematicDiagnosisMt` (D:9-18) and `#$AutomobileDiagnosisMt` (D:378-385), in addition to the already existing microtheory, `#$PCDiagnosisMt` (D:761-767). Of course, the latter two, are extensions of the first one, which in turn is an extension of the `#$BaseKB` microtheory; these relationships are given in Figure 4.11. It is obvious that there is a nice mapping between KADS and CYC: one microtheory corresponds to each domain and to each problem solving method.

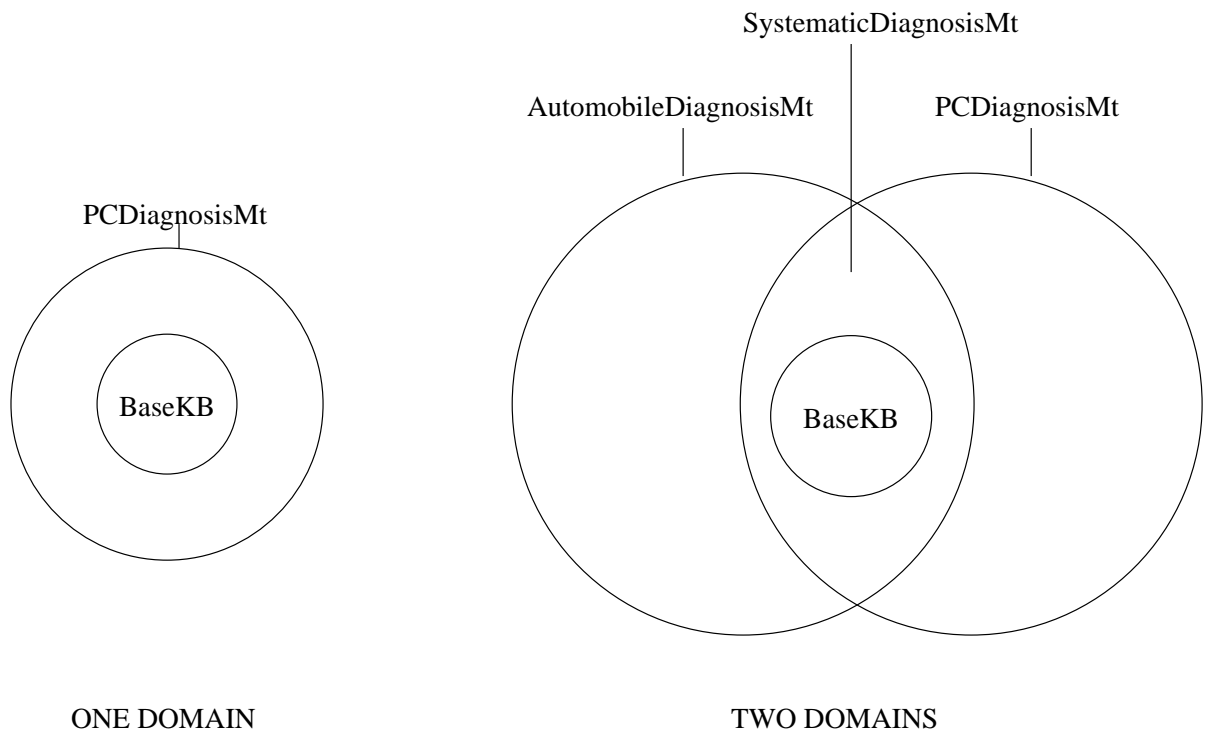


Figure 4.11: Changes in the Microtheories

### 4.5.2 Changes in the Domain Layer

Given the implementation of the Systematic Diagnosis PSM, and the three microtheories, the following changes have to be done in the domain layer (see Figure 4.12) :

1. Generalise concepts, relations and rules in the task-specific microtheory (`#$SystematicDiagnosisMt`). The generalisation is based on the common use of knowledge in both domains. Mainly this kind of knowledge is task and not domain dependent.

The generalisation may demand moving already existing domain structures from the domain-specific microtheories (`#PCDiagnosisMt` and `#AutomobileDiagnosisMt`), e.g. `#Test` (D:106), `#hypothesis` (D:65), to the task-specific one or creating new domain structures in the task-specific microtheory, e.g. `#SubSystem` (D:26), `#Component` (D:36).

2. Specialise concepts, relations and rules in the domain-specific microtheories. The specialisation is based either on connection of the domain knowledge with the task knowledge or on demands for special domain knowledge. In the first case, the new domain structures are specialisations of domain structures in the task microtheory, e.g. `#AutomobileSubSystem` (D:395), `#PCComponent` (D:787). In the second case, the new domain structures express domain-specific variations, e.g. `#physical-Decompositions` (D:44, D:500-518) and `#functionalPartOf` (D:925), which describe the system model in a structural and functional way correspondingly.

### 4.5.3 Changes in the Inference and Task Layers

As expected, the changes in the Inference and Task layers were minimal, practically ignorable. This fact is supported by the following:

1. The rules for testing and decomposition (D:536-738 for `#AutomobileDiagnosisMt` and D:1208-2134 for `#PCDiagnosisMt`), which implement the *select2-3* and *decompose* inference types, have exactly the same structure in both domains and
2. The SubL code, which implements the task structure, has changed only to differentiate the domain microtheory in which the KB operations (`fi-ask`, `fi-assert`, `fi-unassert`) are performed.

The fact that these two layers which constitute the GTM for Systematic Diagnosis were practically unchanged, implies that developing problem solving methods in CYC, with KADS as the developing methodology, provides modularity and ease of extension, knowledge re-use and analogy-driven knowledge acquisition between existing and new domains. These issues will be further discussed in the next chapter.

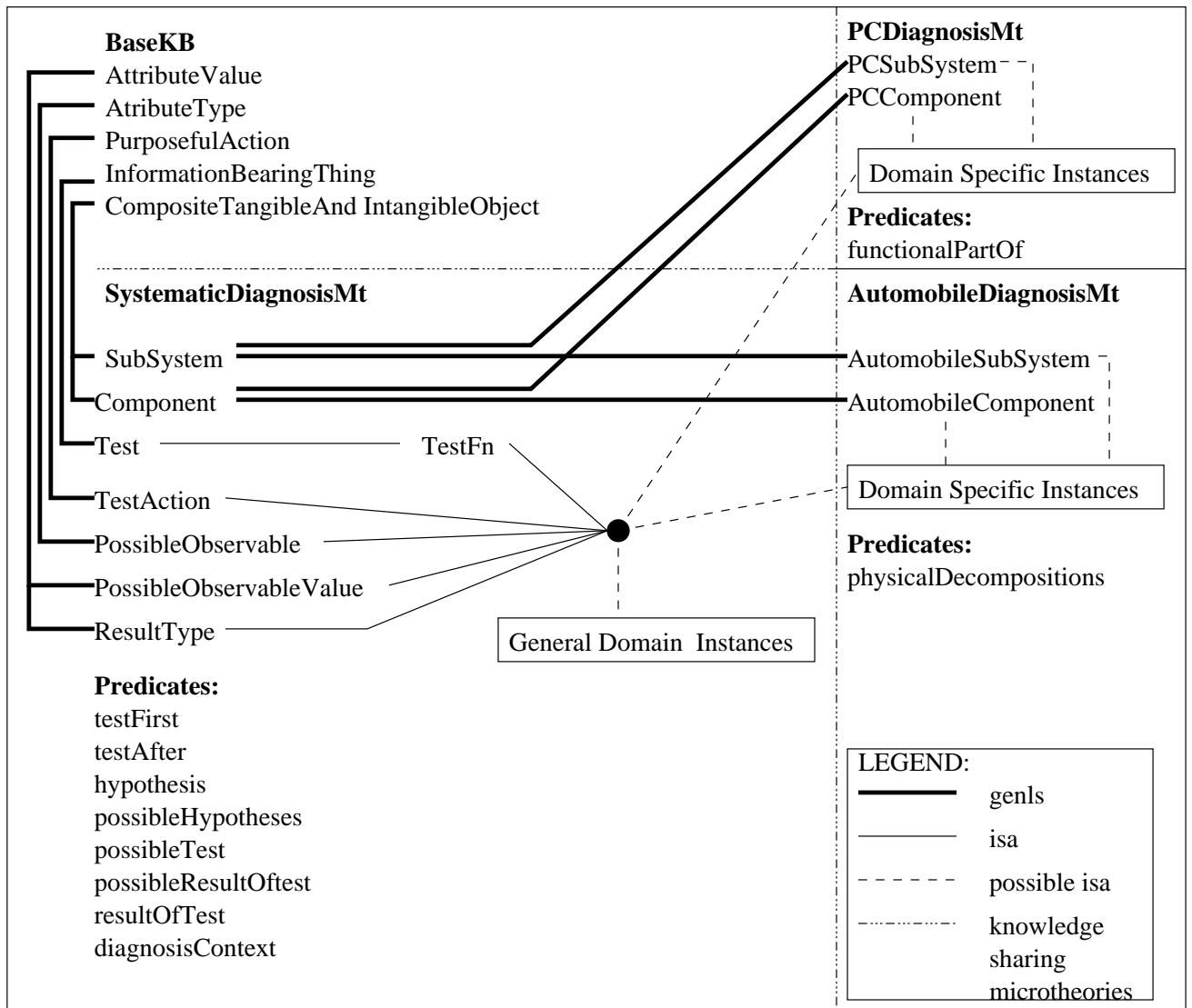


Figure 4.12: The Changes in the Domain Layers

## 4.6 Overview

In this chapter, the implementational decisions for the Systematic Diagnosis PSM in the domain of PC fault diagnosis were discussed. The main decisions were to implement the Domain layer in CYC's KB (using CycL) and the Inference and Task layers to be implemented in Lisp code (using SubL). The whole task is controlled by the Lisp code which operates (asks, asserts and retracts facts) on the KB through the Functional Interface. The main advantage of this implementation is that preserves to a considerable extent the *declarative scheme* of knowledge representation and inference in CYC. A small but conceptually essential extension of the system implemented in the domain of Automobile fault diagnosis was discussed, pointing out the main issues of reusing PSMs under the implementation developed.



# Chapter 5

## Issues and Results

Computers are useless. They can only give you answers.

- Pablo Picasso

An expert is a man who has stopped thinking - he knows!

- Frank Lloyd Wright

### 5.1 Introduction

The principal goal of this research was to implement in CYC a problem solving method (PSM) from the KADS methodology and study the issues that arise with this implementation. The principal issue to study was whether implementing a PSM in CYC would increase its reasoning power. Other issues were:

1. How well does KADS “fit” into CYC? KADS, as a methodology is implementation-independent. Its products, the various models from the analysis and design phases, can potentially be implemented in any implementation environment. CYC provides a declarative representation language, CycL, and a procedural/functional dialect of Lisp, SubL. The issue then is what part of CYC implements each product part of KADS and how.
2. Does KADS contribute to the development of CYC’s Ontology and, if so, in what way?

But first, let us consider the first issue.

## 5.2 Problem Solving in CYC

The final objective of any intelligent agent is to solve problems. Intelligence itself is defined as the ability to solve problems. Expert Systems (ES) are the most developed problem solvers in the field of AI. However, first-generation ES suffered from what was called “the knowledge brittleness”:

- They could not handle missing, incomplete or imprecise data.
- They could not give satisfactory explanations for what they were doing to solve a problem and how they were doing it.
- They could not handle situations which were not foreseen by their programmers and did not have explicit knowledge for them.

In general, the aforementioned problems could all be described as inability of the ES to fall back to “first principles”, to use *common sense*. The motive for developing CYC was overcoming this “brittleness”, this lack of common sense ([Lenat & Guha 90]). To build a system that could perform common sense reasoning, one should provide it *both* with a lot of common sense knowledge, facts and rules of thumb that an average person knows about the world, and common sense inferencing mechanisms like *analogical reasoning* that will perform on that knowledge. In their mid-term report book, [Lenat & Guha 90], the writers admit that the CYC project focused on the development of an *Ontology* of common sense knowledge, as

...successful analogizing depends on ... having a realistically large pool of (millions of) objects, substances, events, sets, ideas, relationships, etc, to which to analogize.

This focus to the common sense knowledge itself led to an imbalance as far as it concerns the problem solving abilities of CYC. The author’s experience with CYC is that, although it provides a quite sophisticated environment for developing any kind of knowledge, it lacks the support for using that knowledge. This imbalance is explained in more detail below.

### 5.2.1 Inferencing in CYC

Currently, CYC starts its inferencing when a question, expressed in its representation language CycL, is asked from its Ask interface. To answer a question, CYC uses two kinds of inference mechanisms:

1. Backward chaining with resolution, as a general, “weak” inference mechanism.
2. Heuristic level, special-purpose inference mechanisms that decide which nodes to open at each stage of this backward chaining, that is which reasoning path to pursue next. These heuristic inference mechanisms are based on the syntactic form of each (sub)goal as well as on the occurrence of special predicates like `#$isa`, `#$genls` and others.

CYC also makes use of *microtheories* to restrict the search space. However, it is the author’s opinion that these inference mechanisms are simply not enough to produce what CYC is intended to produce: *expertise*. “Weak” inference mechanisms are just to support expertise and not to produce it. A discussion about this issue can be found in [Luger & Stubblefield 98]. Moreover, the special-purpose heuristic inference modules only support the implementational level of inferencing and have nothing to do with the knowledge-level ([Newell 82]) which primarily concerns an ES. A discussion about this issue can be found in [Steels 90] and [Chandrasekaran 86]. Most important, the general, “weak” methods of inferencing, like backward reasoning with resolution, suffer heavily from the *combinatorial explosion* of the state space ([Luger & Stubblefield 98]), and this is more severe in the case of CYC’s huge state space, its KB, containing  $10^6$  common sense axioms and still growing ([Lenat 95]). Therefore, it becomes obvious that what is missing from the inferencing power of CYC is what I would call an *Ontology of inference mechanisms*, that is, a large set of elementary inference mechanisms that perform elementary knowledge transformations and can be combined together to generate more complex inferencing for more complex problem-solving situations. In [Lenat & Guha 90] it is explicitly stated that such a set of inference mechanisms should be an indispensable part of any reasoning system:

...Breaking down the phenomenon [i.e. analogical reasoning] into its various subtypes and then handling each one.

### 5.3 Problem Solving Methods in CYC

Problem Solving Methods (PSMs), as they were described in this thesis, provide a source for this ontology of inference mechanisms. Although PSMs have been developed in a background broader than that of KADS, as general building blocks of expertise ([Chandrasekaran 86]), KADS methodology provides an elaborate organisation of these mechanisms, in the form of:

1. Primitive inference mechanisms, the *inference types*, with their elementary input and output, the *domain roles*
2. Structured entities of these primitive inferences, the *Generic Task Models*.

Whether this scheme is the best it is not of importance here. The important is that there are some elementary inferences and a way to combine them together to produce complex problem solving behaviour. It is obvious that this is the analogous of having a common sense knowledge base in the knowledge level, in another level (or meta-level), the inference level. Research in the development of ES has shifted to that level, either in the form of the *knowledge-use* level ([Chandrasekaran 86], [Steels 90]) or in the form of *proof plans* ([Bundy 88]). This shift cannot be ignored, especially in the case of CYC which was built as an ES development platform.

I would try a metaphor here to underline the importance of this level. I think that inference mechanisms are to intelligence what the natural laws are to physics and functions are to mathematics: they describe the fundamental interactions and transformations of the structural materials of each domain, namely of the knowledge, the matter/energy and the quantities correspondingly. This metaphor will be useful right below, where the implementation of the Systematic Diagnosis, the PSM selected for this thesis, is discussed.

#### 5.3.1 Implementing the Systematic Diagnosis PSM

The above discussion gives an answer to the theoretical issue of *why* implement PSMs in CYC. In chapters 3 and 4 a thorough description of *how* this implementation was done is given.

The most important result of this implementation is that *the inferencing power of CYC improved*, as it can now perform systematic fault diagnosis not only in the domain of PCs, but in a range of domains as it was proved by the extension of the system in the domain of Automobiles (§4.5). The importance comes from the fact that CYC could not perform the same task by simply backtracking through its knowledge base; systematic fault diagnosis demands *dynamic* gathering of data through queries from the system to the user and consequent dynamic update of the knowledge base. These dynamic information collection and manipulation operations do not lend themselves to explicit declarative reasoning but naturally demand procedural and functional reasoning. Although CycL, the representation language of CYC, provides the means for calling LISP code (SubL), these means are not enough either to provide interaction with the user or to dynamically update the knowledge base of CYC. Hence, the implementation of the Task Structure, the overall plan of the PSM, in SubL (§4.2), and the use of the functional interface (FI) to ask and update the CYC KB *dynamically*, that is, assert or retract facts.

This latter fact, that simple backtracking and matching with the contents of the KB, in spite of its powerful generality, is not enough to implement any kind of problem solving task, proves in practice that a general reasoning system must provide both a variety of inference mechanisms and the means to combine and control them. The implementation of the Systematic Diagnosis problem solving method, as described in this thesis, establishes both a source for these mechanisms (KADS) and a way to implement them in CYC.

### 5.3.2 The Implemented System

Despite the lack of a variety of inference mechanisms, the CYC system provides an excellent platform for developing expert systems (ES). This is reflected in the expert system developed for this thesis that performs systematic fault diagnosis. The system has all parts of a typical ES:

- A menu-driven user interface, which additionally provides HTML links to any term in the KB, making the system self-explainable and documented by direct reference to its terms.
- Double inference engine: the built in backward chaining CYC inference engine for

answering questions and the Task Structure encoded in SubL for performing the overall task of diagnosis.

- An explanation facility. Although not explicitly implemented, there is an explanation sub-system available through the [Justify] option that the CYC KB Browser provides for assertions in the CYC KB. Since the user can view the contents of the KB at any time during the diagnostic process, he/she can see the justification of any assertion in the KB in terms of which rules/facts prove the assertion. From this limited explanation facility, a more complex could be built by just accumulating in a list the justifications of every assertion relative to the diagnostic process. This feature can be implemented in the SubL code through the `fi-justify` function (see Appendix A).
- A KB editor, built in CYC.
- A general KB, CYC's `#$BaseKB` microtheory.
- Domain-specific knowledge, the `#$PCDiagnosisMt` and `#$AutomobileDiagnosisMt` microtheories.

The implemented system is itself a hybrid rule-based and model-based one (the model of the diagnosed system is implemented via rules), with all the advantages of such systems:

- Direct use of heuristic, unformalised diagnostic knowledge
- Modularity of rules
- Separation of domain and control knowledge which results to easiness of tracing, debugging and extending.
- Good performance in a *limited* domain.

The last characteristic of the system must be discussed more in detail as it touches on the motivating issue for building CYC itself. The motive was to overcome the “brittleness” of ES. This “brittleness” describes the inability of ES to handle novel or unexpected situations. Humans can handle these situations generally by falling back on “first principles” or general (common sense) knowledge. CYC was build to provide this common

sense knowledge as a common substrate for the development of ES. Therefore, one would expect that the system that was developed in this thesis and any other ES that would have been developed in CYC would not suffer from this “brittleness”. This is not the case by any means. The system of this thesis and any other ES developed in CYC would suffer from “brittleness” *unless it was explicitly designed* to take advantage of CYC’s common sense ontology. The reason for this is simple. An ES has as its purpose to produce the same behaviour as a human expert. But, most of the time, a human expert uses *heuristic knowledge*, that is knowledge which is a “distilled” part of both the general and domain-specific knowledge which has proved to be the most important and useful for the expert to perform his task. Of course, the human expert is completely aware *how* he formed this heuristic knowledge and, in case it is not directly applicable for reasons of novelty, can review it and reason about it. But to reason about his heuristic knowledge, the human expert must use the original knowledge from which he formed the heuristic one. These two levels of knowledge are known in the theory of ES as *surface and deep* knowledge correspondingly ([Steels 90]). This is where CYC comes. It provides the “deep” knowledge that is needed for an ES to reason about its heuristic one. But the ES must be designed to be able to perform that kind of reasoning, that is reason about its heuristic knowledge using general knowledge. An example from the actual system will make things clear.

In order to perform diagnosis in PCs, the system has heuristic rules about the *order* of diagnosis. This is because the order is important for the diagnosis (see §3.4.1). Of course, the system does not know why the order in its rules is the one encoded. If, for some reason, this ordering could not be applied, then the system is unable to reason. But, if it should be able to reason about the rules themselves, that is check them for consistency, alter them or even infer them, then more knowledge would be needed. Specifically, in the case of the ordering rules, the system would probably need some or all of the following:

1. A structural model of the system, the way the components are connected together to form the various (sub)systems. This model by itself would need a complete ontology, like the one found in [Borst *et al.* 97].
2. A theory of “ordering” in testing, e.g. easiness, that would require certain criteria (rules) of which systems are easy to test:

- Systems that can be tested sensorially (visually, acoustically) rather than by instruments.
- Systems that are easy to replace.
- Systems that are easy to test in isolation.
- Systems with simple structure (Components vs. Sub-systems)
- Systems that are easy to access (e.g. terminal systems)

Note that the knowledge outlined above could need more knowledge like knowledge about serial and parallel connections, control devices, input and output of devices, etc.

It is obvious that for just a simple group of rules, the rules of ordering, the knowledge of the system would have to grow dramatically. And this can happen for every part of the system's heuristic knowledge: what test to perform, how to perform it, what system variable to test, what makes a system variable to be such and so on. The obvious conclusion is that to overcome "brittleness" two things are needed:

1. A large (huge in the case of CYC) general (and less general) knowledge base and
2. An elaborate connection of the heuristic knowledge of the ES with the appropriate part of the general knowledge.

Consequently, even with CYC's upper ontology in hand, it requires a tremendous effort to build the part of general knowledge that an ES needs to reason about its heuristic knowledge and then to put things together, that is design when, why and how the system will fall back to general knowledge<sup>1</sup>.

With the discussion of the "brittleness" problem concludes the discussion about the main issue of this thesis and the corresponding results. However, in the introduction of this chapter two secondary issues were mentioned, concerning KADS and CYC. These issues are discussed in the next section.

---

<sup>1</sup>This situation is well described in Lenat's report for using CYC in the High Performance Knowledge Base project, which can be found in <http://www.cyc.com/hpkb/proposal-summary-hpkb.html>



## 5.4 KADS and CYC

Problem solving methods (PSMs) were developed in the context of ES development methodologies, such as KADS. Therefore, PSMs can come from somewhere else than KADS. However, KADS, as a complete ES building methodology, provides the following as far as it concerns PSMs:

- A hierarchy of PSMs according to the task they are appropriate to (see Figure 2.5).
- A library of more elementary inference types that can be combined to build new PSMs (see Figure 2.1).
- Domain roles for the various inference types that guide the knowledge acquisition process (see Table 3.1).
- A useful distinction of the four layers of a PSM (domain, inference, task and strategy) and their interrelationships in the Expertise model which they form (see Figure 2.4).

It is obvious that KADS provides much more than a simple source of PSMs and therefore it is worth investigating how much KADS fits into CYC.

The first issue is that of the modelling process of KADS. As described in chapter 2, this is the Expertise model, consisting of four layers, the Domain, Inference, Task and Strategy ones. In chapter 4, a mapping between these layers and some parts of CYC was given, that is:

1. The Domain layer is implemented as a microtheory in CYC KB.
2. The Inference layer is implemented partly in the KB in the form of rules, as far as it concerns the transformation of the domain structures, and partly in the SubL code as functions-inferences, as far as it concerns the management of the transformed domain structures, that is retrieving, asserting and retracting them.
3. The Inference layer is implemented in the SubL code which has the overall inferencing control of the PSM.

4. Knowledge that is specific to the PSM is encoded in the KB in a separate micro-theory.

The advantages of this mapping is that it distinguishes between the various kinds of knowledge needed to perform the task of the PSM and that it preserves to a great extent the declarative scheme of CYC (§4.4.1).

The second issue is how much KADS contributes to the development of CYC's ontology. From the description of the Analysis phase (chapter 3), the system extension (§4.5) and the issue of "brittleness" (§5.3.2), it becomes obvious that KADS provides an excellent methodology for developing CYC's ontology in a bottom-up manner, that is from the heuristic, surface, task-specific knowledge to the common sense, deep, task-independent knowledge, through the development of intermediate ontologies that will fill the gap between these two kinds of knowledge, as shown in Figure 5.1.

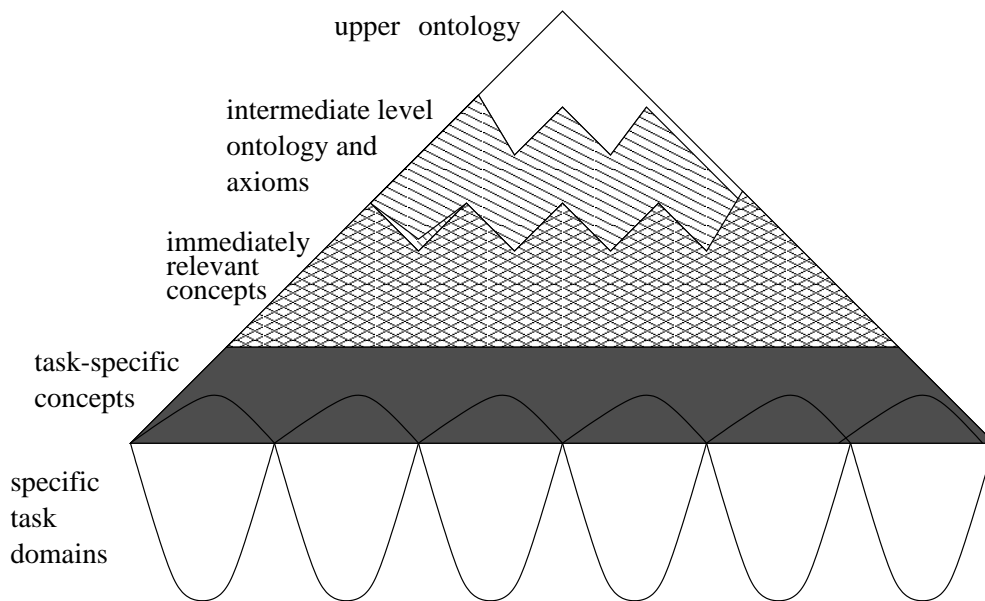


Figure 5.1: Filling the Knowledge Gap

In [Lenat & Guha 90] it is explicitly mentioned that CYC's ontology should be neither an encyclopedia of linearly arranged terms nor a thesaurus of disconnected knowledge. The main point of KADS is that it views ES building not as filling a pool of knowledge but as modelling the expertise behaviour according to specific tasks. This approach is

also justified by the way humans acquire their expertise: by tackling a large diversity of tasks which continuously grow in complexity. Most of our knowledge is task-oriented. It is acquired and connected with the rest of the already existing knowledge that we have acquired, according to how, when and for what purposes it is useful. This is the approach on which KADS is based: KADS supports this approach both by its general modelling directions and the *domain roles* that introduces. On the other hand, KADS is a complete methodology that supports the whole cycle of an ES development, from the knowledge acquisition ([Kingston 94]), to analysis and design ([Tansley & Hayball 93]); it is even possible to use KADS for small ES as described in [Kingston 92]. It is therefore obvious how much KADS can contribute to the development of CYC's ontology in a bottom-up manner. Of course, the top-down direction of developing a global ontology is also useful and necessary, with its own advantages and disadvantages ([Guarino 98], [Smith 98], [Varzi 98]).

## 5.5 Further Work

It was explained before in this thesis that the implemented system was kept to the minimum necessary to investigate the main issue, which was the implementation of PSMs in CYC. However, a number of possible extensions can be considered:

1. Extending the domain knowledge: in section 3.2, it is mentioned that although the “Expert” used to make the knowledge acquisition included three troubleshooting contexts for PCs, only the Boot-time context was analysed. It would be a good test for the implementation to be expanded in the other two domains: the Run-time and Component-specific contexts. In the case of Automobiles this is even more necessary, since only a small part of the knowledge about the ignition system was analysed.
2. Developing some “deep” knowledge: in section 5.3.2, in the discussion about the “brittleness” of the system, it was mentioned that a kind of “ordering theory” would be needed if the system should be able to reason about the ordering of the tested systems. Some rules were given but more elaborate work should be carried out.

3. Implementing another PSM either for the same task or for a different one: it was mentioned that Heuristic Classification could be used for troubleshooting, while a configuration task could probably also make use of some of the knowledge encoded for troubleshooting. Implementing PSMs either for the same task or for a different one could promote the investigation of issues of knowledge re-use from another PSM and of how well the decisions for the implementation of Systematic Diagnosis work for another PSM.
4. The three previous extensions would provide a good paradigm for investigating how easily KADS contributes to the development of intermediate ontologies in CYC.
5. A more elaborate justification mechanism could be developed, by accessing CYC's built-in justification mechanism, keeping the necessary justifications for every inference step during the diagnosis process.
6. Although improving its reasoning power, the various parts of the PSM like the *inference types*, *inference structure*, *task structure* and *generic task model* are not known to CYC since they were implemented in SubL code. It could be a very interesting extension to investigate to what degree all these could be encoded *declaratively* using CycL, and consequently be used by CYC in its usual reasoning schema. A possible application could be to develop an ES that would guide a knowledge engineer to use KADS for developing knowledge-based systems, as described in [Kingston 95].

## 5.6 Conclusions

From the discussion in this chapter, the following conclusion come out of this thesis:

1. The inference mechanism of CYC, backward chaining with resolution, is not strong enough to perform more complex tasks that require dynamic information collection and KB updating, as is the case in the systematic diagnosis of faults in PCs. Moreover, this “weak” inference mechanism suffers from the *combinatorial explosion* problem.

2. The implementation of problem solving methods (PSMs) reinforces the inferencing power of CYC and enriches it with new inference mechanisms.
3. KADS methodology provides a rich source for PSMs and a structured way for using them in every aspect of problem solving, from knowledge acquisition to analysis and design of the problem solving model.
4. Moreover, KADS provides a systematic, task-oriented way for developing CYC's Ontology in a bottom-up way: from the task-specific concepts to the task-independent ones which form CYC's Upper Ontology.

All these conclusions should motivate further research in combining these two very promising state-of-the-art components of Artificial Intelligence.



# Bibliography

- [Blair *et al.* 92] P. Blair, R.V. Guha, and W. Pratt. Microtheories: An Ontological Engineer's Guide. Cyc Technical Report CYC-050-92, Microelectronics and Computer Technology Corporation (MCC), March 1992.
- [Borst *et al.* 97] P. Borst, H. Akkerman, and J. Top. Engineering Ontologies. *International Journal of Human Computer Studies*, 46(213), 1997.
- [Bundy 88] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Ninth Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988.
- [Chandrasekaran 86] B. Chandrasekaran. Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert*, 1(3), 1986.
- [Clancey & Letsinger 82] W.J. Clancey and R. Letsinger. Neomycin: Reconfiguring A Rule-Based Expert System for Application to Teaching. Technical Report STAN-CS-82-908, Department of Computer Science, Stanford University, May 1982.
- [Clancey 85] W.J. Clancey. Heuristic Classification. *Artificial Intelligence*, (27), 1985.

- [Guarino 98] N. Guarino. Formal Ontology and Information Systems. In N. Guarino, editor, *Proceedings of the First International Conference on Formal Ontology in Information Systems*, pages 3–15. IOS Press, 1998.
- [Hamscher 88] W.C. Hamscher. Model-Based Troubleshooting of Digital Systems. Technical Report AI-TR 1074, Artificial Intelligence Laboratory, MIT, August 1988.
- [Hesketh & Barrett 90] P.H. Hesketh and T. Barrett. An Introduction to the KADS Methodology. ESPRIT P1098, Deliverable M1, STC Technology Ltd., March 1990.
- [Kingston 92] J. Kingston. PragmaticKADS: A methodological approach to a small knowledge based systems project. Technical Report AIAI-TR-110, Artificial Intelligence Applications Institute, University of Edinburgh, November 1992.
- [Kingston 94] J. Kingston. Linking Knowledge Acquisition with CommonKADS Knowledge Representation. Technical Report AIAI-TR-156, Artificial Intelligence Applications Institute, University of Edinburgh, July 1994.
- [Kingston 95] J. Kingston. Applying KADS to KADS: knowledge based guidance for knowledge engineering. Technical Report AIAI-TR-158, Artificial Intelligence Applications Institute, University of Edinburgh, January 1995.
- [Lenat & Guha 90] D.B. Lenat and R.V. Guha. *Building large knowledge-based systems. Representation and in-*



- ference in the Cyc project.* Addison-Wesley, Reading, Massachusetts, 1990.
- [Lenat 95] D.B. Lenat. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38(11), November 1995.
- [Luger & Stubblefield 98] G.F. Luger and W.W. Stubblefield. *Artificial Intelligence. Structures and Strategies for Complex Problem Solving. 3rd ed.* Addison-Wesley, 1998.
- [Newell 82] A. Newell. The Knowledge Level. *AI Journal*, 19(2), 1982.
- [Rademakers & Vanwelkenhuysen 93] P. Rademakers and J. Vanwelkenhuysen. Generic Models and their Support in Modelling Problem Solving Behaviour. In Simons R. David, J.M. and J.P. Krivine, editors, *Second Generation Expert Systems.* Springer-Verlag, 1993.
- [Schreiber *et al.* 93] A. Th. Schreiber, B. J. Wielinga, and J.A. Breuker. *KADS: A Principled Approach to Knowledge-Based System Development.* Academic Press, London, 1993.
- [Smith 98] B. Smith. Basic Concepts of Formal Ontology. In N. Guarino, editor, *Proceedings of the First International Conference on Formal Ontology in Information Systems.* IOS Press, 1998.
- [Steels 90] L. Steels. Components of Expertise. *Artificial Intelligence Magazine*, 1990.
- [Tansley & Hayball 93] D.S.W. Tansley and C.C. Hayball. *Knowledge-Based Systems Analysis and Design. A KADS Developer's Handbook.* Prentice Hall, London, 1993.

- [Vanwelkenhuysen 92] J. Vanwelkenhuysen. Scaling-up Model-Based Troubleshooting by Exploiting Design Functionalities. In *Proceedings of the Fifth International Conference on Industrial and Engineering Applications and Expert Systems*. Springer-Verlag, 1992.
- [Varzi 98] A.C. Varzi. Basic Problems of Mereotopology. In N. Guarino, editor, *Proceedings of the First International Conference on Formal Ontology in Information Systems*, pages 29–38. IOS Press, 1998.
- [Wielinga *et al.* 92] B.J. Wielinga, A.Th. Schreiber, and J.A. Breuker. KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition*, (4), 1992.

# Appendix A

## The CYC FI Function Reference

### Core Functions

`fi-assert` : formula mt &optional (el-tv :default) direction -> [boolean]

**function:**

Add a local argument for formula to the KB within microtheory mt.

el-tv is either :default or :monotonic.

direction is either :forward, :backward, or NIL.

If direction is NIL,

ground formulas are entered as :forward,

rules are entered as :backward.

If formula is already present in mt with a different "hl truth value" (:true-monotonic, :true-default, :false-monotonic, or :false-default), change it to the new "hl truth value" determined by formula and el-tv.

If formula is already present in mt with a different direction, its direction is changed.

If direction is :forward, and formula was not already present in mt with a forward direction, then forward inference is performed on the formula.

**return:**

NIL if operation had an error.

T if operation succeeded.

**errors:**

:arg-error

one of the arguments was invalid

:not-well-formed

formula was not well-formed

**warnings:**

:redundant-local-assertion

the assertion is already in the KB locally

:change-local-tv

the assertion is already in the KB locally with a different truth value

`fi-unassert` : formula mt -> [boolean]

**function:**

Remove any local argument for formula within mt.

**return**

NIL if operation had an error.

T if operation succeeded.

**errors:**

:arg-error

one of the arguments was invalid

:not-well-formed

formula was not well-formed

**warnings:**

```
:assertion-not-present
    the formula is not in the KB at all
:assertion-not-local
    the formula is in the KB, but has no local support
```

```
fi-justify : formula mt &optional full? -> [argument]
```

```
function:
    Provide an argument justifying belief in formula within mt.
    If full? is NIL, only provide one level of argument, which may include non-ground facts.
    If full? is non-NIL, follow argument down as far as necessary to reach ground facts.
return
    NIL if operation had an error.
    NIL if it could not be justified.
    Argument if it could be justified.
    [argument] ::= list of [support]
    [support] ::= ([module] [formula] [mt])
    [module] is one of :axiom :isa :genls :equality :eval :reflexive :symmetric :transitive :external
    [formula] is the formula of the support
    [mt] is the microtheory of the support
errors:
    :arg-error
        one of the arguments was invalid
    :not-well-formed
        formula was not well-formed
warnings:
    :assertion-not-present
        the formula is not in the KB at all
```

```
fi-ask : formula mt &optional backchain? number time depth -> [ask-result]
```

```
function:
    Ask for bindings for free variables which will satisfy formula within mt.
    If backchain? is NIL, no inference is performed.
    If backchain? is T or an integer, inference is performed.
    If backchain? is an integer, then at most that many backchaining steps using rules are used.
    If number is an integer, then at most number bindings are returned.
    If time is an integer, then at most time seconds are spent on the operation.
    If depth is an integer, then the inference paths are limited to depth total steps.
return
    NIL if operation had an error.
    An ask-result if operation completed.
    [ask-result]
        ::= list of [bindings]
        ::= ((T . T)) if there were no free variables
    [bindings] ::= list of [binding]
    [binding] ::= ([variable] . [value])
    [variable] is a free variable from the formula
    [value] is the binding satisfying the formula
errors:
    :arg-error
        one of the arguments was invalid
    :not-well-formed
        formula was not well-formed
```

```
fi-continue-last-ask : &optional backchain? number time depth reconsider-deep ->
[ask-result]
```

```
function:
    Continue the last ask that was performed with more resources.
    If backchain? is NIL, no inference is performed.
    If backchain? is t or an integer, inference is performed.
    If backchain? is an integer, then at most that many rules are used.
    If number is an integer, then at most number bindings are returned.
    If time is an integer, then at most time seconds are spent on the operation.
    If depth is an integer, then the inference search cuts off at depth.
```

```

    If reconsider-deep is non-nil, then previous inference paths which were cut off for going past depth
    are reconsidered.
return
  NIL if operation had an error.
  An ask-result if operation completed.
  [ask-result]
    ::= list of [bindings]
    ::= ((T . T)) if there were no free variables
  [bindings] ::= list of [binding]
  [binding] ::= ([variable] . [value])
  [variable] is a free variable from formula.
  [value] is the binding satisfying formula.
errors:
  :arg-error
    one of the arguments was invalid
  :not-well-formed
    formula was not well-formed

fi-ask-status : -> [ask-status]

function:
  Explain why the last ask completed.
return
  an ask-status which is
    :EXHAUST if the search space was exhausted.
    :DEPTH if the search space was exhausted, but some nodes were too deep.
    :NUMBER if the requested number was reached.
    :TIME if the time allotted expired.

```



# Appendix B

## The CYC KE-Text for PC Domain

```
1   ;;; PROJECT: 628-Implementing Problem Solving Methods (PSMs) in Cyc
2   ;;; FILENAME: diagnosisKE.txt
3   ;;; AUTHOR: Dimitrios Sklavakis
4   ;;; PURPOSE: Contains Cyc's Knowledge Entering (KE) text defining the
5   ;;; knowledge (Domain and Expertise) for the implementation of the
6   ;;; 'Systematic diagnosis' PSM from KADS methodology, for PC faults
7   ;;; diagnosis.

8   ;;; LAST UPDATED: 04/08/1998.

9   ;***** DOMAIN KNOWLEDGE *****

10  ;***** THE PC DIAGNOSIS MICROTHEORY *****
11
12  ;;; The whole knowledge for performing PC fault diagnosis will be
13  ;;; entered in the PCDiagnosisMt microtheory, which is a more specific
14  ;;; microtheory of the BaseKB microtheory:
15
16  ;;; (#$genlMt #$PCDiagnosisMt #$BaseKB).
17
18  constant: PCDiagnosisMt.
19  isa: Microtheory.
20  genls: BaseKB.
21  comment: "$PCDiagnosisMt is the $Microtheory that contains all the assertions
22  about performing Personal Computer(PC) fault diagnosis.".
23
24  ; ***** THE SYSTEM MODEL *****
25
26  Default Mt: PCDiagnosisMt. ;Change default microtheory to $PCDiagnosisMt.
27
28  constant: PCSubSystem.
29  isa: Collection.
30  genls: CompositeTangibleAndIntangibleObject.
31  comment: "The collection of all PC sub-systems, like the
32  $VideoSystem, $PowerSystem, $KeyboardSystem. Each instance of
33  $PCSubSystem may include several $PCComponents and/or other
34  $PCSubSystems. Different $PCSubSystems may include the same
35  $PCComponents. In the context of $PCDiagnosisMt any $PCSubSystem is
36  an intermediate level of analysis for the $PersonalComputer; the
37  diagnosis continues until a faulty $PCComponent is located".
38
39  constant: PCComponent.
40  isa: Collection.
41  genls: PCSubSystem.
42  comment: "The collection of all PC components such as the
43  $PowerSupply, $VideoCard, $FloppyDiskDrive. In the context of
```

```

38 $$PCDiagnosisMt any $$PCComponent is the lowest level of analysis for
39 the $$PersonalComputer; the diagnosis terminates when a faulty
40 $$PCComponent is located.".
41
42 constant: PCSystem.
43 isa: Individual PCSubSystem.
44 comment: "The $$PCSystem is used to refer to the $$PersonalComputer as
45 a $$PCSubSystem. It includes the following $$PCComponents:
46 $$PowerSystem, $$VideoSystem, e.t.c.".

47 constant: PowerSystem.
48 isa: Individual PCSubSystem.
49 comment: "The power $$PCSubSystem. Includes the following
50 $$PCComponents: $$PowerSocket, $$PowerCable, $$PowerProtectionDevice
51 (optionally) and $$PowerSupply.".

52 constant: PowerSocket.
53 isa: Individual PCComponent.
54 comment: "$$PowerSocket is the socket that provides electric power to
55 the PC. It is a component of the $$PowerSystem.".

56 constant: PowerCable.
57 isa: Individual PCComponent.
58 comment: "$$PowerCable is the cable that connects the $$PowerSocket
59 with the $$PowerSupply. It is a component of the $$PowerSystem.".

60 constant: PowerProtectionDevice.
61 isa: Individual PCComponent.
62 comment: "$$PowerProtectionDevice is any device (supprocessor, UPS)
63 connected between the $$PowerSocket and the $$PowerSupply to protect
64 the $$PersonalComputer from power failures. It is an optional
65 component of the $$PowerSystem.".

66 constant: PowerSupply.
67 isa: Individual PCComponent.
68 comment: "$$PowerSupply is the component located inside the
69 $$PersonalComputer case that supplies the $$MotherBoard with electric
70 power. It is a component of the $$PowerSystem.".

71 constant: VideoSystem.
72 isa: Individual PCSubSystem.
73 comment: "The video $$PCSubSystem. Includes the following
74 $$PCComponents: $$MotherBoard, $$VideoCard, $$Monitor (and the
75 $$Speaker)".

76 constant: VideoCard.
77 isa: Individual PCComponent.
78 comment: "The $$VideoCard transforms the video information to video
79 signal and sends it to the $$Monitor. It is a component of the
80 $$VideoSystem.".

81 constant: Monitor.
82 isa: Individual PCComponent.
83 comment: "The $$Monitor transforms the video signal sent by the
84 $$VideoCard into visual image. It is a component of the
85 $$VideoSystem.".

86 constant: MotherBoard.
87 isa: Individual PCComponent.
88 comment: "The $$MotherBoard is the main $$PCComponent. Most of the
89 rest $$PCComponents are connected onto the $$MotherBoard and
90 controlled by it. It is actually a sub-system by itself as it includes
91 other components but in the context of PC diagnosis it will be
92 regarded as a $$PCComponent to avoid increasing the complexity of the
93 $$PersonalComputer analysis.".
```



94 constant: Speaker.  
95 isa: Individual PCComponent.  
96 comment: "The PC speaker. It refers to the  
97 internal speaker that is connected on the motherboard and not the  
98 external ones that are part of a multi-media system and require a  
99 sound-card on which they are connected."

100 constant: BIOSStartupSystem.  
101 isa: Individual PCSubSystem.  
102 comment: "The BIOS startup #PCSubSystem. Includes the following  
103 #PCComponents: #MotherBoard, #VideoCard."

104 constant: BIOSsettings.  
105 isa: Individual PCComponent.  
106 comment: "The Basic Input Output System (BIOS) settings record the  
107 system parameters for all its operations. Wrong BIOS settings can be  
108 responsible for a PC malfunction, e.g., the #FloppyDiskDrive being  
109 disabled and therefore being non-existent to the system."

110 constant: MemorySystem.  
111 isa: Individual PCSubSystem.  
112 comment: "The memory #PCSubSystem. Includes the following  
113 #PCComponents: #RAM, #MotherBoard."

114 constant: RAM.  
115 isa: Individual PCComponent.  
116 comment: "The Random Access Memory #PCComponent. Usually, it is a  
117 set of Single In-Line Memory Modules (SIMMs), plugged in special  
118 positions on the #MotherBoard."

119 constant: FloppySystem.  
120 isa: Individual PCSubSystem.  
121 comment: "The #FloppySystem consists of the #FloppyDiskDrive and the  
122 #BIOSsettings for the enabling/disabling of the #FloppyDiskDrive."

123 constant: FloppyDiskDrive.  
124 isa: Individual PCComponent PossibleObservableValue.  
125 comment: "The PC storage device which drives a removable floppy disk to  
126 store/retrieve information."

127 constant: HardDiskDrive.  
128 isa: Individual PCComponent PossibleObservableValue.  
129 comment: "The PC main mass storage device. Usually it is fixed and  
130 non-removable."

131 constant: CDROMdrive.  
132 isa: Individual PCComponent.  
133 comment: "".

134 constant: PlugAndPlaySystem.  
135 isa: Individual PCSubSystem.  
136 comment: "This system comprises all peripherals that are connected to  
137 the PC via expansion cards and they are (usually) automatically  
138 recognised by MS Windows without any extra software drivers or  
139 configuration procedures. However, sometimes there may be some  
140 problems with their recognition. This #PCSubSystem may have as  
141 functional parts a wide variety of peripherals. In the current  
142 implementation of Systematic diagnosis, it is regarded as consisting  
143 of peripherals and their #ExpansionCards. There isn't any further  
144 decomposition into the specific peripherals as these vary in each  
145 configuration."

146 constant: ExpansionCard.  
147 isa: Individual PCComponent.  
148 comment: "This PC component is used in conjunction with various  
149 peripherals. In the current implementation, it is regarded as a

```

150 specific $$PCComponent, although in a specific PC configuration there
151 could be none, one or more expansion cards. Please, refer to the
152 $$PlugAndPlaySystem collection for more information.".

153 constant: PlugAndPlaySystem.
154 isa: Individual PCSubSystem.
155 comment: "It is the system responsible for loading the operating
156 system. In general, it comprises the $$FloppyDiskDrive with a floppy
157 disk containing the operating system ( $$OSfloppyDisk) and the
158 $$HardDiskDrive, although a PC can be configured via the
159 $$BIOSsettings to use only one of them or both.".

160 constant: OSfloppyDisk.
161 isa: Individual PCComponent.
162 comment: "It is the floppy disk containing the operating system. It is
163 used by the $$BootSystem to load the OS from the $$FloppyDiskDrive.".

164 constant: BootSystem.
165 isa: Individual PCSubSystem.
166 comment: "It is the system responsible for loading the operating
167 system (OS). It includes the $$FloppyDiskDrive together with the floppy
168 disk containing the OS ( $$OSfloppyDisk) and the $$HardDiskDrive. It
169 also includes the $$BootSequence-BIOSsetting which defines the sequence
170 in which these media will be used by the BIOS to load the OS.".

171 constant: functionalPartOf.
172 isa : TransitiveBinaryPredicate.
173 arg1Isa: PCSubSystem.
174 arg2Isa: PCSubSystem.
175 comment : "Predicate functionalPartOf is used to define a functional
176 model of the PC under diagnosis
177 functionalPartOf(WholeSubSystem PartialSubSystem) means
178 that the PCSubSystem WholeSubSystem is using the function of
179 PartialSubsystem to perform its own function. E.g.,
180 functionalPartOf(PowerSystem PowerSupply) means that for the
181 PowerSystem to function the PowerSupply must function. This
182 predicate is used to systematically disassemble the PC system into
183 simpler PCSubSystems until a PCComponent is reached that it is
184 faulty.".

185 constant: testFirst.
186 isa: UnaryPredicate.
187 arg1Isa: PCSubSystem.
188 comment: "This predicate is used to declare which PCSubSystem from
189 these occurring after a $$Decompose inference type will be the
190 the first to consider for diagnosis, i.e., the $$hypothesis.".

191 constant: testAfter.
192 isa: BinaryPredicate.
193 arg1Isa: PCSubSystem.
194 arg2Isa: PCSubSystem.
195 comment: "This predicate is used to declare the order of considering
196 PC subsystems for diagnosis. For example, ( $$testAfter SUBSYSTEM1
197 SUBSYSTEM2) means that that the $$PCSubSystem SUBSYSTEM2 will
198 be considered for diagnosis immediately after SUBSYSTEM1.".

199 Default Mt: PCDiagnosisMt.

200 constant: hypothesis.
201 isa: UnaryPredicate.
202 arg1Isa: PCSubSystem.
203 comment: "The predicate is used to record in the KB which
204 $$PCSubSystem is currently being diagnosed. E.g.,
205 $$hypothesis( $$VideoSystem) means that it is the $$VideoSystem that is

```

```

206 currently being checked for possible faults.".

207 constant: possibleHypotheses.
208 isa: UnaryPredicate.
209 arg1Isa: PCSubSystem.
210 comment: "The predicate is used to record in the KB which
211 ##PCSubSystems are currently candidates for being diagnosed. E.g.,
212 ##possibleHypotheses(##VideoCard) means that the ##VideoCard is
213 a candidate to be checked for possible faults.".

214 ;***** SYSTEM TESTING KNOWLEDGE *****

215 ;;; The basic tool in the Systematic Diagnosis problem-solving method
216 ;;; (PSM), as well as in any other diagnostic PSM, for carrying out
217 ;;; the diagnostic procedure, is various TESTS that must be done to
218 ;;; provide information (knowledge) about the state of the
219 ;;; system. This knowledge may concern the actual behaviour of the
220 ;;; system's components, e.g. the absence of electric power, control
221 ;;; information produced by the system, e.g., beep codes or screen
222 ;;; messages. Conceptually, a TEST is a question that the user must
223 ;;; make to the system under diagnosis to extract knowledge about
224 ;;; it. Here, it is implemented as a structure consisting of three
225 ;;; other concepts:
226 ;;; 1. The ##PCSubSystem to which it is related, i.e., to which the
227 ;;; question is addressed.
228 ;;; 2. The ##TestAction which is the action one has really to perform
229 ;;; for the test
230 ;;; 3. The ##PossibleObservable (system variable) that the TEST is
231 ;;; asking about .

232 ;;; Although not part of a TEST structure, there is a fourth concept
233 ;;; related with it, the ##PossibleObservableValue (system variable
234 ;;; value) which is the result (answer) of the TEST's question.

235 ;;; Each TEST is represented as a non-atomic term (NAT) in CycL with
236 ;;; the use of a ##NonPredicateFunction, ##TestFn, which takes as
237 ;;; arguments instances of the three constituent concepts and returns
238 ;;; a TEST structure, Schematically:

239 ;;; (##TestFN ##PCSubSystem ##TestAction ##PossibleObservable) -> ##Test

240 Default Mt: PCDiagnosisMt.

241 constant: Test.
242 isa: Collection.
243 genls: InformationBearingThing.
244 comment: "The basic tool in the Systematic Diagnosis problem-solving
245 method (PSM), as well as in any other diagnostic PSM, for carrying out
246 the diagnostic procedure, is various TESTS that must be done to
247 provide information (knowledge) about the state of the system. This
248 knowledge may concern the actual behaviour of the system's components,
249 e.g. the absence of electric power, control information produced by
250 the system, e.g., beep codes or screen messages. Conceptually, a TEST
251 is a question that the user must make to the system under diagnosis to
252 extract knowledge about it. Here, it is implemented as a structure
253 consisting of three other concepts: 1. The ##PCSubSystem to which it
254 is related, i.e., to which the question is addressed, 2. The
255 ##TestAction which is the action one has really to perform for the
256 test and 3. The ##PossibleObservable (system variable) that the TEST
257 is asking about.".

258 constant: TestFn.
259 isa: NonPredicateFunction.
260 arity: 3.

```

```

261 arg1Isa: PCSubSystem.
262 arg2Isa: TestAction.
263 arg3Isa: PossibleObservable.
264 resultIsa: Test.
265 comment: "Every #Test is a structure consisting of three
266 concepts. The #PCSubSystem to which it is related, the actual
267 #TestAction that must be performed and the #PossibleObservable
268 (system variable) that is being observed. Each TEST is represented as a
269 non-atomic term (NAT) in CycL
270 with the use of the #NonPredicateFunction, #TestFn, which takes as
271 arguments instances of the three constituent concepts and returns a
272 TEST structure, Schematically:
273 (#TestFN #PCSubSystem #TestAction #PossibleObservable) -> #Test".

274 constant: possibleTest.
275 isa: UnaryPredicate.
276 arg1Isa: Test.
277 comment: "The #Tests available to be performed in any stage of the
278 Diagnosis.".

279 constant: TestAction.
280 isa: Collection.
281 genls: PurposefulAction.
282 comment: "The collection of all possible test actions that may be
283 performed from the user on a #PCSubSystem to determine the
284 #PossibleObservableValues of
285 a #PossibleObservable. These values are compared to the expected
286 ones. If they are different, the fault lies somewhere in the
287 #PCSubSystem which is further decomposed and its functional parts are
288 checked one by one. If not, the fault lies in another #PCSubSystem.".
289
290 constant: PossibleObservable.
291 isa: Collection.
292 genls: AttributeType.
293 comment: "The collection of system variables (possible observables)
294 which are used to decide if the currently checked #PCSubSystem actually
295 contains a faulty #PCComponent or not.".

296 constant: PossibleObservableValue.
297 isa: Collection.
298 genls: Thing.
299 comment: "The collection of all possible values of all
300 #PossibleObservables. In terms of the Systematic Diagnosis
301 problem-solving method, the instances of #PossibleObservable
302 correspond to the system variables that one can test during diagnosis and the
303 instances of #PossibleObservableValue correspond to the possible
304 outcomes of these tests. These outcomes can be anything, therefore a
305 #PossibleObservableValue is a sub-collection of #Thing, the topmost
306 #Collection ".

307 constant: ResultType.
308 isa: Collection.
309 genls: AttributeValue.
310 comment: "The collection of all various types of
311 #PossibleObservableValues. These types are characterised from the
312 kind of conclusions they lead relative to the #PCSubSystem being
313 currently diagnosed ( #Hypothesis(#PCSubSystem) ). E.g., such a type
314 can be #Normal which denotes that the #PCSubSystem currently being
315 diagnosed is not faulty and therefore must be discarded as a
316 hypothesis and a new hypothesis must be selected.".

317 constant: possibleResultOfTest.
318 isa: Predicate.
319 arity: 3.
320 arg1Isa: Test.
321 arg2Isa: PossibleObservableValue.

```

```

322 arg3Isa: ResultType.
323 comment: "The predicate is correlating an individual #Test with its
324 actual result and the type of this result. The assertion
325 possibleResultOfTest(TEST VALUE TYPE)
326 express the fact that for the specific TEST, VALUE is a possible
327 result of type TYPE. E.g., possibleResultOfTest((TestFn
328 PowerSystem ConfirmSensorially ElectricPower) Yes Normal) indicates that
329 it is #Normal to observe the existence of #ElectricPower when
330 diagnosing the #PowerSystem. Of course, this immediately would imply
331 that the #PowerSystem is not faulty and therefore should be discarded
332 as a #Hypothesis.".

333 constant: resultOfTest.
334 isa: BinaryPredicate.
335 arg1Isa: Test.
336 arg2Isa: PossibleObservableValue.
337 comment: "The predicate is correlating an individual #Test with its
338 actual result. The assertion resultOfTest(?TEST ?VALUE) means that
339 ?VALUE is the actual result of ?TEST.".

340 ;***** DEFINITIONS OF INSTANCES FOR #TestAction COLLECTION *****

341 constant: ConfirmSensorially.
342 isa: TestAction.
343 comment: "The action of confirming the existence of a
344 #PossibleObservable only by one's senses, e.g., visually,
345 acoustically.".

346 constant: CheckIndependently.
347 isa: TestAction.
348 comment: "This #TestAction means that the user performing
349 diagnosis must check the function of the related #PCSubSystem
350 isolated from the rest of the #PCSubSystem of which it is a
351 #functionalPartOf. The way to do that is not specifically described
352 by the name of the action. It is assumed that the user has some
353 knowledge for performing such isolated tests. E.g., to test the
354 #PowerSocket one can plug another device - known to be working - in
355 it and confirm that the device has #ElectricPower.".

356 constant: Remove.
357 isa: TestAction.
358 comment: "This #TestAction means that the user must remove the
359 related #PCSubSystem from the #PCSubSystem of which it is a
360 #functionalPartOf".

361 constant: Replace.
362 isa: TestAction.
363 comment: "This #TestAction means that the user must replace the
364 related #PCSubSystem with a new one".

365 constant: ChangeVoltage.
366 isa: TestAction.
367 comment: "This #TestAction means that the user must change the
368 voltage setting in the #PowerSupply. It is a specific #TestAction
369 related only to the #PowerSupply".

370 constant: TroubleshootComponent.
371 isa: TestAction.
372 comment: "This #TestAction means that the diagnosis reached at the
373 level of a specific PCComponent but there is not sufficient
374 information to confirm that it is faulty. Therefore, the user must
375 enter the stage of troubleshooting it specifically.".

```

```
376 ;***** DEFINITIONS OF INSTANCES FOR $$PossibleObservable COLLECTION *****
377 constant: ProblemContext.
378 isa: PossibleObservable.
379 comment: "This $$PossibleObservable refers to the general context of
380 diagnosis, i.e., $$BootTime, $$RunTime, $$ComponentSpecific. The
381 value of this $$PossibleObservable determines which rules are
382 applicable, appearing as a condition in their antecedent part".

383 constant: ElectricPower.
384 isa: PossibleObservable.
385 comment: "The electric power that any $$PCSystem needs to operate.".

386 constant: VoltageCorrect.
387 isa: PossibleObservable.
388 comment: "This $$PossibleObservable refers to the voltage setting in
389 the $$PowerSupply being correct, i.e., 110V or 220V.".

390 constant: VideoSignal.
391 isa: PossibleObservable.
392 comment: "This $$PossibleObservable refers to the existence of any
393 video signal on the $$Monitor screen.".

394 constant: SpeakerBeep.
395 isa: PossibleObservable.
396 comment: "This $$PossibleObservable refers to any beep pattern coming
397 out of the $$Speaker.".

398 constant: VideoBIOSMessage.
399 isa: PossibleObservable.
400 comment: "This $$PossibleObservable refers to the display of the
401 video BIOS message.".

402 constant: BootContinues.
403 isa: PossibleObservable.
404 comment: "This $$PossibleObservable refers to the booting process
405 continuing normally.".

406 constant: StartupScreen.
407 isa: PossibleObservable.
408 comment: "This $$PossibleObservable refers to the display of the BIOS
409 startup screen.".

410 constant: MemoryTest.
411 isa: PossibleObservable.
412 comment: "This $$PossibleObservable refers to the memory test
413 performed by the BIOS during boot-time.".

414 constant: ErrorMessage.
415 isa: PossibleObservable.
416 comment: "This $$PossibleObservable refers to the display of an error
417 message on the screen.".

418 constant: ComponentProblem.
419 isa: PossibleObservable.
420 comment: "This $$PossibleObservable refers to the occasion where a
421 specific $$PCComponent has reached which is possibly faulty and the
422 only way to decide about this involves elaborate and complex $$Tests,
423 which the current implementation of the Systematic diagnosis problem
424 solving method does not cover. Therefore, the user has to perform
425 these $$Tests either based on his knowledge or have a human expert
426 perform them.".

427 constant: AutoDetection-BIOSsetting.
428 isa: PossibleObservable.
429 comment: "This $$PossibleObservable refers to the $$HardDiskDrive
```

```
430 auto-detection setting in the PC BIOS. It may be set to ##Auto for
431 automatic detection or to ##Manual, usually the first one.".

432 constant: BootSequence-BIOSsetting.
433 isa: PossibleObservable.
434 comment: "This ##PossibleObservable refers to the BIOS setting which
435 controls the sequence of the media used to load the operating
436 system. It may be A:-C: for using first the ##FloppyDiskDrive and then
437 the ##HardDiskDrive or C:-A: for the reverse.".

438 constant: BootSource.
439 isa: PossibleObservable.
440 comment: "This ##PossibleObservable refers to the actual medium from
441 which the operating system is loaded, independently from what the
442 ##BootSequence-BIOSsetting is.".

443 constant: FloppyAccess.
444 isa: PossibleObservable.
445 comment: "This ##PossibleObservable refers to whether the
446 ##FloppyDiskDrive is actually accessed by the BIOS during boot-time
447 system test.".

448 constant: DetectionMessage.
449 isa: PossibleObservable.
450 comment: "This ##PossibleObservable is related to BIOS messages
451 concerning the autodetection of the ##HardDiskDrives.".

452 constant: InFloppy.
453 isa: PossibleObservable.
454 comment: "This ##PossibleObservable is related to the ##OSfloppyDisk
455 being inside the ##FloppyDiskDrive.".

456 ;***** DEFINITIONS OF INSTANCES FOR $PossibleObservableValue COLLECTION ****

457 constant: BootTime.
458 isa: PossibleObservableValue.
459 comment: "This ##PossibleObservableValue is related to the
460 ##ProblemContext ##PossibleObservable. It means that the fault being
461 diagnosed occured during boot-time, i.e., from the time the power is
462 turned on until the Operating System starts being loaded.".

463 constant: RunTime.
464 isa: PossibleObservableValue.
465 comment: "This ##PossibleObservableValue is related to the
466 ##ProblemContext ##PossibleObservable. It means that the fault being
467 diagnosed occured during run-time, i.e., from the time the Operating
468 System starts being loaded until the PC is switched off.".

469 constant: ComponentSpecific.
470 isa: PossibleObservableValue.
471 comment: "This ##PossibleObservableValue is related to the
472 ##ProblemContext ##PossibleObservable. It means that the fault being
473 diagnosed is identified to be related with a specific ##PCComponent,
474 e.g., the ##Monitor, ##MotherBoard, ##HardDisk e.t.c.".

475 constant: Yes.
476 isa: PossibleObservableValue.
477 comment: "Most of the ##Tests have as a possible result only 'Yes' or 'No'.".

478 constant: No.
479 isa: PossibleObservableValue.
480 comment: "Most of the ##Tests have as a possible result only 'Yes' or 'No'.".

481 constant: None.
482 isa: PossibleObservableValue.
483 comment: "This kind of result indicates that none of the alternative
```

```
484 results of a specific #Test is observed.".
485 constant: SingleBeep.
486 isa: PossibleObservableValue.
487 comment: "This #PossibleObservableValue is related to the
488 #SpeakerBeep #PossibleObservable. It means that the #Speaker
489 produced a single beep".

490 constant: RingingOrBuzzing.
491 isa: PossibleObservableValue.
492 comment: "This #PossibleObservableValue is related to the
493 #SpeakerBeep #PossibleObservable. It means that the #Speaker is
494 producing a ringing or buzzing sound.".

495 constant: ConsistentPattern.
496 isa: PossibleObservableValue.
497 comment: "This #PossibleObservableValue is related to the
498 #SpeakerBeep #PossibleObservable. It means that the #Speaker is
499 producing a consistent pattern (code) of beeps, e.g., one beep, then
500 two more.".

501 constant: Complete.
502 isa: PossibleObservableValue.
503 comment: "This #PossibleObservableValue is related to some tests
504 performed by the BIOS, e.g., the memory test. It means that the
505 corresponding test is successfully completed.".

506 constant: InComplete.
507 isa: PossibleObservableValue.
508 comment: "This #PossibleObservableValue is related to some tests
509 performed by the BIOS, e.g., the memory test. It means that the
510 corresponding test is not successfully completed.".

511 constant: CannotFind-Message.
512 isa: PossibleObservableValue.
513 comment: "This #PossibleObservableValue is related to BIOS error
514 messages concerning the autodetection of IDE/ATAPI devices,
515 e.g. #HardDiskDrive, #CDROMdrive. This kind of messages indicate
516 that the BIOS cannot detect the corresponding device.".

517 ;"This #PossibleObservableValue is related to BIOS error messages
518 ;concerning the autodetection of the #HardDiskDrives. This kind of
519 ;messages indicate that the BIOS cannot detect any #HardDiskDrive.".

520 constant: Auto.
521 isa: PossibleObservableValue.
522 comment: "This #PossibleObservableValue indicates that some
523 action/process/procedure is (set to be) done automatically.".

524 constant: Manual.
525 isa: PossibleObservableValue.
526 comment: "This #PossibleObservableValue indicates that some
527 action/process/procedure is (set to be) done manually.".

528 constant: FloppyThenHard.
529 isa: PossibleObservableValue.
530 comment: "This #PossibleObservableValue is related to the
531 #BootSequence-BIOSsetting #PossibleObservable. It indicates that
532 this setting is set to A:-C:".

533 constant: HardThenFloppy.
534 isa: PossibleObservableValue.
535 comment: "This #PossibleObservableValue is related to the
536 #BootSequence-BIOSsetting #PossibleObservable. It indicates that
537 this setting is set to C:-A:.".
```



```

538 ;***** DEFINITIONS OF INSTANCES FOR $ResultType COLLECTION ****

539 constant: Normal.
540 isa: ResultType.
541 comment: "This type of result indicates that the result is normally
542 expected when the #PCSubSystem related with it is working
543 properly. Such a kind of result implies that the #PCSubSystem must be
544 discarded as a #Hypothesis.".

545 constant: NotNormal.
546 isa: ResultType.
547 comment: "This type of result indicates that the result is not normally
548 expected when the #PCSubSystem related with it is working
549 properly. Such a kind of result implies that the fault lies in the
550 #PCSubSystem which is the current #Hypothesis.".

551 constant: Insufficient.
552 isa: ResultType.
553 comment: "This type of result indicates that the result cannot
554 undoubtedly indicate either the normal function or the malfunction of
555 the #PCComponent related with it. Such a kind of result implies that
556 further testing is necessary to decide about the functional status of
557 the #PCComponent which is the current #Hypothesis.".

558 constant: Distinguishing.
559 isa: ResultType.
560 comment: "This type of result occurs in a situation where there are
561 two components that are probably faulty and the only way to find
562 which, is to test one of them. In this case, a result of type
563 #Distinguishing indicates simultaneously two things. First, that the
564 #PCSubSystem hypothesised as faulty is not such and second, that the faulty
565 one is the other alternative #PCSubSystem.".

566 ;; ***** IMPLEMENTATION OF KADS SYSTEMATIC DIAGNOSIS PSM *****

567 ;; The Task Structure for Systematic Diagnosis (pseudo-code) is:

568 ;; Systematic Diagnosis(+complaint,+possible observables,-hypothesis) by
569 ;; select1(+complaint, -system model)
570 ;; REPEAT
571 ;; decompose(+system model, -hypothesis)
572 ;; WHILE number of hypotheses > 1
573 ;; select2(+possible observables, -variable value)
574 ;; select3(+hypothesis, -norm)
575 ;; compare(+variable value, +norm, -difference)
576 ;; system model <- current decomposition level of system model
577 ;; UNTIL confirm(+hypothesis), i.e. system model cannot be decomposed further

578 ;; The user interaction in CYC will be done from the SubL Interactor
579 ;; interface, as it is not possible to get any input/output
580 ;; interaction between the user and the SubL code from the ASK
581 ;; interface. The whole Task Structure will be implemented as a SubL
582 ;; function, 'systematic', which will be responsible for calling the
583 ;; appropriate SubL functions that will implement the corresponding
584 ;; inferences. In fact, the 'select1', 'select2', and 'select3'
585 ;; inferences will be implemented as FORWARD rules in the
586 ;; CYC KB. "Forward" means that, according to the results of 'Tests'
587 ;; that the user is asked to give, these rules automatically assert
588 ;; new facts in the KB. These facts describe which are the next
589 ;; #PossibleObservables and Variables that must be tested ('select2'

```

```

590   ;;; inference), what should be done according to the result ('select3'
591   ;;; and 'compare' inferences), e.g., if another test for the same
592   ;;; hypothesis should be performed or if the current hypothesis should
593   ;;; be rejected or the current hypothesis must be decomposed further
594   ;;; or if the faulty component was found ('confirm' inference).

```

```

595   ;;; The following three (3) constant definitions introduce the
596   ;;; #PlausibleInference predicate and #Decompose inference type of
597   ;;; KADS. These two are used in the antecedent part of the
598   ;;; "decomposition" rules. They do not constitute control knowledge
599   ;;; but domain role knowledge. In terms of implementation, they cause
600   ;;; the forward "decomposition" rules to fire only when a Decompose
601   ;;; inference has to be made.

```

```

602   Default Mt: PCDiagnosisMt.

```

```

603   constant: InferenceType-KADS.
604   isa: Collection.
605   genls: PropositionalInformationThing.
606   comment: "The collection of all Inference Types of KADS methodology,
607   e.g., 'select', 'decompose', 'confirm'.".

```

```

608   constant: Decompose.
609   isa: InferenceType-KADS.
610   comment: "The #Decompose inference type of KADS takes a structured
611   hierarchy of objects and gives a less or completely unstructured
612   collection of these objects. In its simplest form it is used for
613   breaking down existing knowledge structures, like hierarchies, where
614   there is no loss of objects but only the structure is removed.".

```

```

615   constant: plausibleInference.
616   isa: UnaryPredicate.
617   arg1Isa: InferenceType-KADS.
618   comment: "The #PlausibleInference predicate is used to record in the
619   KB which inference(s) can be next performed during the 'execution' of
620   a problem solving method.".

```

```

621   ;;; During the Systematic Diagnosis problem solving method (PSM) as
622   ;;; well as during any other PSM, there are certain decisions/choices
623   ;;; that must be done. According to the structure of the PSMs as
624   ;;; Generic Task Models in KADS, these decisions/choices occur during
625   ;;; the performance of specific Inferences. The implementation of
626   ;;; these decisions/choices has to be declarative since this is the
627   ;;; main principle in CYC. Therefore, the implementation of them will
628   ;;; be in terms of FORWARD rules: the ANTECEDENT of each rule will be
629   ;;; the conditions under which a decision is made and the CONSEQUENT
630   ;;; will be the knowledge that is becoming known to the system when
631   ;;; this decision is made. Then, the newly added information will be
632   ;;; used by the SubL code to guide the whole procedure. In the
633   ;;; following, we will examine in detail which these decisions/choices
634   ;;; are, when and where do they occur and how they are actually
635   ;;; implemented as forward rules.

```

```

636   ;;; The PSM starts with a SELECT inference. A general symptom is
637   ;;; entered by the user and an appropriate system model is
638   ;;; chosen. This inference is slightly changed for PC diagnosis. What
639   ;;; is actually asked from the user is to distinguish three (3) major
640   ;;; contexts of diagnosis: (i) Boot-time troubleshooting, (ii)
641   ;;; Run-time troubleshooting and (iii) Component-specific
642   ;;; troubleshooting. This categorisation is significant since
643   ;;; completely different rules are applicable in each
644   ;;; context. Although this contextual dependency of the rules could be
645   ;;; implemented as different #Microtheory contexts, this would make
646   ;;; context shifting more complicated - any ASK operation would have

```

```

647  ;;; to define the #Microtheory. Instead, this dependency will be
648  ;;; embedded in the antecedent part of the rules as an extra
649  ;;; condition. The special predicate diagnosisContext will be used, e.g.,
650  ;;;
651  ;;; (implies
652  ;;;   (and
653  ;;;     (diagnosisContext BootTime)
654  ;;;     (...<more conditions>...)) ;end of antecedent
655  ;;;   (<consequent>))

656  constant: diagnosisContext.
657  isa: UnaryPredicate.
658  arg1Isa: PossibleObservableValue.
659  comment: "This predicate records the current problem-solving context,
660   i.e. #BootTime, #RunTime or #ComponentSpecific.".

661  ;;; Rule to assert a (#diagnosisContext ...) assertion. This assertion
662  ;;; is introduced as a "shorthand" for the assertion:
663  ;;;
664  ;;; (resultOfTest (TestFn PCSystem ConfirmSensorially ProblemContext) ?PROBLEM)
665  ;;;
666  ;;; It is used as a premise in every rule which is applicable to the
667  ;;; corresponding diagnosis context, i.e., #BootTime, #RunTime or
668  ;;; #ComponentSecific.

669  Default Mt: PCDiagnosisMt.

670  Direction: forward.
671   F: (implies
672     (resultOfTest (TestFn PCSystem ConfirmSensorially ProblemContext) ?PROBLEM)
673     (diagnosisContext ?PROBLEM)).

674  ;;; The 'decompose' inference in KADS Systematic Diagnosis PSM takes as input
675  ;;; the current #PCSubSystem (#hypothesis PC_SUBSYSTEM) and decomposes it
676  ;;; into its functional subsystems (#functionalPartOf PC_SUBSYSTEM PART),
677  ;;; generating new hypotheses (#possibleHypotheses PART).

678  ;;; *****
679  ;;; ** Rules and Facts for Decompose **
680  ;;; *****

681  ;;; Every PART which is a functional part of the hypothesis, HYP, is a
682  ;;; possible hypothesis.

683  direction: forward.
684   F: (implies
685     (and
686       (diagnosisContext BootTime)
687       (plausibleInference Decompose)
688       (hypothesis ?HYP)
689       (functionalPartOf ?HYP ?PART))
690     (possibleHypotheses ?PART)).

691   F: (functionalPartOf PCSystem PowerSystem).
692   F: (functionalPartOf PCSystem VideoSystem).
693   F: (functionalPartOf PCSystem BIOSStartupSystem).
694   F: (functionalPartOf PCSystem MemorySystem).
695   F: (functionalPartOf PCSystem FloppySystem).
696   F: (functionalPartOf PCSystem HardDiskDrive).
697   F: (functionalPartOf PCSystem CDRomdrive).
698   F: (functionalPartOf PCSystem PlugAndPlaySystem).
699   F: (functionalPartOf PCSystem BootSystem).

```

```

700 F: (functionalPartOf PowerSystem PowerSocket).
701 F: (functionalPartOf PowerSystem PowerCable).
702 F: (functionalPartOf PowerSystem PowerProtectionDevice).
703 F: (functionalPartOf PowerSystem PowerSupply).

704 F: (functionalPartOf VideoSystem MotherBoard).
705 F: (functionalPartOf VideoSystem VideoCard).
706 F: (functionalPartOf VideoSystem Monitor).

707 F: (functionalPartOf BIOSStartupSystem MotherBoard).
708 F: (functionalPartOf BIOSStartupSystem VideoCard).

709 F: (functionalPartOf MemorySystem RAM).
710 F: (functionalPartOf MemorySystem MotherBoard).

711 F: (functionalPartOf FloppySystem FloppyDiskDrive).
712 F: (functionalPartOf FloppySystem BIOSsettings).

713 F: (functionalPartOf PlugAndPlaySystem ExpansionCard).
714 F: (functionalPartOf PlugAndPlaySystem MotherBoard).

715 F: (functionalPartOf BootSystem FloppyDiskDrive).
716 F: (functionalPartOf BootSystem OSfloppyDisk).
717 F: (functionalPartOf BootSystem HardDiskDrive).

718 ;; *****
719 ;; ** **
720 ;; ** Facts and rules about PC subsystems and components **
721 ;; ** **
722 ;; *****

723 ;; *****
724 ;; ** TEST(S) for the PCSystem **
725 ;; *****

726 Default Mt: PCdiagnosisMt.

727 Direction: forward.
728 F: (implies
729   (hypothesis PCSystem) ;if diagnosis just started, ask for the context
730   (and
731     (possibleTest (TestFn PCSystem ConfirmSensorially ProblemContext))
732     (possibleResultOfTest
733       (TestFn PCSystem ConfirmSensorially ProblemContext) BootTime NotNormal)
734     (possibleResultOfTest
735       (TestFn PCSystem ConfirmSensorially ProblemContext) RunTime NotNormal)
736     (possibleResultOfTest
737       (TestFn PCSystem ConfirmSensorially ProblemContext)
738         ComponentSpecific NotNormal))).

739 ;; *****
740 ;; ** DECOMPOSITION knowledge for the PCSystem **
741 ;; *****

742 Direction: forward.
743 F: (implies
744   (and
745     (diagnosisContext BootTime)
746     (hypothesis PCSystem)
747     (plausibleInference Decompose))
748   (and
749     (testFirst PowerSystem)

```

```

750     (testAfter PowerSystem VideoSystem)
751     (testAfter VideoSystem BIOSStartupSystem)
752     (testAfter BIOSStartupSystem MemorySystem)
753     (testAfter MemorySystem FloppySystem)
754     (testAfter FloppySystem HardDiskDrive)
755     (testAfter HardDiskDrive CDRomdrive)
756     (testAfter CDRomdrive PlugAndPlaySystem)
757     (testAfter PlugAndPlaySystem BootSystem))).

758 ;; *****
759 ;; ** TEST(S) for the PowerSystem **
760 ;; *****

761 Direction: forward.
762 F: (implies
763     (and
764         (diagnosisContext BootTime)
765         (hypothesis PowerSystem)) ;
766     (and
767         (possibleTest (TestFn PowerSystem ConfirmSensorially ElectricPower))
768         (possibleResultOfTest
769             (TestFn PowerSystem ConfirmSensorially ElectricPower) Yes Normal)
770         (possibleResultOfTest
771             (TestFn PowerSystem ConfirmSensorially ElectricPower) No NotNormal))).

772 ;; *****
773 ;; ** DECOMPOSITION knowledge for the PowerSystem **
774 ;; ** ElectricPower=No **
775 ;; *****

776 Direction: forward.
777 F: (implies
778     (and
779         (diagnosisContext BootTime)
780         (hypothesis PowerSystem)
781         (resultOfTest (TestFn PowerSystem ConfirmSensorially ElectricPower) No)
782         (plausibleInference Decompose))
783     (and
784         (testFirst PowerSocket)
785         (testAfter PowerSocket PowerProtectionDevice)
786         (testAfter PowerProtectionDevice PowerCable)
787         (testAfter PowerCable PowerSupply))).

788 ;; *****
789 ;; ** TEST(S) for the PowerSocket **
790 ;; *****

791 Direction: forward.
792 F: (implies
793     (and
794         (diagnosisContext BootTime)
795         (hypothesis PowerSocket)) ;
796     (and
797         (possibleTest (TestFn PowerSocket CheckIndependently ElectricPower))
798         (possibleResultOfTest
799             (TestFn PowerSocket CheckIndependently ElectricPower) Yes Normal)
800         (possibleResultOfTest
801             (TestFn PowerSocket CheckIndependently ElectricPower) No NotNormal))).

802 ;; *****
803 ;; ** TEST(S) for the PowerProtectionDevice **
804 ;; *****

805 Direction: forward.

```

```

806 F: (implies
807   (and
808     (diagnosisContext BootTime)
809     (hypothesis PowerProtectionDevice))
810   (and
811     (possibleTest (TestFn PowerProtectionDevice Remove ElectricPower))
812     (possibleResultOfTest
813       (TestFn PowerProtectionDevice Remove ElectricPower) Yes NotNormal)
814     (possibleResultOfTest
815       (TestFn PowerProtectionDevice Remove ElectricPower) No Normal))).

816 ;*****
817 ;** TEST(S) for the PowerCable **
818 ;*****

819 Direction: forward.
820 F: (implies
821   (and
822     (diagnosisContext BootTime)
823     (hypothesis PowerCable)) ;
824   (and
825     (possibleTest (TestFn PowerCable Replace ElectricPower))
826     (possibleResultOfTest
827       (TestFn PowerCable Replace ElectricPower) Yes NotNormal)
828     (possibleResultOfTest
829       (TestFn PowerCable Replace ElectricPower) No Normal))).

830 ;*****
831 ;** TEST(S) for the PowerSupply **
832 ;*****

833 Direction: forward.
834 F: (implies
835   (and
836     (diagnosisContext BootTime)
837     (hypothesis PowerSupply))
838   (and
839     (possibleTest (TestFn PowerSupply ConfirmSensorially VoltageCorrect))
840     (possibleResultOfTest
841       (TestFn PowerSupply ConfirmSensorially VoltageCorrect) Yes NotNormal)
842     (possibleResultOfTest
843       (TestFn PowerSupply ConfirmSensorially VoltageCorrect) No Insufficient))).

844 ;*****
845 ;** TEST(S) for the PowerSupply **
846 ;** VoltageCorrect=No **
847 ;*****
848
849 Direction: forward.
850 F: (implies
851   (and
852     (diagnosisContext BootTime)
853     (hypothesis PowerSupply)
854     (resultOfTest (TestFn PowerSupply ConfirmSensorially VoltageCorrect) No)) ;
855   (and
856     (possibleTest (TestFn PowerSupply ChangeVoltage ElectricPower))
857     (possibleResultOfTest
858       (TestFn PowerSupply ChangeVoltage ElectricPower) Yes Normal)
859     (possibleResultOfTest
860       (TestFn PowerSupply ChangeVoltage ElectricPower) No NotNormal))).

861 ;*****
862 ;** TEST(S) for the VideoSystem **
863 ;*****

```

```

864 Direction: forward.
865 F: (implies
866   (and
867     (diagnosisContext BootTime)
868     (hypothesis VideoSystem))
869   (and
870     (possibleTest (TestFn VideoSystem ConfirmSensorially VideoSignal))
871     (possibleResultOfTest
872       (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes Insufficient)
873     (possibleResultOfTest
874       (TestFn VideoSystem ConfirmSensorially VideoSignal) No NotNormal))).

875 ;; *****
876 ;; ** DECOMPOSITION knowledge for the VideoSystem **
877 ;; ** VideoSignal=No **
878 ;; *****

879 Direction: forward.
880 F: (implies
881   (and
882     (diagnosisContext BootTime)
883     (hypothesis VideoSystem)
884     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No)
885     (plausibleInference Decompose))
886   (and
887     (testFirst Monitor)
888     (testAfter Monitor MotherBoard)
889     (testAfter MotherBoard Speaker))).

890 ;;*****
891 ;;** TEST(S) for the VideoSystem **
892 ;;** VideoSignal=Yes **
893 ;;*****

894 Direction: forward.
895 F: (implies
896   (and
897     (diagnosisContext BootTime)
898     (hypothesis VideoSystem)
899     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes))
900   (and
901     (possibleTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage))
902     (possibleResultOfTest
903       (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes Insufficient)
904     (possibleResultOfTest
905       (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No NotNormal))).

906 ;; *****
907 ;; ** TEST(S) for the VideoSystem **
908 ;; ** VideoSignal=Yes, VideoBIOSMessage=Yes **
909 ;; *****

910 Direction: forward.
911 F: (implies
912   (and
913     (diagnosisContext BootTime)
914     (hypothesis VideoSystem)
915     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
916     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes))
917   (and
918     (possibleTest (TestFn VideoSystem ConfirmSensorially BootContinues))
919     (possibleResultOfTest
920       (TestFn VideoSystem ConfirmSensorially BootContinues) Yes Normal)
921     (possibleResultOfTest
922       (TestFn VideoSystem ConfirmSensorially BootContinues) No NotNormal))).

```

```

923 ;; *****
924 ;; ** DECOMPOSITION knowledge for the VideoSystem **
925 ;; ** VideoSignal=Yes, VideoBIOSMessage=Yes, BootContinues=No **
926 ;; *****

927 Direction: forward.
928 F: (implies
929   (and
930     (diagnosisContext BootTime)
931     (hypothesis VideoSystem)
932     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
933     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes)
934     (resultOfTest (TestFn VideoSystem ConfirmSensorially BootContinues) No)
935     (plausibleInference Decompose))
936   (and
937     (testFirst VideoCard)
938     (testAfter VideoCard MotherBoard))).

939 ;; *****
940 ;; ** DECOMPOSITION knowledge for the VideoSystem **
941 ;; ** VideoSignal=Yes, VideoBIOSMessage=No **
942 ;; *****

943 Direction: forward.
944 F: (implies
945   (and
946     (diagnosisContext BootTime)
947     (hypothesis VideoSystem)
948     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
949     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No)
950     (plausibleInference Decompose))
951   (and
952     (testFirst MotherBoard)
953     (testAfter MotherBoard VideoCard))).

954 ;; *****
955 ;; ** TEST(S) for the Monitor **
956 ;; ** VideoSignal=No **
957 ;; *****

958 Direction: forward.
959 F: (implies
960   (and
961     (diagnosisContext BootTime)
962     (hypothesis Monitor)
963     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No))
964   (and
965     (possibleTest (TestFn Monitor CheckIndependently VideoSignal))
966     (possibleResultOfTest
967       (TestFn Monitor CheckIndependently VideoSignal) Yes Normal)
968     (possibleResultOfTest
969       (TestFn Monitor CheckIndependently VideoSignal) No NotNormal))).

970 ;; *****
971 ;; ** TEST(S) for the MotherBoard **
972 ;; ** VideoSignal=No, Monitor_Working =Yes **
973 ;; *****

974 Direction: forward.
975 F: (implies
976   (and
977     (diagnosisContext BootTime)
978     (hypothesis MotherBoard)

```



```

979     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No)
980     (resultOfTest (TestFn Monitor CheckIndependently VideoSignal) Yes))
981     (and
982     (possibleTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep))
983     (possibleResultOfTest
984     (TestFn MotherBoard ConfirmSensorially SpeakerBeep) SingleBeep NotNormal)
985     (possibleResultOfTest
986     (TestFn MotherBoard ConfirmSensorially SpeakerBeep)
987     ConsistentPattern NotNormal)
988     (possibleResultOfTest
989     (TestFn MotherBoard ConfirmSensorially SpeakerBeep)
990     RingingOrBuzzing NotNormal)
991     (possibleResultOfTest
992     (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No Insufficient))).

993 ;; *****
994 ;; ** TEST(S) for the MotherBoard **
995 ;; ** VideoSignal=No, Monitor_Working =Yes, SpeakerBeep=No **
996 ;; *****

997 Direction: forward.
998 F: (implies
999     (and
1000     (diagnosisContext BootTime)
1001     (hypothesis MotherBoard)
1002     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No)
1003     (resultOfTest (TestFn Monitor CheckIndependently VideoSignal) Yes)
1004     (resultOfTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No))
1005     (and
1006     (possibleTest (TestFn Speaker CheckIndependently SpeakerBeep))
1007     (possibleResultOfTest
1008     (TestFn Speaker CheckIndependently SpeakerBeep) Yes NotNormal)
1009     (possibleResultOfTest
1010     (TestFn Speaker CheckIndependently SpeakerBeep) No Distinguishing))).

1011 ;;; Important notice: Although the PCSubSystem related to the Test is
1012 ;;; the Speaker, however, the PCSubSystem hypothesised as faulty is
1013 ;;; the MotherBoard. Therefore, the Result Type is defined by what it
1014 ;;; indicates about the MotherBoard and not the Speaker itself. This
1015 ;;; rather peculiar situation is due to the controlling function of
1016 ;;; the Speaker. This means that the Speaker functions as a control
1017 ;;; device for the MotherBoard and we must make sure that this device
1018 ;;; is working properly. If it does, then the lack of any sound is due
1019 ;;; to the MotherBoard and we can deduce that it is faulty. Otherwise,
1020 ;;; we cannot deduce anything before we are certain that the Speaker
1021 ;;; is working properly.

1022 ;; *****
1023 ;; ** TEST(S) for the MotherBoard **
1024 ;; ** VideoSignal=Yes, VideoBIOSMessage=No **
1025 ;; *****

1026 Direction: forward.
1027 F: (implies
1028     (and
1029     (diagnosisContext BootTime)
1030     (hypothesis MotherBoard)
1031     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1032     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No))
1033     (and
1034     (possibleTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep))
1035     (possibleResultOfTest
1036     (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No Insufficient)
1037     (possibleResultOfTest
1038     (TestFn MotherBoard ConfirmSensorially SpeakerBeep) ConsistentPattern

```

```

1039                                     Insufficient))).

1040 ;; *****
1041 ;; ** TEST(S) for the MotherBoard **
1042 ;; ** VideoSignal=Yes, VideoBIOSMessage=No, SpeakerBeep=ConsistentPattern **
1043 ;; *****

1044 Direction: forward.
1045 F: (implies
1046   (and
1047     (diagnosisContext BootTime)
1048     (hypothesis MotherBoard)
1049     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1050     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No)
1051     (resultOfTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep)
1052                   ConsistentPattern))
1053   (and
1054     (possibleTest (TestFn MotherBoard TroubleshootComponent ComponentProblem))
1055     (possibleResultOfTest
1056       (TestFn MotherBoard TroubleshootComponent ComponentProblem) Yes
1057       NotNormal)
1058     (possibleResultOfTest
1059       (TestFn MotherBoard TroubleshootComponent ComponentProblem) No
1060       Distinguishing))).

1061 ;; *****
1062 ;; ** TEST(S) for the MotherBoard **
1063 ;; ** VideoSignal=Yes, VideoBIOSMessage=No, SpeakerBeep=No **
1064 ;; *****

1065 Direction: forward.
1066 F: (implies
1067   (and
1068     (diagnosisContext BootTime)
1069     (hypothesis MotherBoard)
1070     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1071     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No)
1072     (resultOfTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No))
1073   (and
1074     (possibleTest (TestFn MotherBoard TroubleshootComponent ComponentProblem))
1075     (possibleResultOfTest
1076       (TestFn MotherBoard TroubleshootComponent ComponentProblem) Yes
1077       NotNormal)
1078     (possibleResultOfTest
1079       (TestFn MotherBoard TroubleshootComponent ComponentProblem) No
1080       Distinguishing))).

1081 ;; *****
1082 ;; ** TEST(S) for the MotherBoard **
1083 ;; ** StartupScreen=No **
1084 ;; *****

1085 Direction: forward.
1086 F: (implies
1087   (and
1088     (diagnosisContext BootTime)
1089     (hypothesis MotherBoard)
1090     (resultOfTest
1091       (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No))
1092   (and
1093     (possibleTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep))
1094     (possibleResultOfTest
1095       (TestFn MotherBoard ConfirmSensorially SpeakerBeep) Yes NotNormal)
1096     (possibleResultOfTest
1097       (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No Insufficient))).

```

```

1098 ;; *****
1099 ;; ** TEST(S) for the MotherBoard **
1100 ;; ** StartupScreen=No, SpeakerBeep=No **
1101 ;; *****

1102 Direction: forward.
1103 F: (implies
1104   (and
1105     (diagnosisContext BootTime)
1106     (hypothesis MotherBoard)
1107     (resultOfTest
1108       (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No)
1109     (resultOfTest
1110       (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No))
1111   (and
1112     (possibleTest (TestFn MotherBoard ConfirmSensorially ErrorMessage))
1113     (possibleResultOfTest
1114       (TestFn MotherBoard ConfirmSensorially ErrorMessage) Yes NotNormal)
1115     (possibleResultOfTest
1116       (TestFn MotherBoard ConfirmSensorially ErrorMessage) No Insufficient))).

1117 ;; *****
1118 ;; ** TEST(S) for the MotherBoard **
1119 ;; ** StartupScreen=No, SpeakerBeep=No, ErrorMessage=No **
1120 ;; *****

1121 Direction: forward.
1122 F: (implies
1123   (and
1124     (diagnosisContext BootTime)
1125     (hypothesis MotherBoard)
1126     (resultOfTest
1127       (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No)
1128     (resultOfTest
1129       (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No)
1130     (resultOfTest
1131       (TestFn MotherBoard ConfirmSensorially ErrorMessage) No))
1132   (and
1133     (possibleTest (TestFn MotherBoard TroubleshootComponent ComponentProblem))
1134     (possibleResultOfTest
1135       (TestFn MotherBoard TroubleshootComponent ComponentProblem) Yes
1136                                               NotNormal)
1137     (possibleResultOfTest
1138       (TestFn MotherBoard TroubleshootComponent ComponentProblem) No
1139                                               Distinguishing))).
1140

1141 ;; *****
1142 ;; ** TEST(S) for the VideoCard **
1143 ;; ** VideoSignal=Yes, VideoBIOSMessage=Yes, BootContinues=No **
1144 ;; *****

1145 Direction: forward.
1146 F: (implies
1147   (and
1148     (diagnosisContext BootTime)
1149     (hypothesis VideoCard)
1150     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1151     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes)
1152     (resultOfTest (TestFn VideoSystem ConfirmSensorially BootContinues) No))
1153   (and
1154     (possibleTest (TestFn VideoCard TroubleshootComponent ComponentProblem))
1155     (possibleResultOfTest
1156       (TestFn VideoCard TroubleshootComponent ComponentProblem) Yes NotNormal)
1157     (possibleResultOfTest

```

```

1158         (TestFn VideoCard TroubleshootComponent ComponentProblem) No
1159                                     Distinguishing))).

1160 ;; Important notice: The '#$TroubleshootComponent' TestAction and the
1161 ;; '#$ComponentProblem' PossibleObservable are too general and
1162 ;; complex to be of actual use. They are used here as artificial
1163 ;; "terminating points" of the Systematic Diagnosis procedure at the
1164 ;; level of #$PCComponent. In a
1165 ;; complete PC faults diagnosis expert system, more elaborate
1166 ;; 'Test(s)' should follow to troubleshoot a specific #$PCComponent
1167 ;; (here the #$VideoCard), instead of the user having to know how to
1168 ;; test the specific #$PCComponent. However, these 'Tests' are so
1169 ;; elaborate and complicated that are beyond the scope of this
1170 ;; implementation. Such 'Test(s)' could be identifying a beep code
1171 ;; according to the specific version of BIOS (American Megatrends
1172 ;; Inc., Phoenix or Other) or interpreting an error message (there
1173 ;; are 120 error messages documented in the PCGuide Troubleshoot
1174 ;; expert that was used as a knowledge acquisition source).

1175 ;; *****
1176 ;; ** TEST(S) for the BIOSStartupSystem **
1177 ;; ** PowerSystem=ok, VideoSystem=ok **
1178 ;; *****

1179 Direction: forward.
1180 F: (implies
1181     (and
1182       (diagnosisContext BootTime)
1183       (hypothesis BIOSStartupSystem))
1184     (and
1185       (possibleTest (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen))
1186       (possibleResultOfTest
1187         (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) Yes Normal)
1188       (possibleResultOfTest
1189         (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No NotNormal))).

1190 ;; *****
1191 ;; ** DECOMPOSITION knowledge for the BIOSStartupSystem **
1192 ;; ** StartupScreen=No **
1193 ;; *****

1194 Direction: forward.
1195 F: (implies
1196     (and
1197       (diagnosisContext BootTime)
1198       (hypothesis BIOSStartupSystem)
1199       (resultOfTest
1200         (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No)
1201       (plausibleInference Decompose))
1202     (and
1203       (testFirst MotherBoard)
1204       (testAfter MotherBoard VideoCard))).

1205 ;; *****
1206 ;; ** TEST(S) for the MemorySystem **
1207 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1208 ;; *****

1209 Direction: forward.
1210 F: (implies
1211     (and
1212       (diagnosisContext BootTime)
1213       (hypothesis MemorySystem))
1214     (and
1215       (possibleTest (TestFn MemorySystem ConfirmSensorially MemoryTest))
1216       (possibleResultOfTest

```

```

1217         (TestFn MemorySystem ConfirmSensorially MemoryTest) Complete Normal)
1218     (possibleResultOfTest
1219     (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete NotNormal))).
1220
1221     ;; *****
1222     ;; ** DECOMPOSITION knowledge for the MemorySystem **
1223     ;; ** MemoryTest=Incomplete **
1224     ;; *****

1225 Direction: forward.
1226 F: (implies
1227     (and
1228         (diagnosisContext BootTime)
1229         (hypothesis MemorySystem)
1230         (resultOfTest
1231         (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete)
1232         (plausibleInference Decompose))
1233     (and
1234         (testFirst RAM)
1235         (testAfter RAM MotherBoard))).

1236     ;; *****
1237     ;; ** TEST(S) for the RAM **
1238     ;; ** MemoryTest=Incomplete **
1239     ;; *****

1240 Direction: forward.
1241 F: (implies
1242     (and
1243         (diagnosisContext BootTime)
1244         (hypothesis RAM)
1245         (resultOfTest
1246         (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete))
1247     (and
1248         (possibleTest (TestFn RAM ConfirmSensorially ErrorMessage))
1249         (possibleResultOfTest
1250         (TestFn RAM ConfirmSensorially ErrorMessage) Yes NotNormal)
1251         (possibleResultOfTest
1252         (TestFn RAM ConfirmSensorially ErrorMessage) No Insufficient))).

1253     ;; *****
1254     ;; ** TEST(S) for the RAM **
1255     ;; ** MemoryTest=Incomplete, ErrorMessage=No **
1256     ;; *****

1257 Direction: forward.
1258 F: (implies
1259     (and
1260         (diagnosisContext BootTime)
1261         (hypothesis RAM)
1262         (resultOfTest
1263         (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete)
1264         (resultOfTest
1265         (TestFn RAM ConfirmSensorially ErrorMessage) No))
1266     (and
1267         (possibleTest (TestFn RAM TroubleshootComponent ComponentProblem))
1268         (possibleResultOfTest
1269         (TestFn RAM TroubleshootComponent ComponentProblem) Yes NotNormal)
1270         (possibleResultOfTest
1271         (TestFn RAM TroubleshootComponent ComponentProblem) No
1272         Distinguishing))).

1273     ;; *****
1274     ;; ** TEST(S) for the FloppySystem **
1275     ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1276     ;; ** MemorySystem=ok **

```

```

1277 ;; *****
1278 Direction: forward.
1279 F: (implies
1280   (and
1281     (diagnosisContext BootTime)
1282     (hypothesis FloppySystem))
1283   (and
1284     (possibleTest (TestFn FloppySystem ConfirmSensorially FloppyAccess))
1285     (possibleResultOfTest
1286       (TestFn FloppySystem ConfirmSensorially FloppyAccess) Yes Normal)
1287     (possibleResultOfTest
1288       (TestFn FloppySystem ConfirmSensorially FloppyAccess) No NotNormal))).

1289 ;; *****
1290 ;; ** DECOMPOSITION knowledge for the FloppySystem **
1291 ;; ** FloppyAccess=No **
1292 ;; *****

1293 Direction: forward.
1294 F: (implies
1295   (and
1296     (diagnosisContext BootTime)
1297     (hypothesis FloppySystem)
1298     (resultOfTest
1299       (TestFn FloppySystem ConfirmSensorially FloppyAccess) No)
1300     (plausibleInference Decompose))
1301   (and
1302     (testFirst FloppyDiskDrive)
1303     (testAfter FloppyDiskDrive BIOSsettings))).

1304 ;; *****
1305 ;; ** TEST(S) for the FloppyDiskDrive **
1306 ;; ** FloppyAccess=No **
1307 ;; *****
1308
1309 Direction: forward.
1310 F: (implies
1311   (and
1312     (diagnosisContext BootTime)
1313     (hypothesis FloppyDiskDrive)
1314     (resultOfTest
1315       (TestFn FloppySystem ConfirmSensorially FloppyAccess) No))
1316   (and
1317     (possibleTest (TestFn FloppyDiskDrive ConfirmSensorially BootContinues))
1318     (possibleResultOfTest
1319       (TestFn FloppyDiskDrive ConfirmSensorially BootContinues) Yes
1320       Distinguishing)
1321     (possibleResultOfTest
1322       (TestFn FloppyDiskDrive ConfirmSensorially BootContinues) No NotNormal))).

1323 ;; *****
1324 ;; ** TEST(S) for the OSfloppyDisk **
1325 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=Hard/None **
1326 ;; *****

1327 Direction: forward.
1328 F: (implies
1329   (and
1330     (diagnosisContext BootTime)
1331     (hypothesis OSfloppyDisk)
1332     (resultOfTest
1333       (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1334       FloppyThenHard)
1335   (or

```

```

1336         (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource)
1337                         HardDiskDrive)
1338         (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource) None)))
1339     (and
1340     (possibleTest (TestFn OSfloppyDisk ConfirmSensorially InFloppy))
1341     (possibleResultOfTest
1342     (TestFn OSfloppyDisk ConfirmSensorially InFloppy) Yes Insufficient)
1343     (possibleResultOfTest
1344     (TestFn OSfloppyDisk ConfirmSensorially InFloppy) No NotNormal))).

1345 ;; *****
1346 ;; ** TEST(S) for the OSfloppyDisk **
1347 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=Hard/None **
1348 ;; ** InFloppy=Yes **
1349 ;; *****

1350 Direction: forward.
1351 F: (implies
1352     (and
1353     (diagnosisContext BootTime)
1354     (hypothesis OSfloppyDisk)
1355     (resultOfTest
1356     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1357     FloppyThenHard)
1358     (or
1359     (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource)
1360     HardDiskDrive)
1361     (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource) None))
1362     (resultOfTest
1363     (TestFn OSfloppyDisk ConfirmSensorially InFloppy) Yes))
1364     (and
1365     (possibleTest
1366     (TestFn OSfloppyDisk TroubleshootComponent ComponentProblem))
1367     (possibleResultOfTest
1368     (TestFn OSfloppyDisk TroubleshootComponent ComponentProblem) Yes
1369     NotNormal)
1370     (possibleResultOfTest
1371     (TestFn OSfloppyDisk TroubleshootComponent ComponentProblem) No
1372     Distinguishing))).

1373 ;; *****
1374 ;; ** TEST(S) for the HardDiskDrive **
1375 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1376 ;; ** MemorySystem=ok, FloppySystem=ok **
1377 ;; *****

1378 Direction: forward.
1379 F: (implies
1380     (and
1381     (diagnosisContext BootTime)
1382     (hypothesis HardDiskDrive))
1383     (and
1384     (possibleTest (TestFn HardDiskDrive ConfirmSensorially DetectionMessage))
1385     (possibleResultOfTest
1386     (TestFn HardDiskDrive ConfirmSensorially DetectionMessage) Yes Normal)
1387     (possibleResultOfTest
1388     (TestFn HardDiskDrive ConfirmSensorially DetectionMessage) No Insufficient)
1389     (possibleResultOfTest
1390     (TestFn HardDiskDrive ConfirmSensorially DetectionMessage)
1391     CannotFind-Message NotNormal))).

1392 ;; *****
1393 ;; ** TEST(S) for the HardDiskDrive **
1394 ;; ** DetectionMessage=No **
1395 ;; *****

```

```

1396 Direction: forward.
1397 F: (implies
1398   (and
1399     (diagnosisContext BootTime)
1400     (hypothesis HardDiskDrive)
1401     (resultOfTest
1402       (TestFn HardDiskDrive ConfirmSensorially DetectionMessage) No))
1403   (and
1404     (possibleTest
1405       (TestFn HardDiskDrive CheckIndependently AutoDetection-BIOSsetting))
1406     (possibleResultOfTest
1407       (TestFn HardDiskDrive CheckIndependently AutoDetection-BIOSsetting)
1408         Manual Normal)
1409     (possibleResultOfTest
1410       (TestFn HardDiskDrive CheckIndependently AutoDetection-BIOSsetting)
1411         Auto NotNormal))).

1412 ;; *****
1413 ;; ** TEST(S) for the HardDiskDrive **
1414 ;; ** BootSequence-BIOSsetting=HardThenFloppy, BootSource=Floppy/None **
1415 ;; *****

1416 Direction: forward.
1417 F: (implies
1418   (and
1419     (diagnosisContext BootTime)
1420     (hypothesis HardDiskDrive)
1421     (resultOfTest
1422       (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1423         HardThenFloppy)
1424     (or
1425       (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource)
1426         FloppyDiskDrive)
1427       (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource) None)))
1428   (and
1429     (possibleTest
1430       (TestFn HardDiskDrive TroubleshootComponent ComponentProblem))
1431     (possibleResultOfTest
1432       (TestFn HardDiskDrive TroubleshootComponent ComponentProblem) Yes
1433         NotNormal)
1434     (possibleResultOfTest
1435       (TestFn HardDiskDrive TroubleshootComponent ComponentProblem) No
1436         Normal))).

1437 ;; *****
1438 ;; ** TEST(S) for the CDROMdrive **
1439 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1440 ;; ** MemorySystem=ok, FloppySystem=ok, HardDisk=ok **
1441 ;; *****

1442 Direction: forward.
1443 F: (implies
1444   (and
1445     (diagnosisContext BootTime)
1446     (hypothesis CDROMdrive))
1447   (and
1448     (possibleTest (TestFn CDROMdrive ConfirmSensorially DetectionMessage))
1449     (possibleResultOfTest
1450       (TestFn CDROMdrive ConfirmSensorially DetectionMessage) Yes Normal)
1451     (possibleResultOfTest
1452       (TestFn CDROMdrive ConfirmSensorially DetectionMessage) No Insufficient)
1453     (possibleResultOfTest
1454       (TestFn CDROMdrive ConfirmSensorially DetectionMessage) CannotFind-Message
1455         NotNormal))).

1456 ;; *****

```



```

1457 ;; ** TEST(S) for the CDROMdrive **
1458 ;; ** DetectionMessage=No **
1459 ;; *****

1460 Direction: forward.
1461 F: (implies
1462     (and
1463      (diagnosisContext BootTime)
1464      (hypothesis CDROMdrive)
1465      (resultOfTest
1466       (TestFn CDROMdrive ConfirmSensorially DetectionMessage) No))
1467     (and
1468      (possibleTest (TestFn CDROMdrive ConfirmSensorially BootContinues))
1469      (possibleResultOfTest
1470       (TestFn CDROMdrive ConfirmSensorially BootContinues) Yes Normal)
1471      (possibleResultOfTest
1472       (TestFn CDROMdrive ConfirmSensorially BootContinues) No NotNormal))).

1473 ;; *****
1474 ;; ** TEST(S) for the PlugAndPlaySystem **
1475 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1476 ;; ** MemorySystem=ok, FloppySystem=ok, HardDisk=ok **
1477 ;; ** CDROMdrive=ok **
1478 ;; *****

1479 Direction: forward.
1480 F: (implies
1481     (and
1482      (diagnosisContext BootTime)
1483      (hypothesis PlugAndPlaySystem))
1484     (and
1485      (possibleTest (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues))
1486      (possibleResultOfTest
1487       (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) Yes Normal)
1488      (possibleResultOfTest
1489       (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) No NotNormal))).

1490 ;; *****
1491 ;; ** DECOMPOSITION knowledge for the PlugAndPlaySystem **
1492 ;; ** BootContinues=No **
1493 ;; *****

1494 Direction: forward.
1495 F: (implies
1496     (and
1497      (diagnosisContext BootTime)
1498      (hypothesis PlugAndPlaySystem)
1499      (resultOfTest
1500       (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) No)
1501      (plausibleInference Decompose))
1502     (and
1503      (testFirst ExpansionCard)
1504      (testAfter ExpansionCard MotherBoard))).

1505 ;; *****
1506 ;; ** TEST(S) for the ExpansionCard **
1507 ;; ** BootContinues=No **
1508 ;; *****

1509 Direction: forward.
1510 F: (implies
1511     (and
1512      (diagnosisContext BootTime)
1513      (hypothesis ExpansionCard)
1514      (resultOfTest
1515       (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) No))

```

```

1516 (and
1517 (possibleTest
1518 (TestFn ExpansionCard TroubleshootComponent ComponentProblem))
1519 (possibleResultOfTest
1520 (TestFn ExpansionCard TroubleshootComponent ComponentProblem) Yes
1521 NotNormal)
1522 (possibleResultOfTest
1523 (TestFn ExpansionCard TroubleshootComponent ComponentProblem) No
1524 Distinguishing))).

1525 ;; *****
1526 ;; ** TEST(S) for the BootSystem **
1527 ;; ** Everything else=ok **
1528 ;; *****

1529 Direction: forward.
1530 F: (implies
1531 (and
1532 (diagnosisContext BootTime)
1533 (hypothesis BootSystem))
1534 (and
1535 (possibleTest
1536 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting))
1537 (possibleResultOfTest
1538 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1539 FloppyThenHard Insufficient)
1540 (possibleResultOfTest
1541 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1542 HardThenFloppy Insufficient))).

1543 ;; *****
1544 ;; ** TEST(S) for the BootSystem **
1545 ;; ** BootSequence-BIOSsetting=FloppyThenHard **
1546 ;; *****

1547 Direction: forward.
1548 F: (implies
1549 (and
1550 (diagnosisContext BootTime)
1551 (hypothesis BootSystem)
1552 (resultOfTest
1553 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1554 FloppyThenHard))
1555 (and
1556 (possibleTest (TestFn BootSystem ConfirmSensorially BootSource))
1557 (possibleResultOfTest
1558 (TestFn BootSystem ConfirmSensorially BootSource) FloppyDiskDrive Normal)
1559 (possibleResultOfTest
1560 (TestFn BootSystem ConfirmSensorially BootSource)
1561 HardDiskDrive NotNormal)
1562 (possibleResultOfTest
1563 (TestFn BootSystem ConfirmSensorially BootSource) None NotNormal))).

1564 ;; *****
1565 ;; ** DECOMPOSITION knowledge for the BootSystem **
1566 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=Hard **
1567 ;; *****

1568 Direction: forward.
1569 F: (implies
1570 (and
1571 (diagnosisContext BootTime)
1572 (hypothesis BootSystem)
1573 (resultOfTest
1574 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1575 FloppyThenHard)

```

```

1576     (resultOfTest
1577     (TestFn BootSystem ConfirmSensorially BootSource) HardDiskDrive)
1578     (plausibleInference Decompose))
1579     (and
1580     (testFirst OSfloppyDisk)
1581     (testAfter OSfloppyDisk FloppyDiskDrive))).

1582
1583 ;; *****
1584 ;; ** DECOMPOSITION knowledge for the BootSystem **
1585 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=None **
1586 ;; *****

1587 Direction: forward.
1588 F: (implies
1589     (and
1590     (diagnosisContext BootTime)
1591     (hypothesis BootSystem)
1592     (resultOfTest
1593     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1594     FloppyThenHard)
1595     (resultOfTest
1596     (TestFn BootSystem ConfirmSensorially BootSource) None)
1597     (plausibleInference Decompose))
1598     (and
1599     (testFirst OSfloppyDisk)
1600     (testAfter OSfloppyDisk FloppyDiskDrive))).
1601
1602 ;; *****
1603 ;; ** TEST(S) for the BootSystem **
1604 ;; ** BootSequence-BIOSsetting=HardThenFloppy **
1605 ;; *****

1606 Direction: forward.
1607 F: (implies
1608     (and
1609     (diagnosisContext BootTime)
1610     (hypothesis BootSystem)
1611     (resultOfTest
1612     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1613     HardThenFloppy))
1614     (and
1615     (possibleTest (TestFn BootSystem ConfirmSensorially BootSource))
1616     (possibleResultOfTest
1617     (TestFn BootSystem ConfirmSensorially BootSource) HardDiskDrive Normal)
1618     (possibleResultOfTest
1619     (TestFn BootSystem ConfirmSensorially BootSource)
1620     FloppyDiskDrive NotNormal)
1621     (possibleResultOfTest
1622     (TestFn BootSystem ConfirmSensorially BootSource) None NotNormal))).

1623 ;; *****
1624 ;; ** DECOMPOSITION knowledge for the BootSystem **
1625 ;; ** BootSequence-BIOSsetting=HardThenFloppy, BootSource=Floppy **
1626 ;; *****

1627 Direction: forward.
1628 F: (implies
1629     (and
1630     (diagnosisContext BootTime)
1631     (hypothesis BootSystem)
1632     (resultOfTest
1633     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1634     HardThenFloppy)
1635     (resultOfTest
1636     (TestFn BootSystem ConfirmSensorially BootSource) FloppyDiskDrive)

```

```

1637     (plausibleInference Decompose))
1638     (testFirst HardDiskDrive)).

1639
1640 ;; *****
1641 ;; ** DECOMPOSITION knowledge for the BootSystem **
1642 ;; ** BootSequence-BIOSsetting=HardThenFloppy, BootSource=None **
1643 ;; *****

1644 Direction: forward.
1645 F: (implies
1646     (and
1647         (diagnosisContext BootTime)
1648         (hypothesis BootSystem)
1649         (resultOfTest
1650             (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1651                 HardThenFloppy)
1652         (resultOfTest
1653             (TestFn BootSystem ConfirmSensorially BootSource) None)
1654         (plausibleInference Decompose))
1655     (testFirst HardDiskDrive)).
1656

```

# Appendix C

## The CYC SubL Code for PC Domain

```
1   ;;; ***** FUNCTION DEFINITIONS *****
2   ;;; ***** GLOBAL VARIABLES *****
3   (csetq *use-local-queue?* NIL) ;CYC variable
4   (defvar *defaultMt* '##PCdiagnosisMt) ;the microtheory for the fi-ask function
5   (defvar *test* nil) ; the current Test (needed by the 'menu' function)
6   (defvar *results* nil) ; a list of the Possible Results of the current Test
7       ; (needed by the 'menu' function)
8   (defvar *no_of_choices* 0) ;the number of Possible Results of the current Test
9       ; (needed by the 'menu' function)
10
11  ;;; The global variable *terms* is used to record in a list - and in
12  ;;; parallel with what the 'format' expressions print to the CYC SubL
13  ;;; Interactor panel - the CYC terms involved in each step of the
14  ;;; Systematic diagnosis. This variable is returned to the Interactor
15  ;;; panel as the 'Results' of the 'Last Form Evaluated' (see the CYC
16  ;;; SubL Interactor panel). The Interactor panel converts any CYC
17  ;;; term, i.e. symbols starting with '##$' (hash-dollar), as an HTML
18  ;;; link to this term in the CYC KB. This way, the user can browse to
19  ;;; any of the CYC terms appearing on the screen.
20
21  (defvar *terms* nil)
22
23  ;;; *****
24  ;;; ***** SubL CODE FOR SYSTEMATIC *****
25  ;;; *****
26
27  ;;; Function : SYSTEMATIC
28  ;;; Arguments: The system that is going to be diagnosed, i.e. '##PCSystem'.
29  ;;; Result    : Implements the Inference Structure for the Systematic
30  ;;;              Diagnosis problem solving method of KADS applied in PC
31  ;;;              faults diagnosis.
32  ;;; Remarks   :
33
34  (define systematic (system)
35    (pcond
36      ((not (eql system '##PCSystem)) (error "DIAGNOSIS: argument must be
37                                             '##PCSystem'"))
38      (t (pcond
39          ;; if there isn't any hypothesis then start diagnosis
40          ((not (fi-ask '##$hypothesis ?HYP) *defaultMt*))
```

```

36         (csetf *terms* nil)
37         (sd-select1))
38         (t (sd-compare)) ;end of innermost 't' clause
39         )) ;end of innermost 'pcond' and outermost 't' clause
40     ) ;end of outermost 'pcond'
41 )

42 ;;; *****
43 ;;; ***** SubL CODE FOR SD-COMPARE * *****
44 ;;; *****

45 ;;; Function : SD-COMPARE
46 ;;; Arguments: None
47 ;;; Result   : Calls the appropriate Systematic Diagnosis Inference
48 ;;; Remarks  : The *test* global variable holds the last 'Test' that
49 ;;;            was performed. It is used to retrieve the result of
50 ;;;            this 'Test' and its type. According to the 'ResultType'
51 ;;;            of the 'Test', the following decisions may be made:
52 ;;;
53 ;;;            RESULT_TYPE          DECISION
54 ;;;
55 ;;;            Normal                Reject hypothesis; backtrack
56 ;;;            NotNormal             Decompose/Confirm hypothesis
57 ;;;            Insufficient          Perform another 'Test' (Select2-3)
58 ;;;            Distinguishing        Reject hypothesis, backtrack
59 ;;;            and confirm next hypothesis.

60 (define sd-compare ()
61   (csetf *terms* nil)
62   ;; get the 'ResultType' for the most recently performed 'Test'
63   (csetq ask-result
64     (fi-ask (list '#$and
65               (list '#$resultOfTest *test* '?RES)
66               (list '#$possibleResultOfTest *test* '?RES '?TYPE)) *defaultMt*))
67   (format t "LAST TEST: ~S, ~%RESULT: ~S," *test* (get-ask-binding
68           (first ask-result) 1))
69   (csetf *terms* (cons (cons 'LAST (cons 'TEST: *test*)) *terms*))

70   (csetq result (get-ask-binding (first ask-result) 1))
71   (csetq result-type (get-ask-binding (first ask-result) 2))
72
73   ;; according to the 'ResultType' value, decide the next inference
74   (pcond
75     ((eql result-type '#$NotNormal)
76      (format t " RESULT TYPE: #$NotNormal, INFERENCE: Confirm/Decompose")
77      (csetf *terms* (cons (list 'RESULT: result 'RESULT 'TYPE:
78                              result-type 'INFERENCE:CONFIRM) *terms*))
79      (sd-confirm))
80     ((eql result-type '#$Insufficient)
81      (format t " RESULT TYPE: #$Insufficient, INFERENCE: Select2-3")
82      (csetf *terms* (cons (list 'RESULT: result 'RESULT 'TYPE:
83                              result-type 'INFERENCE:SELECT2-3) *terms*))
84      (sd-select2-3))
85     ((cor (eql result-type '#$Normal) (eql result-type '#$Distinguishing))
86      (format t " RESULT TYPE: ~S, INFERENCE: NewHypothesis" result-type)
87      (csetf *terms* (cons (list 'RESULT: result 'RESULT 'TYPE:
88                              result-type 'INFERENCE:New_Hypothesis) *terms*))
89      (sd-new-hypothesis result-type))
90     (t (format t " RESULT TYPE: UNKNOWN!, INFERENCE: Diagnosis interrupted"))
91   ) ;end of 'pcond'
92 )

93 ;;; *****
94 ;;; ***** SubL CODE FOR SD-CONFIRM *****
95 ;;; *****

```

```

96   ;;; Function : SD-CONFIRM
97   ;;; Arguments: None
98   ;;; Result   : If the current 'hypothesis' is a 'PCSubSystem', it
99   ;;;             is decomposed by calling function 'sd-decompose';
100  ;;;             otherwise, it is a PCComponent and the diagnosis
101  ;;;             terminates reporting this 'PCComponent' as faulty.
102  ;;; Remarks   :

103  (define sd-confirm ()
104    (csetq ask-result
105      (fi-ask '($and
106              ($hypothesis ?HYP)
107              ($isa ?HYP #$PCComponent)) *defaultMt*))
108    (pcond
109      (ask-result
110        (format t "~%-% DIAGNOSIS ENDED. FAULTY COMPONENT: ~S"
111              (get-ask-binding (first ask-result) 1))
112        (csetf *terms* (cons (list 'DIAGNOSIS 'ENDED. 'FAULTY 'COMPONENT:
113                                (get-ask-binding (first ask-result) 1)) *terms*))
114        (reverse *terms*))
115      (t (format t "Decomposing..."
116              (safe-fi :assert '($plausibleInference #$Decompose) *defaultMt*)
117              (sd-decompose)
118              (sd-select2-3))
119        )
120    )
121
122  ;;; *****
123  ;;; ***** SubL CODE FOR SD-NEW-HYPOTHESIS *****
124  ;;; *****

125  ;;; Function: SD-NEW-HYPOTHESIS
126  ;;; Arguments: 1. A #$ResultType
127  ;;; Results:
128  ;;; Remarks: This function is called by 'sd-confirm' with two types of
129  ;;;           results, #$Normal and #$Distinguishing. If the result is
130  ;;;           #$Normal, the function just changes the current
131  ;;;           'hypothesis' to the next one (if one exists) and performs
132  ;;;           the appropriate 'Test' by calling 'sd-select2-3'. If the result is
133  ;;;           #$Distinguishing, then it changes the current
134  ;;;           'hypothesis' as before, but doesn't perform any 'Test', as
135  ;;;           the last 'Test' performed indicates that the next
136  ;;;           'hypothesis' is faulty. Therefore, it calls the
137  ;;;           'sd-confirm' function.

138  (define sd-new-hypothesis (result-type)

139  ;; get the next hypothesis from the 'testAfter' assertions, if anyone
140  ;; is left
141  (csetq ask-result
142    (fi-ask '($and
143            ($hypothesis ?HYP)
144            ($testAfter ?HYP ?NEW)HYP)) *defaultMt*))
145  (pcond
146    ((null ask-result) (format t "~% ~%I'M SORRY. THERE IS NO ALTERNATIVE
147                          SYSTEM TO BE CONSIDERED. DIAGNOSIS FAILED"))
148    (t (csetq hypothesis (get-ask-binding (first ask-result) 1))
149      (csetq new_hypothesis (get-ask-binding (first ask-result) 2))
150      (fi-unassert (list '$hypothesis hypothesis) *defaultMt*)
151      (fi-unassert (list '$testAfter hypothesis new_hypothesis) *defaultMt*)
152      (safe-fi :assert (list '$hypothesis new_hypothesis) *defaultMt*)
153      (pcond
154        ((eql result-type '$Normal) (sd-select2-3))
155        ((eql result-type '$Distinguishing) (sd-confirm))
156      )) ;end of inner 'pcond' and 't' clause
157  ) ;end of 'pcond'

```

```

158 )

159 ;;; *****
160 ;;; ***** SubL CODE FOR SD-SELECT1 *****
161 ;;; *****

162 ;;; Function: SD-SELECT1
163 ;;; Arguments: None.
164 ;;; Results: Asserts the '($hypothesis #$PCSystem)' fact to start the
165 ;;;          Systematic Diagnosis problem solving method
166 ;;; Remarks:

167 (define sd-select1 ()
168   (safe-fi :assert '($hypothesis #$PCSystem) *defaultMt*)
169   (sd-select2-3) ; ask the user what the general complaint is
170 )

171 ;;; *****
172 ;;; ***** SubL CODE FOR SD-SELECT2-3 *****
173 ;;; *****

174 ;;; Function : SD-SELECT2-3
175 ;;; Arguments: None
176 ;;; Result   :
177 ;;; Remarks  : According to the situation, there may be a lot of
178 ;;; 'possibleTest' assertions in the KB, but only one of them is the
179 ;;; one that must be performed next. This Test is distinguished by the
180 ;;; fact that it is the only one that doesn't have a corresponding
181 ;;; 'resultOfTest' assertion as it is not yet carried out. The
182 ;;; 'sd-select2-3' function must therefore find this Test and pass it and
183 ;;; its possible results ('possibleresultOfTest' assertions) to the
184 ;;; 'get-test-result' function.

185 (define sd-select2-3 ()

186 ;;get the current 'hypothesis'
187 (csetq hypothesis (fi-ask '($hypothesis ?HYP) *defaultMt*))
188 (csetq hypothesis (get-ask-binding (first hypothesis) 1))
189 (format t "~%~%HYPOTHESIS: ~S" hypothesis)
190 (csetf *terms* (cons (list 'HYPOTHESIS: hypothesis) *terms*))

191 ;; Get all possible tests
192 (csetq possible-tests
193   (fi-ask '($possibleTest ?TEST) *defaultMt*))

194 ;; keep the one that doesn't have a corresponding 'resultOfTest' assertion
195 (cdo
196   ((test
197     (get-ask-binding (first possible-tests) 1)
198     (get-ask-binding (first possible-tests) 1))
199    ) ; end of variables
200   ((cnot (fi-ask (list '##resultOfTest test '?R) *defaultMt*)) t) ;exit condition
201   (csetq possible-tests (rest possible-tests))
202   )
203 (csetq test (get-ask-binding (first possible-tests) 1))
204 (csetf *test* test)

205 ;; Get the possible results for this test
206 (csetq possible-results
207   (fi-ask
208     (list
209       '##possibleResultOfTest (list '##TestFn (second test) (third test)
210                                     (fourth test)) '?VAL '?TYPE) *defaultMt*))
211 (get-test-result test possible-results)

```



```

212 )

213 ;;; *****
214 ;;; ***** SubL CODE FOR GET-TEST-RESULT *****
215 ;;; *****

216 ;;; Function : GET-TEST-RESULT
217 ;;; Arguments: 1. A 'Test' structure, which is a list of the form:
218 ;;;
219 ;;; (TestFn PC_SUBSYSTEM TEST_ACTION POSSIBLE_OBSERVABLE)
220 ;;;
221 ;;;          2. A list of the form:
222 ;;;
223 ;;; (
224 ;;; ((?VAL . RESULT_1) (?TYPE . TYPE_1))
225 ;;;   ...
226 ;;; ((?VAL . RESULT_n) (?TYPE . TYPE_n))
227 ;;; )
228 ;;;
229 ;;; Result   : An 'resultOfTest' Assertion in the KB with the actual result of
230 ;;;            the test.
231 ;;; Remarks  : For the moment, it is the 'menu' function that performs
232 ;;;            the actual task as there is no way to get input from
233 ;;;            the user when in the SubL interactor.

234 (define get-test-result (test possible-results)
235   (present-test-parameters test)
236   (present-test-results possible-results)
237   (format t "~%Please, type '(menu [number_of_result])'")
238   ; (input-test-result no_of_choices)
239   (reverse *terms*) ;return as RESULT a list of all CYC terms appearing on the screen

240 ;;; For the moment, we don't know how to interact with the user when in the
241 ;;; SubL Interactor interface. Therefore, the user must give the command
242 ;;; '(menu <number_of_result>)' to interact with the SubL code.
243 )
244
245 ;;; *****
246 ;;; ***** SubL CODE FOR PRESENT-TEST-PARAMETERS*
247 ;;; *****

248 ;;; Function : PRESENT-TEST-PARAMETERS
249 ;;; Arguments: A 'Test' structure, which is a list of the form:
250 ;;;
251 ;;; (TestFn PC_SUBSYSTEM TEST_ACTION POSSIBLE_OBSERVABLE)
252 ;;;
253 ;;; Result   : Prints to the screen the current 'test' parameters, i.e.,
254 ;;;            the PCSubSystem, TestAction and PossibleObservable.
255 ;;; Remarks  :

256 (define present-test-parameters (test)
257   (format t "~%NEW TEST: ~%PCSubSystem: ~S ~%TestAction: ~S
258           ~%Observable: ~S ~%" (second test) (third test) (fourth test))
259   (csetf *terms* (cons (cons 'NEW (cons 'TEST test)) *terms*))
260 )

261 ;;; *****
262 ;;; ***** SubL CODE FOR PRESENT-TEST-RESULTS **
263 ;;; *****

264 ;;; Function : PRESENT-TEST-RESULTS
265 ;;; Arguments: A list of the form:
266 ;;;
267 ;;; ((?VAL . RESULT_1) (?TYPE . TYPE_1))
268 ;;;   ...

```

```

269 ;;; ((?VAL . RESULT_n) (?TYPE . TYPE_n))
270 ;;; )
271 ;;;
272 ;;; Result   : An enumerated menu of all possible results of the current 'test'
273 ;;; Remarks  :

274 (define present-test-results (possible-results)

275 ;; Get possible results
276 (csetq results (mapcar #'get-ask-binding
277                     possible-results
278                     (position-list (length possible-results) 1)))

279 ;; Get possible results' types (Not needed for the moment. The result
280 ;; type is retrieved by the 'sd-compare' function).

281 ; (csetq result_types (mapcar #'get-ask-binding
282 ;                             possible-results
283 ;                             (position-list (length possible-results) 2)))

284 (csetq counter 1)
285 (cdolist (result results 't)
286   (format t "~A. ~S%" counter result)
287   (csetf *terms* (cons (cons counter result) *terms*))
288   (csetq counter (+ counter 1))
289 )
290 (csetf *no_of_choices* (- counter 1)) ;needed by 'menu'
291 (csetf *results* results) ;needed by 'menu'
292 (csetf *result_type* result_types) ; needed by 'menu'
293 )

294 ;;; *****
295 ;;; ***** SubL CODE FOR POSITION-LIST *
296 ;;; *****

297 ;;; Function : POSITION-LIST
298 ;;; Arguments: 1. A number, n, indicating the number of bindings (lists of
299 ;;;               dotted pairs), in an ask-result of an 'fi-ask' function.
300 ;;;               2. A number, k (1 =< k =< n), indicating which BINDING must
301 ;;;               be returned by the 'get-ask-binding' function.
302 ;;; Result   : A list of n elements equal to k.
303 ;;; Remarks  : An auxiliary function. Creates the second argument to be used
304 ;;;               in a 'mapping' function which collects a list of BINDINGS for
305 ;;;               the same ask variable.

306 (define position-list (n k)
307 (csetq res ())
308 (cdotimes (c n res)
309   (csetq res (cons k res))
310 )
311 res
312 )

313 ;;; *****
314 ;;; ***** SubL CODE FOR GET-ASK-BINDING ***
315 ;;; *****

316 ;;; Function : GET-ASK-BINDING
317 ;;; Arguments: 1. A list of dotted pairs of the form:
318 ;;;               ;;;
319 ;;;               ((?VAR1 . BINDING1)
320 ;;;               (?VAR2 . BINDING2)
321 ;;;               ...
322 ;;;               (?VARn . BINDINGn)
323 ;;; )

```

```

324   ;;;          2. A number defining which BINDING must be returned.

325   ;;; Result   : POSSIBLE_OBSERVABLE_VALUE
326   ;;; Remarks  :

327   (define get-ask-binding (bindings_list bind_no)
328     (rest (nth (- bind_no 1) bindings_list))
329   )
330

331   ;;; *****
332   ;;; ***** SubL CODE FOR MENU *****
333   ;;; *****

334   ;;; Function : MENU
335   ;;; Arguments: 1. A number from the menu of the possible results (local)
336   ;;;            2. The list of possible results (global variable *results*)
337   ;;;            3. The number of choices (global variable *no_of_choices*)
338   ;;; Result    : Asserts into the KB a 'resultOfTest' assertion
339   ;;; Remarks  :

340   (define menu (selection)
341     (pcond
342       ((cor (< selection 1) (> selection *no_of_choices*))
343        (format t "%~%You must give as an argument, a number between 1-~A"
344              *no_of_choices*))
345       (t (csetq test (fi-ask '($possibleTest ?TEST) *defaultMt*)
346          (csetq test (get-ask-binding (first test) 1))
347          ; (csetf *result_type* (nth (- selection 1) *result_type*))
348          (safe-fi :assert
349              (list '$resultOfTest test (nth (- selection 1) *results*)) *defaultMt*))
350       )
351     (systematic '$PCSystem)
352   )

353   ;;; The 'decompose' inference in KADS Systematic Diagnosis PSM takes as input
354   ;;; the current #$PCSubSystem ($hypothesis PC_SUBSYSTEM) and decomposes it
355   ;;; into its functional subsystems ($functionalPartOf PC_SUBSYSTEM PART),
356   ;;; generating new hypotheses ($possibleHypotheses PART).

357   ;;; *****
358   ;;; ***** SubL CODE FOR SD-DECOMPOSE *****
359   ;;; *****

360   ;;; Function : SD-DECOMPOSE
361   ;;; Arguments: None.
362   ;;; Result   :
363   ;;; Remarks  :

364   (define sd-decompose ()
365     ;; Before un-asserting the current hypothesis, its functional parts
366     ;; must be saved
367     (csetq ask-result (fi-ask '($possibleHypotheses ?H) *defaultMt*))
368     (csetq in_hypotheses
369       (mapcar #'get-ask-binding ask-result (position-list (length ask-result) 1)))

370     ;; Before un-asserting the current hypothesis, the order of its subsystems
371     ;; diagnosis must be saved
372     (csetq first_to_test (fi-ask '($testFirst ?SYS) *defaultMt*))
373     (csetq first_to_test (get-ask-binding (first first_to_test) 1))
374     (csetq afters (fi-ask '($testAfter ?S1 ?S2) *defaultMt*))
375     (csetq sl_list

```

```

376         (mapcar #'get-ask-binding afters (position-list (length afters) 1)))
377     (csetq s2_list
378         (mapcar #'get-ask-binding afters (position-list (length afters) 2)))

379 ;; Store hypothesis and un-assert it
380 (csetq hypothesis (fi-ask '($hypothesis ?H) *defaultMt*))
381 (csetq hypothesis (get-ask-binding (first hypothesis) 1))
382 (fi-unassert (list '$hypothesis hypothesis) *defaultMt*)

383 ;; Assert possibleHypotheses.
384 (cdolist (system in_hypotheses t)
385     (safe-fi :assert (list '$possibleHypotheses system) *defaultMt*)
386 )

387 ;; Assert diagnosis order
388 ; (safe-fi :assert (list '$testFirst first_to_test) *defaultMt*) ;unnecessary
389 (cdolist (s1 s1_list t)
390     (csetq s2 (first s2_list))
391     (csetq s2_list (rest s2_list))
392     (safe-fi :assert (list '$testAfter s1 s2) *defaultMt*)
393 )

394 ;; Un-assert the ($plausibleInference #$Decompose) assertion.
395 (fi-unassert '($plausibleInference #$Decompose) *defaultMt*)

396 ;; Assert the new hypothesis
397 (safe-fi :assert (list '$hypothesis first_to_test) *defaultMt*)
398 )

399 ;;; *****
400 ;;; ***** SubL CODE FOR SD-RESET *****
401 ;;; *****

402 ;;; Function : SD-RESET
403 ;;; Arguments: None
404 ;;; Result   : Resets all assertions regarding the following
405 ;;; predicates:
406 ;;;     '$diagnosisContext
407 ;;;     '$hypothesis',
408 ;;;     '$possibleHypotheses'
409 ;;;     '$resultOfTest'
410 ;;;     '$testFirst'
411 ;;;     '$testAfter'
412 ;;;     '$diagnosisContext' (dependant from '$resultOfTest)
413 ;;;     '$possibleTest'    (dependant from '$hypothesis & '$resultOfTest)

414 ;;; Remarks :

415 (define sd-reset ()
416     ;; Set the global variables
417     (csetf *use-local-queue?* NIL)
418     (csetf *defaultMt* '$PCDiagnosisMt)
419     (csetf *test* nil)
420     (csetf *results* nil)
421     (csetf *no_of_choices* 0)

422     ;;UN-ASSERT '$diagnosisContext
423     (csetq diagnosisContext (fi-ask '($diagnosisContext ?C) *defaultMt*))
424     (csetq diagnosisContext (get-ask-binding (first diagnosisContext) 1))
425     (fi-unassert (list '$diagnosisContext diagnosisContext) *defaultMt*)

426     ;; UN-ASSERT '$hypothesis
427     (csetq hypothesis (fi-ask '($hypothesis ?H) *defaultMt*))
428     (csetq hypothesis (get-ask-binding (first hypothesis) 1))
429     (fi-unassert (list '$hypothesis hypothesis) *defaultMt*)

```

```

430 ;; UN-ASSERT #$$possibleHypotheses
431 (csetq ask-result (fi-ask '(#$$possibleHypotheses ?H) *defaultMt*))
432 (csetq results
433   (mapcar #'get-ask-binding ask-result (position-list (length ask-result) 1)))
434 (cdolist (result results t)
435   (fi-unassert (list '#$$possibleHypotheses result) *defaultMt*))
436 )

437 ;; UN-ASSERT #$$resultOfTest
438 (csetq ask-result (fi-ask '(#$$resultOfTest ?T ?R) *defaultMt*))
439 (cdolist (bindings ask-result t)
440   (fi-unassert
441     (list '#$$resultOfTest (get-ask-binding bindings 1)
442           (get-ask-binding bindings 2)) *defaultMt*))
443 )

444 ;;UN-ASSERT #$$testFirst
445 (csetq first_to_test (fi-ask '(#$$testFirst ?SYS) *defaultMt*))
446 (csetq first_to_test (get-ask-binding (first first_to_test) 1))
447 (fi-unassert (list '#$$testFirst first_to_test) *defaultMt*)

448 (csetq afters (fi-ask '(#$$testAfter ?S1 ?S2) *defaultMt*))
449 (csetq s1_list
450   (mapcar #'get-ask-binding afters (position-list (length afters) 1)))
451 (csetq s2_list
452   (mapcar #'get-ask-binding afters (position-list (length afters) 2)))
453 (cdo
454   ((s1 (first s1_list) (first s1_list))
455    (s2 (first s2_list) (first s2_list))
456    (s1_list (rest s1_list) (rest s1_list))
457    (s2_list (rest s2_list) (rest s2_list))
458   ) ;end of variables
459 ((null s1)) ;when no more couples of s1,s2
460 (fi-unassert (list '#$$testAfter s1 s2) *defaultMt*))
461 )
462 )

```



# Appendix D

## The CYC KE-Text for PC/Automobile Domains

```
1   ;;; PROJECT: 628-Implementing Problem Solving Methods (PSMs) in Cyc
2   ;;; FILENAME: systematic_diagnosisKE.txt
3   ;;; AUTHOR: Dimitrios Sklavakis
4   ;;; PURPOSE: Contains Cyc's Knowledge Entering (KE) text defining the
5   ;;;           general knowledge for the implementation of the
6   ;;;           Systematic Diagnosis (Localisation) PSM from KADS
7   ;;;           methodology for Personal Computers (PCs) and Automobiles.

8   ;;; LAST UPDATED: 05/08/1998.

9   ;***** THE $$SystematicDiagnosisMt MICROTHEORY *****

10  ;;; The general knowledge for performing Systematic Diagnosis will be
11  ;;; grouped in the $$SystematicDiagnosisMt microtheory. This one is a
12  ;;; more specific microtheory than the $$BaseKB microtheory,
13  ;;; i.e. ($$genlMt $$SystematicDiagnosisMt $$BaseKB) and more general
14  ;;; than the $$PCDiagnosisMt and $$AutomobileDiagnosisMt
15  ;;; microtheories:

16  ;;; ($$genlMt $$PCDiagnosisMt $$SystematicDiagnosisMt) ($$genlMt
17  ;;; $$AutomobileDiagnosisMt $$SystematicDiagnosisMt)

18  constant: SystematicDiagnosisMt.
19  isa: Microtheory.
20  genls: BaseKB.
21  comment: "$$SystematicDiagnosisMt is the $$Microtheory that contains
22  all the assertions about performing KADS' Systematic Diagnosis
23  (Localisation) problem solving method.".

24  ; ***** SYSTEM MODEL KNOWLEDGE *****

25  Default Mt: SystematicDiagnosisMt.

26  constant: SubSystem.
27  isa: Collection.
28  genls: CompositeTangibleAndIntangibleObject.
29  comment: "The collection of all Psub-systems, like the
30  $$VideoSystem, $$PowerSystem, $$KeyboardSystem. Each instance of
31  $$SubSystem may include several $$Components and/or other
32  $$SubSystems. Different $$SubSystems may include the same
33  $$Components. In the context of $$SystematicDiagnosisMt any $$SubSystem is
```

```

34 an intermediate level of analysis for the system model; the
35 diagnosis continues until a faulty #Component is located".

36 constant: Component.
37 isa: Collection.
38 genls: SubSystem.
39 comment: "The collection of all components such as the
40 #PowerSupply, #VideoCard, #FloppyDiskDrive. In the context of
41 #SystematicDiagnosisMt any #Component is the lowest level of analysis for
42 the system model; the diagnosis terminates when a faulty
43 #Component is located.".

44 ;; For Systematic Diagnosis, a predicate is needed to express the
45 ;; hierarchical system model, consisting of #SubSystems and
46 ;; #Components. However, this predicate is domain dependent and
47 ;; therefore will be defined separately in each domain-specific
48 ;; theory. For example, in #PCDiagnosisMt it is #functionalPartOf,
49 ;; while in #AutomobileDiagnosisMt is #physicalDecompositions.
50
51 constant: testFirst.
52 isa: UnaryPredicate.
53 arg1Isa: SubSystem.
54 comment: "This predicate is used to declare which SubSystem from
55 these occurring after a #Decompose inference type will be the
56 the first to consider for diagnosis, i.e., the #hypothesis.".

57 constant: testAfter.
58 isa: BinaryPredicate.
59 arg1Isa: SubSystem.
60 arg2Isa: SubSystem.
61 comment: "This predicate is used to declare the order of considering
62 subsystems for diagnosis. For example, (#testAfter SUBSYSTEM1
63 SUBSYSTEM2) means that that the #SubSystem SUBSYSTEM2 will
64 be considered for diagnosis immediately after SUBSYSTEM1.".

65 constant: hypothesis.
66 isa: UnaryPredicate.
67 arg1Isa: SubSystem.
68 comment: "The predicate is used to record in the KB which
69 #SubSystem is currently being diagnosed. E.g.,
70 #hypothesis(#VideoSystem) means that it is the #VideoSystem that is
71 currently being checked for possible faults.".

72 constant: possibleHypotheses.
73 isa: UnaryPredicate.
74 arg1Isa: SubSystem.
75 comment: "The predicate is used to record in the KB which
76 #SubSystems are currently candidates for being diagnosed. E.g.,
77 #possibleHypotheses(#VideoCard) means that the #VideoCard is
78 a candidate to be checked for possible faults.".

79 ;***** SYSTEM TESTING KNOWLEDGE *****

80 ;; The basic tool in the Systematic Diagnosis problem-solving method
81 ;; (PSM), as well as in any other diagnostic PSM, for carrying out
82 ;; the diagnostic procedure, is various TESTS that must be done to
83 ;; provide information (knowledge) about the state of the
84 ;; system. This knowledge may concern the actual behaviour of the
85 ;; system's components, e.g. the absence of electric power, control
86 ;; information produced by the system, e.g., beep codes or screen
87 ;; messages. Conceptually, a TEST is a question that the user must
88 ;; make to the system under diagnosis to extract knowledge about
89 ;; it. Here, it is implemented as a structure consisting of three
90 ;; other concepts:
91 ;; 1. The #SubSystem to which it is related, i.e., to which the
92 ;; question is addressed.

```



```

93   ;;; 2. The #$$TestAction which is the action one has really to perform
94   ;;;   for the test
95   ;;; 3. The #$$PossibleObservable (system variable) that the TEST is
96   ;;;   asking about .

97   ;;; Although not part of a TEST structure, there is a fourth concept
98   ;;; related with it, the #$$PossibleObservableValue (system variable
99   ;;; value) which is the result (answer) of the TEST's question.

100  ;;; Each TEST is represented as a non-atomic term (NAT) in CycL with
101  ;;; the use of a #$$NonPredicateFunction, #$$TestFn, which takes as
102  ;;; arguments instances of the three constituent concepts and returns
103  ;;; a TEST structure, Schematically:

104  ;;; (#$$TestFN #$$SubSystem #$$TestAction #$$PossibleObservable) -> #$$Test

105  Default Mt: SystematicDiagnosisMt.

106  constant: Test.
107  isa: Collection.
108  genls: InformationBearingThing.
109  comment: "The basic tool in the Systematic Diagnosis problem-solving
110  method (PSM), as well as in any other diagnostic PSM, for carrying out
111  the diagnostic procedure, is various TESTS that must be done to
112  provide information (knowledge) about the state of the system. This
113  knowledge may concern the actual behaviour of the system's components,
114  e.g. the absence of electric power, control information produced by
115  the system, e.g., beep codes or screen messages. Conceptually, a TEST
116  is a question that the user must make to the system under diagnosis to
117  extract knowledge about it. Here, it is implemented as a structure
118  consisting of three other concepts: 1. The #$$SubSystem to which it
119  is related, i.e., to which the question is adressed, 2. The
120  #$$TestAction which is the action one has really to perform for the
121  test and 3. The #$$PossibleObservable (system variable) that the TEST
122  is asking about.".

123  constant: TestFn.
124  isa: NonPredicateFunction.
125  arity: 3.
126  arg1Isa: SubSystem.
127  arg2Isa: TestAction.
128  arg3Isa: PossibleObservable.
129  resultIsa: Test.
130  comment: "Every #$$Test is a structure consisting of three
131  concepts. The #$$SubSystem to which it is related, the actual
132  #$$TestAction that must be performed and the #$$PossibleObservable
133  (system variable) that is being observed. Each TEST is represented as a
134  non-atomic term (NAT) in CycL
135  with the use of the #$$NonPredicateFunction, #$$TestFn, which takes as
136  arguments instances of the three constituent concepts and returns a
137  TEST structure, Schematically:
138  (#$$TestFN #$$SubSystem #$$TestAction #$$PossibleObservable) -> #$$Test".

139  constant: possibleTest.
140  isa: UnaryPredicate.
141  arg1Isa: Test.
142  comment: "The #$$Tests available to be performed in any stage of the
143  Diagnosis.".

144  constant: TestAction.
145  isa: Collection.
146  genls: PurposefulAction.
147  comment: "The collection of all possible test actions that may be
148  performed from the user on a #$$SubSystem to determine the
149  #$$PossibleObservableValues of

```

```

150 a ##PossibleObservable. These values are compared to the expected
151 ones. If they are different, the fault lies somewhere in the
152 ##SubSystem which is further decomposed and its parts are
153 checked one by one. If not, the fault lies in another ##SubSystem".
154
155 constant: PossibleObservable.
156 isa: Collection.
157 genls: AttributeType.
158 comment: "The collection of system variables (possible observables)
159 which are used to decide if the currently checked ##SubSystem actually
160 contains a faulty ##Component or not.".

161 constant: PossibleObservableValue.
162 isa: Collection.
163 genls: AttributeValue.
164 comment: "The collection of all possible values of all
165 ##PossibleObservables. In terms of the Systematic Diagnosis
166 problem-solving method, the instances of ##PossibleObservable
167 correspond to the system variables that one can test during diagnosis and the
168 instances of ##PossibleObservableValue correspond to the possible
169 outcomes of these tests.".

170 constant: ObservableValueFn.
171 isa: NonPredicateFunction.
172 arity: 1.
173 arg1Isa: Thing.
174 resultIsa: PossibleObservableValue.
175 comment: "Although the collection of all terms that are used as
176 possible answers to the ##Tests of the Systematic Diagnosis problem
177 solving method (PSM) are collected in the ##PossibleObservableValue
178 collection, in general anything can be an instance of
179 ##PossibleObservableValue. For example, the ##HardDiskDrive can be an
180 instance of this collection. It only depends on the questions one
181 makes. Anything can be an answer to a question. For this reason, the
182 ##ObservableValueFn function is introduced. It can take anything as an
183 argument and make it play the role of an instance of
184 ##PossibleObservableValue. This function makes the existence
185 of ##PossibleObservableValue seem redundant, however this latter
186 collection plays a significant role in the analysis of the Systematic
187 Diagnosis PSM and therefore has a reason to exist.".

188 constant: ResultType.
189 isa: Collection.
190 genls: AttributeValue.
191 comment: "The collection of all various types of
192 ##PossibleObservableValues. These types are characterised from the
193 kind of conclusions they lead relative to the ##SubSystem being
194 currently diagnosed ( ##hypothesis(##SubSystem) ). E.g., such a type
195 can be ##Normal which denotes that the ##SubSystem currently being
196 diagnosed is not faulty and therefore must be discarded as a
197 hypothesis and a new hypothesis must be selected.".

198 constant: possibleResultOfTest.
199 isa: Predicate.
200 arity: 3.
201 arg1Isa: Test.
202 arg2Isa: PossibleObservableValue.
203 arg3Isa: ResultType.
204 comment: "The predicate is correlating an individual ##Test with its
205 actual result and the type of this result. The assertion
206 possibleResultOfTest(TEST VALUE TYPE)
207 express the fact that for the specific TEST, VALUE is a possible
208 result of type TYPE. E.g., possibleResultOfTest((TestFn
209 PowerSystem ConfirmSensorially ElectricPower) Yes Normal) indicates that
210 it is ##Normal to observe the existence of ##ElectricPower when
211 diagnosing the ##PowerSystem. Of course, this immediately would imply

```

```

212 that the ##PowerSystem is not faulty and therefore should be discarded
213 as a ##hypothesis.".

214 constant: resultOfTest.
215 isa: BinaryPredicate.
216 arg1Isa: Test.
217 arg2Isa: PossibleObservableValue.
218 comment: "The predicate is correlating an individual ##Test with its
219 actual result. The assertion resultOfTest(?TEST ?VALUE) means that
220 ?VALUE is the actual result of ?TEST.".

221 ;***** DEFINITIONS OF INSTANCES FOR ##TestAction COLLECTION *****

222 constant: ConfirmSensorially.
223 isa: TestAction.
224 comment: "The action of confirming the existence of a
225 ##PossibleObservable only by one's senses, e.g., visually,
226 acoustically.".

227 constant: CheckIndependently.
228 isa: TestAction.
229 comment: "This ##TestAction means that the user performing
230 diagnosis must check the function of the related ##SubSystem
231 isolated from the rest of the ##SubSystem of which it is a
232 part of. The way to do that is not specifically described
233 by the name of the action. It is assumed that the user has some
234 knowledge for performing such isolated tests. E.g., to test the
235 ##PowerSocket one can plug another device - known to be working - in
236 it and confirm that the device has ##ElectricPower.".

237 constant: Remove.
238 isa: TestAction.
239 comment: "This ##TestAction means that the user must remove the
240 related ##SubSystem from the ##SubSystem of which it is a
241 part of".

242 constant: Replace.
243 isa: TestAction.
244 comment: "This ##TestAction means that the user must replace the
245 related ##SubSystem with a new one".

246 constant: TroubleshootComponent.
247 isa: TestAction.
248 comment: "This ##TestAction means that the diagnosis reached at the
249 level of a specific PCComponent but there is not sufficient
250 information to confirm that it is faulty. Therefore, the user must
251 enter the stage of troubleshooting it specifically.".

252 ;***** DEFINITIONS OF INSTANCES FOR $PossibleObservable COLLECTION ****

253 constant: ProblemContext.
254 isa: PossibleObservable.
255 comment: "This ##PossibleObservable refers to the general context of
256 diagnosis, i.e., ##BootTime, ##RunTime, ##ComponentSpecific. The
257 value of this ##PossibleObservable determines which rules are
258 applicable, appearing as a condition in their antecedent part".

259 ;***** DEFINITIONS OF INSTANCES FOR $PossibleObservableValue COLLECTION ****

260 constant: RunTime.
261 isa: PossibleObservableValue.
262 comment: "This ##PossibleObservableValue is related to the
263 ##ProblemContext ##PossibleObservable. It means that the fault being
264 diagnosed occurred during run-time, i.e., during its normal operation".

```

```

265 constant: Yes.
266 isa: PossibleObservableValue.
267 comment: "Most of the ##Tests have as a possible result only 'Yes' or 'No'.".

268 constant: No.
269 isa: PossibleObservableValue.
270 comment: "Most of the ##Tests have as a possible result only 'Yes' or 'No'.".

271 constant: None.
272 isa: PossibleObservableValue.
273 comment: "This kind of result indicates that none of the alternative
274 results of a specific ##Test is observed.".

275 ;***** DEFINITIONS OF INSTANCES FOR $ResultType COLLECTION ****

276 constant: Normal.
277 isa: ResultType.
278 comment: "This type of result indicates that the result is normally
279 expected when the ##SubSystem related with it is working
280 properly. Such a kind of result implies that the ##SubSystem must be
281 discarded as a ##hypothesis.".

282 constant: NotNormal.
283 isa: ResultType.
284 comment: "This type of result indicates that the result is not normally
285 expected when the ##SubSystem related with it is working
286 properly. Such a kind of result implies that the fault lies in the
287 ##SubSystem which is the current ##hypothesis.".

288 constant: Insufficient.
289 isa: ResultType.
290 comment: "This type of result indicates that the result cannot
291 undoubtedly indicate either the normal function or the malfunction of
292 the ##SubSystem related with it. Such a kind of result implies that
293 further testing is necessary to decide about the functional status of
294 the ##SubSystem which is the current ##hypothesis.".

295 constant: Distinguishing.
296 isa: ResultType.
297 comment: "This type of result occurs in a situation where there are
298 two components that are probably faulty and the only way to find
299 which, is to test one of them. In this case, a result of type
300 ##Distinguishing indicates simultaneously two things. First, that the
301 ##SubSystem hypothesised as faulty is not such and second, that the faulty
302 one is the other alternative ##SubSystem.".

303 ;; ***** IMPLEMENTATION OF KADS SYSTEMATIC DIAGNOSIS PSM *****

304 ;;; The Task Structure for Systematic Diagnosis (pseudo-code) is:

305 ;;; Systematic Diagnosis(+complaint,+possible observables,-hypothesis) by
306 ;;; select1(+complaint, -system model)
307 ;;; REPEAT
308 ;;;   decompose(+system model, -hypothesis)
309 ;;;   WHILE number of hypotheses > 1
310 ;;;     select2(+possible observables, -variable value)
311 ;;;     select3(+hypothesis, -norm)
312 ;;;     compare(+variable value, +norm, -difference)
313 ;;;     system model <- current decomposition level of system model
314 ;;; UNTIL confirm(+hypothesis), i.e. system model cannot be decomposed further

315 ;;; The user interaction in CYC will be done from the SubL Interactor
316 ;;; interface, as it is not possible to get any input/output
317 ;;; interaction between the user and the SubL code from the ASK

```

```

318   ;;; interface. The whole Task Structure will be implemented as a SubL
319   ;;; function, 'systematic', which will be responsible for calling the
320   ;;; appropriate SubL functions that will implement the corresponding
321   ;;; inferences. In fact, the 'select1', 'select2', and 'select3'
322   ;;; inferences will be implemented as FORWARD rules in the
323   ;;; CYC KB. "Forward" means that, according to the results of 'Tests'
324   ;;; that the user is asked to give, these rules automatically assert
325   ;;; new facts in the KB. These facts describe which are the next
326   ;;; #PossibleObservables and Variables that must be tested ('select2'
327   ;;; inference), what should be done according to the result ('select3'
328   ;;; and 'compare' inferences), e.g., if another test for the same
329   ;;; hypothesis should be performed or if the current hypothesis should
330   ;;; be rejected or the current hypothesis must be decomposed further
331   ;;; or if the faulty component was found ('confirm' inference).

332   ;;; The following three (3) constant definitions introduce the
333   ;;; #PlausibleInference predicate and #Decompose inference type of
334   ;;; KADS. These two are used in the antecedent part of the
335   ;;; "decomposition" rules. They do not constitute control knowledge
336   ;;; but domain role knowledge. In terms of implementation, they cause
337   ;;; the forward "decomposition" rules to fire only when a Decompose
338   ;;; inference has to be made.

339   Default Mt: SystematicDiagnosisMt.

340   constant: InferenceType-KADS.
341   isa: Collection.
342   genls: PropositionalInformationThing.
343   comment: "The collection of all Inference Types of KADS methodology,
344   e.g., 'select', 'decompose', 'confirm'.".

345   constant: Decompose.
346   isa: InferenceType-KADS.
347   comment: "The #Decompose inference type of KADS takes a structured
348   hierarchy of objects and gives a less or completely unstructured
349   collection of these objects. In its simplest form it is used for
350   breaking down existing knowledge structures, like hierarchies, where
351   there is no loss of objects but only the structure is removed.".

352   constant: plausibleInference.
353   isa: UnaryPredicate.
354   arg1Isa: InferenceType-KADS.
355   comment: "The #PlausibleInference predicate is used to record in the
356   KB which inference(s) can be next performed during the 'execution' of
357   a problem solving method.".

358   constant: diagnosisContext.
359   isa: UnaryPredicate.
360   arg1Isa: PossibleObservableValue.
361   comment: "This predicate records the current problem-solving context
362   in a specific diagnosis domain, e.g., #BootTime, #StaringTime,
363   #RunTime etc. Most of the rules in each domain are
364   context-dependent. This is reflected in the appearance of the
365   #diagnosisContext predicate in their antecedent part.".

366   *****
367   *****
368   END OF systematic_diagnosisKE.txt
369   *****
370   *****

371   ;;; PROJECT: 628-Implementing Problem Solving Methods (PSMs) in Cyc
372   ;;; FILENAME: automobile_diagnosisKE.txt
373   ;;; AUTHOR: Dimitrios Sklavakis
374   ;;; PURPOSE: Contains Cyc's Knowledge Entering (KE) text defining the

```

```

375  ;;;      Automobile domain specific knowledge   for the
376  ;;;      implementation   of the Systematic Diagnosis
377  ;;;      (Localisation) PSM   from KADS methodology.

378 ;***** THE $$AutomobileDiagnosisMt MICROTHEORY *****
379
380  ;;; The domain specific knowledge for performing   Automobile fault
381  ;;; diagnosis will be entered in the $$AutomobileDiagnosisMt
382  ;;; microtheory, which is a more specific microtheory of the
383  ;;; $$SystematicDiagnosisMt microtheory:

384  ;;; (#$genlMt $$AutomobileDiagnosisMt $$SystematicDiagnosisMt).

385 constant: AutomobileDiagnosisMt.
386 isa: Microtheory.
387 genlMt: SystematicDiagnosisMt.
388 comment: "$$AutomobileDiagnosisMt is the $$Microtheory that contains
389 all the assertions about performing Automobile fault diagnosis.".

390  ;;; *****
391  ;;; * Domain specific Automobile diagnosis knowledge *
392  ;;; *****

393 Default Mt: AutomobileDiagnosisMt.

394 ; ***** THE SYSTEM MODEL *****

395 constant: AutomobileSubSystem.
396 isa: Collection.
397 genls: SubSystem.
398 comment: "The collection of all Automobile sub-systems, like the
399 $$IgnitionSystem. Each instance of $$AutomobileSubSystem may include
400 several $$AutomobileComponents and/or other
401 $$AutomobileSubSystems. Different $$AutomobileSubSystems may include
402 the same $$AutomobileComponents. In the context of
403 $$AutomobileDiagnosisMt any $$AutomobileSubSystem is an intermediate
404 level of analysis for the $$Automobile; the diagnosis continues until
405 a faulty $$AutomobileComponent is located".

406 constant: AutomobileComponent.
407 isa: Collection.
408 genls: AutomobileSubSystem Component.
409 comment: "The collection of all Automobile components such as the
410 $$SparkPlugs, $$Points, $$RotorArm. In the context of
411 $$AutomobileDiagnosisMt any $$AutomobileComponent is the lowest level
412 of analysis for the $$Automobile; the diagnosis terminates when a
413 faulty $$AutomobileComponent is located.".
414
415 constant: AutomobileSystem.
416 isa: Individual AutomobileSubSystem.
417 comment: "The $$AutomobileSystem is used to refer to the $$Automobile as
418 an $$AutomobileSubSystem. It includes the following $$AutomobileSubSystems:
419 $$IgnitionSystem etc.".

420 Constant: IgnitionSystem.
421 isa: AutomobileSubSystem.

422 Constant: HighTensionCircuit.
423 isa: AutomobileSubSystem.

424 Constant: LowTensionCircuit.
425 isa: AutomobileSubSystem.

426 Constant: Distributor.
427 isa: AutomobileSubSystem.

```

```

428 Constant: SparkPlugs.
429 isa: AutomobileComponent.

430 Constant: HighTensionWiring.
431 isa: AutomobileComponent.

432 Constant: Points.
433 isa: AutomobileComponent.

434 Constant: RotorArm.
435 isa: AutomobileComponent.

436 Constant: VacuumAdvance.
437 isa: AutomobileComponent.

438 Constant: CentrifugalWeights.
439 isa: AutomobileComponent.

440 ;***** SYSTEM TESTING KNOWLEDGE *****

441 ;***** DEFINITIONS OF INSTANCES FOR #PossibleObservable COLLECTION *****

442 Constant: FuelConsumption. ;;PossibleObservableValue: Integer
443 isa: PossibleObservable. ;;Normal value: 7.0 (miles/litre)

444 ;;HighTensionCircuit test
445 Constant: EngineMisfire. ;;PossibleObservableValue: Yes/No
446 isa: PossibleObservable. ;;Normal value: No

447 ;;LowTensionCircuit test
448 Constant: EngineStarts. ;;PossibleObservableValue: Yes/No
449 isa: PossibleObservable. ;;Normal value: Yes

450 ;;Distributor test
451 Constant: Acceleration0to60. ;;PossibleObservableValue: Integer
452 isa: PossibleObservable. ;;Normal value: 15 (SecondsDuration)

453 ;;SparkPlugs test
454 Constant: ColourOfCeramic. ;;PossibleObservableValue: WhiteColor/
455                               ;;GreyColor/BlackColor
456 isa: PossibleObservable. ;;Normal value: WhiteColor

457 ;;HighTensionWiring test
458 Constant: WiringSecurity. ;;PossibleObservableValue: Secure/Insecure
459 isa: PossibleObservable. ;;Normal value: Secure

460 ;;Points test
461 Constant: SurfaceOfComponent. ;;PossibleObservableValue: Shiny/Dull/Rusty
462 isa: PossibleObservable. ;;Normal value: Shiny

463 ;;CentrifugalWeights test
464 Constant: StrobeTestResult. ;;PossibleObservableValue: Pass/Fail
465 isa: PossibleObservable. ;;Normal value: Pass

466 ;;VacuumAdvance test
467 Constant: VacuumTestResult. ;;PossibleObservableValue: Pass/Fail
468 isa: PossibleObservable. ;;Normal value: Pass

469 Constant: WhiteColor.
470 isa: Color.
471 Constant: GrayColor.
472 isa: Color.
473 Constant: BlackColor.
474 isa: Color.

```

```

475 F: (genls Color PossibleObservableValue).

476 ;**** DEFINITIONS OF INSTANCES FOR $PossibleObservableValue COLLECTION ****

477 constant: StartingTime.
478 isa: PossibleObservableValue.
479 comment: "This #PossibleObservableValue is related to the
480 #ProblemContext #PossibleObservable. It means that the fault being
481 diagnosed occurred during the engine starting time, i.e., from the time
482 the driver turns the key on until the automobile starts moving".

483 Constant: Pass.
484 isa: PossibleObservableValue.

485 Constant: Fail.
486 isa: PossibleObservableValue.

487 Constant: Shiny.
488 isa: PossibleObservableValue.

489 Constant: Dull.
490 isa: PossibleObservableValue.

491 Constant: Rusty.
492 isa: PossibleObservableValue.

493 Constant: Secure.
494 isa: PossibleObservableValue.

495 Constant: Insecure.
496 isa: PossibleObservableValue.

497 ;; *****
498 ;; ** Rules and Facts for Decompose **
499 ;; *****

500 ;; Every PART which is a physical decomposition of the hypothesis, HYP, is a
501 ;; possible hypothesis.

502 direction: forward.
503 F: (implies
504   (and
505     (plausibleInference Decompose)
506     (hypothesis ?HYP)
507     (physicalDecompositions ?HYP ?PART))
508   (possibleHypotheses ?PART)).

509 F: (physicalDecompositions AutomobileSystem IgnitionSystem).

510 F: (physicalDecompositions IgnitionSystem HighTensionCircuit).
511 F: (physicalDecompositions IgnitionSystem LowTensionCircuit).
512 F: (physicalDecompositions IgnitionSystem Distributor).

513 F: (physicalDecompositions HighTensionCircuit SparkPlugs).
514 F: (physicalDecompositions HighTensionCircuit HighTensionWiring).

515 F: (physicalDecompositions Distributor Points).
516 F: (physicalDecompositions Distributor RotorArm).
517 F: (physicalDecompositions Distributor VacuumAdvance).
518 F: (physicalDecompositions Distributor CentrifugalWeights).

519 ;; The following assertion permits for whole CYC formulae to appear
520 ;; as PossibleObservableValues
521

```



```

522 F: (genls CycFormula PossibleObservableValue).

523 ;;; Rule to assert a (#$diagnosisContext ...) assertion. This assertion
524 ;;; is introduced as a "shorthand" for the assertion:
525 ;;;
526 ;;; (resultOfTest
527 ;;;   (TestFn AutomobileSystem ConfirmSensorially ProblemContext) ?PROBLEM)
528 ;;;
529 ;;; It is used as a premise in every rule which is applicable to the
530 ;;; corresponding diagnosis context, i.e., #$StartingTime, #$RunTime etc.

531 Direction: forward.
532 F: (implies
533     (resultOfTest
534       (TestFn AutomobileSystem ConfirmSensorially ProblemContext) ?PROBLEM)
535     (diagnosisContext ?PROBLEM)).

536 ;;; *****
537 ;;; * Testing knowledge for the AutomobileSystem *
538 ;;; *****

539 Direction: forward.

540 F: (implies
541     (hypothesis AutomobileSystem)
542     (and
543       (possibleTest (TestFn AutomobileSystem ConfirmSensorially ProblemContext))
544       (possibleResultOfTest
545         (TestFn AutomobileSystem ConfirmSensorially ProblemContext)
546           StartingTime NotNormal)
547       (possibleResultOfTest
548         (TestFn AutomobileSystem ConfirmSensorially ProblemContext)
549           RunTime NotNormal)))
550 ;;; *****
551 ;;; * Decomposition knowledge for the AutomobileSystem *
552 ;;; *****

553 Direction: forward.

554 F: (implies
555     (and
556       (diagnosisContext RunTime)
557       (hypothesis AutomobileSystem)
558       (plausibleInference Decompose))
559     (and
560       (testFirst IgnitionSystem))).

561 ;;; *****
562 ;;; * Testing knowledge for the IgnitionSystem *
563 ;;; *****

564 Direction: forward.

565 F: (implies
566     (and
567       (diagnosisContext RunTime)
568       (hypothesis IgnitionSystem))
569     (and
570       (possibleTest (TestFn IgnitionSystem CheckIndependently FuelConsumption))
571       (possibleResultOfTest
572         (TestFn IgnitionSystem CheckIndependently FuelConsumption)
573         (and (greaterThanOrEqualTo ?X 5) (greaterThanOrEqualTo 8 ?X)) Normal)
574       (possibleResultOfTest
575         (TestFn IgnitionSystem CheckIndependently FuelConsumption)

```

```

576                                     (greaterThan ?X 8) NotNormal)
577   (possibleResultOfTest
578     (TestFn IgnitionSystem CheckIndependently FuelConsumption)
579     (greaterThan 5 ?X) NotNormal))).

580 ;; *****
581 ;; * Decomposition knowledge for the IgnitionSystem *
582 ;; *****

583 Direction: forward.

584 F: (implies
585   (and
586     (diagnosisContext RunTime)
587     (hypothesis IgnitionSystem)
588     (plausibleInference Decompose))
589   (and
590     (testFirst HighTensionCircuit)
591     (testAfter HighTensionCircuit LowTensionCircuit)
592     (testAfter LowTensionCircuit Distributor))).

593 ;; *****
594 ;; * Testing knowledge for the HighTensionCircuit *
595 ;; *****

596 Direction: forward.

597 F: (implies
598   (and
599     (diagnosisContext RunTime)
600     (hypothesis HighTensionCircuit))
601   (and
602     (possibleTest
603       (TestFn HighTensionCircuit ConfirmSensorially EngineMisfire))
604     (possibleResultOfTest
605       (TestFn HighTensionCircuit ConfirmSensorially EngineMisfire) No Normal)
606     (possibleResultOfTest
607       (TestFn HighTensionCircuit ConfirmSensorially EngineMisfire)
608         Yes NotNormal))).

609 ;; *****
610 ;; * Decomposition knowledge for the HighTensionCircuit *
611 ;; *****

612 Direction: forward.

613 F: (implies
614   (and
615     (diagnosisContext RunTime)
616     (hypothesis HighTensionCircuit)
617     (plausibleInference Decompose))
618   (and
619     (testFirst SparkPlugs)
620     (testAfter HighTensionWiring))).

621 ;; *****
622 ;; * Testing knowledge for the LowTensionCircuit *
623 ;; *****

624 Direction: forward.

625 F: (implies
626   (and
627     (diagnosisContext RunTime)
628     (hypothesis LowTensionCircuit))
629   (and

```

```

630     (possibleTest (TestFn LowTensionCircuit ConfirmSensorially EngineStarts))
631     (possibleResultOfTest
632     (TestFn LowTensionCircuit ConfirmSensorially EngineStarts) Yes Normal)
633     (possibleResultOfTest
634     (TestFn LowTensionCircuit ConfirmSensorially EngineStarts)
635     (greaterThan 15 ?X) Normal)
636     (possibleResultOfTest
637     (TestFn LowTensionCircuit ConfirmSensorially EngineStarts)
638     (greaterThanOrEqualTo ?X 15) NotNormal))).

636   ;;; *****
637   ;;; * Testing knowledge for the Distributor *
638   ;;; *****

639   Direction: forward.

640   F: (implies
641     (and
642     (diagnosisContext RunTime)
643     (hypothesis Distributor))
644     (and
645     (possibleTest (TestFn Distributor CheckIndependently Acceleration0to60))
646     (possibleResultOfTest
647     (TestFn Distributor CheckIndependently Acceleration0to60)
648     (greaterThan 15 ?X) Normal)
649     (possibleResultOfTest
650     (TestFn Distributor CheckIndependently Acceleration0to60)
651     (greaterThanOrEqualTo ?X 15) NotNormal))).

652   ;;; *****
653   ;;; * Decomposition knowledge for the Distributor *
654   ;;; *****

655   Direction: forward.

656   F: (implies
657     (and
658     (diagnosisContext RunTime)
659     (hypothesis Distributor)
660     (plausibleInference Decompose))
661     (and
662     (testFirst Points)
663     (testAfter Points VacuumAdvance)
664     (testAfter VacuumAdvance CentrifugalWeights)
665     (testAfter CentrifugalWeights RotorArm))).

666   ;;; *****
667   ;;; * Testing knowledge for the SparkPlugs *
668   ;;; *****

669   Direction: forward.

670   F: (implies
671     (and
672     (diagnosisContext RunTime)
673     (hypothesis SparkPlugs))
674     (and
675     (possibleTest (TestFn SparkPlugs ConfirmSensorially ColourOfCeramic))
676     (possibleResultOfTest
677     (TestFn SparkPlugs ConfirmSensorially ColourOfCeramic) WhiteColor Normal)
678     (possibleResultOfTest
679     (TestFn SparkPlugs ConfirmSensorially ColourOfCeramic)
680     (greaterThan 15 ?X) GrayColor NotNormal)
681     (possibleResultOfTest
682     (TestFn SparkPlugs ConfirmSensorially ColourOfCeramic)
683     (greaterThanOrEqualTo ?X 15) BlackColor NotNormal))).

684   ;;; *****

```

```

685 ;; * Testing knowledge for the HighTensionWiring *
686 ;; *****

687 Direction: forward.

688 F: (implies
689   (and
690     (diagnosisContext RunTime)
691     (hypothesis HighTensionWiring))
692   (and
693     (possibleTest
694       (TestFn HighTensionWiring ConfirmSensorially WiringSecurity))
695     (possibleResultOfTest
696       (TestFn HighTensionWiring ConfirmSensorially WiringSecurity)
697         Secure Normal)
698     (possibleResultOfTest
699       (TestFn HighTensionWiring ConfirmSensorially WiringSecurity)
700         Insecure NotNormal))).
701 ;; *****
702 ;; * Testing knowledge for the Points *
703 ;; *****

704 Direction: forward.

705 F: (implies
706   (and
707     (diagnosisContext RunTime)
708     (hypothesis Points))
709   (and
710     (possibleTest (TestFn Points ConfirmSensorially SurfaceOfComponent))
711     (possibleResultOfTest
712       (TestFn Points ConfirmSensorially SurfaceOfComponent) Shiny Normal)
713     (possibleResultOfTest
714       (TestFn Points ConfirmSensorially SurfaceOfComponent) Dull NotNormal)
715     (possibleResultOfTest
716       (TestFn Points ConfirmSensorially SurfaceOfComponent) Rusty NotNormal))).

717 ;; *****
718 ;; * Testing knowledge for the CentrifugalWeights *
719 ;; *****

720 Direction: forward.

721 F: (implies
722   (and
723     (diagnosisContext RunTime)
724     (hypothesis CentrifugalWeights))
725   (and
726     (possibleTest
727       (TestFn CentrifugalWeights ConfirmSensorially StrobeTestResult))
728     (possibleResultOfTest
729       (TestFn CentrifugalWeights ConfirmSensorially StrobeTestResult)
730         Pass Normal)
731     (possibleResultOfTest
732       (TestFn CentrifugalWeights ConfirmSensorially StrobeTestResult)
733         Fail NotNormal))).

734 ;; *****
735 ;; * Testing knowledge for the VacuumAdvance *
736 ;; *****

737 Direction: forward.

738 F: (implies
739   (and
740     (diagnosisContext RunTime)

```

```

741     (hypothesis VacuumAdvance))
742     (and
743     (possibleTest (TestFn VacuumAdvance ConfirmSensorially VacuumTestResult))
744     (possibleResultOfTest
745     (TestFn VacuumAdvance ConfirmSensorially VacuumTestResult) Pass Normal)
746     (possibleResultOfTest
747     (TestFn VacuumAdvance ConfirmSensorially VacuumTestResult)
748     Fail NotNormal))).

749 *****
750 *****
751     END OF automobile_diagnosisKE.txt
752 *****
753 *****

754 ;;; PROJECT: 628-Implementing Problem Solving Methods (PSMs) in Cyc
755 ;;; FILENAME: pc_diagnosisKE.txt
756 ;;; AUTHOR: Dimitrios Sklavakis
757 ;;; PURPOSE: Contains Cyc's Knowledge Entering (KE) text defining the
758 ;;;          PC domain specific knowledge for the implementation of
759 ;;;          the Systematic Diagnosis (Localisation) PSM from KADS
760 ;;;          methodology.

761 ;***** THE #PCDiagnosisMt MICROTHEORY *****
762
763 ;;; The whole knowledge for performing PC fault diagnosis will be
764 ;;; entered in the #PCDiagnosisMt microtheory, which is a more specific
765 ;;; microtheory of the #SystematicDiagnosisMt microtheory:

766 ;;; (#genMt #PCDiagnosisMt #SystematicDiagnosisMt).

767 constant: PCDiagnosisMt.
768 isa: Microtheory.
769 genMt: SystematicDiagnosisMt.
770 comment: "#PCDiagnosisMt is the #Microtheory that contains all the assertions
771 about performing Personal Computer(PC) fault diagnosis.".

772 ;;; *****
773 ;;; * Domain specific PC diagnosis knowledge *
774 ;;; *****

775 Default Mt: PCDiagnosisMt.

776 ; ***** THE SYSTEM MODEL *****

777 constant: PCSubSystem.
778 isa: Collection.
779 genls: SubSystem.
780 comment: "The collection of all PC sub-systems, like the
781 #VideoSystem, #PowerSystem, #KeyboardSystem. Each instance of
782 #PCSubSystem may include several #PCComponents and/or other
783 #PCSubSystems. Different #PCSubSystems may include the same
784 #PCComponents. In the context of #PCDiagnosisMt any #PCSubSystem is
785 an intermediate level of analysis for the #PersonalComputer; the
786 diagnosis continues until a faulty #PCComponent is located".

787 constant: PCComponent.
788 isa: Collection.
789 genls: PCSubSystem Component.
790 comment: "The collection of all PC components such as the
791 #PowerSupply, #VideoCard, #FloppyDiskDrive. In the context of
792 #PCDiagnosisMt any #PCComponent is the lowest level of analysis for
793 the #PersonalComputer; the diagnosis terminates when a faulty

```

```
794 #PCComponent is located.".
795
796 constant: PCSystem.
797 isa: Individual PCSubSystem.
798 comment: "The #PCSystem is used to refer to the #PersonalComputer as
799 a #PCSubSystem. It includes the following #PCComponents:
800 #PowerSystem, #VideoSystem, e.t.c.".

801 constant: PowerSystem.
802 isa: Individual PCSubSystem.
803 comment: "The power #PCSubSystem. Includes the following
804 #PCComponents: #PowerSocket, #PowerCable, #PowerProtectionDevice
805 (optionally) and #PowerSupply.".

806 constant: PowerSocket.
807 isa: Individual PCComponent.
808 comment: "#PowerSocket is the socket that provides electric power to
809 the PC. It is a component of the #PowerSystem.".

810 constant: PowerCable.
811 isa: Individual PCComponent.
812 comment: "#PowerCable is the cable that connects the #PowerSocket
813 with the #PowerSupply. It is a component of the #PowerSystem.".

814 constant: PowerProtectionDevice.
815 isa: Individual PCComponent.
816 comment: "#PowerProtectionDevice is any device (suprocessor, UPS)
817 connected between the #PowerSocket and the #PowerSupply to protect
818 the #PersonalComputer from power failures. It is an optional
819 component of the #PowerSystem.".

820 constant: PowerSupply.
821 isa: Individual PCComponent.
822 comment: "#PowerSupply is the component located inside the
823 #PersonalComputer case that supplies the #MotherBoard with electriv
824 power. It is a component of the #PowerSystem.".

825 constant: VideoSystem.
826 isa: Individual PCSubSystem.
827 comment: "The video #PCSubSystem. Includes the following
828 #PCComponents: #MotherBoard, #VideoCard, #Monitor (and the
829 #Speaker)".

830 constant: VideoCard.
831 isa: Individual PCComponent.
832 comment: "The #VideoCard trasforms the video information to video
833 signal and sends it to the #Monitor. It is a component of the
834 #VideoSystem.".

835 constant: Monitor.
836 isa: Individual PCComponent.
837 comment: "The #Monitor trasforms the video signal sent by the
838 #VideoCard into visual image. It is a component of the
839 #VideoSystem.".

840 constant: MotherBoard.
841 isa: Individual PCComponent.
842 comment: "The #MotherBoard is the main #PCComponent. Most of the
843 rest #PCComponents are connected onto the #MotherBoard and
844 controlled by it. It is actually a sub-system by itself as it includes
845 other components but in the context of PC diagnosis it will be
846 regarded as a #PCComponent to avoid increasing the complexity of the
847 #PersonalComputer analysis.".

848 constant: Speaker.
849 isa: Individual PCComponent.
```

850 comment: "The PC speaker. It refers to the  
851 internal speaker that is connected on the motherboard and not the  
852 external ones that are part of a multi-media system and require a  
853 sound-card on which they are connected."

854 constant: BIOSStartupSystem.  
855 isa: Individual PCSubSystem.  
856 comment: "The BIOS startup ##PCSubSystem. Includes the following  
857 ##PCComponents: ##MotherBoard, ##VideoCard."

858 constant: BIOSsettings.  
859 isa: Individual PCComponent.  
860 comment: "The Basic Input Output System (BIOS) settings record the  
861 system parameters for all its operations. Wrong BIOS settings can be  
862 responsible for a PC malfunction, e.g., the ##FloppyDiskDrive being  
863 disabled and therefore being non-existent to the system."

864 constant: MemorySystem.  
865 isa: Individual PCSubSystem.  
866 comment: "The memory ##PCSubSystem. Includes the following  
867 ##PCComponents: ##RAM, ##MotherBoard."

868 constant: RAM.  
869 isa: Individual PCComponent.  
870 comment: "The Random Access Memory ##PCComponent. Usually, it is a  
871 set of Single In-Line Memory Modules (SIMMs), plugged in special  
872 positions on the ##MotherBoard."

873 constant: FloppySystem.  
874 isa: Individual PCSubSystem.  
875 comment: "The ##FloppySystem consists of the ##FloppyDiskDrive and the  
876 ##BIOSsettings for the enabling/disabling of the ##FloppyDiskDrive."

877 constant: FloppyDiskDrive.  
878 isa: Individual PCComponent.  
879 comment: "The PC storage device which drives a removable floppy disk to  
880 store/retrieve information."

881 constant: HardDiskDrive.  
882 isa: Individual PCComponent.  
883 comment: "The PC main mass storage device. Usually it is fixed and  
884 non-removable."

885 constant: CDRomdrive.  
886 isa: Individual PCComponent.  
887 comment: ""

888 constant: PlugAndPlaySystem.  
889 isa: Individual PCSubSystem.  
890 comment: "This system comprises all peripherals that are connected to  
891 the PC via expansion cards and they are (usually) automatically  
892 recognised by MS Windows without any extra software drivers or  
893 configuration procedures. However, sometimes there may be some  
894 problems with their recognition. This ##PCSubSystem may have as  
895 functional parts a wide variety of peripherals. In the current  
896 implementation of Systematic diagnosis, it is regarded as consisting  
897 of peripherals and their ##ExpansionCards. There isn't any further  
898 decomposition into the specific peripherals as these vary in each  
899 configuration."

900 constant: ExpansionCard.  
901 isa: Individual PCComponent.  
902 comment: "This PC component is used in conjunction with various  
903 peripherals. In the current implementation, it is regarded as a  
904 specific ##PCComponent, although in a specific PC configuration there  
905 could be none, one or more expansion cards. Please, refer to the

```

906 ##PlugAndPlaySystem collection for more information.".
907 constant: PlugAndPlaySystem.
908 isa: Individual PCSubSystem.
909 comment: "It is the system responsible for loading the operating
910 system. In general, it comprises the ##FloppyDiskDrive with a floppy
911 disk containing the operating system (##OSfloppyDisk) and the
912 ##HardDiskDrive, although a PC can be configured via the
913 ##BIOSsettings to use only one of them or both.".

914 constant: OSfloppyDisk.
915 isa: Individual PCComponent.
916 comment: "It is the floppy disk containing the operating system. It is
917 used by the ##BootSystem to load the OS from the ##FloppyDiskDrive.".

918 constant: BootSystem.
919 isa: Individual PCSubSystem.
920 comment: "It is the system responsible for loading the operating
921 system (OS). It includes the ##FloppyDiskDrive together with the floppy
922 disk containing the OS (##OSfloppyDisk) and the ##HardDiskDrive. It
923 also includes the ##BootSequence-BIOSsetting which defines the sequence
924 in which these media will be used by the BIOS to load the OS.".

925 constant: functionalPartOf.
926 isa : TransitiveBinaryPredicate.
927 arg1Isa: PCSubSystem.
928 arg2Isa: PCSubSystem.
929 comment : "Predicate functionalPartOf is used to define a functional
930 model of the PC under diagnosis
931 functionalPartOf(WholeSubSystem PartialSubSystem) means
932 that the PCSubSystem WholeSubSystem is using the function of
933 PartialSubsystem to perform its own function. E.g.,
934 functionalPartOf(PowerSystem PowerSupply) means that for the
935 PowerSystem to function the PowerSupply must function. This
936 predicate is used to systematically disassemble the PC system into
937 simpler PCSubSystems until a PCComponent is reached that it is
938 faulty.".

939 ;***** DEFINITIONS OF INSTANCES FOR ##TestAction COLLECTION *****

940 constant: ChangeVoltage.
941 isa: TestAction.
942 comment: "This ##TestAction means that the user must change the
943 voltage setting in the ##PowerSupply. It is a specific ##TestAction
944 related only to the ##PowerSupply".

945 ;***** DEFINITIONS OF INSTANCES FOR ##PossibleObservable COLLECTION *****

946 constant: ElectricPower.
947 isa: PossibleObservable.
948 comment: "The electric power that any ##PCSystem needs to operate.".

949 constant: VoltageCorrect.
950 isa: PossibleObservable.
951 comment: "This ##PossibleObservable refers to the voltage setting in
952 the ##PowerSupply being correct, i.e., 110V or 220V.".

953 constant: VideoSignal.
954 isa: PossibleObservable.
955 comment: "This ##PossibleObservable refers to the existence of any
956 video signal on the ##Monitor screen.".

957 constant: SpeakerBeep.
958 isa: PossibleObservable.
959 comment: "This ##PossibleObservable refers to any beep pattern coming
960 out of the ##Speaker.".

```



961 constant: VideoBIOSMessage.  
962 isa: PossibleObservable.  
963 comment: "This ##PossibleObservable refers to the display of the  
964 video BIOS message."

965 constant: BootContinues.  
966 isa: PossibleObservable.  
967 comment: "This ##PossibleObservable refers to the booting process  
968 cointinuing normally."

969 constant: StartupScreen.  
970 isa: PossibleObservable.  
971 comment: "This ##PossibleObservable refers to the display of the BIOS  
972 stratup screen."

973 constant: MemoryTest.  
974 isa: PossibleObservable.  
975 comment: "This ##PossibleObservable refers to the memeory test  
976 performed by the BIOS during boot-time."

977 constant: ErrorMessage.  
978 isa: PossibleObservable.  
979 comment: "This ##PossibleObservable refers to the display of an error  
980 message on the screen."

981 constant: ComponentProblem.  
982 isa: PossibleObservable.  
983 comment: "This ##PossibleObservable refers to the occasion where a  
984 specific ##PCComponent has reached which is possibly faulty and the  
985 only way to decide about this involves elaborate and complex ##Tests,  
986 which the current implementation of the Systematic diagnosis problem  
987 solving method does not cover. Therefore, the user has to perform  
988 these ##Tests either based on his knowledge or have a human expert  
989 perform them."

990 constant: AutoDetection-BIOSsetting.  
991 isa: PossibleObservable.  
992 comment: "This ##PossibleObservable refers to the ##HardDiskDrive  
993 auto-detection setting in the PC BIOS. It may be set to ##Auto for  
994 automatic detection or to ##Manual, usually the first one."

995 constant: BootSequence-BIOSsetting.  
996 isa: PossibleObservable.  
997 comment: "This ##PossibleObservable refers to the BIOS setting which  
998 controls the sequence of the media used to load the operating  
999 system. It may be A:-C: for using first the ##FloppyDiskDrive and then  
1000 the ##HardDiskDrive or C:-A: for the reverse."

1001 constant: BootSource.  
1002 isa: PossibleObservable.  
1003 comment: "This ##PossibleObservable refers to the actual medium from  
1004 which the operating system is loaded, independtly from what the  
1005 ##BootSequence-BIOSsetting is."

1006 constant: FloppyAccess.  
1007 isa: PossibleObservable.  
1008 comment: "This ##PossibleObservable refers to whether the  
1009 ##FloppyDiskDrive is actually accessed by the BIOS during boot-time  
1010 system test."

1011 constant: DetectionMessage.  
1012 isa: PossibleObservable.  
1013 comment: "This ##PossibleObservable is related to BIOS messages  
1014 concerning the autodetection of the ##HardDiskDrives."

```
1015 constant: InFloppy.
1016 isa: PossibleObservable.
1017 comment: "This ##PossibleObservable is related to the ##OSfloppyDisk
1018 being inside the ##FloppyDiskDrive.".

1019 ;***** DEFINITIONS OF INSTANCES FOR $PossibleObservableValue COLLECTION ****

1020 constant: BootTime.
1021 isa: PossibleObservableValue.
1022 comment: "This ##PossibleObservableValue is related to the
1023 ##ProblemContext ##PossibleObservable. It means that the fault being
1024 diagnosed occurred during boot-time, i.e., from the time the power is
1025 turned on until the Operating System starts being loaded.".

1026 constant: ComponentSpecific.
1027 isa: PossibleObservableValue.
1028 comment: "This ##PossibleObservableValue is related to the
1029 ##ProblemContext ##PossibleObservable. It means that the fault being
1030 diagnosed is identified to be related with a specific ##PCComponent,
1031 e.g., the ##Monitor, ##MotherBoard, ##HardDisk e.t.c.".

1032 constant: SingleBeep.
1033 isa: PossibleObservableValue.
1034 comment: "This ##PossibleObservableValue is related to the
1035 ##SpeakerBeep ##PossibleObservable. It means that the ##Speaker
1036 produced a single beep".

1037 constant: RingingOrBuzzing.
1038 isa: PossibleObservableValue.
1039 comment: "This ##PossibleObservableValue is related to the
1040 ##SpeakerBeep ##PossibleObservable. It means that the ##Speaker is
1041 producing a ringing or buzzing sound.".

1042 constant: ConsistentPattern.
1043 isa: PossibleObservableValue.
1044 comment: "This ##PossibleObservableValue is related to the
1045 ##SpeakerBeep ##PossibleObservable. It means that the ##Speaker is
1046 producing a consistent pattern (code) of beeps, e.g., one beep, then
1047 two more.".

1048 constant: Complete.
1049 isa: PossibleObservableValue.
1050 comment: "This ##PossibleObservableValue is related to some tests
1051 performed by the BIOS, e.g., the memory test. It means that the
1052 corresponding test is successfully completed.".

1053 constant: InComplete.
1054 isa: PossibleObservableValue.
1055 comment: "This ##PossibleObservableValue is related to some tests
1056 performed by the BIOS, e.g., the memory test. It means that the
1057 corresponding test is not successfully completed.".

1058 constant: CannotFind-Message.
1059 isa: PossibleObservableValue.
1060 comment: "This ##PossibleObservableValue is related to BIOS error
1061 messages concerning the autodetection of IDE/ATAPI devices,
1062 e.g. ##HardDiskDrive, ##CDROMdrive. This kind of messages indicate
1063 that the BIOS cannot detect the corresponding device.".

1064 constant: Auto.
1065 isa: PossibleObservableValue.
1066 comment: "This ##PossibleObservableValue indicates that some
1067 action/process/procedure is (set to be) done automatically.".

1068 constant: Manual.
1069 isa: PossibleObservableValue.
```

```

1070 comment: "This ##PossibleObservableValue indicates that some
1071 action/process/procedure is (set to be) done manually.".

1072 constant: FloppyThenHard.
1073 isa: PossibleObservableValue.
1074 comment: "This ##PossibleObservableValue is related to the
1075 ##BootSequence-BIOSsetting ##PossibleObservable. It indicates that
1076 this setting is set to A:-C:.".

1077 constant: HardThenFloppy.
1078 isa: PossibleObservableValue.
1079 comment: "This ##PossibleObservableValue is related to the
1080 ##BootSequence-BIOSsetting ##PossibleObservable. It indicates that
1081 this setting is set to C:-A:.".

1082 ;; ***** IMPLEMENTATION OF KADS SYSTEMATIC DIAGNOSIS PSM *****

1083 ;; The Task Structure for Systematic Diagnosis (pseudo-code) is:

1084 ;; Systematic Diagnosis(+complaint,+possible observables,-hypothesis) by
1085 ;; select1(+complaint, -system model)
1086 ;; REPEAT
1087 ;; decompose(+system model, -hypothesis)
1088 ;; WHILE number of hypotheses > 1
1089 ;; select2(+possible observables, -variable value)
1090 ;; select3(+hypothesis, -norm)
1091 ;; compare(+variable value, +norm, -difference)
1092 ;; system model <- current decomposition level of system model
1093 ;; UNTIL confirm(+hypothesis), i.e. system model cannot be decomposed further

1094 ;; The user interaction in CYC will be done from the SubL Interactor
1095 ;; interface, as it is not possible to get any input/output
1096 ;; interaction between the user and the SubL code from the ASK
1097 ;; interface. The whole Task Structure will be implemented as a SubL
1098 ;; function, 'systematic', which will be responsible for calling the
1099 ;; appropriate SubL functions that will implement the corresponding
1100 ;; inferences. In fact, the 'select1', 'select2', and 'select3'
1101 ;; inferences will be implemented as FORWARD rules in the
1102 ;; CYC KB. "Forward" means that, according to the results of 'Tests'
1103 ;; that the user is asked to give, these rules automatically assert
1104 ;; new facts in the KB. These facts describe which are the next
1105 ;; ##PossibleObservables and Variables that must be tested ('select2'
1106 ;; inference), what should be done according to the result ('select3'
1107 ;; and 'compare' inferences), e.g., if another test for the same
1108 ;; hypothesis should be performed or if the current hypothesis should
1109 ;; be rejected or the current hypothesis must be decomposed further
1110 ;; or if the faulty component was found ('confirm' inference).

1111 ;; During the Systematic Diagnosis problem solving method (PSM) as
1112 ;; well as during any other PSM, there are certain decisions/choices
1113 ;; that must be done. According to the structure of the PSMs as
1114 ;; Generic Task Models in KADS, these decisions/choices occur during
1115 ;; the performance of specific Inferences. The implementation of
1116 ;; these decisions/choices has to be declarative since this is the
1117 ;; main principle in CYC. Therefore, the implementation of them will
1118 ;; be in terms of FORWARD rules: the ANTECEDENT of each rule will be
1119 ;; the conditions under which a decision is made and the CONSEQUENT
1120 ;; will be the knowledge that is becoming known to the system when
1121 ;; this decision is made. Then, the newly added information will be
1122 ;; used by the SubL code to guide the whole procedure. In the
1123 ;; following, we will examine in detail which these decisions/choices
1124 ;; are, when and where do they occur and how they are actually
1125 ;; implemented as forward rules.

1126 ;; The PSM starts with a SELECT inference. A general symptom is
1127 ;; entered by the user and an appropriate system model is

```

```

1128   ;;; chosen. This inference is slightly changed for PC diagnosis. What
1129   ;;; is actually asked from the user is to distinguish three (3) major
1130   ;;; contexts of diagnosis: (i) Boot-time troubleshooting, (ii)
1131   ;;; Run-time troubleshooting and (iii) Component-specific
1132   ;;; troubleshooting. This categorisation is significant since
1133   ;;; completely different rules are applicable in each
1134   ;;; context. Although this contextual dependency of the rules could be
1135   ;;; implemented as different #Microtheory contexts, this would make
1136   ;;; context shifting more complicated - any ASK operation would have
1137   ;;; to define the #Microtheory. Instead, this dependency will be
1138   ;;; embedded in the antecedent part of the rules as an extra
1139   ;;; condition. The special predicate diagnosisContext (see
1140   ;;; #SystematicDiagnosis microtheory) will be used, e.g.,
1141   ;;;
1142   ;;; (implies
1143   ;;; (and
1144   ;;; (diagnosisContext BootTime)
1145   ;;; (...<more conditions>...)) ;end of antecedent
1146   ;;; (<consequent>))

1147   ;;; Rule to assert a (#diagnosisContext ...) assertion. This assertion
1148   ;;; is introduced as a "shorthand" for the assertion:
1149   ;;;
1150   ;;; (resultOfTest (TestFn PCSystem ConfirmSensorially ProblemContext) ?PROBLEM)
1151   ;;;
1152   ;;; It is used as a premise in every rule which is applicable to the
1153   ;;; corresponding diagnosis context, i.e., #BootTime, #RunTime or
1154   ;;; #ComponentSecific.

1155   Direction: forward.
1156   F: (implies
1157       (resultOfTest (TestFn PCSystem ConfirmSensorially ProblemContext) ?PROBLEM)
1158       (diagnosisContext ?PROBLEM)).

1159   ;;; The 'decompose' inference in KADS Systematic Diagnosis PSM takes as input
1160   ;;; the current #PCSubSystem (#hypothesis PC_SUBSYSTEM) and decomposes it
1161   ;;; into its functional subsystems (#functionalPartOf PC_SUBSYSTEM PART),
1162   ;;; generating new hypotheses (#possibleHypotheses PART).

1163   ;;; *****
1164   ;;; ** Rules and Facts for Decompose **
1165   ;;; *****

1166   ;; Every PART which is a functional part of the hypothesis, HYP, is a
1167   ;; possible hypothesis.

1168   direction: forward.
1169   F: (implies
1170       (and
1171         (diagnosisContext BootTime)
1172         (plausibleInference Decompose)
1173         (hypothesis ?HYP)
1174         (functionalPartOf ?HYP ?PART))
1175       (possibleHypotheses ?PART)).

1176   F: (functionalPartOf PCSystem PowerSystem).
1177   F: (functionalPartOf PCSystem VideoSystem).
1178   F: (functionalPartOf PCSystem BIOSStartupSystem).
1179   F: (functionalPartOf PCSystem MemorySystem).
1180   F: (functionalPartOf PCSystem FloppySystem).
1181   F: (functionalPartOf PCSystem HardDiskDrive).
1182   F: (functionalPartOf PCSystem CDROMdrive).

```

```

1183 F: (functionalPartOf PCSystem PlugAndPlaySystem).
1184 F: (functionalPartOf PCSystem BootSystem).

1185 F: (functionalPartOf PowerSystem PowerSocket).
1186 F: (functionalPartOf PowerSystem PowerCable).
1187 F: (functionalPartOf PowerSystem PowerProtectionDevice).
1188 F: (functionalPartOf PowerSystem PowerSupply).

1189 F: (functionalPartOf VideoSystem MotherBoard).
1190 F: (functionalPartOf VideoSystem VideoCard).
1191 F: (functionalPartOf VideoSystem Monitor).

1192 F: (functionalPartOf BIOSStartupSystem MotherBoard).
1193 F: (functionalPartOf BIOSStartupSystem VideoCard).

1194 F: (functionalPartOf MemorySystem RAM).
1195 F: (functionalPartOf MemorySystem MotherBoard).

1196 F: (functionalPartOf FloppySystem FloppyDiskDrive).
1197 F: (functionalPartOf FloppySystem BIOSsettings).

1198 F: (functionalPartOf PlugAndPlaySystem ExpansionCard).
1199 F: (functionalPartOf PlugAndPlaySystem MotherBoard).

1200 F: (functionalPartOf BootSystem FloppyDiskDrive).
1201 F: (functionalPartOf BootSystem OSfloppyDisk).
1202 F: (functionalPartOf BootSystem HardDiskDrive).

1203 ;; *****
1204 ;; ** **
1205 ;; ** Facts and rules about PC subsystems and components **
1206 ;; ** **
1207 ;; *****

1208 ;; *****
1209 ;; ** TEST(S) for the PCSystem **
1210 ;; *****

1211 Default Mt: PCDiagnosisMt.

1212 Direction: forward.
1213 F: (implies
1214   (hypothesis PCSystem) ;if diagnosis just started, ask for the context
1215   (and
1216     (possibleTest (TestFn PCSystem ConfirmSensorially ProblemContext))
1217     (possibleResultOfTest
1218       (TestFn PCSystem ConfirmSensorially ProblemContext) BootTime NotNormal)
1219     (possibleResultOfTest
1220       (TestFn PCSystem ConfirmSensorially ProblemContext) RunTime NotNormal)
1221     (possibleResultOfTest
1222       (TestFn PCSystem ConfirmSensorially ProblemContext)
1223         ComponentSpecific NotNormal))).

1224 ;; *****
1225 ;; ** DECOMPOSITION knowledge for the PCSystem **
1226 ;; *****

1227 Direction: forward.
1228 F: (implies
1229   (and
1230     (diagnosisContext BootTime)

```

```

1231     (hypothesis PCSystem)
1232     (plausibleInference Decompose))
1233     (and
1234     (testFirst PowerSystem)
1235     (testAfter PowerSystem VideoSystem)
1236     (testAfter VideoSystem BIOSStartupSystem)
1237     (testAfter BIOSStartupSystem MemorySystem)
1238     (testAfter MemorySystem FloppySystem)
1239     (testAfter FloppySystem HardDiskDrive)
1240     (testAfter HardDiskDrive CDROMdrive)
1241     (testAfter CDROMdrive PlugAndPlaySystem)
1242     (testAfter PlugAndPlaySystem BootSystem))).

1243 ;; *****
1244 ;; ** TEST(S) for the PowerSystem **
1245 ;; *****

1246 Direction: forward.
1247 F: (implies
1248     (and
1249     (diagnosisContext BootTime)
1250     (hypothesis PowerSystem)) ;
1251     (and
1252     (possibleTest (TestFn PowerSystem ConfirmSensorially ElectricPower))
1253     (possibleResultOfTest
1254     (TestFn PowerSystem ConfirmSensorially ElectricPower) Yes Normal)
1255     (possibleResultOfTest
1256     (TestFn PowerSystem ConfirmSensorially ElectricPower) No NotNormal))).

1257 ;; *****
1258 ;; ** DECOMPOSITION knowledge for the PowerSystem **
1259 ;; ** ElectricPower=No **
1260 ;; *****

1261 Direction: forward.
1262 F: (implies
1263     (and
1264     (diagnosisContext BootTime)
1265     (hypothesis PowerSystem)
1266     (resultOfTest (TestFn PowerSystem ConfirmSensorially ElectricPower) No)
1267     (plausibleInference Decompose))
1268     (and
1269     (testFirst PowerSocket)
1270     (testAfter PowerSocket PowerProtectionDevice)
1271     (testAfter PowerProtectionDevice PowerCable)
1272     (testAfter PowerCable PowerSupply))).

1273 ;; *****
1274 ;; ** TEST(S) for the PowerSocket **
1275 ;; *****

1276 Direction: forward.
1277 F: (implies
1278     (and
1279     (diagnosisContext BootTime)
1280     (hypothesis PowerSocket)) ;
1281     (and
1282     (possibleTest (TestFn PowerSocket CheckIndependently ElectricPower))
1283     (possibleResultOfTest
1284     (TestFn PowerSocket CheckIndependently ElectricPower) Yes Normal)
1285     (possibleResultOfTest
1286     (TestFn PowerSocket CheckIndependently ElectricPower) No NotNormal))).

1287 ;;*****

```

```

1288 ;; ** TEST(S) for the PowerProtectionDevice **
1289 ;*****

1290 Direction: forward.
1291 F: (implies
1292   (and
1293    (diagnosisContext BootTime)
1294    (hypothesis PowerProtectionDevice))
1295   (and
1296    (possibleTest (TestFn PowerProtectionDevice Remove ElectricPower))
1297    (possibleResultOfTest
1298     (TestFn PowerProtectionDevice Remove ElectricPower) Yes NotNormal)
1299    (possibleResultOfTest
1300     (TestFn PowerProtectionDevice Remove ElectricPower) No Normal))).

1301 ;*****
1302 ;; ** TEST(S) for the PowerCable **
1303 ;*****

1304 Direction: forward.
1305 F: (implies
1306   (and
1307    (diagnosisContext BootTime)
1308    (hypothesis PowerCable)) ;
1309   (and
1310    (possibleTest (TestFn PowerCable Replace ElectricPower))
1311    (possibleResultOfTest
1312     (TestFn PowerCable Replace ElectricPower) Yes NotNormal)
1313    (possibleResultOfTest
1314     (TestFn PowerCable Replace ElectricPower) No Normal))).

1315 ;*****
1316 ;;** TEST(S) for the PowerSupply **
1317 ;*****

1318 Direction: forward.
1319 F: (implies
1320   (and
1321    (diagnosisContext BootTime)
1322    (hypothesis PowerSupply))
1323   (and
1324    (possibleTest (TestFn PowerSupply ConfirmSensorially VoltageCorrect))
1325    (possibleResultOfTest
1326     (TestFn PowerSupply ConfirmSensorially VoltageCorrect) Yes NotNormal)
1327    (possibleResultOfTest
1328     (TestFn PowerSupply ConfirmSensorially VoltageCorrect) No Insufficient))).

1329 ;*****
1330 ;;** TEST(S) for the PowerSupply **
1331 ;;** VoltageCorrect=No **
1332 ;*****
1333
1334 Direction: forward.
1335 F: (implies
1336   (and
1337    (diagnosisContext BootTime)
1338    (hypothesis PowerSupply)
1339    (resultOfTest (TestFn PowerSupply ConfirmSensorially VoltageCorrect) No)) ;
1340   (and
1341    (possibleTest (TestFn PowerSupply ChangeVoltage ElectricPower))
1342    (possibleResultOfTest
1343     (TestFn PowerSupply ChangeVoltage ElectricPower) Yes Normal)
1344    (possibleResultOfTest
1345     (TestFn PowerSupply ChangeVoltage ElectricPower) No NotNormal))).

```

```

1346 ;*****
1347 ;** TEST(S) for the VideoSystem **
1348 ;*****

1349 Direction: forward.
1350 F: (implies
1351   (and
1352     (diagnosisContext BootTime)
1353     (hypothesis VideoSystem))
1354   (and
1355     (possibleTest (TestFn VideoSystem ConfirmSensorially VideoSignal))
1356     (possibleResultOfTest
1357       (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes Insufficient)
1358     (possibleResultOfTest
1359       (TestFn VideoSystem ConfirmSensorially VideoSignal) No NotNormal))).

1360 ; *****
1361 ; ** DECOMPOSITION knowledge for the VideoSystem **
1362 ; ** VideoSignal=No **
1363 ; *****

1364 Direction: forward.
1365 F: (implies
1366   (and
1367     (diagnosisContext BootTime)
1368     (hypothesis VideoSystem)
1369     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No)
1370     (plausibleInference Decompose))
1371   (and
1372     (testFirst Monitor)
1373     (testAfter Monitor MotherBoard)
1374     (testAfter MotherBoard Speaker))).

1375 ;*****
1376 ;** TEST(S) for the VideoSystem **
1377 ;** VideoSignal=Yes **
1378 ;*****

1379 Direction: forward.
1380 F: (implies
1381   (and
1382     (diagnosisContext BootTime)
1383     (hypothesis VideoSystem)
1384     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes))
1385   (and
1386     (possibleTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage))
1387     (possibleResultOfTest
1388       (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes Insufficient)
1389     (possibleResultOfTest
1390       (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No NotNormal))).

1391 ; *****
1392 ; ** TEST(S) for the VideoSystem **
1393 ; ** VideoSignal=Yes, VideoBIOSMessage=Yes **
1394 ; *****

1395 Direction: forward.
1396 F: (implies
1397   (and
1398     (diagnosisContext BootTime)
1399     (hypothesis VideoSystem)
1400     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1401     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes))
1402   (and
1403     (possibleTest (TestFn VideoSystem ConfirmSensorially BootContinues))
1404     (possibleResultOfTest

```



```

1405         (TestFn VideoSystem ConfirmSensorially BootContinues) Yes Normal)
1406         (possibleResultOfTest
1407         (TestFn VideoSystem ConfirmSensorially BootContinues) No NotNormal))).

1408 ;; *****
1409 ;; ** DECOMPOSITION knowledge for the VideoSystem **
1410 ;; ** VideoSignal=Yes, VideoBIOSMessage=Yes, BootContinues=No **
1411 ;; *****

1412 Direction: forward.
1413 F: (implies
1414     (and
1415         (diagnosisContext BootTime)
1416         (hypothesis VideoSystem)
1417         (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1418         (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes)
1419         (resultOfTest (TestFn VideoSystem ConfirmSensorially BootContinues) No)
1420         (plausibleInference Decompose))
1421     (and
1422         (testFirst VideoCard)
1423         (testAfter VideoCard MotherBoard))).

1424 ;; *****
1425 ;; ** DECOMPOSITION knowledge for the VideoSystem **
1426 ;; ** VideoSignal=Yes, VideoBIOSMessage=No **
1427 ;; *****

1428 Direction: forward.
1429 F: (implies
1430     (and
1431         (diagnosisContext BootTime)
1432         (hypothesis VideoSystem)
1433         (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1434         (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No)
1435         (plausibleInference Decompose))
1436     (and
1437         (testFirst MotherBoard)
1438         (testAfter MotherBoard VideoCard))).

1439 ;; *****
1440 ;; ** TEST(S) for the Monitor **
1441 ;; ** VideoSignal=No **
1442 ;; *****

1443 Direction: forward.
1444 F: (implies
1445     (and
1446         (diagnosisContext BootTime)
1447         (hypothesis Monitor)
1448         (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No))
1449     (and
1450         (possibleTest (TestFn Monitor CheckIndependently VideoSignal))
1451         (possibleResultOfTest
1452             (TestFn Monitor CheckIndependently VideoSignal) Yes Normal)
1453         (possibleResultOfTest
1454             (TestFn Monitor CheckIndependently VideoSignal) No NotNormal))).

1455 ;; *****
1456 ;; ** TEST(S) for the MotherBoard **
1457 ;; ** VideoSignal=No, Monitor_Working =Yes **
1458 ;; *****

1459 Direction: forward.

```

```

1460 F: (implies
1461   (and
1462     (diagnosisContext BootTime)
1463     (hypothesis MotherBoard)
1464     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No)
1465     (resultOfTest (TestFn Monitor CheckIndependently VideoSignal) Yes))
1466   (and
1467     (possibleTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep))
1468     (possibleResultOfTest
1469       (TestFn MotherBoard ConfirmSensorially SpeakerBeep) SingleBeep NotNormal)
1470     (possibleResultOfTest
1471       (TestFn MotherBoard ConfirmSensorially SpeakerBeep)
1472         ConsistentPattern NotNormal)
1473     (possibleResultOfTest
1474       (TestFn MotherBoard ConfirmSensorially SpeakerBeep)
1475         RingingOrBuzzing NotNormal)
1476     (possibleResultOfTest
1477       (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No Insufficient))).

1478 ;; *****
1479 ;; ** TEST(S) for the MotherBoard **
1480 ;; ** VideoSignal=No, Monitor_Working =Yes, SpeakerBeep=No **
1481 ;; *****

1482 Direction: forward.
1483 F: (implies
1484   (and
1485     (diagnosisContext BootTime)
1486     (hypothesis MotherBoard)
1487     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) No)
1488     (resultOfTest (TestFn Monitor CheckIndependently VideoSignal) Yes)
1489     (resultOfTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No))
1490   (and
1491     (possibleTest (TestFn Speaker CheckIndependently SpeakerBeep))
1492     (possibleResultOfTest
1493       (TestFn Speaker CheckIndependently SpeakerBeep) Yes NotNormal)
1494     (possibleResultOfTest
1495       (TestFn Speaker CheckIndependently SpeakerBeep) No Distinguishing))).

1496 ;;; Important notice: Although the PCSubSystem related to the Test is
1497 ;;; the Speaker, however, the PCSubSystem hypothesised as faulty is
1498 ;;; the MotherBoard. Therefore, the Result Type is defined by what it
1499 ;;; indicates about the MotherBoard and not the Speaker itself. This
1500 ;;; rather peculiar situation is due to the controlling function of
1501 ;;; the Speaker. This means that the Speaker functions as a control
1502 ;;; device for the MotherBoard and we must make sure that this device
1503 ;;; is working properly. If it does, then the lack of any sound is due
1504 ;;; to the MotherBoard and we can deduce that it is faulty. Otherwise,
1505 ;;; we cannot deduce anything before we are certain that the Speaker
1506 ;;; is working properly.

1507 ;; *****
1508 ;; ** TEST(S) for the MotherBoard **
1509 ;; ** VideoSignal=Yes, VideoBIOSMessage=No **
1510 ;; *****

1511 Direction: forward.
1512 F: (implies
1513   (and
1514     (diagnosisContext BootTime)
1515     (hypothesis MotherBoard)
1516     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1517     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No))
1518   (and
1519     (possibleTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep))

```

```

1520     (possibleResultOfTest
1521     (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No Insufficient)
1522     (possibleResultOfTest
1523     (TestFn MotherBoard ConfirmSensorially SpeakerBeep) ConsistentPattern
1524     Insufficient))).

1525 ;; *****
1526 ;; ** TEST(S) for the MotherBoard **
1527 ;; ** VideoSignal=Yes, VideoBIOSMessage=No, SpeakerBeep=ConsistentPattern **
1528 ;; *****

1529 Direction: forward.
1530 F: (implies
1531     (and
1532     (diagnosisContext BootTime)
1533     (hypothesis MotherBoard)
1534     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1535     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No)
1536     (resultOfTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep)
1537     ConsistentPattern))
1538     (and
1539     (possibleTest (TestFn MotherBoard TroubleshootComponent ComponentProblem))
1540     (possibleResultOfTest
1541     (TestFn MotherBoard TroubleshootComponent ComponentProblem) Yes
1542     NotNormal)
1543     (possibleResultOfTest
1544     (TestFn MotherBoard TroubleshootComponent ComponentProblem) No
1545     Distinguishing))).

1546 ;; *****
1547 ;; ** TEST(S) for the MotherBoard **
1548 ;; ** VideoSignal=Yes, VideoBIOSMessage=No, SpeakerBeep=No **
1549 ;; *****

1550 Direction: forward.
1551 F: (implies
1552     (and
1553     (diagnosisContext BootTime)
1554     (hypothesis MotherBoard)
1555     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1556     (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) No)
1557     (resultOfTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No))
1558     (and
1559     (possibleTest (TestFn MotherBoard TroubleshootComponent ComponentProblem))
1560     (possibleResultOfTest
1561     (TestFn MotherBoard TroubleshootComponent ComponentProblem) Yes
1562     NotNormal)
1563     (possibleResultOfTest
1564     (TestFn MotherBoard TroubleshootComponent ComponentProblem) No
1565     Distinguishing))).

1566 ;; *****
1567 ;; ** TEST(S) for the MotherBoard **
1568 ;; ** StartupScreen=No **
1569 ;; *****

1570 Direction: forward.
1571 F: (implies
1572     (and
1573     (diagnosisContext BootTime)
1574     (hypothesis MotherBoard)
1575     (resultOfTest
1576     (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No))
1577     (and
1578     (possibleTest (TestFn MotherBoard ConfirmSensorially SpeakerBeep))
1579     (possibleResultOfTest

```

```

1580      (TestFn MotherBoard ConfirmSensorially SpeakerBeep) Yes NotNormal)
1581      (possibleResultOfTest
1582      (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No Insufficient))).

1583 ;; *****
1584 ;; ** TEST(S) for the MotherBoard      **
1585 ;; ** StartupScreen=No, SpeakerBeep=No **
1586 ;; *****

1587 Direction: forward.
1588 F: (implies
1589      (and
1590      (diagnosisContext BootTime)
1591      (hypothesis MotherBoard)
1592      (resultOfTest
1593      (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No)
1594      (resultOfTest
1595      (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No))
1596      (and
1597      (possibleTest (TestFn MotherBoard ConfirmSensorially ErrorMessage))
1598      (possibleResultOfTest
1599      (TestFn MotherBoard ConfirmSensorially ErrorMessage) Yes NotNormal)
1600      (possibleResultOfTest
1601      (TestFn MotherBoard ConfirmSensorially ErrorMessage) No Insufficient))).

1602 ;; *****
1603 ;; ** TEST(S) for the MotherBoard      **
1604 ;; ** StartupScreen=No, SpeakerBeep=No, ErrorMessage=No **
1605 ;; *****

1606 Direction: forward.
1607 F: (implies
1608      (and
1609      (diagnosisContext BootTime)
1610      (hypothesis MotherBoard)
1611      (resultOfTest
1612      (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No)
1613      (resultOfTest
1614      (TestFn MotherBoard ConfirmSensorially SpeakerBeep) No)
1615      (resultOfTest
1616      (TestFn MotherBoard ConfirmSensorially ErrorMessage) No))
1617      (and
1618      (possibleTest (TestFn MotherBoard TroubleshootComponent ComponentProblem))
1619      (possibleResultOfTest
1620      (TestFn MotherBoard TroubleshootComponent ComponentProblem) Yes
1621      NotNormal)
1622      (possibleResultOfTest
1623      (TestFn MotherBoard TroubleshootComponent ComponentProblem) No
1624      Distinguishing))).
1625

1626 ;; *****
1627 ;; ** TEST(S) for the VideoCard      **
1628 ;; ** VideoSignal=Yes, VideoBIOSMessage=Yes, BootContinues=No **
1629 ;; *****

1630 Direction: forward.
1631 F: (implies
1632      (and
1633      (diagnosisContext BootTime)
1634      (hypothesis VideoCard)
1635      (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoSignal) Yes)
1636      (resultOfTest (TestFn VideoSystem ConfirmSensorially VideoBIOSMessage) Yes)
1637      (resultOfTest (TestFn VideoSystem ConfirmSensorially BootContinues) No))
1638      (and

```

```

1639     (possibleTest (TestFn VideoCard TroubleshootComponent ComponentProblem))
1640     (possibleResultOfTest
1641     (TestFn VideoCard TroubleshootComponent ComponentProblem) Yes NotNormal)
1642     (possibleResultOfTest
1643     (TestFn VideoCard TroubleshootComponent ComponentProblem) No
1644     Distinguishing))).

1645 ;;; Important notice: The '#$TroubleshootComponent' TestAction and the
1646 ;;; '#$ComponentProblem' PossibleObservable are too general and
1647 ;;; complex to be of actual use. They are used here as artificial
1648 ;;; "terminating points" of the Systematic Diagnosis procedure at the
1649 ;;; level of #$PCComponent. In a
1650 ;;; complete PC faults diagnosis expert system, more elaborate
1651 ;;; 'Test(s)' should follow to troubleshoot a specific #$PCComponent
1652 ;;; (here the #$VideoCard), instead of the user having to know how to
1653 ;;; test the specific #$PCComponent. However, these 'Tests' are so
1654 ;;; elaborate and complicated that are beyond the scope of this
1655 ;;; implementation. Such 'Test(s)' could be identifying a beep code
1656 ;;; according to the specific version of BIOS (American Megatrends
1657 ;;; Inc., Pheonix or Other) or interpreting an error message (there
1658 ;;; are 120 error messages documented in the PCGuide Troubleshoot
1659 ;;; expert that was used as a knowledge acquisistion source).

1660 ;; *****
1661 ;; ** TEST(S) for the BIOSStartupSystem **
1662 ;; ** PowerSystem=ok, VideoSystem=ok **
1663 ;; *****

1664 Direction: forward.
1665 F: (implies
1666     (and
1667     (diagnosisContext BootTime)
1668     (hypothesis BIOSStartupSystem))
1669     (and
1670     (possibleTest (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen))
1671     (possibleResultOfTest
1672     (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) Yes Normal)
1673     (possibleResultOfTest
1674     (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No NotNormal))).

1675 ;; *****
1676 ;; ** DECOMPOSITION knowledge for the BIOSStartupSystem **
1677 ;; ** StartupScreen=No **
1678 ;; *****

1679 Direction: forward.
1680 F: (implies
1681     (and
1682     (diagnosisContext BootTime)
1683     (hypothesis BIOSStartupSystem)
1684     (resultOfTest
1685     (TestFn BIOSStartupSystem ConfirmSensorially StartupScreen) No)
1686     (plausibleInference Decompose))
1687     (and
1688     (testFirst MotherBoard)
1689     (testAfter MotherBoard VideoCard))).

1690 ;; *****
1691 ;; ** TEST(S) for the MemorySystem **
1692 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1693 ;; *****

1694 Direction: forward.
1695 F: (implies
1696     (and
1697     (diagnosisContext BootTime)

```

```

1698     (hypothesis MemorySystem))
1699     (and
1700     (possibleTest (TestFn MemorySystem ConfirmSensorially MemoryTest))
1701     (possibleResultOfTest
1702     (TestFn MemorySystem ConfirmSensorially MemoryTest) Complete Normal)
1703     (possibleResultOfTest
1704     (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete NotNormal))).
1705
1706 ;; *****
1707 ;; ** DECOMPOSITION knowledge for the MemorySystem **
1708 ;; ** MemoryTest=Incomplete **
1709 ;; *****

1710 Direction: forward.
1711 F: (implies
1712     (and
1713     (diagnosisContext BootTime)
1714     (hypothesis MemorySystem)
1715     (resultOfTest
1716     (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete)
1717     (plausibleInference Decompose))
1718     (and
1719     (testFirst RAM)
1720     (testAfter RAM MotherBoard))).

1721 ;; *****
1722 ;; ** TEST(S) for the RAM **
1723 ;; ** MemoryTest=Incomplete **
1724 ;; *****

1725 Direction: forward.
1726 F: (implies
1727     (and
1728     (diagnosisContext BootTime)
1729     (hypothesis RAM)
1730     (resultOfTest
1731     (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete))
1732     (and
1733     (possibleTest (TestFn RAM ConfirmSensorially ErrorMessage))
1734     (possibleResultOfTest
1735     (TestFn RAM ConfirmSensorially ErrorMessage) Yes NotNormal)
1736     (possibleResultOfTest
1737     (TestFn RAM ConfirmSensorially ErrorMessage) No Insufficient))).

1738 ;; *****
1739 ;; ** TEST(S) for the RAM **
1740 ;; ** MemoryTest=Incomplete, ErrorMessage=No **
1741 ;; *****

1742 Direction: forward.
1743 F: (implies
1744     (and
1745     (diagnosisContext BootTime)
1746     (hypothesis RAM)
1747     (resultOfTest
1748     (TestFn MemorySystem ConfirmSensorially MemoryTest) InComplete)
1749     (resultOfTest
1750     (TestFn RAM ConfirmSensorially ErrorMessage) No))
1751     (and
1752     (possibleTest (TestFn RAM TroubleshootComponent ComponentProblem))
1753     (possibleResultOfTest
1754     (TestFn RAM TroubleshootComponent ComponentProblem) Yes NotNormal)
1755     (possibleResultOfTest
1756     (TestFn RAM TroubleshootComponent ComponentProblem) No
1757     Distinguishing))).

```

```

1758 ;; *****
1759 ;; ** TEST(S) for the FloppySystem **
1760 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1761 ;; ** MemorySystem=ok **
1762 ;; *****

1763 Direction: forward.
1764 F: (implies
1765     (and
1766         (diagnosisContext BootTime)
1767         (hypothesis FloppySystem))
1768     (and
1769         (possibleTest (TestFn FloppySystem ConfirmSensorially FloppyAccess))
1770         (possibleResultOfTest
1771             (TestFn FloppySystem ConfirmSensorially FloppyAccess) Yes Normal)
1772         (possibleResultOfTest
1773             (TestFn FloppySystem ConfirmSensorially FloppyAccess) No NotNormal))).

1774 ;; *****
1775 ;; ** DECOMPOSITION knowledge for the FloppySystem **
1776 ;; ** FloppyAccess=No **
1777 ;; *****

1778 Direction: forward.
1779 F: (implies
1780     (and
1781         (diagnosisContext BootTime)
1782         (hypothesis FloppySystem)
1783         (resultOfTest
1784             (TestFn FloppySystem ConfirmSensorially FloppyAccess) No)
1785         (plausibleInference Decompose))
1786     (and
1787         (testFirst FloppyDiskDrive)
1788         (testAfter FloppyDiskDrive BIOSsettings))).

1789 ;; *****
1790 ;; ** TEST(S) for the FloppyDiskDrive **
1791 ;; ** FloppyAccess=No **
1792 ;; *****
1793
1794 Direction: forward.
1795 F: (implies
1796     (and
1797         (diagnosisContext BootTime)
1798         (hypothesis FloppyDiskDrive)
1799         (resultOfTest
1800             (TestFn FloppySystem ConfirmSensorially FloppyAccess) No))
1801     (and
1802         (possibleTest (TestFn FloppyDiskDrive ConfirmSensorially BootContinues))
1803         (possibleResultOfTest
1804             (TestFn FloppyDiskDrive ConfirmSensorially BootContinues) Yes
1805             Distinguishing)
1806         (possibleResultOfTest
1807             (TestFn FloppyDiskDrive ConfirmSensorially BootContinues) No NotNormal))).

1808 ;; *****
1809 ;; ** TEST(S) for the OSfloppyDisk **
1810 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=Hard/None **
1811 ;; *****

1812 Direction: forward.
1813 F: (implies
1814     (and
1815         (diagnosisContext BootTime)
1816         (hypothesis OSfloppyDisk)

```

```

1817 (resultOfTest
1818 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1819 FloppyThenHard)
1820 (or
1821 (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource)
1822 (ObservableValueFn HardDiskDrive))
1823 (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource) None)))
1824 (and
1825 (possibleTest (TestFn OSfloppyDisk ConfirmSensorially InFloppy))
1826 (possibleResultOfTest
1827 (TestFn OSfloppyDisk ConfirmSensorially InFloppy) Yes Insufficient)
1828 (possibleResultOfTest
1829 (TestFn OSfloppyDisk ConfirmSensorially InFloppy) No NotNormal))).

1830 ;; *****
1831 ;; ** TEST(S) for the OSfloppyDisk **
1832 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=Hard/None **
1833 ;; ** InFloppy=Yes **
1834 ;; *****

1835 Direction: forward.
1836 F: (implies
1837 (and
1838 (diagnosisContext BootTime)
1839 (hypothesis OSfloppyDisk)
1840 (resultOfTest
1841 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1842 FloppyThenHard)
1843 (or
1844 (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource)
1845 (ObservableValueFn HardDiskDrive))
1846 (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource) None))
1847 (resultOfTest
1848 (TestFn OSfloppyDisk ConfirmSensorially InFloppy) Yes))
1849 (and
1850 (possibleTest
1851 (TestFn OSfloppyDisk TroubleshootComponent ComponentProblem))
1852 (possibleResultOfTest
1853 (TestFn OSfloppyDisk TroubleshootComponent ComponentProblem) Yes
1854 NotNormal)
1855 (possibleResultOfTest
1856 (TestFn OSfloppyDisk TroubleshootComponent ComponentProblem) No
1857 Distinguishing))).

1858 ;; *****
1859 ;; ** TEST(S) for the HardDiskDrive **
1860 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1861 ;; ** MemorySystem=ok, FloppySystem=ok **
1862 ;; *****

1863 Direction: forward.
1864 F: (implies
1865 (and
1866 (diagnosisContext BootTime)
1867 (hypothesis HardDiskDrive))
1868 (and
1869 (possibleTest (TestFn HardDiskDrive ConfirmSensorially DetectionMessage))
1870 (possibleResultOfTest
1871 (TestFn HardDiskDrive ConfirmSensorially DetectionMessage) Yes Normal)
1872 (possibleResultOfTest
1873 (TestFn HardDiskDrive ConfirmSensorially DetectionMessage) No Insufficient)
1874 (possibleResultOfTest
1875 (TestFn HardDiskDrive ConfirmSensorially DetectionMessage)
1876 CannotFind-Message NotNormal))).

1877 ;; *****

```



```

1878 ;; ** TEST(S) for the HardDiskDrive **
1879 ;; ** DetectionMessage=No **
1880 ;; *****

1881 Direction: forward.
1882 F: (implies
1883   (and
1884     (diagnosisContext BootTime)
1885     (hypothesis HardDiskDrive)
1886     (resultOfTest
1887       (TestFn HardDiskDrive ConfirmSensorially DetectionMessage) No))
1888   (and
1889     (possibleTest
1890       (TestFn HardDiskDrive CheckIndependently AutoDetection-BIOSsetting))
1891     (possibleResultOfTest
1892       (TestFn HardDiskDrive CheckIndependently AutoDetection-BIOSsetting)
1893         Manual Normal)
1894     (possibleResultOfTest
1895       (TestFn HardDiskDrive CheckIndependently AutoDetection-BIOSsetting)
1896         Auto NotNormal))).

1897 ;; *****
1898 ;; ** TEST(S) for the HardDiskDrive **
1899 ;; ** BootSequence-BIOSsetting=HardThenFloppy, BootSource=Floppy/None **
1900 ;; *****

1901 Direction: forward.
1902 F: (implies
1903   (and
1904     (diagnosisContext BootTime)
1905     (hypothesis HardDiskDrive)
1906     (resultOfTest
1907       (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
1908         HardThenFloppy)
1909     (or
1910       (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource)
1911         (ObservableValueFn FloppyDiskDrive))
1912       (resultOfTest (TestFn BootSystem ConfirmSensorially BootSource) None)))
1913   (and
1914     (possibleTest
1915       (TestFn HardDiskDrive TroubleshootComponent ComponentProblem))
1916     (possibleResultOfTest
1917       (TestFn HardDiskDrive TroubleshootComponent ComponentProblem) Yes
1918         NotNormal)
1919     (possibleResultOfTest
1920       (TestFn HardDiskDrive TroubleshootComponent ComponentProblem) No
1921         Normal))).

1922 ;; *****
1923 ;; ** TEST(S) for the CDROMdrive **
1924 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1925 ;; ** MemorySystem=ok, FloppySystem=ok, HardDisk=ok **
1926 ;; *****

1927 Direction: forward.
1928 F: (implies
1929   (and
1930     (diagnosisContext BootTime)
1931     (hypothesis CDROMdrive))
1932   (and
1933     (possibleTest (TestFn CDROMdrive ConfirmSensorially DetectionMessage))
1934     (possibleResultOfTest
1935       (TestFn CDROMdrive ConfirmSensorially DetectionMessage) Yes Normal)
1936     (possibleResultOfTest
1937       (TestFn CDROMdrive ConfirmSensorially DetectionMessage) No Insufficient)
1938     (possibleResultOfTest

```

```

1939         (TestFn CDRomdrive ConfirmSensorially DetectionMessage) CannotFind-Message
1940         NotNormal))).

1941 ;; *****
1942 ;; ** TEST(S) for the CDRomdrive **
1943 ;; ** DetectionMessage=No          **
1944 ;; *****

1945 Direction: forward.
1946 F: (implies
1947     (and
1948         (diagnosisContext BootTime)
1949         (hypothesis CDRomdrive)
1950         (resultOfTest
1951             (TestFn CDRomdrive ConfirmSensorially DetectionMessage) No))
1952     (and
1953         (possibleTest (TestFn CDRomdrive ConfirmSensorially BootContinues))
1954         (possibleResultOfTest
1955             (TestFn CDRomdrive ConfirmSensorially BootContinues) Yes Normal)
1956         (possibleResultOfTest
1957             (TestFn CDRomdrive ConfirmSensorially BootContinues) No NotNormal))).

1958 ;; *****
1959 ;; ** TEST(S) for the PlugAndPlaySystem **
1960 ;; ** PowerSystem=ok, VideoSystem=ok, StartupSystem=ok **
1961 ;; ** MemorySystem=ok, FloppySystem=ok, HardDisk=ok **
1962 ;; ** CDRomdrive=ok **
1963 ;; *****

1964 Direction: forward.
1965 F: (implies
1966     (and
1967         (diagnosisContext BootTime)
1968         (hypothesis PlugAndPlaySystem))
1969     (and
1970         (possibleTest (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues))
1971         (possibleResultOfTest
1972             (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) Yes Normal)
1973         (possibleResultOfTest
1974             (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) No NotNormal))).

1975 ;; *****
1976 ;; ** DECOMPOSITION knowledge for the PlugAndPlaySystem **
1977 ;; ** BootContinues=No          **
1978 ;; *****

1979 Direction: forward.
1980 F: (implies
1981     (and
1982         (diagnosisContext BootTime)
1983         (hypothesis PlugAndPlaySystem)
1984         (resultOfTest
1985             (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) No)
1986         (plausibleInference Decompose))
1987     (and
1988         (testFirst ExpansionCard)
1989         (testAfter ExpansionCard MotherBoard))).

1990 ;; *****
1991 ;; ** TEST(S) for the ExpansionCard **
1992 ;; ** BootContinues=No          **
1993 ;; *****

1994 Direction: forward.
1995 F: (implies
1996     (and

```

```

1997     (diagnosisContext BootTime)
1998     (hypothesis ExpansionCard)
1999     (resultOfTest
2000     (TestFn PlugAndPlaySystem ConfirmSensorially BootContinues) No))
2001     (and
2002     (possibleTest
2003     (TestFn ExpansionCard TroubleshootComponent ComponentProblem))
2004     (possibleResultOfTest
2005     (TestFn ExpansionCard TroubleshootComponent ComponentProblem) Yes
2006                                     NotNormal)
2007     (possibleResultOfTest
2008     (TestFn ExpansionCard TroubleshootComponent ComponentProblem) No
2009                                     Distinguishing))).

2010 ;; *****
2011 ;; ** TEST(S) for the BootSystem **
2012 ;; ** Everything else=ok **
2013 ;; *****

2014 Direction: forward.
2015 F: (implies
2016     (and
2017     (diagnosisContext BootTime)
2018     (hypothesis BootSystem))
2019     (and
2020     (possibleTest
2021     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting))
2022     (possibleResultOfTest
2023     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2024                                     FloppyThenHard Insufficient)
2025     (possibleResultOfTest
2026     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2027                                     HardThenFloppy Insufficient))).

2028 ;; *****
2029 ;; ** TEST(S) for the BootSystem **
2030 ;; ** BootSequence-BIOSsetting=FloppyThenHard **
2031 ;; *****

2032 Direction: forward.
2033 F: (implies
2034     (and
2035     (diagnosisContext BootTime)
2036     (hypothesis BootSystem)
2037     (resultOfTest
2038     (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2039                                     FloppyThenHard))
2040     (and
2041     (possibleTest (TestFn BootSystem ConfirmSensorially BootSource))
2042     (possibleResultOfTest
2043     (TestFn BootSystem ConfirmSensorially BootSource)
2044                                     (ObservableValueFn FloppyDiskDrive) Normal)
2045     (possibleResultOfTest
2046     (TestFn BootSystem ConfirmSensorially BootSource)
2047                                     (ObservableValueFn HardDiskDrive) NotNormal)
2048     (possibleResultOfTest
2049     (TestFn BootSystem ConfirmSensorially BootSource) None NotNormal))).

2050 ;; *****
2051 ;; ** DECOMPOSITION knowledge for the BootSystem **
2052 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=Hard **
2053 ;; *****

2054 Direction: forward.
2055 F: (implies
2056     (and

```

```

2057 (diagnosisContext BootTime)
2058 (hypothesis BootSystem)
2059 (resultOfTest
2060 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2061 FloppyThenHard)
2062 (resultOfTest
2063 (TestFn BootSystem ConfirmSensorially BootSource)
2064 (ObservableValueFn HardDiskDrive))
2065 (plausibleInference Decompose))
2066 (and
2067 (testFirst OSfloppyDisk)
2068 (testAfter OSfloppyDisk FloppyDiskDrive))).

2069
2070 ;; *****
2071 ;; ** DECOMPOSITION knowledge for the BootSystem **
2072 ;; ** BootSequence-BIOSsetting=FloppyThenHard, BootSource=None **
2073 ;; *****

2074 Direction: forward.
2075 F: (implies
2076 (and
2077 (diagnosisContext BootTime)
2078 (hypothesis BootSystem)
2079 (resultOfTest
2080 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2081 FloppyThenHard)
2082 (resultOfTest
2083 (TestFn BootSystem ConfirmSensorially BootSource) None)
2084 (plausibleInference Decompose))
2085 (and
2086 (testFirst OSfloppyDisk)
2087 (testAfter OSfloppyDisk FloppyDiskDrive))).
2088
2089 ;; *****
2090 ;; ** TEST(S) for the BootSystem **
2091 ;; ** BootSequence-BIOSsetting=HardThenFloppy **
2092 ;; *****

2093 Direction: forward.
2094 F: (implies
2095 (and
2096 (diagnosisContext BootTime)
2097 (hypothesis BootSystem)
2098 (resultOfTest
2099 (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2100 HardThenFloppy))
2101 (and
2102 (possibleTest (TestFn BootSystem ConfirmSensorially BootSource))
2103 (possibleResultOfTest
2104 (TestFn BootSystem ConfirmSensorially BootSource)
2105 (ObservableValueFn HardDiskDrive) Normal)
2106 (possibleResultOfTest
2107 (TestFn BootSystem ConfirmSensorially BootSource)
2108 (ObservableValueFn FloppyDiskDrive) NotNormal)
2109 (possibleResultOfTest
2110 (TestFn BootSystem ConfirmSensorially BootSource) None NotNormal))).

2111 ;; *****
2112 ;; ** DECOMPOSITION knowledge for the BootSystem **
2113 ;; ** BootSequence-BIOSsetting=HardThenFloppy, BootSource=Floppy **
2114 ;; *****

2115 Direction: forward.
2116 F: (implies
2117 (and

```

```

2118     (diagnosisContext BootTime)
2119     (hypothesis BootSystem)
2120     (resultOfTest
2121       (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2122                                     HardThenFloppy)
2123     (resultOfTest
2124       (TestFn BootSystem ConfirmSensorially BootSource)
2125                                     (ObservableValueFn FloppyDiskDrive))
2126     (plausibleInference Decompose))
2127     (testFirst HardDiskDrive)).

2128
2129 ;; *****
2130 ;; ** DECOMPOSITION knowledge for the BootSystem **
2131 ;; ** BootSequence-BIOSsetting=HardThenFloppy, BootSource=None **
2132 ;; *****

2133 Direction: forward.
2134 F: (implies
2135     (and
2136       (diagnosisContext BootTime)
2137       (hypothesis BootSystem)
2138       (resultOfTest
2139         (TestFn BootSystem CheckIndependently BootSequence-BIOSsetting)
2140                                     HardThenFloppy)
2141       (resultOfTest
2142         (TestFn BootSystem ConfirmSensorially BootSource) None)
2143       (plausibleInference Decompose))
2144     (testFirst HardDiskDrive)).
2145
2146 *****
2147 *****
2148             END OF pc_diagnosisKE.txt
2149 *****
2150 *****

```



# Appendix E

## The CYC SubL Code for PC/Automobile Domains

```
1  ;;; ***** FUNCTION DEFINITIONS *****
2  ;;; ***** GLOBAL VARIABLES *****
3  (csetq *use-local-queue?* NIL) ;CYC variable
4  (defvar *defaultMt* nil) ;the microtheory for the fi-ask function
5  (defvar *test* nil) ; the current Test (needed by the 'menu' function)
6  (defvar *results* nil) ; a list of the Possible Results of the current Test
7  ; (needed by the 'menu' function)
8  (defvar *no_of_choices* 0) ;the number of Possible Results of the current Test
9  ; (needed by the 'menu' function)
10 (defvar *system* nil) ;the system being diagnosed (see 'systematic')
11 ;;; The global variable *terms* is used to record in a list - and in
12 ;;; parallel with what the 'format' expressions print to the CYC SubL
13 ;;; Interactor panel - the CYC terms involved in each step of the
14 ;;; Systematic diagnosis. This variable is returned to the Interactor
15 ;;; panel as the 'Results' of the 'Last Form Evaluated' (see the CYC
16 ;;; SubL Interactor panel). The Interactor panel converts any CYC
17 ;;; term, i.e. symbols starting with '#$' (hash-dollar), as an HTML
18 ;;; link to this term in the CYC KB. This way, the user can browse to
19 ;;; any of the CYC terms appearing on the screen.
20 (defvar *terms* nil)
21 ;;; *****
22 ;;; ***** SubL CODE FOR SYSTEMATIC *****
23 ;;; *****
24 ;;; Function : SYSTEMATIC
25 ;;; Arguments: The system that is going to be diagnosed. For the
26 ;;;             moment, any one of the symbols '$PCSystem' or
27 ;;;             '$AutomobileSystem'.
28 ;;; Result    : Implements the Inference Structure for the Systematic
29 ;;;             Diagnosis problem solving method of KADS applied in PC
30 ;;;             and Automobile faults diagnosis.
31 ;;; Remarks  :
```

```

32 (define systematic (system)
33   (pcond
34     ((cnot (cor (eql system '$PCSystem) (eql system '$AutomobileSystem)))
35       (error "SYSTEMATIC:argument must be '$PCSystem' or '$AutomobileSystem"))
36     (t (pcond
37       ((eql system '$PCSystem) (csetf *defaultMt* '$PCDiagnosisMt))
38       ((eql system '$AutomobileSystem)
39         (csetf *defaultMt* '$AutomobileDiagnosisMt))
40     ) ;end of pcond
41     (pcond
42       ;; if there isn't any hypothesis then start diagnosis
43       ((cnot (fi-ask '($hypothesis ?HYP) *defaultMt*))
44         (csetf *terms* nil)
45         (csetf *system* system)
46         (sd-select1 system))
47       (t (sd-compare)) ;end of innermost 't' clause
48     )) ;end of innermost 'pcond' and outermost 't' clause
49 ) ;end of outermost 'pcond'
50 )

51 ;;; *****
52 ;;; ***** SubL CODE FOR SD-COMPARE * *****
53 ;;; *****

54 ;;; Function : SD-COMPARE
55 ;;; Arguments: None
56 ;;; Result : Calls the appropriate Systematic Diagnosis Inference
57 ;;; Remarks : The *test* global variable holds the last 'Test' that
58 ;;; was performed. It is used to retrieve the result of
59 ;;; this 'Test' and its type. According to the 'ResultType'
60 ;;; of the 'Test', the following decisions may be made:
61 ;;;
62 ;;;          RESULT_TYPE          DECISION
63 ;;;
64 ;;;          Normal                Reject hypothesis; backtrack
65 ;;;          NotNormal             Decompose/Confirm hypothesis
66 ;;;          Insufficient          Perform another 'Test' (Select2-3)
67 ;;;          Distinguishing        Reject hypothesis, backtrack
68 ;;;                               and confirm next hypothesis.

69 (define sd-compare ()
70   (csetf *terms* nil)
71   ;; get the 'ResultType' for the most recently performed 'Test'
72   (csetq ask-result
73     (fi-ask (list '$and
74               (list '$resultOfTest *test* '?RES)
75               (list '$possibleResultOfTest *test* '?RES '?TYPE)) *defaultMt*))
76   (format t "LAST TEST: ~S, ~%RESULT: ~S," *test* (get-ask-binding
77     (first ask-result) 1))
78   (csetf *terms* (cons (cons 'LAST (cons 'TEST: *test*)) *terms*))

79   (csetq result (get-ask-binding (first ask-result) 1))
80   (csetq result-type (get-ask-binding (first ask-result) 2))
81
82   ;; according to the 'ResultType' value, decide the next inference
83   (pcond
84     ((eql result-type '$NotNormal)
85       (format t " RESULT TYPE: '$NotNormal, INFERENCE: Confirm/Decompose")
86       (csetf *terms* (cons (list 'RESULT: result 'RESULT 'TYPE:
87         result-type 'INFERENCE:CONFIRM) *terms*)))
88     (sd-confirm))
89     ((eql result-type '$Insufficient)
90       (format t " RESULT TYPE: '$Insufficient, INFERENCE: Select2-3")
91       (csetf *terms* (cons (list 'RESULT: result 'RESULT 'TYPE:
92         result-type 'INFERENCE:SELECT2-3) *terms*)))
93     (sd-select2-3))

```



```

94      ((cor (eql result-type '$Normal) (eql result-type '$Distinguishing))
95         (format t " RESULT TYPE: ~S, INFERENCE: NewHypothesis" result-type)
96         (csetf *terms* (cons (list 'RESULT: result 'RESULT 'TYPE:
97                                result-type 'INFERENCE:New_Hypothesis) *terms*)))
98         (sd-new-hypothesis result-type))
99      (t (format t " RESULT TYPE: UNKNOWN!, INFERENCE: Diagnosis interrupted"))
100     ) ;end of 'pcond'
101  )

102  ;;; *****
103  ;;; ***** SubL CODE FOR SD-CONFIRM *****
104  ;;; *****

105  ;;; Function : SD-CONFIRM
106  ;;; Arguments: None
107  ;;; Result   : If the current 'hypothesis' is a 'SubSystem', it
108  ;;;           : is decomposed by calling function 'sd-decompose';
109  ;;;           : otherwise, it is a Component and the diagnosis
110  ;;;           : terminates reporting this 'Component' as faulty.
111  ;;; Remarks  :

112  (define sd-confirm ()
113    (csetq ask-result
114      (fi-ask '($and
115              ($hypothesis ?HYP)
116              ($isa ?HYP $Component)) *defaultMt*))
117    (pcond
118      (ask-result
119        (format t "~%~% DIAGNOSIS ENDED. FAULTY COMPONENT: ~S"
120              (get-ask-binding (first ask-result) 1))
121        (csetf *terms* (cons (list 'DIAGNOSIS 'ENDED. 'FAULTY 'COMPONENT:
122                                (get-ask-binding (first ask-result) 1)) *terms*)))
123      (reverse *terms*))
124    (t (format t " Decomposing...")
125      (safe-fi :assert '($plausibleInference $Decompose) *defaultMt*)
126      (sd-decompose)
127      (sd-select2-3))
128    )
129  )
130
131  ;;; *****
132  ;;; ***** SubL CODE FOR SD-NEW-HYPOTHESIS *****
133  ;;; *****

134  ;;; Function: SD-NEW-HYPOTHESIS
135  ;;; Arguments: 1. A $ResultType
136  ;;; Results:
137  ;;; Remarks: This function is called by 'sd-confirm' with two types of
138  ;;;           results, $Normal and $Distinguishing. If the result is
139  ;;;           $Normal, the function just changes the current
140  ;;;           'hypothesis' to the next one (if one exists) and performs
141  ;;;           the appropriate 'Test' by calling 'sd-select2-3'. If the result is
142  ;;;           $Distinguishing, then it changes the current
143  ;;;           'hypothesis' as before, but doesn't perform any 'Test', as
144  ;;;           the last 'Test' performed indicates that the next
145  ;;;           'hypothesis' is faulty. Therefore, it calls the
146  ;;;           'sd-confirm' function.

147  (define sd-new-hypothesis (result-type)

148  ;; get the next hypothesis from the 'testAfter' assertions, if anyone
149  ;; is left
150  (csetq ask-result
151    (fi-ask '($and
152            ($hypothesis ?HYP)
153            ($testAfter ?HYP ?NEW_HYP)) *defaultMt*))

```

```

154 (pcond
155 ((null ask-result) (format t "~% ~%I'M SORRY. THERE IS NO ALTERNATIVE
156                      SYSTEM TO BE CONSIDERED. DIAGNOSIS FAILED"))
157 (t (csetq hypothesis (get-ask-binding (first ask-result) 1))
158    (csetq new_hypothesis (get-ask-binding (first ask-result) 2))
159    (fi-unassert (list '#$hypothesis hypothesis) *defaultMt*)
160    (fi-unassert (list '#$testAfter hypothesis new_hypothesis) *defaultMt*)
161    (safe-fi :assert (list '#$hypothesis new_hypothesis) *defaultMt*)
162    (pcond
163      ((eql result-type '$Normal) (sd-select2-3))
164      ((eql result-type '$Distinguishing) (sd-confirm))
165    )) ;end of inner 'pcond' and 't' clause
166 ) ;end of 'pcond'
167 )

168 ;;; *****
169 ;;; ***** SubL CODE FOR SD-SELECT1*****
170 ;;; *****

171 ;;; Function: SD-SELECT1
172 ;;; Arguments: The system that is going to be diagnosed. For the
173 ;;;              moment, any one of the symbols '$PCSystem' or
174 ;;;              '$AutomobileSystem'.
175 ;;; Results: Asserts the '$hypothesis system)' fact to start the
176 ;;;              Systematic Diagnosis problem solving method
177 ;;; Remarks:

178 (define sd-select1 (system)
179   (safe-fi :assert (list '$hypothesis system) *defaultMt*)
180   (sd-select2-3) ; ask the user what the general complaint is
181 )

182 ;;; *****
183 ;;; ***** SubL CODE FOR SD-SELECT2-3 *****
184 ;;; *****

185 ;;; Function : SD-SELECT2-3
186 ;;; Arguments: None
187 ;;; Result :
188 ;;; Remarks : According to the situation, there may be a lot of
189 ;;; 'possibleTest' assertions in the KB, but only one of them is the
190 ;;; one that must be performed next. This Test is distinguished by the
191 ;;; fact that it is the only one that doesn't have a corresponding
192 ;;; 'resultOfTest' assertion as it is not yet carried out. The
193 ;;; 'sd-select2-3' function must therefore find this Test and pass it and
194 ;;; its possible results ('possiblereultOfTest' assertions) to the
195 ;;; 'get-test-result' function.

196 (define sd-select2-3 ()
197   ;;get the current 'hypothesis'
198   (csetq hypothesis (fi-ask '($hypothesis ?HYP) *defaultMt*))
199   (csetq hypothesis (get-ask-binding (first hypothesis) 1))
200   (format t "~%~%HYPOTHESIS: ~S" hypothesis)
201   (csetf *terms* (cons (list 'HYPOTHESIS: hypothesis) *terms*))

202   ;; Get all possible tests
203   (csetq possible-tests
204     (fi-ask '($possibleTest ?TEST) *defaultMt*))

205   ;; keep the one that doesn't have a corresponding 'resultOfTest' assertion
206   (cdo
207     ((test
208       (get-ask-binding (first possible-tests) 1)

```

```

209     (get-ask-binding (first possible-tests) 1))
210   ) ; end of variables
211   ((cnot (fi-ask (list '#$resultOfTest test '?R) *defaultMt*)) t) ;exit condition
212   (csetq possible-tests (rest possible-tests))
213   )
214   (csetq test (get-ask-binding (first possible-tests) 1))
215   (csetf *test* test)

216 ;; Get the possible results for this test
217   (csetq possible-results
218     (fi-ask
219       (list
220         '#$possibleResultOfTest (list '#$TestFn (second test) (third test)
221                                       (fourth test)) '?VAL '?TYPE) *defaultMt*))
222   (get-test-result test possible-results)
223   )

224 ;;; *****
225 ;;; ***** SubL CODE FOR GET-TEST-RESULT *****
226 ;;; *****

227 ;;; Function : GET-TEST-RESULT
228 ;;; Arguments: 1. A 'Test' structure, which is a list of the form:
229 ;;;
230 ;;; (TestFn SUBSYSTEM TEST_ACTION POSSIBLE_OBSERVABLE)
231 ;;;
232 ;;;           2. A list of the form:
233 ;;;
234 ;;; (
235 ;;; ((?VAL . RESULT_1) (?TYPE . TYPE_1))
236 ;;;   ...
237 ;;; ((?VAL . RESULT_n) (?TYPE . TYPE_n))
238 ;;; )
239 ;;;
240 ;;; Result   : An 'resultOfTest' Assertion in the KB with the actual result of
241 ;;;            the test.
242 ;;; Remarks  : For the moment, it is the 'menu' function that performs
243 ;;;            the actual task as there in no way to get input from
244 ;;;            the user when in the SubL interactor.

245 (define get-test-result (test possible-results)
246   (present-test-parameters test)
247   (present-test-results possible-results)
248   (format t "~%Please, type '(menu [number_of_result])'")
249   ; (input-test-result no_of_choices)
250   (reverse *terms*) ;return as RESULT a list of all CYC terms appearing on the screen

251 ;;; For the moment, we don't know how to interact with the user when in the
252 ;;; SubL Interactor interface. Therefore, the user must give the command
253 ;;; '(menu <number_of_result>)' to interact with the SubL code.
254   )
255
256 ;;; *****
257 ;;; ***** SubL CODE FOR PRESENT-TEST-PARAMETERS*
258 ;;; *****

259 ;;; Function : PRESENT-TEST-PARAMETERS
260 ;;; Arguments: A 'Test' structure, which is a list of the form:
261 ;;;
262 ;;; (TestFn SUBSYSTEM TEST_ACTION POSSIBLE_OBSERVABLE)
263 ;;;
264 ;;; Result   : Prints to the screen the current 'test' parameters, i.e.,
265 ;;;            the SubSystem, TestAction and PossibleObservable.
266 ;;; Remarks  :

267 (define present-test-parameters (test)

```

```

268 (format t "~%NEW TEST: ~%SubSystem: ~S ~%TestAction: ~S
      ~%Observable: ~S ~%" (second test) (third test) (fourth test))
269 (csetf *terms* (cons (cons 'NEW (cons 'TEST test)) *terms*))
270 )

271 ;; *****
272 ;; ***** SubL CODE FOR PRESENT-TEST-RESULTS **
273 ;; *****

274 ;;; Function : PRESENT-TEST-RESULTS
275 ;;; Arguments: A list of the form:
276 ;;;
277 ;;; (
278 ;;; ((?VAL . RESULT_1) (?TYPE . TYPE_1))
279 ;;;   ..
280 ;;; ((?VAL . RESULT_n) (?TYPE . TYPE_n))
281 ;;; )
282 ;;;
283 ;;; Result : An enumerated menu of all possible results of the current 'test'
284 ;;; Remarks :

285 (define present-test-results (possible-results)

286 ;;; Get possible results
287 (csetq results (mapcar #'get-ask-binding
288                       possible-results
289                       (position-list (length possible-results) 1)))

290 ;;; Get possible results' types (Not needed for the moment. The result
291 ;;; type is retrieved by the 'sd-compare' function).

292 ; (csetq result_types (mapcar #'get-ask-binding
293 ;                             possible-results
294 ;                             (position-list (length possible-results) 2)))

295 (csetq counter 1)
296 (cdolist (result results 't)
297 (format t "~A. ~S~%" counter result)
298 (csetf *terms* (cons (cons counter result) *terms*))
299 (csetq counter (+ counter 1))
300 )
301 (csetf *no_of_choices* (- counter 1)) ;needed by 'menu'
302 (csetf *results* results) ;needed by 'menu'
303 ; (csetf *result_type* result_types) ; needed by 'menu'
304 )

305 ;;; *****
306 ;;; ***** SubL CODE FOR POSITION-LIST *
307 ;;; *****

308 ;;; Function : POSITION-LIST
309 ;;; Arguments: 1. A number, n, indicating the number of bindings (lists of
310 ;;;             dotted pairs), in an ask-result of an 'fi-ask' function.
311 ;;;             2. A number, k (1 <= k <= n), indicating which BINDING must
312 ;;;             be returned by the 'get-ask-binding' function.
313 ;;; Result : A list of n elements equal to k.
314 ;;; Remarks : An auxiliary function. Creates the second argument to be used
315 ;;;             in a 'mapping' function which collects a list of BINDINGS for
316 ;;;             the same ask variable.

317 (define position-list (n k)
318 (csetq res ())
319 (cdotimes (c n res)
320 (csetq res (cons k res))
321 )
322 res

```

```

323 )

324 ;;; *****
325 ;;; ***** SubL CODE FOR GET-ASK-BINDING ***
326 ;;; *****

327 ;;; Function : GET-ASK-BINDING
328 ;;; Arguments: 1. A list of dotted pairs of the form:
329 ;;;
330 ;;; ((?VAR1 . BINDING1)
331 ;;; (?VAR2 . BINDING2)
332 ;;; ...
333 ;;; (?VARn . BINDINGn)
334 ;;; )
335 ;;;          2. A number defining which BINDING must be returned.

336 ;;; Result   : POSSIBLE_OBSERVABLE_VALUE
337 ;;; Remarks  :

338 (define get-ask-binding (bindings_list bind_no)
339   (rest (nth (- bind_no 1) bindings_list))
340 )
341

342 ;;; *****
343 ;;; ***** SubL CODE FOR MENU *****
344 ;;; *****

345 ;;; Function : MENU
346 ;;; Arguments: 1. A number from the menu of the possible results (local)
347 ;;;            2. The list of possible results (global variable *results*)
348 ;;;            3. The number of choices (global variable *no_of_choices*)
349 ;;; Result   : Asserts into the KB a 'resultOfTest' assertion
350 ;;; Remarks  :

351 (define menu (selection)
352   (pcond
353     ((cor (< selection 1) (> selection *no_of_choices*))
354       (format t "~%~%You must give as an argument, a number between 1-~A"
355               *no_of_choices*))
356     (t (csetq test (fi-ask '($possibleTest ?TEST) *defaultMt*))
357        (csetq test (get-ask-binding (first test) 1))
358      ; (csetf *result_type* (nth (- selection 1) *result_type*))
359        (safe-fi :assert
360                (list '$resultOfTest test (nth (- selection 1) *results*)) *defaultMt*))
361     )
362   (systematic *system*)
363 )

364 ;;; The 'decompose' inference in KADS Systematic Diagnosis PSM takes as input
365 ;;; the current $$SubSystem ($$hypothesis SUBSYSTEM) and decomposes it
366 ;;; into its subsystems (PART-OF-PREDICATE SUBSYSTEM PART),
367 ;;; generating new hypotheses ($$possibleHypotheses PART).

368 ;;; *****
369 ;;; ***** SubL CODE FOR SD-DECOMPOSE *****
370 ;;; *****

371 ;;; Function : SD-DECOMPOSE
372 ;;; Arguments: None.
373 ;;; Result   :

```

```

374 ;;; Remarks :

375 (define sd-decompose ()
376 ;; Before un-asserting the current hypothesis, its functional parts
377 ;; must be saved
378 (csetq ask-result (fi-ask '($possibleHypotheses ?H) *defaultMt*))
379 (csetq in_hypotheses
380 (mapcar #'get-ask-binding ask-result (position-list (length ask-result) 1)))

381 ;; Before un-asserting the current hypothesis, the order of its subsystems
382 ;; diagnosis must be saved
383 (csetq first_to_test (fi-ask '($testFirst ?SYS) *defaultMt*))
384 (csetq first_to_test (get-ask-binding (first first_to_test) 1))
385 (csetq afters (fi-ask '($testAfter ?S1 ?S2) *defaultMt*))
386 (csetq s1_list
387 (mapcar #'get-ask-binding afters (position-list (length afters) 1)))
388 (csetq s2_list
389 (mapcar #'get-ask-binding afters (position-list (length afters) 2)))

390 ;; Store hypothesis and un-assert it
391 (csetq hypothesis (fi-ask '($hypothesis ?H) *defaultMt*))
392 (csetq hypothesis (get-ask-binding (first hypothesis) 1))
393 (fi-unassert (list '$hypothesis hypothesis) *defaultMt*)

394 ;; Assert possibleHypotheses.
395 (cdolist (system in_hypotheses t)
396 (safe-fi :assert (list '$possibleHypotheses system) *defaultMt*)
397 )

398 ;; Assert diagnosis order
399 ; (safe-fi :assert (list '$testFirst first_to_test) *defaultMt*) ;unnecessary
400 (cdolist (s1 s1_list t)
401 (csetq s2 (first s2_list))
402 (csetq s2_list (rest s2_list))
403 (safe-fi :assert (list '$testAfter s1 s2) *defaultMt*)
404 )

405 ;; Un-assert the ($plausibleInference #$Decompose) assertion.
406 (fi-unassert '($plausibleInference #$Decompose) *defaultMt*)

407 ;; Assert the new hypothesis
408 (safe-fi :assert (list '$hypothesis first_to_test) *defaultMt*)
409 )

410 ;;; *****
411 ;;; ***** SubL CODE FOR SD-RESET *****
412 ;;; *****

413 ;;; Function : SD-RESET
414 ;;; Arguments: The system that is going to be diagnosed. For the
415 ;;; moment, any one of the symbols '$PCSystem' or
416 ;;; '$AutomobileSystem'.
417 ;;; Result : Resets all assertions, in the appropriate Microtheory,
418 ;;; regarding the following predicates :

419 ;;; '$diagnosisContext
420 ;;; '$hypothesis',
421 ;;; '$possibleHypotheses'
422 ;;; '$resultOfTest'
423 ;;; '$testFirst'
424 ;;; '$testAfter'
425 ;;; '$diagnosisContext' (dependant from '$resultOfTest')
426 ;;; '$possibleTest' (dependant from '$hypothesis & '$resultOfTest)

427 ;;; Remarks :

```

```

428 (define sd-reset (system)

429 (pcond
430 ((eql system '$PCSystem) (csetf *defaultMt* '$PCDiagnosisMt))
431 ((eql system '$AutomobileSystem)
432 (csetf *defaultMt* '$AutomobileDiagnosisMt))
433 (t (error "SD-RESET: argument must be '$PCSystem' or '$AutomobileSystem"))
434 ) ;end of pcond
435
436 ;; Set the global variables
437 (csetf *use-local-queue?* NIL)
438 (csetf *test* nil)
439 (csetf *results* nil)
440 (csetf *no_of_choices* 0)
441 (csetf *system* nil)

442 ;;UN-ASSERT #$diagnosisContext
443 (csetq diagnosisContext (fi-ask '(#$diagnosisContext ?C) *defaultMt*))
444 (csetq diagnosisContext (get-ask-binding (first diagnosisContext) 1))
445 (fi-unassert (list '$diagnosisContext diagnosisContext) *defaultMt*))

446 ;; UN-ASSERT #$hypothesis
447 (csetq hypothesis (fi-ask '(#$hypothesis ?H) *defaultMt*))
448 (csetq hypothesis (get-ask-binding (first hypothesis) 1))
449 (fi-unassert (list '$hypothesis hypothesis) *defaultMt*))

450 ;; UN-ASSERT #$possibleHypotheses
451 (csetq ask-result (fi-ask '(#$possibleHypotheses ?H) *defaultMt*))
452 (csetq results
453 (mapcar #'get-ask-binding ask-result (position-list (length ask-result) 1)))
454 (cdolist (result results t)
455 (fi-unassert (list '$possibleHypotheses result) *defaultMt*))
456 )

457 ;; UN-ASSERT #$resultOfTest
458 (csetq ask-result (fi-ask '(#$resultOfTest ?T ?R) *defaultMt*))
459 (cdolist (bindings ask-result t)
460 (fi-unassert
461 (list '$resultOfTest (get-ask-binding bindings 1)
462 (get-ask-binding bindings 2)) *defaultMt*))
463 )

464 ;;UN-ASSERT #$testFirst
465 (csetq first_to_test (fi-ask '(#$testFirst ?SYS) *defaultMt*))
466 (csetq first_to_test (get-ask-binding (first first_to_test) 1))
467 (fi-unassert (list '$testFirst first_to_test) *defaultMt*))

468 (csetq afters (fi-ask '(#$testAfter ?S1 ?S2) *defaultMt*))
469 (csetq s1_list
470 (mapcar #'get-ask-binding afters (position-list (length afters) 1)))
471 (csetq s2_list
472 (mapcar #'get-ask-binding afters (position-list (length afters) 2)))
473 (cdo
474 ((s1 (first s1_list) (first s1_list))
475 (s2 (first s2_list) (first s2_list))
476 (s1_list (rest s1_list) (rest s1_list))
477 (s2_list (rest s2_list) (rest s2_list))
478 ) ;end of variables
479 ((null s1)) ;when no more couples of s1,s2
480 (fi-unassert (list '$testAfter s1 s2) *defaultMt*))
481 )
482 )

```