KNOWLEDGE REPRESENTATION AND PROBLEM SOLVING FOR AN INTELLIGENT TUTORING SYSTEM

By

Vincent Li

B. A. Sc. (Engineering Physics) University of British Columbia

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER'S IN APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

June 1990

Abstract

As part of an effort to develop an intelligent tutoring system, a set of knowledge representation frameworks was proposed to represent expert domain knowledge. A general representation of time points and temporal relations was developed to facilitate temporal concept deductions as well as facilitating explanation capabilities vital in an intelligent advisor system. Conventional representations of time use a single-referenced timeline and assigns a single unique value to the time of occurrence of an event. They fail to capture the notion of events, such as changes in signal states in microcomputer systems, which do not occur at precise points in time, but rather over a range of time with some probability distribution. Time is, fundamentally, a relative quantity. In conventional representations, this relative relation is implicitly defined with a fixed reference, "time-zero", on the timeline. This definition is insufficient if an explanation of the temporal relations is to be constructed. The proposed representation of time solves these two problems by representing a time point as a time-range and making the reference point explicit.

An architecture of the system was also proposed to provide a means of integrating various modules as the system evolves, as well as a modular development approach. A production rule EXPERT based on the rule framework used in the Graphic Interactive LISP tutor (GIL) [44, 45], an intelligent tutor for LISP programming, was implemented to demonstrate the inference process using this time point representation. The EXPERT is goal-driven and is intended to be an integral part of a complete intelligent tutoring system.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I would like to express my appreciation for my supervisor, Dr. Kai P. Lam, of the Electrical Engineering department for his guidance and support in this project. In particular, for the directions to various reference sources and both the hardware and software that made this project possible. I would also like to thank Dr. C. S. Phan for his suggestions in the project.

Also, I would like to thank the ELEC464 students who provided valuable feedback on this project.

Chapter 1

Introduction

University students, engineering students in particular in our case, are given tutorial problem sets to be solved regularly, which are handed in, and marked. The idea behind these problem sets is to reinforce concepts taught in lectures, to provide feedback to the student on his or her strengths and weaknesses in the course, and to develop his or her problem solving skills. There are two main drawbacks to this conventional procedure. First, these assignments are often viewed as chores that are required to pass a course rather than an opportunity to acquire some basic problem solving skills and correct any misconceptions the student may have on the subject. Second, when an error is made, the student often has to figure out for herself why there is an error by comparing her solution with an "ideal" solution. However, the ideal solution often lacks explicit explanation of the lines of reasoning taken to solve the problems. The student often cannot deduce how a solution was arrived at, thus cannot learn from her mistakes. Third, there is a delay between the time the student does a problem and when he gets his marked assignments back. The problems and problem solving processes are no longer fresh in his mind, and attempts to determine the difference between how she arrived at the wrong solution and how the expert (the tutor or professor) produced the ideal solution are thus made more difficult and tedious. Often the student must go to the tutor in person to gain further understanding or, if a tutor is not available, just give up in the attempt, defeating the purpose of the problem sets.

Ideally, an expert tutor should be monitoring the student as he solves the problems.

When the student deviates or seems to deviate from the solution path, the tutor can ask the student why he is doing what he is doing, and possibly correct his misconceptions and give suggestions to bring him back on the solution path. Clearly, to have a private tutor for each and every student is not possible in practice. The next best situation would be to have a simulation, or model, of the expert available while the student solves the problems. This intelligent tutoring system (ITS) would guide the student, providing hints to lead the student toward the solution. ITS is a system "that can make inferences about student knowledge and can interact intelligently with students based upon individual representations of what those students know" [33][preface, p. vi].

We are interested in developing such an intelligent tutoring system to advise students on problem solving in tutorial problems, in effect helping them to organize their thoughts, yet not restricting them to any one predetermined solution path. The domain selected is ELEC464, a fourth year electrical engineering course in mini-/micro-computer systems design.

As a first step toward developing a full scale intelligent advisory system, we need to first understand how the expert solves the problem. In this work, we have established a representation scheme for the expert knowledge and developed an *expert model* which is able to solve a given problem.

Important knowledge elements in the selected domain are time and temporal relations. Signals within a system change over time and these changes must meet certain constraints in order for the system to function properly. Furthermore, these constraints are specified as time range maximums and minimums relative to different reference points in time. A knowledge representation framework is proposed to tackle these problems by making the reference explicit, and representing the time range as a set of values bounded by its maximum and minimum. The details of the representation framework are discussed in Chapter 5. The EXPERT module is a goal-oriented production system with its roots founded in the GIL system developed at Princeton University [44, 45]. A modular architecture is also established to facilitate relatively independent development of the different modules required in a full scale system.

In the following chapters, we will first give a brief summary of developments in the area of ITS, followed by a brief description of our problem domain. We will then proceed to describe the overall architecture of our system, followed by a description of the knowledge representation framework and our approach to the implementation of the expert module. Finally, some of the initial results and an outline of the direction of future research is presented.

Chapter 2

Background

In the past decade, a great deal of research has been carried out in developing an intelligent tutoring system (ITS) which would act as a personalized tutor to a student [21, 35, 41, 42, 43, 44, 50, 58]. Previous researches have been implemented in a variety of domains such as subtraction [8, 10], geometry [4], high school algebra [34, 48, 49], symbolic integration [30], electronic troubleshooting [9, 55], medical diagnosis [13], informal game environments [11, 25], and computer programming [35, 38, 43, 44], among others.

These systems have demonstrated that, where there is a lack of individual attention, using an ITS is more effective than conventional classroom teaching. We have focused our attention on a couple of recent projects in particular as the basis for our research.

John Anderson's group at Carnegie-Mellon University (CMU) produced two intelligent tutoring systems based on Anderson's theory of cognition known as ACT (Adaptive Control of Thought), and a later version ACT* [2]. One is the GEOMETRY tutor [4], and the other a LISP tutor [4, 43]. The LISP tutor will be discussed further here, being better documented in the literatures.

The CMU LISP tutor is comprised of a set of goal-restricted production rules, each production rule representing a unit of knowledge to be taught to the student. These rules are organized by goals. The tutor initially contained 325 *correct* rules as well as 475 *mal-rules*, "buggy" variants of the ideal model [4]. The rules are modelled after cognitive processes and encoded at different levels of problem solving protocols. Advance rules are turned off when novice students are being tutored to avoid presenting them with information beyond their level, as well as confining the tutor's search space in interpreting the students' steps.

The CMU LISP tutor and similar rule based systems [13, 48] have proven to be successful in analyzing student behaviour, but once the analysis is done, explanations must be constructed separately, using knowledge outside the production rules used in the analysis. These systems have their roots in conventional expert systems, where explanation requirements are much less demanding than those required in a tutoring environment. Clancey's GUIDON [13, 14], a tutorial system for the MYCIN expert system, is another such system. In building GUIDON, Clancey found the justification for the expert's action inadequate, and additional knowledge is required to give acceptable tutorial explanations. Also, in the CMU LISP tutor, a set of mal-rules must be constructed beforehand. If unanticipated student action is encountered in which no production rule's conditions are matched, problems may arise. Some sort of approach has to be developed to handle the situation where no rule can be fired. Reiser et al. attempted to address some of these shortcomings in the CMU LISP tutor in the Graphic Interactive LISP tutor (GIL) [44, 45].

In Reiser's work with GIL, the rules used by the expert to solve problems are also used to generate the explanations. GIL is a goal-oriented system which allows forward reasoning, from the given of a problem, as well as backward reasoning, from the desired goal decomposed into smaller subgoals. The rules used by GIL contain explicitly the properties of the input and output data objects as well as the conditions required for a rule to execute, or fire. Thus, if the student's action matches a given rule in any two of its three components (input, output, and conditions), then an explanation can be generated for the component where the error was made. This system leads to a fairly robust explanation system.

We have adopted the goal-driven production system used in GIL. We propose two major modifications in the structure. First, in the implementation of the production rules, the condition side of the rule is separated into a goal part and a condition part only. The input and output statements are absorbed by the condition to allow for a more uniform and flexible production structure. The goal aspect of the rule structure offers "why" a rule was fired, a feature lacking in many conventional expert systems. Absorbing the input and output into a single conditions list does not affect the explanation capability of our system. In fact, the explanation is enhanced, as all failed conditions are kept track of. The production rule structure is discussed in more detail in Chapter 6.

Second, GIL's domain does not deal with temporal elements explicitly. Each step of the program is shown as one event following another. However, the exact time of their occurrence is unimportant. This is not the case in the domain of microprocessor system design. In fact, temporal values are an essential part of problem solving in this domain. This reason has led us to develop a general time point representation system from which we can constructing explanations. There are two major classes of formalism in representing time and temporal concepts [46, 47], namely the McDermott formalism and Allen formalism. Both of these systems attempt to provide a logical representation of time by a tuple of temporal and an atemporal elements. In McDermott's formalism [37], the primitive temporal objects are time points. Time intervals are represented by the two end points of the interval. The building blocks of the atemporal concepts are divided into facts and events types. In Allen's formalism [1], the temporal objects are the intervals themselves. Allen classified the atemporal elements into properties that hold true over a time interval, events that occur over a time interval, and processes that are occuring over a time interval. As pointed out by Shoham in [46], there are weaknesses in both of these formalisms and Shoham proposed a compromised logical representation. Shoham's proposal requires the primitive time elements to be time points and bears a strong resemblence to McDermott's representation, as time point is a more concise and intuitive a representation than intervals. Time point is also less awkward in practical implementations. We have thus chosen a representation framework for time based on Shoham's logic. We have further expanded the time point to incorporate a time range representation for imprecise event occurrence. This framework may be used in other domains dealing with temporal concepts and is further detailed in Chapter 5.

Although not a major focus in this work, an overall architecture of the ITS is required to provide a means of integrating the various modules of the advisor system as the system evolves. Barbara Hayes-Roth [28, 29] has been involved in the study of artificial intelligence applications of the *blackboard architecture* and has successfully applied it in a task planning application. The essence of a blackboard architecture is to provide a common database, the *blackboard*, as a means of communication between several expert knowledge sources [20]. Elements of a problem's solution are placed on the blackboard, and different experts will augment the solution when enough information is present on the blackboard. This architecture is an efficient way of integrating several expert systems together and allows a highly modular development approach. The McCalla ITS research team at the University of Saskatchewan has adopted this approach to their ITS project, called SCENT [35].

In our present work, we have also adopted a blackboard type architecture, mainly for its modular development approach. This architecture allows the separate development of the expert, advisor, and interface modules. Details of the system architecture are discussed in Chapter 4. We shall first discuss in more detail the problems in the selected domain.

Chapter 3

Problem Domain

Our problem domain is based on the ELEC464 course, a fourth year electrical engineering course dealing with mini-/micro-computer systems design. First a brief outline of the course, and then the details of the actual problem sets we would like the ITS to be able to handle are presented.

3.1 ELEC464

ELEC464 is based on the text, *Microprocessor Systems Design*, by Alan Clements [15]. This is a one term course dealing mainly with design of mini- and micro-computer systems based on the Motorola MC68000^{TM} microprocessor. The course deals with topics in microprocessor-memory interfacing, designing with static and dynamic memory components, interrupt handling, multitasking designs, direct memory access techniques, synchronous peripheral interfaces, and standard bus interfaces. The students are given problem sets once every two to three weeks, to reinforce the concepts presented in class and to show the students how these concepts are applied in practice. ELEC464 is a difficult course involving many concepts. It was decided to concentrate only on assignment one of the fall 1989 session initially. This assignment deals with address decoding and timing diagrams of CPU-memory systems. In this assignment alone, we have found the amount of basic knowledge required to solve the entire problem set to be overwhelming, as will be shown.

3.2 Assignment One

A copy of Assignment One is shown in Appendix A. This assignment consists of three questions, and is typical of problems given in the course. The questions may be classified into three major categories:

1. **Analysis** problems, in which, typically, the schematic of a circuit is given. The student would be asked questions concerning the function(s) of the circuit. For example, in question (1a) of Assignment One, the student is given an address decoding circuit for a microprocessor-memory system and is asked to identify what type of decoding strategies are used and what the logical address ranges of the memory are.

2. **Modification** problems, in which the student is asked to modify a given circuit to satisfy a set of constraints or specifications. In assignment one, for example, question (2b) asks the student how to modify the given circuit to integrate the unit into another circuit.

3. **Synthesis** problems. This type of question is of the essence in problem solving in the engineering domain. The student is given a set of specifications and constraints, and is asked to design a circuit satisfying these givens. For example, in question (3) of the assignment, the student is asked to design an address decoding circuit to satisfy a given memory map configuration.

These three types of problems require different problem solving skills and different levels of understanding of the *concepts* [1] required. Analysis questions require the student

[1]The term *concept* is used in this work to denote a unit of knowledge, which may be of variable grain size. A concept may stand alone at one level of understanding while it may consist of numerous sub-concepts at a lower level.

to understand the concepts that have been taught explicitly in class. Solutions may be arrived at with precisely definable rules and definitions. For this reason, the solutions to this type of question are often unique, and only one or a few solution paths exist. This type of problem is used to reinforce concepts taught in class explicitly.

Modification problems are slightly more difficult. This type of problem requires the student to have a thorough understanding of how some given circuit works. The solution to this type of problem is not necessarily unique. As long as the modification meets the desired goal, it is a solution. Of course, some solutions are more elegant or optimal while some may be completely impractical. Even if practicality is used to constrain the type of acceptable solutions, often there is still a fair number of choices. For example, Figure A.10 shows two possible solutions for question (2b). Different solution paths may be drawn to arrive at the solutions. This type of problem is used to show students how concepts taught in class are applied and reinforces the concepts implicitly.

Synthesis problems are the most difficult to solve. Many students have the most difficulty with this type of problem. In our domain, for example, the students are asked to design a circuit given the purpose and constraints on the devices. The solution to this type of problem is far from unique and the number of solution paths is large. This type of problem requires the student to have a thorough understanding of the concepts taught in class and how and when to apply them.

3.3 Analogy with GIL

We have based our approach on GIL because we feel that there are some parallels between the problems in our domain of microprocessor system design and that of LISP programming in GIL. In the synthesis problems, we have found that we can draw a very good analogy between the two domains. The environment in GIL is to tutor program creation in LISP. The goal is for the student to create a LISP program to perform a specific task. In our domain, the student is asked to create a system or circuit that would perform a specific task in synthesis problems. The basic operators in GIL are LISP commands. Each command operates on an input and produces an output. The inputs and outputs are lists and atoms. Similarly, we may consider the basic operators in our domain to range from digital components such as logic gates at the lowest levels, to entire blocks in a block diagram at a higher levels of abstraction. These components operate on a set of input signals, producing a set of output signals. Since the production rules in the GIL framework encode the LISP operators, we can thus, in principle, use a similar framework to encode the components.

We still have to deal with analysis and modification problems. We can look at modification problems as a combination of analysis and synthesis problems, understanding what has been given and then generating the desired result from what has been given. Thus, we can concentrate on the analysis problem. We can, of course, encode separate production rules to deal with these type of problems. However, in GIL's production rule framework, the purpose of the operator is also encoded as a goal, which is what we want in most analysis problems. That is, given the circuit, the student sees a number of components and how they are interconnected. From this schematic, the student is expected to deduce how the circuit would function and the purpose of the components and connections. Thus, given the input, output, and component, the same rules describing the components can be used to discover their purpose – why a device or parameter is used.

3.4 Timing Problems

One factor that GIL does not deal with, but is of critical importance in our domain, is time. In a program, when an operator operates on the inputs and produces the output, time is not a critical issue. This is not the case in microprocessor system design. In fact, the timing relation between component signals is one of the most important parts of the analysis. A signal changes, or a signal is required to change to a given state within some constraint specified by the device. The changes are propagated through the circuit, but the time of change is not precisely known. This work has concentrated on the representation and propagation of this imprecision in time, and an attempt to develop an expert that is able to use this representation to determine if a given timing constraint is met. A subproblem of question (1b) in assignment one has been chosen to demonstrate the reasoning process involved.

We will now give an overview of the overall architecture of the proposed advisor system, and then deal more in depth with the knowledge representation and implementation of the expert in the following chapters.

Chapter 4

Architecture

A typical intelligent tutoring system architecture consists of four parts: an expert solver, a student model, a tutor module and an interface [41, 50].

The expert solver, or the expert model, contains the procedural and heuristic knowledge used by experts to solve problems within a domain. This solver is essentially an expert system. However, in a tutoring environment, conventional expert systems are usually inadequate since they are unable to explain and justify their actions. The student model contains information about the student. It represents the state of the student knowledge about the chosen domain, and is used in various ways by other ITS modules such as offering advice, generating problems, or producing adaptive explanations [56]. The tutor or advisor module evaluates the difference between the steps suggested by the expert and those taken by the student, from which the sources of any student error are inferred and appropriate corrective actions will be effected. It contains the tutorial strategies and teaching methods used by the system. The interface translates the student's direct input into a form the expert and tutor can understand and manipulate. The interface also provides a user friendly environment for the student to work in.

4.1 Architecture of Our Advisor

The architecture we have developed for our system is shown in Figure 4.1.

It is a modified *blackboard* type architecture [20]. A conventional expert system architecture consists of a knowledge base, an inference engine that uses the knowledge base,

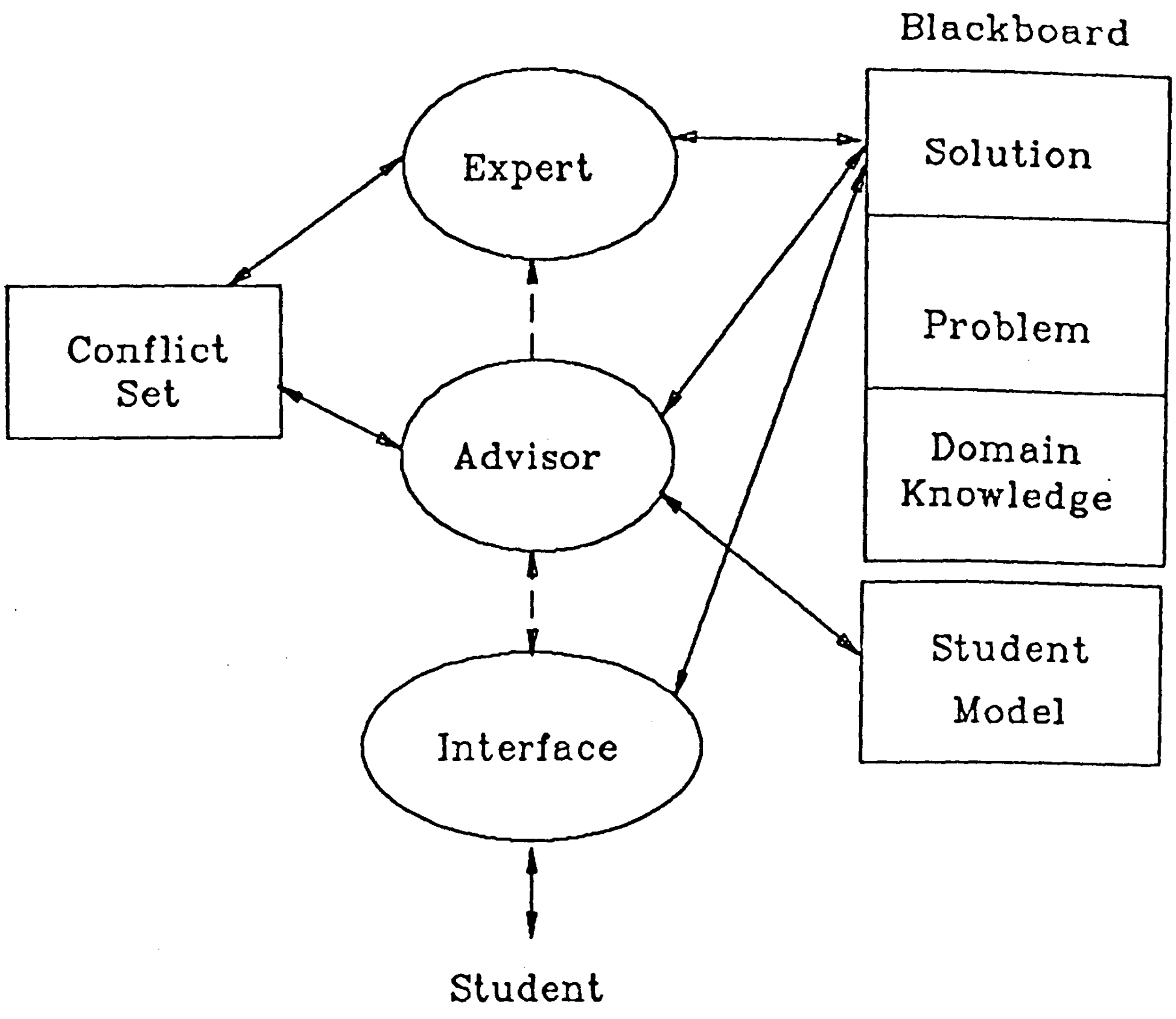


Figure 4.1: The system architecture is made up of the *expert*, *advisor*, *interface*, and the *student model*. Communications between the modules are done via common databases, collectively called the *blackboard*. The circles indicate *active* components while rectangles indicate *passive* databases. Solid arrows indicate actual data flow and dashed arrows indicate command flow.

and a working memory onto which inputs and results are written. This system has two inherent weaknesses. The control is implicitly embedded in the knowledge base, and the knowledge representation is fixed by the inference engine [20]. In a blackboard system, related knowledge bases are grouped together, each having their own inference engine. Together, these "mini-experts", known as *knowledge sources*, communicate their part of the solution to each other via a common global database called the blackboard. Problem solving is thus opportunistic; that is, the knowledge source will add its part of the solution to the blackboard whenever it has enough data to do so. The control of such a system is made explicit.

An example of the application of the blackboard architecture is the SCENT project, a Lisp tutor being developed at the University of Saskatchewan [35]. The expert, advisor, student model, and interface are highly interdependent because they must communicate their results to each other. A blackboard type architecture enables the separation of these highly dependent modules via the blackboard, which is effectively a communication database. Thus, the introduction of this modularity allows a more independent development of the different units of the system and simple integration of these units. This is the reason why we have selected this type of architecture. In our case, the knowledge bases are shared by the various modules in order that explanation and justification of the solution steps may be facilitated from the same knowledge that solves the problem. The databases and modules are briefly described below:

- **Expert** – The purpose of the expert module, when invoked, is to suggest what may be done next for a given problem state. It is a production control system that selects a number of possible next steps, that is, all the domain knowledge production rules that apply to the current problem state. It is responsible for solving the problem, and contains the knowledge of how to manipulate the domain knowledge.

- **Advisor** – The purpose of the advisor is to compare the student's step with those suggested by the expert. If the student's step matches a suggested step (a production rule) from the expert, then the advisor remains silent, adding the production rule to the student model. The results of applying the production rule are added to the solution state. If a match cannot be found, the advisor then proceeds to select a possible next step from the set of possibilities suggested by the expert. When a relevant production rule is found, the advisor then passes the rule to the interface to translate the representation into an explanation for the student. The corresponding student model is also updated. It is also responsible for selecting the appropriate next step to suggest to the student when the student explicitly asks for suggestions.

- **Interface** – The interface module simply translate the information on the solution blackboard into a representation the human user can understand. The output may be natural language sentences, diagrams, or animations.

- **Student Model** – The student model contains the past history of the student. It is a representation of the student's knowledge state and is used to select the appropriate next step to suggest to the student.

- **Blackboard** – The *blackboard* is the global database which acts as the "memory" and communication interface of the system. It is partitioned into several sub-blackboards for organizational purposes as follows:

  - **Domain Knowledge** – All the domain knowledge is stored in this database. It contains the facts used in the domain. The elements in this database do not change over the course of the tutorial session.

- – **Problem** – The problems are defined here as a set of given elements and goal elements. This database does not change over the course of the problem solving.

- – **Solution** – This is the work space where all the solution elements are recorded. A solution element is simply a statement of facts that are true for a given solution state. They are affected by the productions executed. The elements in this database may be asserted and retracted over the course of the problem solving process. The contents of this database are always cleared before the beginning of each problem.

- **Conflict Set** – The conflict set holds the set of rules that may be applied at a given solution state. The term "conflict set" holds a slightly different connotation than its conventional use in production expert systems. Here, it merely denotes a set of productions that was selected. Actual conflicts between the rules are resolved by a filtering process explained in more detail in Chapter 6.

We shall now examine the representation of the domain knowledge and solution elements in detail.

Chapter 5

Knowledge Representation

In any intelligent system, one of the key questions that needs to be answered is how to represent knowledge. In our domain, the use of schematic diagrams to represent electronic circuits and timing diagrams to represent signal relations is extensive. Thus, it is logical to encode these graphical representations in some manner, and the following sections details the convention implemented.

5.1 Circuit Representation

The schematics can be represented by the interconnections among the nodes, or pins of devices, hence:

```
connected(node_1,node_2,Reason). % node_1 and node_2 are connected because
                                   of Reason
```

where *node_i* is a unique point in a circuit, and can be represented by three pieces of information as:

```
node = [device_name, device_number, name_of_node]
```

The *device_name* and *name_of_node* are particular to a device. For example, an OR-gate may have two-inputs *in1* and *in2* and output *out.* A generic OR-gate output may then be labelled as *[or,N,out]*, say, for the N^{th} OR-gate. *Device_number* defines the specific device being dealt with in a circuit.

The implicit assumption made is the commonality principle. Under this principle, if node 1 is connected to node 2, and node 2 is connected to node 3, then node 1 is connected to node 3 as well. A separate statement of the connection between node 1 and node 3 in the above example would be recorded along with the reason that node 1 and node 3 are connected, namely because node 1 and node 2 are connected. Thus, if node 1 and node 2 are disconnected, node 1 and node 3 must also be disconnected. This commonality principle is implicit in the sense that the fact that node 1 and node 3 are connected is stated automatically, whenever the above situation occurs.

5.2 Signal Representation

Next, we need to consider the status or state of the signals from the nodes. The state of a node in state S is represented as:

```
state(node,S). % the state of node is S
```

where S={high|low|float|valid|invalid|asserted|negated|unknown} [1,2]. These states can be divided into two groups: functional states and logical states. The functional states are more useful in explaining what a node or signal is doing while the logical state is the actual electrical characteristics assumed. The logical states are chosen from the set {high|low|unknown}. The functional states are defined to be {float|valid|invalid|asserted|negated|unknown}. States *valid* and *invalid* are used with bus type signals (a group of nodes) such as the address bus and data bus, while *asserted* and *negated* are used with control signals. Only *asserted* and *negated* can be converted to an equivalent known logical signal of *high* or *low*, depending on whether the node is

[1]The notation {*choice1*|*choice2*} is used to indicate either *choice 1* or *choice 2* is selected.

[2]The CPU clock signal is the only exception. The *clk* signal state is defined as S0, S1, S2, etc. for the bus cycle and W1, W2, W3, etc. for any wait states inserted, as defined in the M68000 user's manual [39]. The logical state is inferred from the definition of these states.

active high or active low.

Of great interest in our domain is when a node assumes a particular state and how these states change over time. We define an event to be a condition, in this case the state of the signals, which occurs at a point in time, T. The events of interest in our domain are when a node changes state, represented by:

```
event(T,state(N,S)). % the state of node N goes to S at time T
```

That is, the state of node N goes to state S at time T. The assumption is that before T, N may assume any state other than S. A collection of these *event* statements will completely define the changes in the signals of interest. This forms the encoded representation of the timing diagram, the graphical method used by human experts to represent changes in states and to solve timing problems in this domain. A key question that needs to be answered at this point is how time T should be represented.

5.3 Time Representation

Conventionally, T is represented as a single value, say an integer, with an implicit reference "time 0" on the timeline. This system allows different times to be simply compared, but the time point when an event occurs must be precisely known. This is not the case in many practical situations, as in our domain of CPU-memory systems, where the timing parameters are not and cannot be precisely specified. Rather, a range of constraint values are given. Also, timing constraint parameters is given relative to some reference which is different for different parameters. Converting all the parameters to a single time reference à priori may be difficult and very inefficient. For example, for the 68000 memory read cycle, the parameter t_{AVSL}, time from address valid to $\overline{AS}$ low is the time when the *overlineAS* signal asserts *relative* to when the address bus is valid. But the parameter t_{CHSL}, clock high to $\overline{AS}$ low, also gives the time when $\overline{AS}$ asserts. In this case, if both parameters are converted to the same reference, say when clock is low, they should be identical and choice of either would be arbitrary. Then it does not make sense to have two parameters referring to the same event, $\overline{AS}$ asserts. However, suppose some system configuration requires the address bus valid to $\overline{AS}$ assert meets some constraint. Then, to convert this constraint to a new time reference is obviously more work than simply comparing it with the t_{AVSL} parameter. In any case, this practice is not desirable. A more dynamic derivation would be more robust and flexible.

Time is fundamentally a relative quantity. Human beings talk about time in relative terms. For example, when we say "3:00 p.m. tomorrow", we are really saying "3 hours from noon tomorrow relative to today". Terms such as *tomorrow*, *yesterday*, and *last week* are all relative terms, where the time *reference* is understood. Such knowledge must be made explicit to the machine in order for it to understand time and produce sensible explanations. Hence we propose to make the reference an explicit and integral part of the time point representation itself. Therefore, a given time point t is represented by a pair of values $t = [< s_{ref} >, < t_{rel} >]$, where time t is the point in time t_{rel} after reference s_{ref}. Making the reference point explicit allows us to use a multiple-referenced timeline instead of a single-reference timeline. The advantage of this system is that now we can talk about a time relative to a local reference instead of some possibly distant and implicit time in the past or future, facilitating more sensible explanations. If only a single s_{ref} is used, then the representation is collapsed back into the conventional single reference system. Thus, the proposed representation is a superset of the existing system.

In our implementation, t_{rel} is a single parameter name (e.g. tDICL), and s_{ref} is a single or several conjunctive reference **conditions** or **states**. Hence

```
event(T,state(N,S)) = event([sREF,tREL],state(N,S))
```

is interpreted as: "The event of node N changes to state S at time tREL after the time when all the conditions sREF are satisfied". For example, the statement `event([state([cpu,0,clk],s2),tCHSL],state([cpu,0,as_],asserted)`

means that $\overline{AS}$ of CPU zero is asserted t_{CHSL} after the CPU clock enters bus cycle state S2. For time at $\pm\infty$, we may use t=[X,'-inf'] and t=[X,'inf'] to represent them, X being any reference. That is, infinity is the same point in time relative to any reference.

Different time points can be compared by first finding a common reference between them, then evaluating the resulting value ranges.

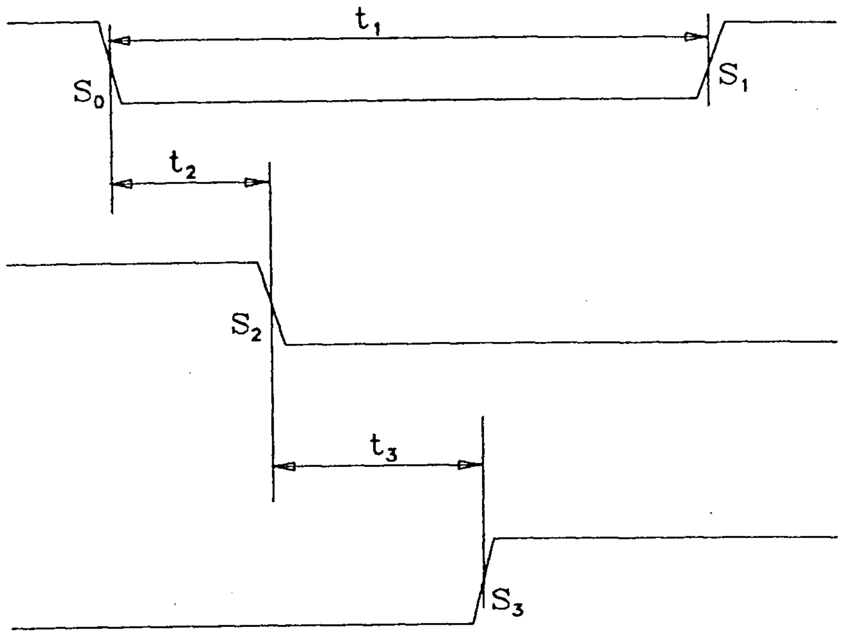


Figure 5.2: A typical timing diagram with of 4 state changes. State S_1 occurs at t_1 after state change to S_0 and S_3 occurs at $t_2 + t_3$ after S_0.

For example, Figure 5.2 shows a typical timing diagram consisting of four events, and is described by the following sequence of statements:

- event([S_0,t_1],S_1).
- event([S_0,t_2],S_2).
- event([S_2,t_3],S_3).

In order to determine the relation between S_1 and S_3, say, to determine which event happens first, we first need to find a common reference between the time of the events, namely S_0. Graphically, we can see immediately that the parameters we wish to compare are t_1 and $t_2 + t_3$. We can discover these relations from the the *event* statements also. By tracing back, we find that S_1 occurs at t_1 relative to S_0 and S_3 occurs at t_3 relative to S_2, but S_2 occurs at t_2 relative to S_0. Thus, the event of S_3 occurs at time $t_1 + t_2$ relative to S_0. We have derived the parameter equations that are required for the comparison, and need only substitute the quantitative values of these parameters for a simple comparison. When S_0 occurs is of no concern to us because this is our reference. Two questions arise here: What happens when the reference is made up of more than one states change occurring at once, and what are the values of the relative times t_i?

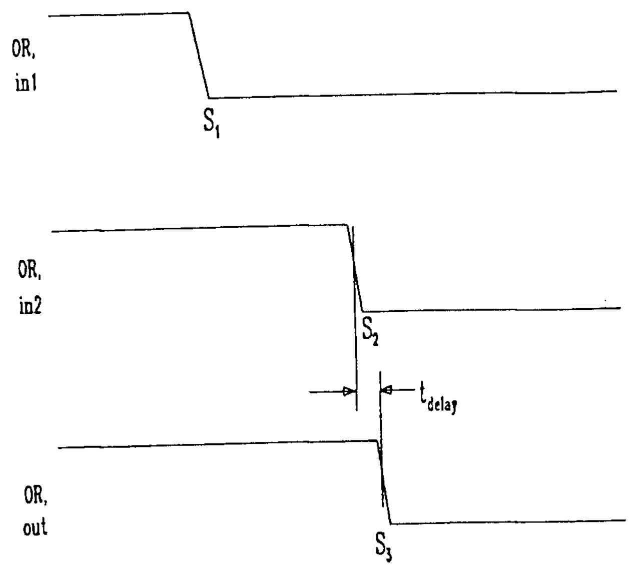


Figure 5.3: Timing relation of a typical 2-input OR-gate. S_3, output low, occurs when both S_1, input 1 is low, and S_2, input 2 is low.

In the case of more than one event referenced in conjunction, we can reduce the conjunction to a single equivalent event. This equivalent would be the last of the conjunctive

events to occur. For example, consider the timing relation of a two-input OR-gate as shown in Figure 5.3. The relation can be described by:
event([[S_1, S_2],t_{delay}],S_3),
event S_3, the OR-gate output is low, occurs t_{delay}, propagation delay, after *both* inputs are low, i.e. state changes S_1 and S_2. For Figure 5.3, we can restate the relation as:
event([S_2,t_{delay}],S_3)
as S_2 occurs later than S_1. Similarly, the reference would be S_1 if it is to occur after S_2. If both events occur at the same time, we may select either event, either arbitrarily or by some preferred choice rules.

The quantitative value the relative times t_i are parameters that may have certain constraint requirements. The constraint is represented as:

```
constraint([sREF,ti],[tMin,tMax]).
```

where t_i assumes the minimum and maximum values tMin and tMax respectively, relative to reference sREF. The constraint value does not change from circuit to circuit, but is dictated by the device. The actual value, however, may change and is represented as:

```
value([sREF,ti],[tMin,tMax]).
```

Thus, the value of a given parameter is found by determining what the value of the parameter is relative to the given reference.

When given two arbitrary time points, their common reference is not known, or even if a common reference exists at all. How then can one find the common reference? The answer is derived from the timing diagram as represented by the collection of *events*. These *event* statements together with *value* and *constraint* are used by the expert to determine the common reference timing in the problem solving process. We will now first take a closer look at the expert, then discuss how the expert and the proposed knowledge representations work together to find the common time reference and solve problems in Chapter 7.

Chapter 6

The Expert

The EXPERT is invoked by the ADVISOR to provide information about the correctness of a student's action or to solve a given problem. It is a goal-oriented production rule expert system. Each rule will satisfy a single goal or may create a number of subgoals when executed, or fired. The EXPERT performs a simulation on a given circuit by a simple recognize-act loop. All productions that can be fired at a given time are placed in a conflict set. These rules are then filtered. All the rules that remain will be acted upon, as opposed to the case in conventional systems, where only one of the rules is fired. The reason for firing all the rules is that several goals may be satisfied at the same time by different rules, which may all be present in the conflict set. Also, more than one rule may satisfy the same goal, each representing different solution paths. The loop will terminate when no more rules can be fired for a given solution state. The expert's action is further detailed in the next chapter.

The structure of the rules will now be discussed further.

6.1 Production Rules

The general knowledge representation scheme is adopted from the production framework in GIL [44]. The implementation is in a Prolog style rule: [1]

```
concept(Concept_Name,Concept_number,Rule_Info_List) :-
```

[1]The representations are presented in the Prolog programming language style and readers are referred to [16] and [53] for details on the syntax.

```
(current_goal(Goal),
 conditions(ConditionsList)
) ->
(properties(PropertiesList),
 new_goals(NewGoalList)).
```

Each production is uniquely identified by its Concept_Name and Concept_number. The Rule_Info_List provides various information about the rule that may be used by the advisor to formulate its explanations, such as the direction of reasoning of the rule. The left-hand, or condition, side (LHS) of the rule consists of *current_goal* and *conditions*. The right-hand, or action, side (RHS) of the rule consists of *properties* and *new_goals*. A rule is added to the *conflict_set* database when all its left-hand side conditions are satisfied. The LHS is satisfied by matching the pattern of the rule with solution elements in the *solution* database. When the conflict set is completed, this set is filtered, or resolved, to remove some of the rules in the set by some criteria. For example, a rule that was fired previously should not be fired again for satisfying exactly the same conditions, since this action contributes no progress toward a solution. After filtering, the remaining rules in the conflict set are fired. When a rule is fired, the properties and new goals are added to the solution database as new solution elements.

The EXPERT is a goal-oriented production system. When solving a problem, the expert attempts to solve a certain set of goals, or objectives, in the *solution* database. These objectives are marked as either *active* or *solved*. Only active goals will trigger the rules. The ConditionsList describes the properties that must be satisfied in order for the rule to be considered for execution. This is a list of terms that are in the solution database, are obtainable from another database (in which case the term would be added to the solution database), or is a provable [2] goal. The *properties* describes the properties of the output as a result of firing the rule, and is also a list of terms that are either provable or added to the solution database. The *new_goals* sets up the subproblems and new objectives to be solved next. This is a list of active objectives that are added to the solution.

```
concept(tCLAV,1,[direction(forward),object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,addressbus([A1,A2])],valid]),
 conditions([
=(Addressbus,[cpu,N,addressbus([A1,A2])]),
  =(T,[state([cpu,N,clk],s1),tCLAV]),
  frequency(8)
  ])
) ->
(properties([
solves([determine,when,[cpu,N,addressbus([A1,A2])],valid]),
  event(T,state(Addressbus,valid)),
  constraint(T,[undefined,62]),
  value(T,[undefined,62])
  ]),
 new_goals([[propagate,event(T,state(Addressbus,valid))]])).
```

Figure 6.4: Example of a definition rule on the timing parameter t_{CLAV}, clock low (S1) to address valid.

Figure 6.4 shows an example of a production rule about the timing parameter t_{CLAV}, which is the time from clock enters bus cycle state S1 (clock low) to address bus valid. The *=/2* [3] predicate is used to bind the variables for a cleaner looking rule. The goal is of the form: *[determine,when,Node,State]*, determining when a *Node* assumes a particular

[2] *Provable* in the sense that it may be evaluated as true or false. In the case of Prolog implementation, the term is the head of a Prolog clause.

[3] The */2* means the predicate takes 2 arguments. In Prolog notation, */n* indicates predicates with *n* arguments. For more details, the reader is referred to [16] and [53].

State. The conditions required are that the clock frequency is 8MHz, and that given time *T* and address bus *Addressbus*, this rule will be satisfied. When executed, the expert will place the *event*, *constraint*, and *value* elements into the solution database. The *properties* also indicates that this rule solves the goal. The *solves* predicate is a provable goal that explicitly modifies the current goal to the *solved* status, so that the goal will no longer be considered. *New_goals* sets up the new goal to propagate the event to all the nodes connected with the CPU address bus.

The production rules used are listed in Appendix B.

6.2 The Blackboard

The blackboard is the set of common databases via which the various modules communicate with each other and with the student. The blackboard is divided into three sub-databases: domain knowledge, problem definition, and solution.

Domain knowledge contains all the facts and production rules about the domain. In our case, it would contain information about the MC68000 processor, memory components, discrete SSI and MSI components, etc. This information may exist in the memory of the computer while the advisor system is running or as a single or multiple file(s) residing on disks.

The problem definition contains all the information given in the problem and what the objective of the problem is (i.e. the set of goals to solve). It is the encoded form of the problems on the assignment sheets. The selected problem would put the relevant information in the solution database, from which the problem may be solved. In Prolog implementation, this is achieved by a series of assertions. A simple example is shown in Appendix C, where the file *problem.pro*, the problem definition file, is shown. In our example, the problem is not loaded into the problem definition, but is defined directly from the file.

The solution database is where all the work is done and recorded. It is made up of *solution elements* that may be created, modified, or destroyed by the expert and advisor. Each solution element is a triplet in the format:

```
element(<tag>,<element>,<reason>).
```

The $< tag >$ is simply an integer label to identify the elements. The $< element >$ is the solution element itself, represented in our implementation in Prolog predicate logic. This is the part which is used to match the LHS of the production rules, and the products produced by the RHS of the productions that are fired. The $< reason >$ is the justification for the existence of $< element >$. There are three major reasons for the existence of an element:

1. An $< element >$ may be given by the problem definition, or
2. It may be a domain fact that can be looked up, represented as `domain(Origin)` for a fact coming for the domain or subdomain, `Origin`, or
3. The $< element >$ is deduced from a production rule, identified as `rule(concept(rule_name,rule_number,rule_info))`, which uniquely identifies the rule effected.

The rules that are executed are recorded in the solution as well in the format:

```
element(<tag>,rule(Name,Number,Info_List),Element_List).
```

where `Element_List` is a list of the tags of the elements in the solution database that contributed to satisfying the LHS of the rule. The tags provide the reason why the rule was fired in the first place. Appendix C shows the content of the solution database at various stages of solving the problem of determining the parameter t_{DICL} for the circuit in assignment one.

In addition, there is a *conflict_set* database, which contains the set of rules that the expert deems appropriate and may be used for a given solution state. Two predicates exist in this database: *size* and *rule*. *size* simply indicates the number of rules in the conflict set. The *rule* predicate contains all the information about a rule selected by the expert. Its format is:

```
rule(RuleHead,RHS,TTs,FailedLHS).
```

`RuleHead` uniquely identifies the rule that was selected. `RHS` is the right-hand side of the production that is to be executed if the rule fires. The RHS is recorded to save the expert from having to look up the rule again if the rule is to be fired. `TTs` is a list of the solution element tags which identifies the solution elements that satisfies the LHS of the production. `FailedLHS` is a list of the LHS of the production not satisfied. This information is used to create an explanation and justification for the expert's actions.

Next, we will examine how the EXPERT performs problem solving with the knowledge representation presented earlier.

Chapter 7

Solving Problems

When asked to solve a problem given a set of initial objectives, the EXPERT would create a conflict set containing all the rules that can be fired at a given solution state. These rules are then filtered or resolved by the expert and the remaining rules are executed. By executing the rules, new goals are added to the solution which must be satisfied. This process then repeats until the size of the conflict set is zero, that is, no more rules known to the expert can be fired. At this point, two conditions may exist: Either there are still active objectives in the solution database, or the problem has been solved, (i.e. all objectives were solved). The expert will return a failure in the former case and a success in the latter, indicating whether the given problem was solved or not.

7.1 Equivalent Goals

For each loop, after the rules have been fired, the solution database is cleaned up by removing duplicate active goals that may be generated as well as propagating the results of solved goals. A given goal may assume several forms in the solution database at once, each of which is equivalent. Consider the circuit in Appendix A for problem one. Consider the $\overline{CS}$ node of the RAM. The goal *determine when* $\overline{CS}$ *asserts* is equivalent to *determine when* $\overline{CS}$ *low* because $\overline{CS}$ is an active low signal. Furthermore, since $\overline{CS}$ is connected to $\overline{OE}$ as well as the output of the or gate, the following goals are all equivalent:

- determine when $\overline{OE}$ asserted.

- determine when $\overline{OE}$ low.
- determine when [or,1,out] asserted.
- determine when [or,1,out] low.

Note that rules cannot be written to satisfy a goal such as *determine when [or,N,out] asserted* (determine when the output of the N^{th} or gate is asserted), since the state of *asserted* is not defined for logical devices such as an or-gate. Whereas, although one can define a rule with the goal *determine when $\overline{CS}$ low*, in terms of conceptual explanation, one would prefer instead to define a rule for the goal *determine when $\overline{CS}$ asserted.* By noting that all these goals are equivalent, when any rule that solves one of these goals has all the other conditions satisfied, the rule can fire to solve that goal, and all the equivalent goals will also be solved at the same time. This allows us to create rules about a device that can be used in any circuit configuration. For a human expert solver, the equivalence of these goals is implicitly expressed, graphically, with timing diagrams. This equivalence needs to be stated explicitly by the expert model.

After a set of rules has been selected to create a conflict set, these rules need to be filtered. The rule filtering that is done by the expert is done primarily to prevent infinite loops. Suppose a rule had been fired, but the elements triggering it are still in the solution database. If the expert is asked to suggest a new set of rules that can be executed, the same rule will be selected for the same elements (i.e. the same reasons). If this is the only rule in the database, clearly the loop will not terminate. Furthermore, firing this rule again will not be profitable, since it does not contribute toward the solution. Hence, it should be eliminated from the conflict set. Elimination is easily accomplished, as we have a record of the rules fired and the reasons for firing them in the solution database. So, a rule will not be fired twice for the same reasons.

7.2 Determining Time

The problems solved in our domain are concerned with the timing of signals. The constraint of when a signal assumes a certain state must be satisfied in order for a given circuit to work properly. We have already examined how time is represented, and quantified with *constraint* and *value*. We will now examine how these predicates are used in conjunction with the collection of *events* to solve the timing problem.

For t, a point in time, represented by $t = [s_i, T_i]$, where s_i is the set of conjunctive references S_1, S_2, etc. Then, *event* is of the form:
event(t_i,S_i) = event([s_i,T_i],S_i) = event([[$S_1, S_2, \ldots, S_n$],T_i],S_i).

Each *event* represents a change of state of a node in our domain. Our objective is to determine the value of T_i, the timing parameter of a given circuit, in order to determine if it satisfies the required constraint. First we note that for references s_i for state S_i, the reference may be reduced to a single equivalent state reference S_j, which is the latest of the set of conjunctive reference states s_i. For example, if
s_i = [state([ram,N,cs_],asserted),state([ram,N,oe_],asserted)],
that for the N^{th} RAM device, both the nodes $\overline{CS}$ and $\overline{OE}$ are asserted. If $\overline{CS}$ is asserted before $\overline{OE}$, then we can say s_i is equivalent to the state S_j, $\overline{OE}$ is asserted, and vice-versa. Once the equivalent single event is established, we may write the same event as: event([S_j,T_i],S_i) (see Figure 7.5).

Hence, to find the value of T_i relative to event S_j, we simply look for a common reference, S_k, of the two events such that:

- event([S_k, T_i'],S_i), and
- event([S_k, T_j'],S_j).

From Figure 7.5, we can see that in order to to find the value of T_i, we simply find the

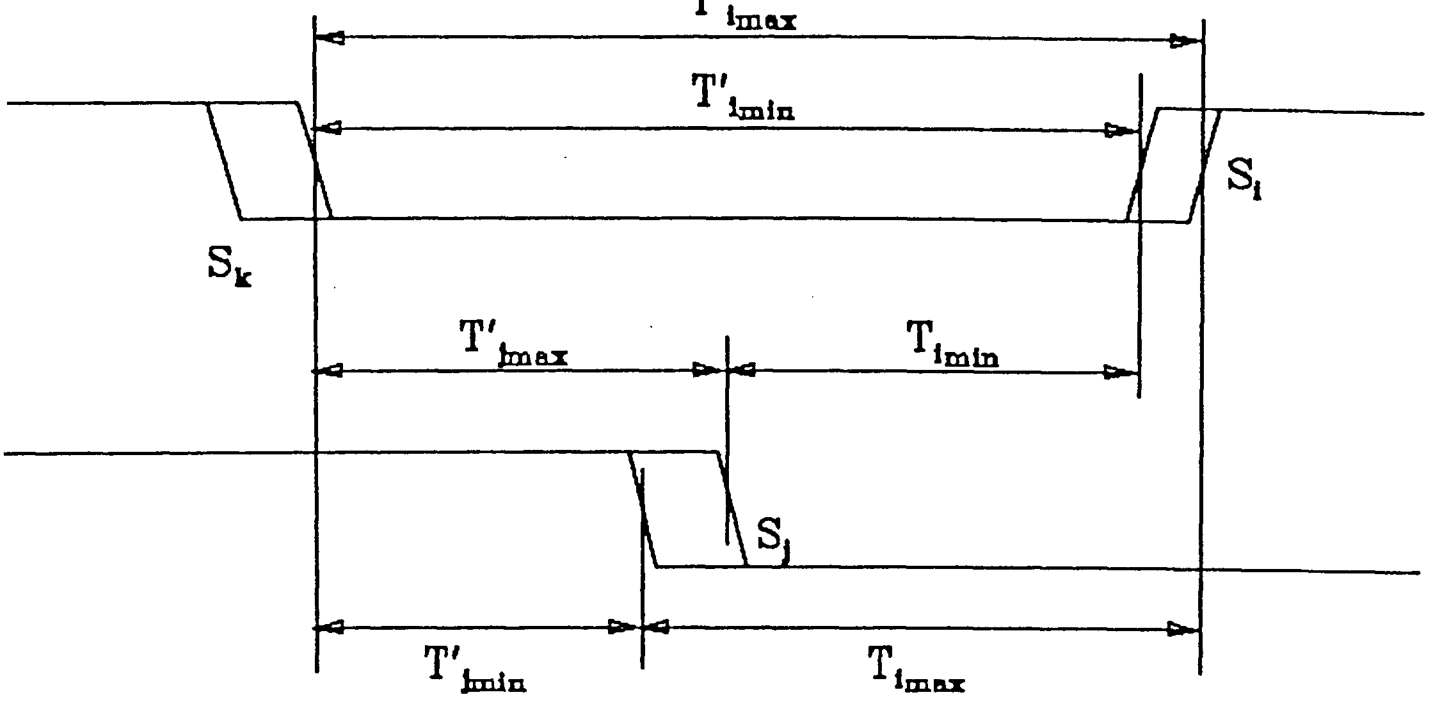


Figure 7.5: Timing diagram of events that occur over a range of time. $T_{i_{min}} = T'_{i_{min}} - T'_{j_{max}}$, and $T_{i_{max}} = T'_{i_{max}} - T'_{j_{min}}$.

difference of the values of T'_i and T'_j. That is, in the implementation, we look for:

- value($[E_k, T'_i]$,$[T'_{i_{min}}, T'_{i_{max}}]$)
- value($[E_k, T'_j]$,$[T'_{j_{min}}, T'_{j_{max}}]$)

from which we can evaluate:

- value($[E_j, T_i]$,$[T_{i_{min}}, T_{i_{max}}]$), where

  $T_{i_{min}} = T'_{i_{min}} - T'_{j_{max}}$, and

  $T_{i_{max}} = T'_{i_{max}} - T'_{j_{min}}$

Note that the exact time of occurrence of S_k is irrelevant.

7.3 Determining Reference

Next, how the four required items – two *events* and two *values* – are evaluated must be determined. For the *event* requirements, we simply look for the set of times τ_i whose members are t_i such that *event*(t_i, S_i) is satisfied. Similarly, define the set τ_j for which *event*(t_j, S_j) is satisfied. The intersection of the **reference events** for the sets of times τ_i and τ_j will be the set of common reference events, say ρ.

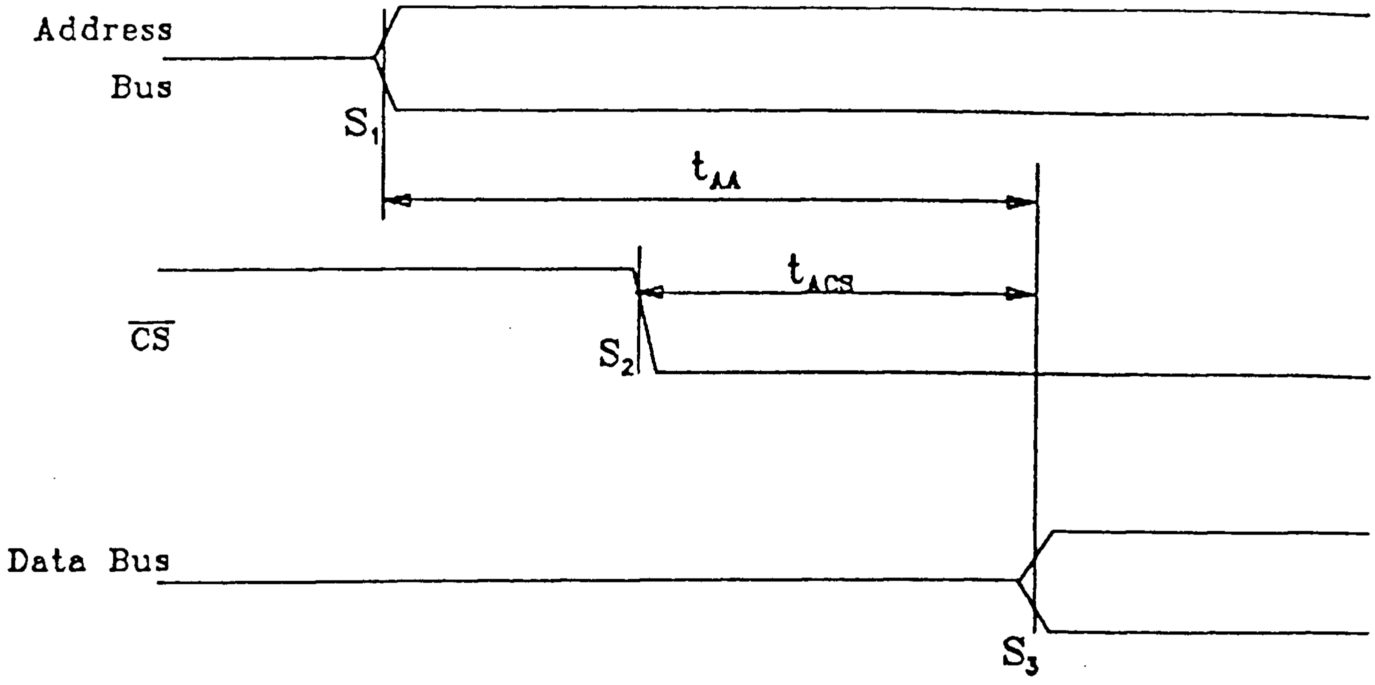


Figure 7.6: A simplified timing diagram of a typical static RAM read cycle. Data bus is valid at most t_{AA} after address bus valid *and* t_{ACS} after $\overline{CS}$ valid.

For example, consider Figure 7.6, a simplified timing diagram of a memory read of a static RAM. The three events described are:

- S_1 = state([ram,N,addressbus],valid)

- S_2 = state([ram,N,cs_],asserted)
- S_3 = state([ram,N,databus],valid)).

For the event E_i =event(t_3,S_3), the set of t_i that satisfies the condition is t_3=$[S_1, t_{AA}]$ and $t_3 = [S_2, t_{ACS}]$. Let this set be τ_i. The reference events in the set of times τ_i are then S_1 and S_2. Suppose for another event, E_j, with event(t_j, S_j) giving the set $\tau_j = \{[S_1, t_{AVSL}]\}$, say, then the common event between the two sets of time τ_i and τ_j is then ρ=$\{S_1\}$, or *state([ram,N,addressbus],valid)*. Note that we are interested in the reference only. The parameters will give us the quantitative values, which is not important at the moment.

There are two situations to consider for the set ρ, namely for $\rho = \emptyset$, an empty set, and $\rho \neq \emptyset$.

7.3.1 Common References Exists

First, consider the case $\rho \neq \emptyset$. This means a potential common reference exists. We may then proceed to determine the quantitative value of the times of with common reference. That is, determine the times for which *value* is defined for both events, say E_i and E_j. In the previous example, then, we would be looking for value($[S_1, t_{AA}]$,v_1) and value($[S_1, t_{AVSL}]$,v_2), where v_i are the time range of the respective parameters.

If the *value* of the time is defined for only one of the two events of interest, the subgoal of determining the value of the other event is added to the solution. When this subgoal is satisfied, the goal of determining the value of T_i can also be satisfied, given T_i is defined as event($[S_j, T_i]$,S_i) as before. If *value* is not defined for any of the common events, then two subgoals are generated to determine the value relative to an event selected from ρ. Domain dependent heuristics may be employed here to determine which of the common events to pursue in case where there are more than one common events. For example, suppose ρ = {state([cpu,1,clk],s0), state([cpu,1,addressbus],valid)}, then we may choose to use the *clk* (clock) signal instead of the the address bus signal. The *clk* signal might be chosen over the address bus signal because the *clk* is a well defined, periodic signal. This rule is heuristical because cases may exist where either reference event can be used, but the address bus will yield a more optimal solution, using fewer parameters (i.e. a shorter solution path).

7.3.2 No Common Reference

In the case $\rho = \emptyset$, one of the sets of time τ_i or τ_j, will be selected and pursued. Note first that for a given set, say τ_i, all the times in the set are equal. Each element t_i of the set τ_i satisfies *event*(t_i, S_i). S_i occurs at only one point on the timeline. Hence, pursuing any one time t_i in the set τ_i will establish the actual quantitative time value of S_i; that is, the time when S_i occurs, relative to some reference S_k, where $t_i = [S_k, T_k]$. Suppose we select a time $t_k = [S_k, T_k]$ from the set τ_i to pursue. Now, the problem is reduced to finding a common reference between S_j and S_k instead of S_i and S_j. This has now been reduced to the original problem. This line of reasoning is illustrated by the following example.

Consider, for example, Figure 7.7. Suppose we wish to determine the value of the time t_4, or equivalently, value($[S_3, t_4]$,v_4), defining the event event($[S_3, t_4]$,S_1). Then, the two states of interest are S_3 and S_4. The *events* defined are event($[S_0, t_1]$,S_1) and event($[S_2, t_3]$,S_3). The set $\tau_1 = \{[S_0, t_1]\}$ and $\tau_2 = \{[S_2, t_3]\}$. Then, the set of the intersection of reference events $\rho = \emptyset$. If we then choose $[S_2, t_3]$ from τ_2, we have reduced the problem to finding a common reference between S_1 and S_2, since we know we can get the value for S_3 relative to S_2. Between S_1 and S_2, we are able, in this example, to find the common reference S_0, as shown in Figure 7.7.

Why is the time $[S_2, t_3]$ is selected, or in fact, why is the set τ_2 selected instead of,

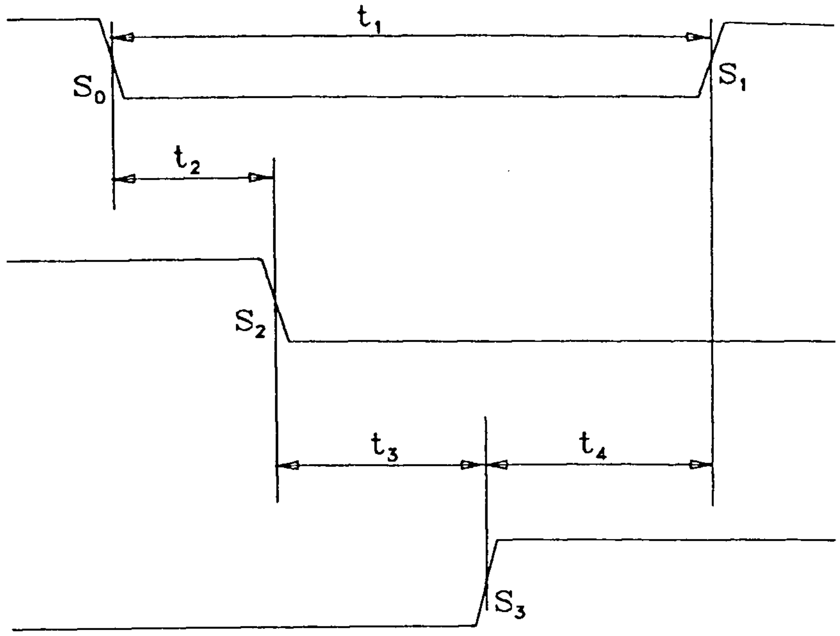


Figure 7.7: A typical timing diagram with of 4 events. The timing parameter t_1 is equivalent to $t_2 + t_3 + t_4$.

say, τ_1 in the above example? The decision may be arbitrary in general, or, more likely, based on some domain dependent heuristics. These heuristics are the same sets of rules an expert might use to solve problems in the domain, and are an integral part of the expert knowledge. In our domain, for example, the CPU clock is a key reference signal. If the S_j, say, is a clock signal state while S_i is another signal state, then we would use the set τ_j since S_j is a well defined state.

These heuristic rules might also be an interpretation of the representation to allow the machine to "see" what is obvious. For example, a human expert (or any human being, for that matter), can see immediately that the common reference between S_1 and S_3 is S_0 in Figure 7.7. This fact is not "obvious" to a machine expert. The line of reasoning may be: S_2 is a control signal that triggers S_1, and S_1 a signal generated by the CPU, say. Then, base on either fact (or both), the expert may conclude that there is likely to be a definition of S_2 relative to S_1, but not the other way around. Human deduction is aided by a visual interpretation not available to the machine expert.

The following example will demonstrate the issues outlined above more clearly.

7.4 Determining t_{DICL}: An Example

To give an example from Assignment One shown in Appendix A, one of the subproblems the student must solve is to determine that the data-in set up time, t_{DICL}, of a memory read cycle. The data-in set up time is defined in the timing diagram in Figure A.9 in Appendix A. The t_{DICL} parameter is established by the statement:
`event([state([cpu,1,databus([D0,D1])]`[1]`,valid),tDICL],state([cpu,1,clk],s7))`.
That is, the CPU clock enters bus cycle state S7 t_{DICL} after the data bus is valid. The two events of interest are thus: `state([cpu,1,clk],s7)`, when the clock of the CPU enters bus cycle state S7; and `state([cpu,1,databus],valid)`, when the data bus of the CPU is valid. The event `state([cpu,1,clk],s7)` is a clock signal event. It is defined with respect to itself and does not need to be pursued any further to find a reference. We need only find when data bus is valid with respect to a clock state and the problem is solved.

7.4.1 Determining When Data Bus Valid

Since the data is input to the CPU, the event `state([cpu,1,databus],valid)` cannot be determined directly, but can only be determined indirectly, via the CPU's connections. By the connection definitions of the circuit, the CPU data bus and the RAM data bus signals are identical, hence, the events `state([cpu,1,databus],valid)` and `state([ram,1,databus],valid)` are equal. Then, the objective *determine when [cpu,1,databus] valid* is equivalent to *determine when [ram,1,databus] valid.* Production

[1]For buses, an argument is used to indicate the range of the lines that are involved. This involves the address bus and data bus only. The argument will be left out in the rest of this work for the sake of brevity.

rules exist to solve the latter goal. There are three timing parameters given for the μPD43256 static RAM, the RAM used in the circuit, concerning when the data bus is valid, namely: t_{ACS}, chip select asserted to data bus valid; t_{AA}, address bus valid to data bus valid; and t_{OE}, output enable asserted to data bus valid. There are no direct common reference events between any of the data bus valid times and clock low at S7. Since the clock event is a "regular" event as stated above, the data bus valid times are pursued. To select which one of the three timing parameters to pursue, it is necessary to invoke the fact that the $\overline{CS}$ signal controls both the address decoder and $\overline{OE}$ signals going into the RAM array for the device being used. This fact implies that $\overline{CS}$ must be asserted for the device to begin working. Hence, these timing parameters are actually referenced as conjunctive events with the $\overline{CS}$ signal, rather than as a single event, say address bus valid for t_{AA}. The rule to resolve which parameter to use is thus determined by when the address bus, $\overline{CS}$, and $\overline{OE}$ is valid or asserted. In this example, we find $\overline{CS}$ is asserted after the address bus is valid, thus, the parameter to use is t_{ACS}.

7.4.2 Determining When $\overline{CS}$ Asserted

Going backwards through the circuit, we can establish that the output of the OR-gate [2] connected to $\overline{CS}$ must be asserted. This assertion implies that the OR-gate output goes low, since $\overline{CS}$ is an active low signal. This in turn implies that all the inputs of the OR-gate must also be low. This is a conjunct of two events for a two input OR-gate, in which we must establish the value of each in order to select the latest event. A propagation delay of the logic gate may be introduced here without affecting the reasoning process. Continuing the backtracking, we can establish that the $\overline{DS}$ signal and one of the OR-gate inputs are equal by connection, and the 'LS138 line decoder output is equal to the other OR-gate input. The $\overline{DS}$ signal timing is a well defined CPU parameter and requires

[2] By DeMorgan's rule, the AND-gate with negated inputs and outputs can be replaced by an OR-gate.

no further pursuit. The line decoder output requires that the device be activated. The 'LS138 is activated by satisfying the conjunctive event of having all of its three enable inputs in the enable state, namely, $\overline{e0}$ and $\overline{e1}$ at logical low and $e2$ at logical high. $\overline{e1}$ is connected to ground and $e2$ is connected to power, so these will always assume the enabling state. The third enable, $\overline{e0}$, is connected to $\overline{AS}$ of the CPU, which is also a well defined CPU signal. Hence, we can now propagate this information back down the chain, and establish the value of the t_{ACS} for this circuit.

7.4.3 Determining the Value of t_{DICL}

Both the $\overline{AS}$ and $\overline{DS}$ signals assert at the same time, t_{CHSL} from bus cycle state S2, and have value `value([state([cpu,1,clk],s2),tCHSL],[3,60])` [3]. By introducing a propagation delay of `value([_,tgate_delay],[0,10])` [4] the 'LS138 output will be active in the time range $< 3, 70 >$ ns from state S2. The OR-gate will introduce a second propagation delay, and the output goes low in $< 3, 80 >$ ns after S2. This is the time when the RAM $\overline{CS}$ signal asserts. Using `value([state([ram,1,cs_],tACS],[undefined,100])` [5], the data bus valid event can be established at time $< undefined, 180 >$ ns. Note that whenever an *undefined* value enters a calculation, the result is also undefined, since we do not know the value. The value of the other event of interest, `state([cpu,1,clk],s7)`, with respect to `state([cpu,1,clk],s2)`, can easily be established to be $< 312.5, 625 >$ ns at a frequency of 8MHz. Then, the t_{DICL} value is thus $< 312.5-180, 625-undefined >$, or $< 132.5, undefined >$ ns. Comparing this with the constraint on t_{DICL} $< 10, undefined >$, the constraint is satisfied with a large margin.

[3]All values are in nanoseconds (ns).

[4]The "_" is a variable that can be bind to any atoms in Prolog implementation. Here, it can be interpreted as the reference event is unimportant.

[5]Only the maximum for t_{ACS} is given on this RAM's data sheet, as the minimum is not important as a constraint.

7.5 The Human Solution

The expert model is an attempt to emulate the problem solving process of the human expert. In solving the example problem, the human expert is able to do so by "simply" reading off the relevant parameters on a timing diagram. However, when constructing the timing diagram, the human expert must first discover the relations between the signals through the connections in the circuit. The reasoning process of following the connections and then producing the timing diagram is similar to the process used by the expert model proposed. While the human expert produces the timing diagram, the expert model dynamically constructs the collection of *event* statements used in the solution as it traces through the circuit.

Chapter 8

Implementation Results

8.1 The Expert

Following the approach outlined in previous chapters, we have implemented a simple expert using the time-range representation and demonstrated how the reasoning process works with the example given in the previous chapter.

The expert was implemented in Advance A.I. Systems' Prolog^{TM} in the Edinburgh syntax. The expert is invoked to solve the timing problem via the command *simulate/0*, which performs a simulation of the circuit. Before *simulate/0* is issued, the user must set up the problem in the solution database. The problem is defined primarily by the goals the problem is attempting to solve and the circuit involved. Appendix C shows the file that sets up the solution database for the example of finding the value of the parameter t_{DICL} described in the previous chapter. The circuit is defined by a series of connections between nodes in the circuits as outlined in Chapter 5. The goals are initialized as *active* to let the expert use them with the production rules. Note that goals are solved concurrently, a task made possible by the use of the blackboard architecture. Goals that depend on the solutions of other goals in the database will maintain their dependence. These parent goals will not have enough solution elements in the database to execute their productions until their subgoals have been solved.

8.1.1 Simulation

The simulation cycle consists of three parts: suggestion of productions that may be executed, execution of the conflict set productions, and elimination of duplicate elements on the database.

When asked to suggest productions to execute, the expert proceeds to attempt to match the rules' goals and conditions with active goals and solution elements in the databases. The head of the rule *concept(Name,ID,Info)*, the RHS of the rule, and the solution element tags that satisfy the LHS are put in the conflict set database when the LHS of the rule is completely satisfied as stated in Chapter 6. Each production rule's LHS pattern is matched against all available solution elements until the solution elements are exhausted. When all the productions in the database have been processed, the expert then proceeds to filter the conflict set.

Filtering the conflict set means reducing the size of the conflict set, eliminating some of the rules in the conflict set that may be redundant or do not contribute to the problem solution. Currently, the expert will remove any rules that had been fired for the same reasons to prevent futile production executions and infinite loops. Additional criteria can be added by simply replacing the filter program. Some additional criteria commonly used in conventional expert systems include *recency* and *specificity* filtering [12]. Recency filtering removes rules that are selected using older solution elements (i.e. solutions elements added at an earlier time to the solution). Specificity filtering simply states that rules that have more specific conditions are preferred over those that are less specific.

8.1.2 Rule Execution

After the conflict set has been filtered, all the rules remaining will be executed. There are two reasons why all the rules are fired instead of only a single rule, as in the case in conventional expert systems. First, goals in the solution database are solved concurrently. Firing a single rule can only solve a single objective. Second, more than one rule solving a single goal may be true at the same time, these constituting different solution paths that may be taken. The presence of the various paths in the database enables the advisor to compare the student's solution with the expert's more effectively. This implementation is not without its drawbacks, however. Interactions between rules can become a major problem. We have approached this problem by not eliminating any unique solution elements in the solution database once they were put there. In a blackboard system, each production rule is independent of every other production rule. Rules interact solely via the blackboard. Their actions will interfere with each other only when one rule eliminates an element from the blackboard that satisfies a condition in another rule. When the former rule is executed, the advisor will not be able to trace back and explain why the latter rule was executed, since one of its condition elements was erased from the blackboard. Hence, we decided not to allow any rule to erase any elements from the blackboard, and only a limited amount of modification is allowed. We do not believe this is a great hindrance in the construction of the rules. Any element that is in the solution database should be a true fact to begin with. Additional rules may be introduced to resolve ambiguous facts. For example, there is a set of rules in the current database which determines, for the μPD43256 static memory, whether the address bus is valid before the chip select is asserted (see Appendix B). This fact is used to determine which parameter will be used in the timing calculations. For this implementation, only the selected parameter has its *value* added to the database.

Assumptions can be incorporated by stating them explicitly as solution elements. Solution deductions may continue with a set of assumptions, and the resulting solution elements that are proven inconsistent at the end need only be labelled so. The advisor can then say a fact is inconsistent with the given because certain assumptions were made which led to the conclusion.

The RHS has two components that need to be executed when the rule is fired: New properties that were deduced and new subgoals that need to be solved. The properties are Prolog statements that can either be proven as a Prolog program, or facts that are put in the solution blackboard. An important note in the properties is a *solves(Goal)* statement which must be present to indicate that the rule solves the current goal. The RHS of a rule being satisfied and executed does not automatically imply that the current goal is solved. Several subgoals may first need to be solved before the current goal can be solved. Thus, a rule that solves a goal is made explicit as part of the rule's properties. This information may also be carried in the information list in the rule's header, but we have adopted the former approach to simplify the implementation. In fact, the information list which appears as the third argument in the rule header is not used by the expert at all in the current version of implementation. The new goals are added to the solution blackboard as active objectives. These goals will be attempted to be matched in the following cycle of the simulation.

8.1.3 Tidying Up

Finally, after all the rules have been executed, duplicate new goals may appear on the solution blackboard as well as goals that has already been solved. These redundant goals are erased from the blackboard. This action does not affects the solution in any way. There is no point in solving the same goals more than once.

The simulation cycle continues until the suggested conflict set size is zero – that is, no more productions can be executed. This condition can only occur in two instances. First, when all the active objectives are solved, then the problem is solved, and no more productions need to be executed. In this case, the simulation is successful. In the second case, there are still one or more active goals in the database. In this case, the production set is insufficient to solve the problem posed and the simulation will return a failure.

8.2 Solving Goals

When a goal is put in the database, no productions may satisfy that goal. However, we may find an equivalent goal for which there is a production rule that can be satisfied. For example, for the circuit in Appendix A, one of the subgoals encountered in solving the timing problem is to determine when one of the inputs to the OR-gate is low, *determine when [or,1,in1] low*. In general, no production would exist for this goal, since there are no specifications on inputs of devices. However, we note that the node [or,1,in1] is connected to [cpu,1,lds_], the $\overline{LDS}$ of the CPU. Thus, this goal is equivalent to solving the goal *determine when [cpu,1,lds_] low*. To continue, although we may specify a rule for when $\overline{LDS}$ is low, it is more appropriate in an explanatory system to specify when $\overline{LDS}$ is asserted and note that $\overline{LDS}$ is an active low signal. Note that we require $\overline{LDS}$ to be asserted regardless of whether $\overline{LDS}$ is an active high or an active low signal. If we specify a rule to solve for when $\overline{LDS}$ is low, a different rule needs to be specified if $\overline{LDS}$ is active high. Furthermore, we can now say that the output is low because the signal is active low. Hence, because $\overline{LDS}$ is active low, the goal *determine when [cpu,1,lds_] asserted* is equivalent to our original goal of *determine when [or,1,in1] low* also. The equivalence of all these goals is noted in the database. Now, rules exist to solve the goal *determine when [cpu,1,lds_] asserted*, namely, the parameter rules t_{CHSL} and t_{AVSL} [1]. When these rules solve the *determine when [cpu,1,lds_] asserted* goal, they also solve the other two goals in the database. Thus, all these goals are marked as solved. The *solves* predicate will take a goal, mark it as solved, and trace and mark as solved all goals equivalent to

[1]Actually, the rules in the rule set does not distinguish between the lower and upper data strobe of the 68000, the $\overline{LDS}$ and $\overline{UDS}$ signals respectively. They are collectively referred to as $\overline{DS}$. See Appendix B

it which are still active.

8.3 Discussion of Results

The expert currently implemented uses a database of approximately 40 production rules. The rules are shown in Appendix B. These rules are made up of several subsets of rules, including a set of rules for the 68000 microprocessor timing parameters as well as for the μPD43256 static RAM. A number of *control rules* were defined to set up equivalent goals on the blackboard and to propagate events. Parts of some of the rules were written as provable Prolog clauses for efficiency. An example is the rule to propagate an established event. The state of a node is propagated to all other nodes connected to it and all equivalent objectives are set to *solved*. The implementation of the time comparison was also written entirely in Prolog code. The decision as to whether to encode a rule as a set of productions or as Prolog codes depended on an assumption of the user's basic level of understanding as well as efficiency. The user's level of understanding means simply what type of information should the user know already before using the system. For example, if an explanation says "event E_1 is *before* E_2", the meaning of the statement would be obvious to the human user. There is no need to explain what is meant by *before*. The term *before*, however, must be precisely defined for various conditions for the machine expert.

To solve the problem of determining t_{DICL}, approximately 55 solution elements were put on the blackboard initially. These elements are shown in Appendix C. Most of the elements simply defines the circuit connections and the clock signal relations. The final solution state contains over 380 solution elements and is shown in Appendix C as well. The simulation took about 15 iterations and up to 30 minutes to complete. This is unacceptably slow from a practical point of view. The major inefficiency is caused by the expert having to match all the LHS elements of all the rules in the database. An exhaustive matching is required when an explanation of why a rule or set of rules do or do not apply in a given solution state. This matching allows the generation of a list of goals that have failed. However, when the expert is simply solving a problem, it may be more efficient for the expert to give no further consideration to a rule when the goal of the rule is not active. The current version of the expert, however, uses the same matching program for both modes of operation. Furthermore, as the number of elements in the solution database increases, so does the time required per iteration since the expert has to test all the elements for each iteration. Improvements can be made in the expert by adopting a faster production rule matching algorithm.

8.3.1 Interface

We have not dealt with other parts of the ITS in this project in any depth due to the lack of time and resources, as this is still an infant project. Some preliminary experiments were done at the beginning of the project regarding the interface module and its integration with the rest of the system at the beginning of the project.

A simple CAI (computer assisted instruction) style system was set up using a hypertext system [27, 52] Hyperbase^{TM}. Hypertext, or more generally referred as hypermedia, systems are made up of a number of nodes and links. Each node is made up of a single page that can be presented on the screen and the links provides association between the nodes. Students from ELEC464 were asked to use and compare solutions for assignment one presented in hypertext with conventional posted solutions. The reaction were mixed. Generally, the "flashcard-style" presentation of the hypertext system was well received, but there were a number of complaints that the layout of the screen was confusing. After an exchange with some of these students, we concluded that the problems were mainly caused by our inexperience in designing the system rather than any intrinsic problem with the hypertext itself. Work in this area was continued by two fourth year students using the HyperCard^{TM} [26] implementation of hypertext on the Apple Macintosh II^{TM}.

Chapter 9

Conclusions

A general time point representation using a multiple-referenced timeline and imprecise time range values has been developed to facilitate temporal concept deductions and explanations. This representation framework was incorporated into an intelligent advisor system being developed to facilitate instruction to students in tutorial problem solutions. A prototype expert module of the system was implemented to demonstrate the effectiveness of this knowledge representation framework. The expert was implemented as a goal-oriented production rule system, and demonstrated its problem solving techniques in a CPU-memory timing subproblem. The production rule set currently contains approximately 40 production rules.

The expert is by no means problem free, however. The production rules, although written in the form of Prolog clauses, are not being "proved" as normal Prolog clauses are. The expert actually finds the rule, takes it apart, and attempts to unify its components individually. The reason for not using the built-in inference engine directly is to allow the expert to keep track of the components of the LHS of the production which failed. Furthermore, this prototype expert is also relatively slow in performing a complete simulation.

9.1 Future Directions & Recommendations

With the knowledge representation established and the system architecture specified, other components of the system can now begin to be developed.

The expert itself requires some improvement. The production rule matching algorithm employed at the moment is far too slow in problem solving. A quicker, more efficient algorithm or approach needs to be developed for practical applications. The use of off-the-shelf expert system shells, such as CLIPS, should also be considered in the short term for the project. Integration of these off-the-shelf expert systems with the ITS is a major technical problem that must be resolved. Expansion and completion of the production rule set is also required in the near future.

An associated project in developing the interface module was done by two graduating students with HyperCard^{TM} [26] on the Macintosh II^{TM} microcomputer. Hypermedia provides an excellent environment for user and machine interaction [27, 52]. Rapid prototyping of the interface module is possible. Integration between the interface and the expert module should be one of the next major steps in this project to provide a functioning prototype system.

We have not addressed any issues concerning the tutor or tutoring approach in this work. These issues should form an important part of the system's development and requires attention in the near future also.

Appendix A

Assignment 1

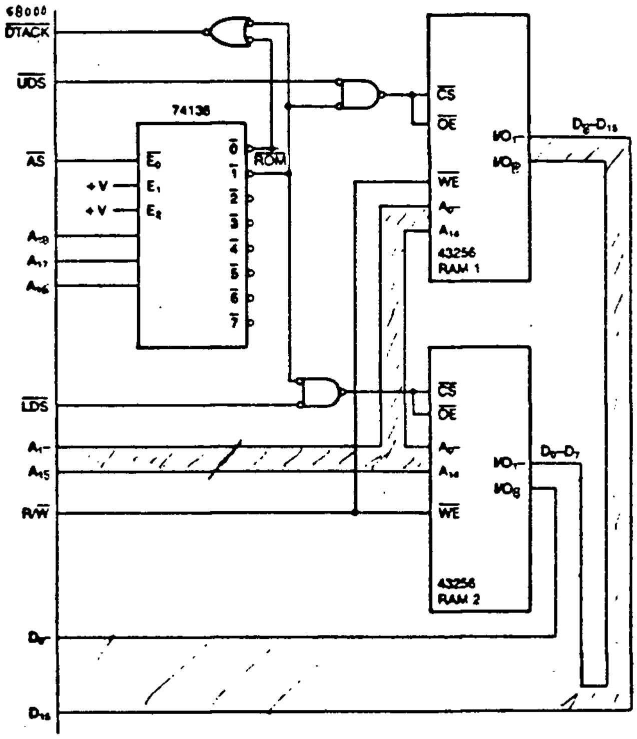
Figure A.8: Assignment one of ELEC464 from the fall 1989 session. (next page)

University of British Columbia
Department of Electrical Engineering
ELEC 464 (Fall 1988): Assignment 1

The due date for this assignment is *September 29, 1988 (Thursday).* Please submit your solutions to the EE464 box outside the Systems Lab (room 332) on or before the due date.

Problem 1:

The μPD43256 is a family of 32K × 8 CMOS static RAMs which have an access time ranging from 100 ns to 150 ns. Consider the following circuit in which two 43256 are connected to a 68000 CPU running at 8 MHz.



(a) Which type of address decoding strategy (or strategies) is being used in this circuit? Give your reasons. What are the memory address ranges (in hexidecimal) for RAM1 and RAM2?

(b) Draw the read cycle and write cycle timing diagrams for the 68000–μPD43256-10L combination and determine whether any timing restrictions have been violated.

(c) If the clock frequency of the 68000 is changed to 12.5 MHz and two μPD43256-15L are used for RAM1 and RAM2, will the given circuit still work? Give your reasons.

Problem 2:

Figures 4.21a and b of your textbook give the ciruit diagrams of two typical $\overline{DTACK}$ generators.

(a) Draw suitable timing diagrams to illustrate the operation of these $\overline{DTACK}$ generators.

(b) Show how you would integrate the $\overline{DTACK}$ generator with the circuit shown in Fig. 4.11 of your textbook.

(c) For each $\overline{DTACK}$ generator, estimate the number of wait states inserted in a read bus cycle.

Problem 3:

The memory map for a small system is to be set up as follow:

ROM1S	00 0000 - 00 3FFF	supervisor ROM
ROM1U	00 0000 - 00 3FFF	user ROM
ROM2	00 4000 - 00 7FFF	
RAM1S	01 0000 - 03 FFFF	supervisor RAM
RAM1U	01 0000 - 03 FFFF	user RAM
PERI1	04 0000 - 04 00FF	
PERI2	04 0100 - 04 01FF	

The supervisor memory is accessible only when the CPU is in supervisor mode, and the user memory is accessible only in the user mode. ROM2, PERI1, and PERI2 are accessible in either mode. You may assume that 32K $\times$ 1 RAMs and 8K $\times$ 1 ROMs are available as well as discrete logic components. Design an address decoder strobed with $\overline{AS}$ using:

(a) m-line to n-line decoders.

(b) 16 $\times$ 48 $\times$ 8 FPLA. Show all fused links and list all product and sum terms. You should try to minimize the parts count in both (a) and (b).

(c) What are the advantages and disadvantages of each decoder?

(*Hint: consider FC0–FC2*)

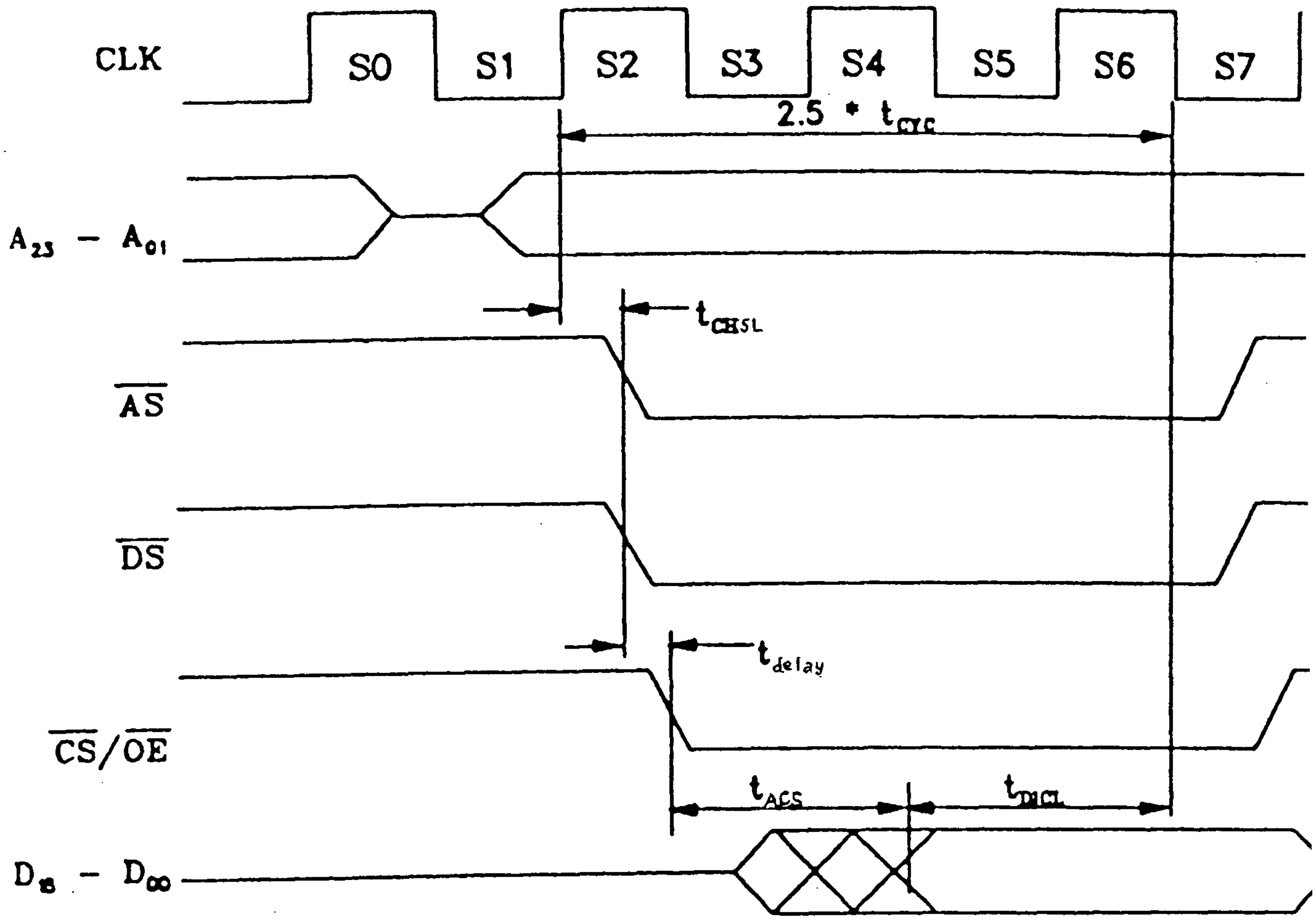


Figure A.9: A simplified timing diagram for problem one of assignment one.

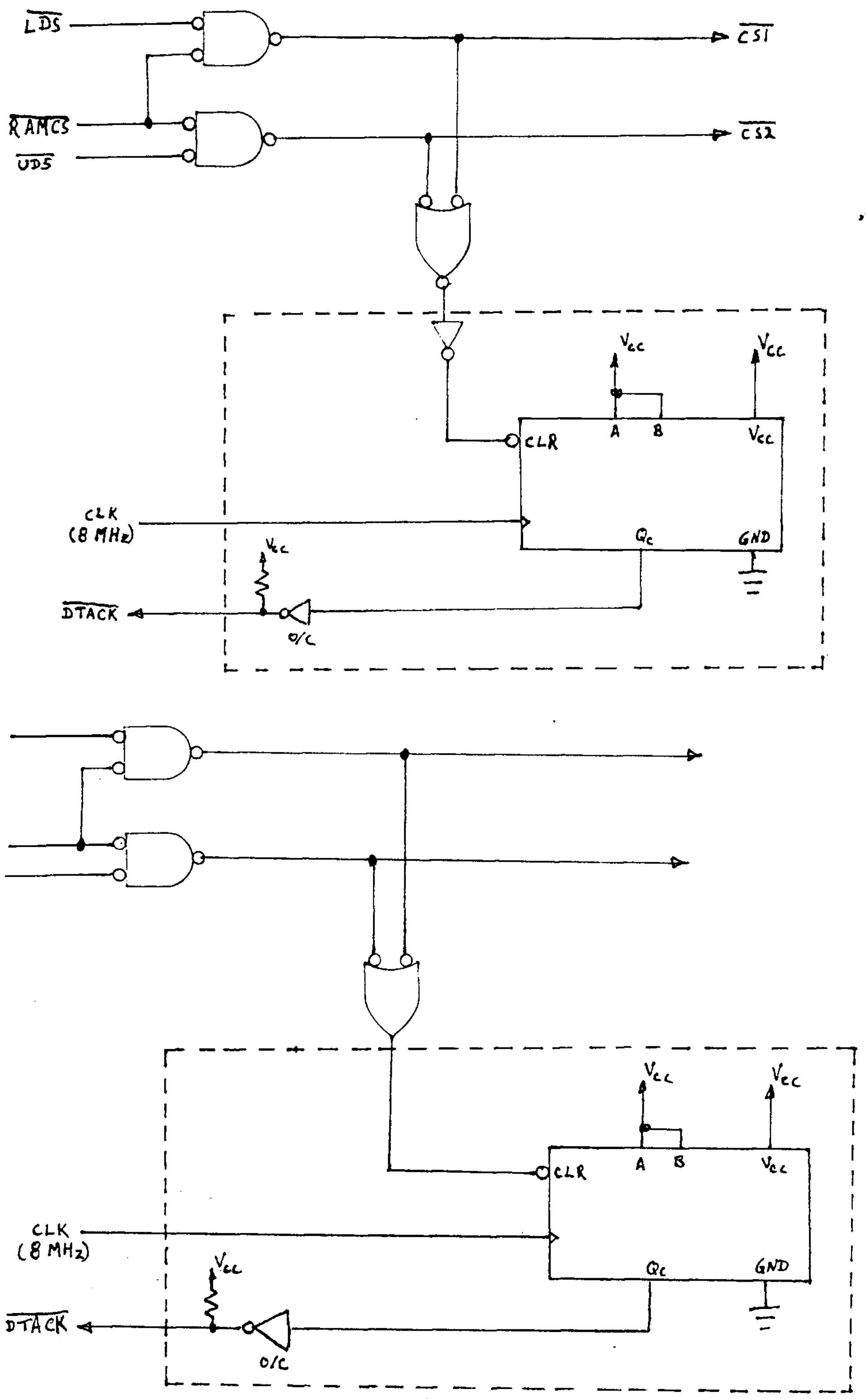


Figure A.10: Two solutions that satisfy question (2b) from assignment 1, integration of a shift register $\overline{DTACK}$ generator.

Appendix B

Production Rules

The following are the files containing the production rules. These files are not totally complete, but serves to prove the framework. The files are *68000.rul*, *upd43256.rul*, *time.rul*, *control.pro*, and *logic.pro*. *68000.rul* contains rules and facts about the MC68000 processor parameters. *UPD43256.rul* contains rules and facts about the NEC μPD43256 static RAM. *Time.rul* contains rules to deal with time determination. *Control.pro* contains rules to resolve equivalent goals. *Logic.pro* contains rules about logic gates and small scale integrated (SSI) devices.

```
%%%%%%%%%%%%%%%%%%%%%%%%%% 68000.rul %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%% domain knowledge: facts about MC68000 %%%%%%%%%%%%%%%

active([cpu,as_],low).

active([cpu,ds_],low).

active([cpu,lds_],low).

active([cpu,uds_],low).

active([cpu,dtack_],low).


% bus cycle state definition (clock)

:-

% Note: S7 and S0 of the previous and next bus cycle are different states

ps_store(_,event([state([cpu,N,clk],s7p),tCYC],state([cpu,N,clk],s1)),given),

ps_store(_,event([state([cpu,N,clk],s0),tCYC],state([cpu,N,clk],s2)),given),

ps_store(_,event([state([cpu,N,clk],s1),tCYC],state([cpu,N,clk],s3)),given),

ps_store(_,event([state([cpu,N,clk],s2),tCYC],state([cpu,N,clk],s4)),given),

ps_store(_,event([state([cpu,N,clk],s3),tCYC],state([cpu,N,clk],s5)),given),

ps_store(_,event([state([cpu,N,clk],s4),tCYC],state([cpu,N,clk],s6)),given),

ps_store(_,event([state([cpu,N,clk],s5),tCYC],state([cpu,N,clk],s7)),given),

ps_store(_,event([state([cpu,N,clk],s6),tCYC],state([cpu,N,clk],s0n)),given),

% half cycle times have to be given explicitly

ps_store(_,event([state([cpu,N,clk],s7p),'0.5*tCYC'],state([cpu,N,clk],s0)),

given),

ps_store(_,event([state([cpu,N,clk],s0),'0.5*tCYC'],state([cpu,N,clk],s1)),

given),

ps_store(_,event([state([cpu,N,clk],s1),'0.5*tCYC'],state([cpu,N,clk],s2)),

given),
ps_store(_,event([state([cpu,N,clk],s2),'0.5*tCYC'],state([cpu,N,clk],s3)),
given),
ps_store(_,event([state([cpu,N,clk],s3),'0.5*tCYC'],state([cpu,N,clk],s4)),
given),
ps_store(_,event([state([cpu,N,clk],s4),'0.5*tCYC'],state([cpu,N,clk],s5)),
given),
ps_store(_,event([state([cpu,N,clk],s5),'0.5*tCYC'],state([cpu,N,clk],s6)),
given),
ps_store(_,event([state([cpu,N,clk],s6),'0.5*tCYC'],state([cpu,N,clk],s7)),
given),
ps_store(_,event([state([cpu,N,clk],s7),'0.5*tCYC'],state([cpu,N,clk],s0n)),
given).

concept(clk,1,[object(m68000)]) :-
(current_goal([set,clock]),
 conditions([frequency(8)])
) ->
properties([
solves([set,clock]),
constraint([_,tCYC],[125,250]),
value([_,tCYC],[125,250]),
value([_,'0.5*tCYC'],[62.5,125])]).
concept(clk,2,[object(m68000)]) :-
(current_goal([set,clock]),
 conditions([frequency(12.5)])
) ->
```

```
properties([
solves([set,clock]),
constraint([_,tCYC],[80,250]),
value([_,tCYC],[80,250]),value([_,'0.5*tCYC'],[40,125])]).

% read cycle

% timing parameters: cpu generate & device constraint signals
% cpu generates
concept(tCLAV,1,[object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,addressbus([A1,A2])],valid]),
 conditions([
=(Addressbus,[cpu,N,addressbus([A1,A2])]),
  =(T,[state([cpu,N,clk],s1),tCLAV]),
  frequency(8)
  ])
)->
(properties([
solves([determine,when,[cpu,N,addressbus([A1,A2])],valid]),
  event(T,state(Addressbus,valid)),
  constraint(T,[undefined,62]),
  value(T,[undefined,62])
  ]),
 new_goals([[propagate,event(T,state(Addressbus,valid))]])).

concept(tCHSL,1,[object(mc68000)]) :-
```

```
(current_goal([determine,when,[cpu,N,as_],asserted]),
 conditions([
  =(T,[state([cpu,N,clk],s2),tCHSL]),
  frequency(8)
  ])
)->
(properties([
solves([determine,when,[cpu,N,as_],asserted]),
  event(T,state([cpu,N,as_],asserted)),
  constraint(T,[3,60]),
  value(T,[3,60])]),
 new_goals([[propagate,event(T,state([cpu,N,as_],asserted))]])).
concept(tCHSL,2,[object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,ds_],asserted]),
 conditions([
  =(T,[state([cpu,N,clk],s2),tCHSL]),
  frequency(8),buscycle(read)
  ])
)->
(properties([
solves([determine,when,[cpu,N,ds_],asserted]),
  event(T,state([cpu,N,lds_],asserted)),
  event(T,state([cpu,N,uds_],asserted)),
  constraint(T,[3,60]),
  value(T,[3,60])]),
 new_goals([[propagate,event(T,state([cpu,N,lds_],asserted))],
  [propagate,event(T,state([cpu,N,uds_],asserted))]])).
concept(tCHSL,3,[object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,ds_],asserted]),
 conditions([
  =(T,[state([cpu,N,clk],s4),tCHSL]),
  frequency(8),buscycle(write)
  ])
)->
(properties([
solves([determine,when,[cpu,N,as_],asserted]),
  event(T,state([cpu,N,lds_],asserted)),
  event(T,state([cpu,N,uds_],asserted)),
  constraint(T,[3,60])],
  value(T,[3,60])),
 new_goals([[propagate,event(T,state([cpu,N,lds_],asserted))],
  [propagate,event(T,state([cpu,N,uds_],asserted))]])).

concept(tAVSL,1,[object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,as_],asserted]),
 conditions([
=(Addressbus,[cpu,N,addressbus([A1,A2])]),
  =(T,[state(Addressbus,valid),tAVSL]),
  frequency(8)
  ])
) ->
(properties([
solves([determine,when,[cpu,N,as_],asserted]),
  event(T,state([cpu,N,as_],asserted)),
  constraint(T,[30,undefined]),
  value(T,[30,undefined])]),
 new_goals([[determine,when,Addressbus,valid],
  [propagate,event(T,state([cpu,N,as_],asserted))]])).
concept(tAVSL,2,[object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,ds_],asserted]),
 conditions([
=(Addressbus,[cpu,N,addressbus([A1,A2])]),
  =(T,[state(Addressbus,valid),tAVSL]),
  frequency(8)
  ])
) ->
(properties([
solves([determine,when,[cpu,N,ds_],asserted]),
  event(T,state([cpu,N,lds_],asserted)),
  event(T,state([cpu,N,uds_],asserted)),
  constraint(T,[30,undefined]),
  value(T,[30,undefined])]),
 new_goals([[determine,when,Addressbus,valid],
  [propagate,event(T,state([cpu,N,lds_],asserted))],
 [propagate,event(T,state([cpu,N,uds_],asserted))]])).

% contraint
concept(tDICL,1,[object(mc68000)]) :-
```

```
(current_goal([check,setup,time,of,[cpu,N,databus([D1,D2])]]),
 conditions([
=(Databus,[cpu,N,databus([D1,D2])]),
  =(T,[state(Databus,valid),tDICL]),
  buscycle(read)
  ])
) ->
(properties([
solves([check,setup,time,of,[cpu,N,databus([D1,D2])]]),
event(T,state([cpu,N,clk],s7)),
constraint(T,[10,undefined])
]),
 new_goals([[determine,when,Databus,valid],
  [determine,value,of,T]])).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% UPD43256.rul %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% definition of NEC uPD43256 32K x 8 static ram %%%%%%%%%%%%
active([_,cs_],low).
active([_,oe_],low).
active([_,we_],low).

% timing parameters
% read cycle
% property definitions
concept(tAA,1,[object(static_ram)]) :-
current_goal([determine,when,[ram,N,databus([D1,D2])],valid]),
conditions([
=(Databus,[ram,N,databus([D1,D2])]),
=(Addressbus,[ram,N,addressbus([A1,A2])]),
  =(T,[state(Addressbus,valid),tAA]),
  buscycle(read),device([ram,N],'uPD43256-10L')
 ]) ->
(properties([
solves([determine,when,[ram,N,databus([D1,D2])],valid]),
  event(T,state(Databus,valid)),
  constraint(T,[undefined,100])]),
 new_goals([
  [propagate,event(T,state([ram,N,databus([D1,D2])],valid))],
  [determine,when,Addressbus,valid],
  [determine,when,[ram,N,cs_],asserted],
[determine,ealier,of,[state(Addressbus,valid),
state([ram,N,cs_],asserted)]]
])
).
concept(tACS,1,[object(static_ram)]) :-
current_goal([determine,when,[ram,N,databus([D1,D2])],valid]),
conditions([
=(Databus,[ram,N,databus([D1,D2])]),
  =(T,[state([ram,N,cs_],asserted),tACS]),
  buscycle(read),device([ram,N],'uPD43256-10L')
 ]) ->
(properties([
solves([determine,when,[ram,N,databus([D1,D2])],valid]),
  event(T,state(Databus,valid)),
  constraint(T,[undefined,100])]),
 new_goals([
  [propagate,event(T,state([ram,N,databus([D1,D2])],valid))],
  [determine,when,[ram,N,cs_],asserted],
[determine,ealier,of,[state(Addressbus,valid),
state([ram,N,cs_],asserted)]]
])).
concept(tOE,1,[object(static_ram)]) :-
current_goal([determine,when,[ram,N,databus([D1,D2])],valid]),
conditions([
=(Databus,[ram,N,databus([D1,D2])]),
  =(T,[state([ram,N,oe_],asserted),tOE]),
  buscycle(read),device([ram,N],'uPD43256-10L')
 ]) ->
(properties([
solves([determine,when,[ram,N,databus([D1,D2])],valid]),
  event(T,state(Databus,valid)),
  constraint(T,[undefined,50])]),
 new_goals([
  [propagate,event(T,state([ram,N,databus([D1,D2])],valid))],
  [determine,when,[ram,N,oe_],asserted]
])
).

% rules determining whether tACS or tAA is used
concept(resolve,1,[]) :-
current_goal([determine,ealier,of,
[state(Addressbus,valid),state(CS_,asserted)]]),
conditions([
event(T1,state(Addressbus,S1)),functional_state([Addressbus,S1],valid),
event(T2,state(CS_,S2)),functional_state([CS_,S2],asserted),
before(T1,T2),
constraint([state(CS_,asserted),T],V)
 ]) ->
(properties([
solves([determine,ealier,of,
[state(Addressbus,valid),state(CS_,asserted)]]),
  equivalent([state(Addressbus,valid),state(CS_,asserted)],
   state(CS_,asserted)),
  value([state(CS_,asserted),T],V)]),
 new_goals([])
).
concept(resolve,2,[]) :-
current_goal([determine,ealier,of,
[state(Addressbus,valid),state(CS_,asserted)]]),
conditions([
event(T1,state(Addressbus,S1)),functional_state([Addressbus,S1],valid),
event(T2,state(CS_,S2)),functional_state([CS_,S2],asserted),
before(T2,T1),
constraint([state(Addressbus,valid),T],V)
 ]) ->
(properties([
solves([determine,ealier,of,
[state(Addressbus,valid),state(CS_,asserted)]]),
  equivalent([state(Addressbus,valid),state(CS_,asserted)],
   state(CS_,asserted)),
  value([state(Addressbus,valid),T],V)]),
 new_goals([])
).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%% time.rul %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% value exists with single common event
concept(determine_value,1,[]) :-
(current_goal([determine,value,of,[E2,T]]),
 conditions([
  event([E2,T],E1),
event([Er,Tr2],E2),value([Er,Tr2],[T2min,T2max]),
event([Er,Tr1],E1),value([Er,Tr1],[T1min,T1max]),
sub_val(T1max,T2min,Tmax),sub_val(T1min,T2max,Tmin)])
) ->
(properties([
solves([determine,value,of,[E2,T]]),
value([E2,T],[Tmin,Tmax])
]),
 new_goals([])).
concept(determine_value,2,[]) :-
(current_goal([determine,value,of,[E2,T]]),
 conditions([
  event([E2,T],E1),common_ref([E1,E2],Er),
eval_val(Er,E1,[T1min,T1max]),eval_val(Er,E2,[T2min,T2max]),
sub_val(T1min,T2max,Tmin),sub_val(T1max,T2min,Tmax)])
) ->
(properties([
solves([determine,value,of,[E2,T]]),
value([E2,T],[Tmin,Tmax])
]),
 new_goals([])).
concept(determine_value,3,[]) :-
(current_goal([determine,value,of,[E2,T]]),
 conditions([
  event([E2,T],E1),event([Er1,Tr1],E1),event([Er2,Tr2],E2),
 \==(Er1,Er2)])
) ->
(properties([]),
 new_goals([[find,common,reference,of,E1,E2]])).


% finding common reference
concept(common_ref,1,[]) :-
(current_goal([find,common,reference,of,E1,E2]),
 conditions([=(E1,state([cpu,N,clk],S1)),=(E2,state([cpu,N,clk],S2)),
  event(T1,E1),event(T2,E2),
  before(T1,T2)])
) ->
(properties([solves([find,common,reference,of,E1,E2]),
common_ref([E1,E2],E1)]),
 new_goals([])).
concept(common_ref,2,[]) :-
(current_goal([find,common,reference,of,E1,E2]),
 conditions([=(E1,state([cpu,N,clk],S1)),=(E2,state([cpu,N,clk],S2)),
  event(T1,E1),event(T2,E2),
  before(T2,T1)])
) ->
```

```
(properties([solves([find,common,reference,of,E1,E2]),
common_ref([E1,E2],E2)]),
 new_goals([])).

% heuristic, use clock as reference if possible
concept(common_ref,3,[]) :-
(current_goal([find,common,reference,of,E1,E2]),
 conditions([=(E1,state([cpu,N,clk],S1)),\==(E2,state([cpu,N,clk],_)),
  track(E1,E2,state([cpu,N,clk],S0))])
) ->
(properties([
solves([find,common,reference,of,E1,E2]),
common_ref([E1,E2],state([cpu,N,clk],S0))]),
 new_goals([])).
concept(common_ref,4,[]) :-
(current_goal([find,common,reference,of,E1,E2]),
 conditions([\==(E1,state([cpu,N,clk],_)),=(E2,state([cpu,N,clk],S2)),
  track(E1,E2,state([cpu,N,clk],S0))])
) ->
(properties([solves([find,common,reference,of,E1,E2]),
common_ref([E1,E2],state([cpu,N,clk],S0))]),
 new_goals([])).
```

```
%%%%%%%%%%%%%%%%%%%%%%%% control.rul %%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%% control rules  %%%%%%%%%%%%%%%%%%%%%%%%%

% create equivalent goals for connected nodes

concept(equivalent_goal,1,[object(device)]) :-

(current_goal([determine,when,Node,State]),

 conditions([

  =(Goal,[determine,when,Node,State]),

  setof(N,R^connected(Node,N,R),Nodes),

  equiv_goals(Goal,Nodes,3,NewGoalList)

  ])

) ->

(properties([

equiv_goals(Goal,NewGoalList)]),

 new_goals(NewGoalList)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% create a list of goals because Nodes are connected

% Prolog program to improve efficiency


equiv_goals(_,[],_,[]) :- !.

equiv_goals(Goal,[N|NList],I,[NewGoal|NewGoalList]) :-

substitute(N,Goal,I,NewGoal),

equiv_goals(Goal,NList,I,NewGoalList).


% asserts the property Goal & NewGoalList goals are equivalent

equiv_goals(_,[]) :- !.
equiv_goals(Goal,[G|GList]) :-
(ps_recorded(_,equivalent(Goal,G),_);
 (ps_recorded(_,equivalent(G,Goal),_);
  ps_store(_,equivalent(Goal,G),
concept(equivalent_goal,1,[object(device)]))
 )
),equiv_goals(Goal,GList).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create equivalent goals for similar states
concept(equivalent_goal,2,[object(device)]) :-
(current_goal([determine,when,Node,State]),
 conditions([
  functional_state(State),
  logical_state([Node,State],S),\==(S,unknown)])
) ->
(properties([
equivalent([determine,when,Node,State],
[determine,when,Node,S])
]),
 new_goals([[determine,when,Node,S]])).
concept(equivalent_goal,3,[object(device)]) :-
(current_goal([determine,when,Node,State]),
 conditions([
  logical_state(State),
  functional_state([Node,State],S),\==(S,unknown)])
) ->
(properties([
equivalent([determine,when,Node,State],
[determine,when,Node,S])
]),
 new_goals([[determine,when,Node,S]])).

% use ds* instead of lds*/uds*
concept(equivalent_goal,4,[object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,lds_],S]),
 conditions([])
) ->
(properties([
equivalent([determine,when,[cpu,N,lds_],S],
[determine,when,[cpu,N,ds_],S])]),
 new_goals([[determine,when,[cpu,N,ds_],S]])).
concept(equivalent_goal,5,[object(mc68000)]) :-
(current_goal([determine,when,[cpu,N,uds_],S]),
 conditions([])
) ->
(properties([
equivalent([determine,when,[cpu,N,uds_],State],
[determine,when,[cpu,N,ds_],S])]),
 new_goals([[determine,when,[cpu,N,ds_],S]])).

% event propagation
```

```
concept(propagate,1,[type(achieve),object(mc68000)]) :-
(current_goal([propagate,event(T,state(Node,S))]),
 conditions([])
) ->
(properties([
solves([propagate,event(T,state(Node,S))]),
propagate(T,Node,S)]),
 new_goals([])).

concept(solved_goal,1,[type(achieve)]) :-
(current_goal([determine,when,Node,S]),
 conditions([event(T,state(Node,S))])
) ->
(properties([
solves([determine,when,Node,S])]),
 new_goals([])).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% logic.rul %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- ps_store(_,value([_,tgate_delay],[0,10]),'propagation delay').


concept(power,1,[object(power)]) :-
(current_goal([power,on]),
 conditions([])
) ->
(properties([solves([power,on]),propagate_power]),
 new_goals([])).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% initialize nodes connected to vcc and gnd
propagate_power :-
bagof(N,I^R^connected([power,I,vcc],N,R),NHi),set_state_high(NHi),
bagof(N,I^R^connected([power,I,gnd],N,R),NLo),set_state_low(NLo).
set_state_high([]) :- !.
set_state_high([N|Ns]) :-
ps_store(_,event([[],'-inf'],state(N,high)),connected([power,_,vcc],N)),
set_state_high(Ns).
set_state_low([]) :- !.
set_state_low([N|Ns]) :-
ps_store(_,event([[],'-inf'],state(N,low)),connected([power,_,gnd],N)),
set_state_low(Ns).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2-input OR-gate definitions
```

```
concept(or_gate,1,[object(logic_gate)]) :-
(current_goal([determine,when,[or,N,out],high]),
 conditions([
logical_state(T,[or,1,in1],high),
=(T1,[state([or,1,in1],high),tgate_delay])
])
) ->
(properties([
solves([determine,when,[or,N,out],high]),
  event(T1,state([or,N,out],high)),
  propagate(T1,[or,N,out],high)]),
 new_goals([])).
concept(or_gate,2,[object(logic_gate)]) :-
(current_goal([determine,when,[or,N,out],high]),
 conditions([
logical_state(T,[or,1,in2],high),
=(T1,[state([or,1,in2],high),tgate_delay])
])
) ->
(properties([
solves([determine,when,[or,N,out],high]),
  event(T1,state([or,N,out],high)),
  propagate(T1,[or,N,out],high)]),
 new_goals([])).
concept(or_gate,3,[object(logic_gate)]) :-
(current_goal([determine,when,[or,N,out],low]),
 conditions([
logical_state(T,[or,1,in1],low),
logical_state(T,[or,1,in2],low),
=(T1,[state([or,1,in1],low),tgate_delay])
])
) ->
(properties([
solves([determine,when,[or,N,out],low]),
  event(T1,state([or,N,out],low)),
  propagate(T1,[or,N,out],low)]),
 new_goals([])).
concept(or_gate,4,[object(logic_gate)]) :-
(current_goal([determine,when,[or,N,out],low]),
 conditions([
logical_state(T,[or,1,in2],low),
logical_state(T,[or,1,in1],low),
=(T1,[state([or,1,in2],low),tgate_delay])
])
) ->
(properties([
solves([determine,when,[or,N,out],low]),
  event(T1,state([or,N,out],low)),
  propagate(T1,[or,N,out],low)]),
 new_goals([])).

concept(or_gate,5,[object(logic_gate)]) :-
```

```
(current_goal([determine,when,[or,N,out],low]),
 conditions([])
) ->
(properties([]),
 new_goals([[determine,when,[or,N,in1],low],
      [determine,when,[or,N,in2],low]])).
concept(or_gate,6,[object(logic_gate)]) :-
(current_goal([determine,when,[or,N,out],high]),
 conditions([])
) ->
(properties([]),
 new_goals([[determine,when,[or,N,in1],high]])).
concept(or_gate,7,[object(logic_gate)]) :-
(current_goal([determine,when,[or,N,out],high]),
 conditions([])
) ->
(properties([]),
 new_goals([[determine,when,[or,N,in2],high]])).

%%%%%%%%%%%%%%%%%%%%%%%%%% LS138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
active([ls138,y0_],low).
active([ls138,y1_],low).
active([ls138,y2_],low).
active([ls138,y3_],low).
active([ls138,y4_],low).
active([ls138,y5_],low).
active([ls138,y6_],low).

active([ls138,y7_],low).

active([ls138,e0_],low).

active([ls138,e1_],low).

active([ls138,e2],high).


concept(ls138,1,[object(device)]) :-
(current_goal([determine,when,[ls138,N,y1_],asserted]),
 conditions([
  logical_state(T,[ls138,N,e0_],low),
  logical_state(T,[ls138,N,e1_],low),
  logical_state(T,[ls138,N,e2],high),
  =(T1,[state([ls138,N,e0_],low),tgate_delay])])
) ->
(properties([
solves([determine,when,[ls138,N,y1_],asserted]),
event(T1,state([ls138,N,y1_],asserted)),
propagate(T1,[ls138,N,y1_],asserted)
]),
 new_goals([])).
concept(ls138,2,[object(device)]) :-
(current_goal([determine,when,[ls138,N,y1_],asserted]),
 conditions([])
) ->
(properties([]),
 new_goals([
  [determine,when,[ls138,N,e0_],low],
  [determine,when,[ls138,N,e1_],low],
  [determine,when,[ls138,N,e2],high]
  ])
 ).
```

Appendix C

The Solution Database

The following is the file that sets up the example problem and snapshot of the solution database at various stages of problem solving.

```
%%%%%%%%%%%%%%%%%%%%%%%%% file: problem.pro %%%%%%%%%%%%%%%%%%%%%%%%
% defines example problem to find tDICL

% define problem givens
% define circuit
circuit :-
   connect([cpu,1,as_],[ls138,1,e0_]),
   connect([power,1,gnd],[ls138,1,e1_]),
   connect([power,1,vcc],[ls138,1,e2]),
   connect([cpu,1,addressbus(a16)],[ls138,1,a]),
   connect([cpu,1,addressbus(a17)],[ls138,1,b]),
   connect([cpu,1,addressbus(a18)],[ls138,1,c]),
   connect([cpu,1,uds_],[or,1,in1]),
   connect([cpu,1,lds_],[or,2,in1]),
   connect([ls138,1,y1_],[or,1,in2]),
   connect([ls138,1,y1_],[or,2,in2]),
   connect([ls138,1,y0_],[and,1,in1]),
   connect([ls138,1,y1_],[and,1,in2]),
   connect([or,1,out],[ram,1,cs_]),
   connect([or,1,out],[ram,1,oe_]),
   connect([or,2,out],[ram,2,cs_]),
   connect([or,2,out],[ram,2,oe_]),
   connect([cpu,1,rw_],[ram,1,we_]),
   connect([cpu,1,rw_],[ram,2,we_]),
   connect([cpu,1,databus([d0,d7])],[ram,1,databus([d0,d7])]),
   connect([cpu,1,databus([d0,d7])],[ram,2,databus([d0,d7])]),
   connect([cpu,1,addressbus([a1,a15])],[ram,1,addressbus([a0,a14])]),
   connect([cpu,1,addressbus([a1,a15])],[ram,2,addressbus([a0,a14])]).

% set up objectives to solve & housekeeping elements
:- ps_init([],[problem],solution),
       ps_store(_,objective(active,[power,on]),given),
       ps_store(_,objective(active,[set,clock]),given),
       ps_store(_,
        objective(active,[check,setup,time,of,[cpu,1,databus([d0,d7])]]),
        given),
       ps_store(_,frequency(8),given),
       ps_store(_,buscycle(read),given),
       ps_store(_,device([ram,1],'uPD43256-10L'),given),
       ps_store(_,device([ram,2],'uPD43256-10L'),given),circuit.
```

```
%%%% state of solution database at beginning of problem solving %%%%
%%  ** highlights objective elements, >> highlights rule elements %%

time_tag(55).
** element(1,objective(active,[power,on]),given).
** element(2,objective(active,[set,clock]),given).
** element(3,objective(active,[check,setup,time,of,[cpu,1,databus([d0,d7])]]),
           given).
   element(4,frequency(8),given).
   element(5,buscycle(read),given).
   element(6,device([ram,1],uPD43256-10L),given).
   element(7,device([ram,2],uPD43256-10L),given).
   element(8,connected([cpu,1,as_],[ls138,1,e0_]),given).
   element(9,connected([power,1,gnd],[ls138,1,e1_]),given).
   element(10,connected([power,1,vcc],[ls138,1,e2]),given).
   element(11,connected([cpu,1,addressbus(a16)],[ls138,1,a]),given).
   element(12,connected([cpu,1,addressbus(a17)],[ls138,1,b]),given).
   element(13,connected([cpu,1,addressbus(a18)],[ls138,1,c]),given).
   element(14,connected([cpu,1,uds_],[or,1,in1]),given).
   element(15,connected([cpu,1,lds_],[or,2,in1]),given).
   element(16,connected([ls138,1,y1_],[or,1,in2]),given).
   element(17,connected([ls138,1,y1_],[or,2,in2]),given).
   element(18,connected([or,2,in2],[or,1,in2]),
           connected([ls138,1,y1_],[or,2,in2])).
   element(19,connected([ls138,1,y0_],[and,1,in1]),given).
   element(20,connected([ls138,1,y1_],[and,1,in2]),given).
   element(21,connected([and,1,in2],[or,1,in2]),
           connected([ls138,1,y1_],[and,1,in2])).
   element(22,connected([and,1,in2],[or,2,in2]),
           connected([or,1,in2],[and,1,in2])).
   element(23,connected([or,1,out],[ram,1,cs_]),given).
   element(24,connected([or,1,out],[ram,1,oe_]),given).
   element(25,connected([ram,1,oe_],[ram,1,cs_]),
           connected([or,1,out],[ram,1,oe_])).
   element(26,connected([or,2,out],[ram,2,cs_]),given).
   element(27,connected([or,2,out],[ram,2,oe_]),given).
   element(28,connected([ram,2,oe_],[ram,2,cs_]),
           connected([or,2,out],[ram,2,oe_])).
   element(29,connected([cpu,1,rw_],[ram,1,we_]),given).
   element(30,connected([cpu,1,rw_],[ram,2,we_]),given).
   element(31,connected([ram,2,we_],[ram,1,we_]),
           connected([cpu,1,rw_],[ram,2,we_])).
   element(32,connected([cpu,1,databus([d0,d7])],[ram,1,databus([d0,d7])]),
           given).
   element(33,connected([cpu,1,databus([d0,d7])],[ram,2,databus([d0,d7])]),
           given).
   element(34,connected([ram,2,databus([d0,d7])],[ram,1,databus([d0,d7])]),
           connected([cpu,1,databus([d0,d7])],[ram,2,databus([d0,d7])])).
   element(35,connected([cpu,1,addressbus([a1,a15])],
                        [ram,1,addressbus([a0,a14])]),given).
   element(36,connected([cpu,1,addressbus([a1,a15])],
                        [ram,2,addressbus([a0,a14])]),given).
   element(37,connected([ram,2,addressbus([a0,a14])],
                        [ram,1,addressbus([a0,a14])]),
              connected([cpu,1,addressbus([a1,a15])],
                        [ram,2,addressbus([a0,a14])])).
   element(38,event([state([cpu,_20,clk],s7p),tCYC],state([cpu,_20,clk],s1)),
           given).
   element(39,event([state([cpu,_20,clk],s0),tCYC],state([cpu,_20,clk],s2)),
        given).
element(40,event([state([cpu,_20,clk],s1),tCYC],state([cpu,_20,clk],s3)),
        given).
element(41,event([state([cpu,_20,clk],s2),tCYC],state([cpu,_20,clk],s4)),
        given).
element(42,event([state([cpu,_20,clk],s3),tCYC],state([cpu,_20,clk],s5)),
        given).
element(43,event([state([cpu,_20,clk],s4),tCYC],state([cpu,_20,clk],s6)),
        given).
element(44,event([state([cpu,_20,clk],s5),tCYC],state([cpu,_20,clk],s7)),
        given).
element(45,event([state([cpu,_20,clk],s6),tCYC],state([cpu,_20,clk],s0n)),
        given).
element(46,event([state([cpu,_20,clk],s7p),0.5*tCYC],
                 state([cpu,_20,clk],s0)),given).
element(47,event([state([cpu,_20,clk],s0),0.5*tCYC],
                 state([cpu,_20,clk],s1)),given).
element(48,event([state([cpu,_20,clk],s1),0.5*tCYC],
                  state([cpu,_20,clk],s2)),given).
element(49,event([state([cpu,_20,clk],s2),0.5*tCYC],
                  state([cpu,_20,clk],s3)),given).
element(50,event([state([cpu,_20,clk],s3),0.5*tCYC],
                  state([cpu,_20,clk],s4)),given).
element(51,event([state([cpu,_20,clk],s4),0.5*tCYC],
                  state([cpu,_20,clk],s5)),given).
element(52,event([state([cpu,_20,clk],s5),0.5*tCYC],
                  state([cpu,_20,clk],s6)),given).
element(53,event([state([cpu,_20,clk],s6),0.5*tCYC],
                  state([cpu,_20,clk],s7)),given).
element(54,event([state([cpu,_20,clk],s7),0.5*tCYC],
                  state([cpu,_20,clk],s0n)),given).
element(55,value([_20,tgate_delay],[0,10]),propagation delay).
```

```
%%%%%%%%% state of solution database after 3 iterations %%%%%%%%%
%%  ** highlights objective elements, >> highlights rule elements %%

time_tag(110).
** element(1,objective(solved,[power,on]),given).
** element(2,objective(solved,[set,clock]),given).
** element(3,objective(solved,[check,setup,time,of,[cpu,1,databus([d0,d7])]]),
           given).
   element(4,frequency(8),given).
   element(5,buscycle(read),given).
   element(6,device([ram,1],uPD43256-10L),given).
   element(7,device([ram,2],uPD43256-10L),given).
   element(8,connected([cpu,1,as_],[ls138,1,e0_]),given).
   element(9,connected([power,1,gnd],[ls138,1,e1_]),given).
   element(10,connected([power,1,vcc],[ls138,1,e2]),given).
   element(11,connected([cpu,1,addressbus(a16)],[ls138,1,a]),given).
   element(12,connected([cpu,1,addressbus(a17)],[ls138,1,b]),given).
   element(13,connected([cpu,1,addressbus(a18)],[ls138,1,c]),given).
   element(14,connected([cpu,1,uds_],[or,1,in1]),given).
   element(15,connected([cpu,1,lds_],[or,2,in1]),given).
   element(16,connected([ls138,1,y1_],[or,1,in2]),given).
   element(17,connected([ls138,1,y1_],[or,2,in2]),given).
   element(18,connected([or,2,in2],[or,1,in2]),
           connected([ls138,1,y1_],[or,2,in2])).
   element(19,connected([ls138,1,y0_],[and,1,in1]),given).
   element(20,connected([ls138,1,y1_],[and,1,in2]),given).
   element(21,connected([and,1,in2],[or,1,in2]),
           connected([ls138,1,y1_],[and,1,in2])).
   element(22,connected([and,1,in2],[or,2,in2]),
           connected([or,1,in2],[and,1,in2])).
   element(23,connected([or,1,out],[ram,1,cs_]),given).
   element(24,connected([or,1,out],[ram,1,oe_]),given).
   element(25,connected([ram,1,oe_],[ram,1,cs_]),
           connected([or,1,out],[ram,1,oe_])).
   element(26,connected([or,2,out],[ram,2,cs_]),given).
   element(27,connected([or,2,out],[ram,2,oe_]),given).
   element(28,connected([ram,2,oe_],[ram,2,cs_]),
           connected([or,2,out],[ram,2,oe_])).
   element(29,connected([cpu,1,rw_],[ram,1,we_]),given).
   element(30,connected([cpu,1,rw_],[ram,2,we_]),given).
   element(31,connected([ram,2,we_],[ram,1,we_]),
           connected([cpu,1,rw_],[ram,2,we_])).
   element(32,connected([cpu,1,databus([d0,d7])],[ram,1,databus([d0,d7])]),
           given).
   element(33,connected([cpu,1,databus([d0,d7])],[ram,2,databus([d0,d7])]),
           given).
   element(34,connected([ram,2,databus([d0,d7])],[ram,1,databus([d0,d7])]),
           connected([cpu,1,databus([d0,d7])],[ram,2,databus([d0,d7])])).
   element(35,connected([cpu,1,addressbus([a1,a15])],
                        [ram,1,addressbus([a0,a14])]),given).
   element(36,connected([cpu,1,addressbus([a1,a15])],
                        [ram,2,addressbus([a0,a14])]),given).
   element(37,connected([ram,2,addressbus([a0,a14])],
                        [ram,1,addressbus([a0,a14])]),
              connected([cpu,1,addressbus([a1,a15])],
                        [ram,2,addressbus([a0,a14])])).
   element(38,event([state([cpu,_20,clk],s7p),tCYC],state([cpu,_20,clk],s1)),
           given).
   element(39,event([state([cpu,_20,clk],s0),tCYC],state([cpu,_20,clk],s2)),
         given).
   element(40,event([state([cpu,_20,clk],s1),tCYC],state([cpu,_20,clk],s3)),
         given).
   element(41,event([state([cpu,_20,clk],s2),tCYC],state([cpu,_20,clk],s4)),
         given).
   element(42,event([state([cpu,_20,clk],s3),tCYC],state([cpu,_20,clk],s5)),
         given).
   element(43,event([state([cpu,_20,clk],s4),tCYC],state([cpu,_20,clk],s6)),
         given).
   element(44,event([state([cpu,_20,clk],s5),tCYC],state([cpu,_20,clk],s7)),
         given).
   element(45,event([state([cpu,_20,clk],s6),tCYC],state([cpu,_20,clk],s0n)),
         given).
   element(46,event([state([cpu,_20,clk],s7p),0.5*tCYC],
                  state([cpu,_20,clk],s0)),given).
   element(47,event([state([cpu,_20,clk],s0),0.5*tCYC],
                  state([cpu,_20,clk],s1)),given).
   element(48,event([state([cpu,_20,clk],s1),0.5*tCYC],
                   state([cpu,_20,clk],s2)),given).
   element(49,event([state([cpu,_20,clk],s2),0.5*tCYC],
                   state([cpu,_20,clk],s3)),given).
   element(50,event([state([cpu,_20,clk],s3),0.5*tCYC],
                   state([cpu,_20,clk],s4)),given).
   element(51,event([state([cpu,_20,clk],s4),0.5*tCYC],
                   state([cpu,_20,clk],s5)),given).
   element(52,event([state([cpu,_20,clk],s5),0.5*tCYC],
                   state([cpu,_20,clk],s6)),given).
   element(53,event([state([cpu,_20,clk],s6),0.5*tCYC],
                   state([cpu,_20,clk],s7)),given).
   element(54,event([state([cpu,_20,clk],s7),0.5*tCYC],
                   state([cpu,_20,clk],s0n)),given).
   element(55,value([_20,tgate_delay],[0,10]),propagation delay).
>> element(56,rule(concept(clk,1,[object(m68000)])),elements([4,2])).
   element(57,constraint([_4444,tCYC],[125,250]),
         concept(clk,1,[object(m68000)])).
   element(58,value([_4444,tCYC],[125,250]),concept(clk,1,[object(m68000)])).
   element(59,value([_4444,0.5*tCYC],[62.500000,125]),
         concept(clk,1,[object(m68000)])).
>> element(60,rule(concept(tDICL,1,[object(mc68000)])),elements([5,3])).
** element(61,objective(active,[determine,value,of,
                       [state([cpu,1,databus([d0,d7])],valid),tDICL]]),
                       concept(tDICL,1,[object(mc68000)])).
** element(62,objective(solved,[determine,when,
             [cpu,1,databus([d0,d7])],valid]),
             concept(tDICL,1,[object(mc68000)])).
   element(63,event([state([cpu,1,databus([d0,d7])],valid),tDICL],
                   state([cpu,1,clk],s7)),
                   concept(tDICL,1,[object(mc68000)])).
   element(64,constraint([state([cpu,1,databus([d0,d7])],valid),tDICL],
                        [10,undefined]),
                        concept(tDICL,1,[object(mc68000)])).
>> element(65,rule(concept(power,1,[object(power)])),elements([1])).
   element(66,event([[],-inf],state([ls138,1,e2],high)),
         connected([power,_4444,vcc],[ls138,1,e2])).
   element(67,event([[],-inf],state([ls138,1,e1_],low)),
         connected([power,_4444,gnd],[ls138,1,e1_])).
>> element(68,rule(concept(equivalent_goal,1,[object(device)])),
         elements([62])).
** element(69,objective(solved,
             [determine,when,[ram,2,databus([d0,d7])],valid]),
                concept(equivalent_goal,1,[object(device)])).
** element(70,objective(solved,
                [determine,when,[ram,1,databus([d0,d7])],valid]),
                concept(equivalent_goal,1,[object(device)])).
   element(71,equivalent([determine,when,[cpu,1,databus([d0,d7])],valid],
                [determine,when,[ram,1,databus([d0,d7])],valid]),
                concept(equivalent_goal,1,[object(device)])).
   element(72,equivalent([determine,when,[cpu,1,databus([d0,d7])],valid],
                [determine,when,[ram,2,databus([d0,d7])],valid]),
                concept(equivalent_goal,1,[object(device)])).
>> element(73,rule(concept(tAA,1,[object(static_ram)])),elements([6,5,70])).
** element(74,objective(active,
             [determine,ealier,of,
              [state([ram,1,addressbus([_4444,_4445])],valid),
               state([ram,1,cs_],asserted)]]),
             concept(tAA,1,[object(static_ram)])).
** element(75,objective(active,[determine,when,[ram,1,cs_],asserted]),
             concept(tAA,1,[object(static_ram)])).
** element(76,objective(active,
             [determine,when,[ram,1,addressbus([_4444,_4445])],valid]),
             concept(tAA,1,[object(static_ram)])).
** element(77,objective(active,
             [propagate,event(
              [state([ram,1,addressbus([_4444,_4445])],valid),tAA],
               state([ram,1,databus([d0,d7])],valid))]),
             concept(tAA,1,[object(static_ram)])).
   element(78,event([state([ram,1,addressbus([_4444,_4445])],valid),tAA],
             state([ram,1,databus([d0,d7])],valid)),
             concept(tAA,1,[object(static_ram)])).
   element(79,constraint([state([ram,1,addressbus([_4444,_4445])],valid),tAA],
             [undefined,100]),
             concept(tAA,1,[object(static_ram)])).
>> element(80,rule(concept(tAA,1,[object(static_ram)])),elements([7,5,69])).
** element(81,objective(active,
             [determine,ealier,of,
              [state([ram,2,addressbus([_4444,_4445])],valid),
               state([ram,2,cs_],asserted)]]),
             concept(tAA,1,[object(static_ram)])).
** element(82,objective(active,[determine,when,[ram,2,cs_],asserted]),
                concept(tAA,1,[object(static_ram)])).
** element(83,objective(active,
              [determine,when,[ram,2,addressbus([_4444,_4445])],valid]),
              concept(tAA,1,[object(static_ram)])).
** element(84,objective(active,
              [propagate,event(
                 [state([ram,2,addressbus([_4444,_4445])],valid),tAA],
                 state([ram,2,databus([d0,d7])],valid))]),
              concept(tAA,1,[object(static_ram)])).
   element(85,event([state([ram,2,addressbus([_4444,_4445])],valid),tAA],
                       state([ram,2,databus([d0,d7])],valid)),
                       concept(tAA,1,[object(static_ram)])).
   element(86,constraint([state([ram,2,addressbus([_4444,_4445])],valid),tAA],
                           [undefined,100]),
                           concept(tAA,1,[object(static_ram)])).
>> element(87,rule(concept(tACS,1,[object(static_ram)])),elements([6,5,70])).
** element(88,objective(active,
                [propagate,event([state([ram,1,cs_],asserted),tACS],
                state([ram,1,databus([d0,d7])],valid))]),
                concept(tACS,1,[object(static_ram)])).
   element(89,event([state([ram,1,cs_],asserted),tACS],
                       state([ram,1,databus([d0,d7])],valid)),
                       concept(tACS,1,[object(static_ram)])).
   element(90,constraint([state([ram,1,cs_],asserted),tACS],[undefined,100]),
          concept(tACS,1,[object(static_ram)])).
>> element(91,rule(concept(tACS,1,[object(static_ram)])),elements([7,5,69])).
** element(92,objective(active,
                           [propagate,event([state([ram,2,cs_],asserted),tACS],
                           state([ram,2,databus([d0,d7])],valid))]),
                           concept(tACS,1,[object(static_ram)])).
   element(93,event([state([ram,2,cs_],asserted),tACS],
                        state([ram,2,databus([d0,d7])],valid)),
                        concept(tACS,1,[object(static_ram)])).
   element(94,constraint([state([ram,2,cs_],asserted),tACS],[undefined,100]),
          concept(tACS,1,[object(static_ram)])).
>> element(95,rule(concept(tOE,1,[object(static_ram)])),elements([6,5,70])).
** element(96,objective(active,[determine,when,[ram,1,oe_],asserted]),
          concept(tOE,1,[object(static_ram)])).
** element(97,objective(active,
             [propagate,event([state([ram,1,oe_],asserted),tOE],
             state([ram,1,databus([d0,d7])],valid))]),
             concept(tOE,1,[object(static_ram)])).
   element(98,event([state([ram,1,oe_],asserted),tOE],
                        state([ram,1,databus([d0,d7])],valid)),
                        concept(tOE,1,[object(static_ram)])).
   element(99,constraint([state([ram,1,oe_],asserted),tOE],[undefined,50]),
          concept(tOE,1,[object(static_ram)])).
>> element(100,rule(concept(tOE,1,[object(static_ram)])),elements([7,5,69])).
** element(101,objective(active,[determine,when,[ram,2,oe_],asserted]),
          concept(tOE,1,[object(static_ram)])).
** element(102,objective(active,
                            [propagate,event([state([ram,2,oe_],asserted),tOE],
                            state([ram,2,databus([d0,d7])],valid))]),
                            concept(tOE,1,[object(static_ram)])).
   element(103,event([state([ram,2,oe_],asserted),tOE],
                         state([ram,2,databus([d0,d7])],valid)),
                         concept(tOE,1,[object(static_ram)])).
   element(104,constraint([state([ram,2,oe_],asserted),tOE],[undefined,50]),
          concept(tOE,1,[object(static_ram)])).
>> element(105,rule(concept(equivalent_goal,1,[object(device)])),
          elements([70])).
   element(108,equivalent([determine,when,[ram,1,databus([d0,d7])],valid],
                           [determine,when,[ram,2,databus([d0,d7])],valid]),
                           concept(equivalent_goal,1,[object(device)])).
>> element(109,rule(concept(equivalent_goal,1,[object(device)])),
         elements([69])).
```

```
%%%%%%%%%%%%%%%% final state of solution database %%%%%%%%%%%%%%%%
%%  ** highlights objective elements, >> highlights rule elements %%

time_tag(381).
** element(1,objective(solved,[power,on]),given).
** element(2,objective(solved,[set,clock]),given).
** element(3,objective(solved,
                  [check,setup,time,of,[cpu,1,databus([d0,d7])]]),given).
   element(4,frequency(8),given).
   element(5,buscycle(read),given).
   element(6,device([ram,1],uPD43256-10L),given).
   element(7,device([ram,2],uPD43256-10L),given).
   element(8,connected([cpu,1,as_],[ls138,1,e0_]),given).
   element(9,connected([power,1,gnd],[ls138,1,e1_]),given).
   element(10,connected([power,1,vcc],[ls138,1,e2]),given).
   element(11,connected([cpu,1,addressbus(a16)],[ls138,1,a]),given).
   element(12,connected([cpu,1,addressbus(a17)],[ls138,1,b]),given).
   element(13,connected([cpu,1,addressbus(a18)],[ls138,1,c]),given).
   element(14,connected([cpu,1,uds_],[or,1,in1]),given).
   element(15,connected([cpu,1,lds_],[or,2,in1]),given).
   element(16,connected([ls138,1,y1_],[or,1,in2]),given).
   element(17,connected([ls138,1,y1_],[or,2,in2]),given).
   element(18,connected([or,2,in2],[or,1,in2]),
                   connected([ls138,1,y1_],[or,2,in2])).
   element(19,connected([ls138,1,y0_],[and,1,in1]),given).
   element(20,connected([ls138,1,y1_],[and,1,in2]),given).
   element(21,connected([and,1,in2],[or,1,in2]),
                   connected([ls138,1,y1_],[and,1,in2])).
   element(22,connected([and,1,in2],[or,2,in2]),
                   connected([or,1,in2],[and,1,in2])).
   element(23,connected([or,1,out],[ram,1,cs_]),given).
   element(24,connected([or,1,out],[ram,1,oe_]),given).
   element(25,connected([ram,1,oe_],[ram,1,cs_]),
                   connected([or,1,out],[ram,1,oe_])).
   element(26,connected([or,2,out],[ram,2,cs_]),given).
   element(27,connected([or,2,out],[ram,2,oe_]),given).
   element(28,connected([ram,2,oe_],[ram,2,cs_]),
                   connected([or,2,out],[ram,2,oe_])).
   element(29,connected([cpu,1,rw_],[ram,1,we_]),given).
   element(30,connected([cpu,1,rw_],[ram,2,we_]),given).
   element(31,connected([ram,2,we_],[ram,1,we_]),
                   connected([cpu,1,rw_],[ram,2,we_])).
   element(32,connected([cpu,1,databus([d0,d7])],
                   [ram,1,databus([d0,d7])]),given).
   element(33,connected([cpu,1,databus([d0,d7])],
                   [ram,2,databus([d0,d7])]),given).
   element(34,connected([ram,2,databus([d0,d7])],
                   [ram,1,databus([d0,d7])]),
                   connected([cpu,1,databus([d0,d7])],
                   [ram,2,databus([d0,d7])])).
   element(35,connected([cpu,1,addressbus([a1,a15])],
                   [ram,1,addressbus([a0,a14])]),given).
   element(36,connected([cpu,1,addressbus([a1,a15])],
                   [ram,2,addressbus([a0,a14])]),given).
   element(37,connected([ram,2,addressbus([a0,a14])],
                   [ram,1,addressbus([a0,a14])]),
                   connected([cpu,1,addressbus([a1,a15])],
                   [ram,2,addressbus([a0,a14])])).
   element(38,event([state([cpu,_233,clk],s7p),tCYC],
                        state([cpu,_233,clk],s1)),given).
   element(39,event([state([cpu,_233,clk],s0),tCYC],
                        state([cpu,_233,clk],s2)),given).
   element(40,event([state([cpu,_233,clk],s1),tCYC],
                        state([cpu,_233,clk],s3)),given).
   element(41,event([state([cpu,_233,clk],s2),tCYC]
                        ,state([cpu,_233,clk],s4)),given).
   element(42,event([state([cpu,_233,clk],s3),tCYC],
                        state([cpu,_233,clk],s5)),given).
   element(43,event([state([cpu,_233,clk],s4),tCYC],
                        state([cpu,_233,clk],s6)),given).
   element(44,event([state([cpu,_233,clk],s5),tCYC],
                        state([cpu,_233,clk],s7)),given).
   element(45,event([state([cpu,_233,clk],s6),tCYC],
                        state([cpu,_233,clk],s0n)),given).
   element(46,event([state([cpu,_233,clk],s7p),0.5*tCYC],
                        state([cpu,_233,clk],s0)),given).
   element(47,event([state([cpu,_233,clk],s0),0.5*tCYC],
                        state([cpu,_233,clk],s1)),given).
   element(48,event([state([cpu,_233,clk],s1),0.5*tCYC],
                        state([cpu,_233,clk],s2)),given).
   element(49,event([state([cpu,_233,clk],s2),0.5*tCYC],
                        state([cpu,_233,clk],s3)),given).
   element(50,event([state([cpu,_233,clk],s3),0.5*tCYC],
                        state([cpu,_233,clk],s4)),given).
   element(51,event([state([cpu,_233,clk],s4),0.5*tCYC],
                        state([cpu,_233,clk],s5)),given).
   element(52,event([state([cpu,_233,clk],s5),0.5*tCYC],
                        state([cpu,_233,clk],s6)),given).
   element(53,event([state([cpu,_233,clk],s6),0.5*tCYC],
                        state([cpu,_233,clk],s7)),given).
   element(54,event([state([cpu,_233,clk],s7),0.5*tCYC],
                        state([cpu,_233,clk],s0n)),given).
   element(55,value([_233,tgate_delay],[0,10]),propagation delay).
>> element(56,rule(concept(clk,1,[object(m68000)])),elements([4,2])).
   element(57,constraint([_233,tCYC],[125,250]),
                        concept(clk,1,[object(m68000)])).
   element(58,value([_233,tCYC],[125,250]),concept(clk,1,[object(m68000)])).
   element(59,value([_233,0.5*tCYC],[62.500000,125]),
                        concept(clk,1,[object(m68000)])).
>> element(60,rule(concept(tDICL,1,[object(mc68000)])),elements([5,3])).
** element(61,objective(solved,
                     [determine,value,of,[state([cpu,1,databus([d0,d7])],valid)
                     ,tDICL]]),concept(tDICL,1,[object(mc68000)])).
** element(62,objective(solved,
                     [determine,when,[cpu,1,databus([d0,d7])],valid]),
                     concept(tDICL,1,[object(mc68000)])).
   element(63,event([state([cpu,1,databus([d0,d7])],valid),tDICL],
                        state([cpu,1,clk],s7)),
                        concept(tDICL,1,[object(mc68000)])).
   element(64,constraint([state([cpu,1,databus([d0,d7])],valid),tDICL],
                       [10,undefined]),concept(tDICL,1,[object(mc68000)])).
>> element(65,rule(concept(power,1,[object(power)])),elements([1])).
   element(66,event([[],-inf],state([ls138,1,e2],high)),
                        connected([power,_233,vcc],[ls138,1,e2])).
   element(67,event([[],-inf],state([ls138,1,e1_],low)),
                        connected([power,_233,gnd],[ls138,1,e1_])).
>> element(68,rule(concept(equivalent_goal,1,[object(device)])),elements([62])).
** element(69,objective(solved,
                     [determine,when,[ram,2,databus([d0,d7])],valid]),
                    concept(equivalent_goal,1,[object(device)])).
** element(70,objective(solved,
                    [determine,when,[ram,1,databus([d0,d7])],valid]),
                    concept(equivalent_goal,1,[object(device)])).
   element(71,equivalent([determine,when,[cpu,1,databus([d0,d7])],valid],
                       [determine,when,[ram,1,databus([d0,d7])],valid]),
                       concept(equivalent_goal,1,[object(device)])).
   element(72,equivalent([determine,when,[cpu,1,databus([d0,d7])],valid],
                       [determine,when,[ram,2,databus([d0,d7])],valid]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(73,rule(concept(tAA,1,[object(static_ram)])),elements([6,5,70])).
** element(74,objective(solved,[determine,ealier,of,
                    [state([ram,1,addressbus([a0,a14])],valid),
                    state([ram,1,cs_],asserted)]]),
                    concept(tAA,1,[object(static_ram)])).
** element(75,objective(solved,[determine,when,[ram,1,cs_],asserted]),
                    concept(tAA,1,[object(static_ram)])).
** element(76,objective(solved,[determine,when,[ram,1,addressbus([a0,a14])],
                    valid]),concept(tAA,1,[object(static_ram)])).
** element(77,objective(solved,[propagate,
                    event([state([ram,1,addressbus([_233,_234])],valid),tAA],
                    state([ram,1,databus([d0,d7])],valid))]),
                    concept(tAA,1,[object(static_ram)])).
   element(78,event([state([ram,1,addressbus([_233,_234])],valid),tAA],
                       state([ram,1,databus([d0,d7])],valid)),
                       concept(tAA,1,[object(static_ram)])).
   element(79,constraint([state([ram,1,addressbus([_233,_234])],valid),tAA],
                       [undefined,100]),concept(tAA,1,[object(static_ram)])).
>> element(80,rule(concept(tAA,1,[object(static_ram)])),elements([7,5,69])).
** element(81,objective(solved,[determine,ealier,of,
                    [state([ram,2,addressbus([a0,a14])],valid),
                    state([ram,2,cs_],asserted)]]),
                    concept(tAA,1,[object(static_ram)])).
** element(82,objective(solved,[determine,when,[ram,2,cs_],asserted]),
                    concept(tAA,1,[object(static_ram)])).
** element(83,objective(solved,[determine,when,[ram,2,addressbus([a0,a14])],
                    valid]),concept(tAA,1,[object(static_ram)])).
** element(84,objective(solved,[propagate,
                    event([state([ram,2,addressbus([_233,_234])],valid),tAA],
                    state([ram,2,databus([d0,d7])],valid))]),
                    concept(tAA,1,[object(static_ram)])).
   element(85,event([state([ram,2,addressbus([_233,_234])],valid),tAA],
                       state([ram,2,databus([d0,d7])],valid)),
                       concept(tAA,1,[object(static_ram)])).
   element(86,constraint([state([ram,2,addressbus([_233,_234])],valid),tAA],
                       [undefined,100]),concept(tAA,1,[object(static_ram)])).
>> element(87,rule(concept(tACS,1,[object(static_ram)])),elements([6,5,70])).
** element(88,objective(solved,[propagate,event([state([ram,1,cs_],asserted),
                    tACS],state([ram,1,databus([d0,d7])],valid))]),
                    concept(tACS,1,[object(static_ram)])).
   element(89,event([state([ram,1,cs_],asserted),tACS],
                       state([ram,1,databus([d0,d7])],valid)),
                       concept(tACS,1,[object(static_ram)])).
   element(90,constraint([state([ram,1,cs_],asserted),tACS],[undefined,100]),
                       concept(tACS,1,[object(static_ram)])).
>> element(91,rule(concept(tACS,1,[object(static_ram)])),elements([7,5,69])).
** element(92,objective(solved,[propagate,event([state([ram,2,cs_],asserted),
                    tACS],state([ram,2,databus([d0,d7])],valid))]),
                    concept(tACS,1,[object(static_ram)])).
   element(93,event([state([ram,2,cs_],asserted),tACS],
                          state([ram,2,databus([d0,d7])],valid)),
                          concept(tACS,1,[object(static_ram)])).
   element(94,constraint([state([ram,2,cs_],asserted),tACS],[undefined,100]),
                          concept(tACS,1,[object(static_ram)])).
>> element(95,rule(concept(tOE,1,[object(static_ram)])),elements([6,5,70])).
** element(96,objective(solved,[determine,when,[ram,1,oe_],asserted]),
                       concept(tOE,1,[object(static_ram)])).
** element(97,objective(solved,[propagate,event([state([ram,1,oe_],asserted),
                       tOE],state([ram,1,databus([d0,d7])],valid))]),
                       concept(tOE,1,[object(static_ram)])).
   element(98,event([state([ram,1,oe_],asserted),tOE],
                          state([ram,1,databus([d0,d7])],valid)),
                          concept(tOE,1,[object(static_ram)])).
   element(99,constraint([state([ram,1,oe_],asserted),tOE],[undefined,50]),
                          concept(tOE,1,[object(static_ram)])).
>> element(100,rule(concept(tOE,1,[object(static_ram)])),elements([7,5,69])).
** element(101,objective(solved,[determine,when,[ram,2,oe_],asserted]),
                       concept(tOE,1,[object(static_ram)])).
** element(102,objective(solved,
                       [propagate,event([state([ram,2,oe_],asserted),tOE],
                       state([ram,2,databus([d0,d7])],valid))]),
                       concept(tOE,1,[object(static_ram)])).
   element(103,event([state([ram,2,oe_],asserted),tOE],
                          state([ram,2,databus([d0,d7])],valid)),
                          concept(tOE,1,[object(static_ram)])).
   element(104,constraint([state([ram,2,oe_],asserted),tOE],[undefined,50]),
                          concept(tOE,1,[object(static_ram)])).
>> element(105,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([70])).
   element(108,equivalent([determine,when,[ram,1,databus([d0,d7])],valid],
                          [determine,when,[ram,2,databus([d0,d7])],valid]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(109,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([69])).
>> element(111,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([101])).
** element(112,objective(solved,[determine,when,[or,2,out],asserted]),
                       concept(equivalent_goal,1,[object(device)])).
   element(113,equivalent([determine,when,[ram,2,oe_],asserted],
                          [determine,when,[or,2,out],asserted]),
                          concept(equivalent_goal,1,[object(device)])).
   element(114,equivalent([determine,when,[ram,2,oe_],asserted],
                          [determine,when,[ram,2,cs_],asserted]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(115,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([96])).
** element(116,objective(solved,[determine,when,[or,1,out],asserted]),
                       concept(equivalent_goal,1,[object(device)])).
   element(117,equivalent([determine,when,[ram,1,oe_],asserted],
                          [determine,when,[or,1,out],asserted]),
                          concept(equivalent_goal,1,[object(device)])).
   element(118,equivalent([determine,when,[ram,1,oe_],asserted],
                          [determine,when,[ram,1,cs_],asserted]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(119,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([83])).
** element(120,objective(solved,[determine,when,[cpu,1,addressbus([a1,a15])],
                       valid]),concept(equivalent_goal,1,[object(device)])).
   element(121,equivalent([determine,when,[ram,2,addressbus([a0,a14])],valid]
                          ,[determine,when,[cpu,1,addressbus([a1,a15])],valid]),
                          concept(equivalent_goal,1,[object(device)])).
   element(122,equivalent([determine,when,[ram,2,addressbus([a0,a14])],valid]
                          ,[determine,when,[ram,1,addressbus([a0,a14])],valid]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(123,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([82])).
   element(124,equivalent([determine,when,[ram,2,cs_],asserted],
                          [determine,when,[or,2,out],asserted]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(125,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([76])).
   element(126,equivalent([determine,when,[ram,1,addressbus([a0,a14])],valid]
                          ,[determine,when,[cpu,1,addressbus([a1,a15])],valid]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(127,rule(concept(equivalent_goal,1,[object(device)])),
                       elements([75])).
   element(128,equivalent([determine,when,[ram,1,cs_],asserted],
                          [determine,when,[or,1,out],asserted]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(129,rule(concept(equivalent_goal,2,[object(device)])),
                       elements([101])).
** element(130,objective(solved,[determine,when,[ram,2,oe_],low]),
                       concept(equivalent_goal,2,[object(device)])).
   element(131,equivalent([determine,when,[ram,2,oe_],asserted],
                          [determine,when,[ram,2,oe_],low]),
                          concept(equivalent_goal,2,[object(device)])).
>> element(132,rule(concept(equivalent_goal,2,[object(device)])),
                       elements([96])).
** element(133,objective(solved,[determine,when,[ram,1,oe_],low]),
                       concept(equivalent_goal,2,[object(device)])).
   element(134,equivalent([determine,when,[ram,1,oe_],asserted],
                          [determine,when,[ram,1,oe_],low]),
                          concept(equivalent_goal,2,[object(device)])).
>> element(135,rule(concept(equivalent_goal,2,[object(device)])),
                       elements([82])).
** element(136,objective(solved,[determine,when,[ram,2,cs_],low]),
                       concept(equivalent_goal,2,[object(device)])).
   element(137,equivalent([determine,when,[ram,2,cs_],asserted],
                          [determine,when,[ram,2,cs_],low]),
                          concept(equivalent_goal,2,[object(device)])).
>> element(138,rule(concept(equivalent_goal,2,[object(device)])),
                       elements([75])).
** element(139,objective(solved,[determine,when,[ram,1,cs_],low]),
                       concept(equivalent_goal,2,[object(device)])).
   element(140,equivalent([determine,when,[ram,1,cs_],asserted],
                          [determine,when,[ram,1,cs_],low]),
                          concept(equivalent_goal,2,[object(device)])).
>> element(141,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
                       elements([102])).
   element(142,event([state([ram,2,oe_],asserted),tOE],
                          state([ram,1,databus([d0,d7])],valid)),
                          connected([ram,1,databus([d0,d7])],
                          [ram,2,databus([d0,d7])])).
   element(143,event([state([ram,2,oe_],asserted),tOE],
                          state([cpu,1,databus([d0,d7])],valid)),
                          connected([cpu,1,databus([d0,d7])],
                          [ram,2,databus([d0,d7])])).
>> element(144,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
                       elements([97])).
   element(145,event([state([ram,1,oe_],asserted),tOE],
                        state([ram,2,databus([d0,d7])],valid)),
                        connected([ram,2,databus([d0,d7])],
                        [ram,1,databus([d0,d7])])).
   element(146,event([state([ram,1,oe_],asserted),tOE],
                        state([cpu,1,databus([d0,d7])],valid)),
                        connected([cpu,1,databus([d0,d7])],
                        [ram,1,databus([d0,d7])])).
>> element(147,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
                    elements([92])).
   element(148,event([state([ram,2,cs_],asserted),tACS],
                        state([ram,1,databus([d0,d7])],valid)),
                        connected([ram,1,databus([d0,d7])],
                        [ram,2,databus([d0,d7])])).
   element(149,event([state([ram,2,cs_],asserted),tACS],
                        state([cpu,1,databus([d0,d7])],valid)),
                        connected([cpu,1,databus([d0,d7])],
                        [ram,2,databus([d0,d7])])).
>> element(150,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
                    elements([88])).
   element(151,event([state([ram,1,cs_],asserted),tACS],
                        state([ram,2,databus([d0,d7])],valid)),
                        connected([ram,2,databus([d0,d7])],[ram,1,databus([d0,d7])
   element(152,event([state([ram,1,cs_],asserted),tACS],
                        state([cpu,1,databus([d0,d7])],valid)),
                        connected([cpu,1,databus([d0,d7])],[ram,1,databus([d0,d7])
>> element(153,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
                    elements([84])).
   element(154,event([state([ram,2,addressbus([_233,_234])],valid),tAA],
                        state([ram,1,databus([d0,d7])],valid)),
                        connected([ram,1,databus([d0,d7])],
                        [ram,2,databus([d0,d7])])).
   element(155,event([state([ram,2,addressbus([_233,_234])],valid),tAA],
                        state([cpu,1,databus([d0,d7])],valid)),
                        connected([cpu,1,databus([d0,d7])],
                        [ram,2,databus([d0,d7])])).
>> element(156,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
                    elements([77])).
   element(157,event([state([ram,1,addressbus([_233,_234])],valid),tAA],
                        state([ram,2,databus([d0,d7])],valid)),
                        connected([ram,2,databus([d0,d7])],
                        [ram,1,databus([d0,d7])])).
   element(158,event([state([ram,1,addressbus([_233,_234])],valid),tAA],
                        state([cpu,1,databus([d0,d7])],valid)),
                        connected([cpu,1,databus([d0,d7])],
                        [ram,1,databus([d0,d7])])).
>> element(159,rule(concept(tCLAV,1,[object(mc68000)])),elements([4,120])).
** element(160,objective(solved,[propagate,
                    event([state([cpu,1,clk],s1),tCLAV],
                    state([cpu,1,addressbus([a1,a15])],valid))]),
                    concept(tCLAV,1,[object(mc68000)])).
   element(161,event([state([cpu,1,clk],s1),tCLAV],
                        state([cpu,1,addressbus([a1,a15])],valid)),
                        concept(tCLAV,1,[object(mc68000)])).
   element(162,constraint([state([cpu,1,clk],s1),tCLAV],[undefined,62]),
                        concept(tCLAV,1,[object(mc68000)])).
   element(163,value([state([cpu,1,clk],s1),tCLAV],[undefined,62]),
                        concept(tCLAV,1,[object(mc68000)])).
>> element(164,rule(concept(equivalent_goal,1,[object(device)])),
                    elements([139])).
** element(165,objective(solved,[determine,when,[or,1,out],low]),
                     concept(equivalent_goal,1,[object(device)])).
   element(166,equivalent([determine,when,[ram,1,cs_],low],
                         [determine,when,[or,1,out],low]),
                         concept(equivalent_goal,1,[object(device)])).
   element(167,equivalent([determine,when,[ram,1,cs_],low],
                         [determine,when,[ram,1,oe_],low]),
                         concept(equivalent_goal,1,[object(device)])).
>> element(168,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([136])).
** element(169,objective(solved,[determine,when,[or,2,out],low]),
                     concept(equivalent_goal,1,[object(device)])).
   element(170,equivalent([determine,when,[ram,2,cs_],low],
                         [determine,when,[or,2,out],low]),
                         concept(equivalent_goal,1,[object(device)])).
   element(171,equivalent([determine,when,[ram,2,cs_],low],
                         [determine,when,[ram,2,oe_],low]),
                         concept(equivalent_goal,1,[object(device)])).
>> element(172,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([133])).
   element(173,equivalent([determine,when,[ram,1,oe_],low],
                         [determine,when,[or,1,out],low]),
                         concept(equivalent_goal,1,[object(device)])).
>> element(174,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([130])).
   element(175,equivalent([determine,when,[ram,2,oe_],low],
                         [determine,when,[or,2,out],low]),
                         concept(equivalent_goal,1,[object(device)])).
>> element(176,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([120])).
>> element(179,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([116])).
>> element(180,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([112])).
>> element(181,rule(concept(equivalent_goal,3,[object(device)])),
                     elements([139])).
   element(182,equivalent([determine,when,[ram,1,cs_],low],
                         [determine,when,[ram,1,cs_],asserted]),
                         concept(equivalent_goal,3,[object(device)])).
>> element(183,rule(concept(equivalent_goal,3,[object(device)])),
                     elements([136])).
   element(184,equivalent([determine,when,[ram,2,cs_],low],
                         [determine,when,[ram,2,cs_],asserted]),
                         concept(equivalent_goal,3,[object(device)])).
>> element(185,rule(concept(equivalent_goal,3,[object(device)])),
                     elements([133])).
   element(186,equivalent([determine,when,[ram,1,oe_],low],
                         [determine,when,[ram,1,oe_],asserted]),
                         concept(equivalent_goal,3,[object(device)])).
>> element(187,rule(concept(equivalent_goal,3,[object(device)])),
                     elements([130])).
   element(188,equivalent([determine,when,[ram,2,oe_],low],
                         [determine,when,[ram,2,oe_],asserted]),
                         concept(equivalent_goal,3,[object(device)])).
>> element(189,rule(concept(determine_value,3,[])),elements([158,53,63,61])).
** element(190,objective(solved,
                     [find,common,reference,of,state([cpu,1,clk],s7),
                     state([cpu,1,databus([d0,d7])],valid)]),
                     concept(determine_value,3,[])).
>> element(191,rule(concept(or_gate,5,[object(logic_gate)])),elements([169])).
** element(192,objective(solved,[determine,when,[or,2,in2],low]),
                     concept(or_gate,5,[object(logic_gate)])).
** element(193,objective(solved,[determine,when,[or,2,in1],low]),
                     concept(or_gate,5,[object(logic_gate)])).
>> element(194,rule(concept(or_gate,5,[object(logic_gate)])),elements([165])).
** element(195,objective(solved,[determine,when,[or,1,in2],low]),
                     concept(or_gate,5,[object(logic_gate)])).
** element(196,objective(solved,[determine,when,[or,1,in1],low]),
                     concept(or_gate,5,[object(logic_gate)])).
>> element(197,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([169])).
>> element(198,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([165])).
>> element(199,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
                     elements([160])).
   element(200,event([state([cpu,1,clk],s1),tCLAV],
                       state([ram,2,addressbus([a0,a14])],valid)),
                       connected([ram,2,addressbus([a0,a14])],
                                 [cpu,1,addressbus([a1,a15])])).
   element(201,event([state([cpu,1,clk],s1),tCLAV],
                       state([ram,1,addressbus([a0,a14])],valid)),
                       connected([ram,1,addressbus([a0,a14])],[cpu,1,addressbus([
>> element(202,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([196])).
** element(203,objective(solved,[determine,when,[cpu,1,uds_],low]),
                     concept(equivalent_goal,1,[object(device)])).
   element(204,equivalent([determine,when,[or,1,in1],low],
                       [determine,when,[cpu,1,uds_],low]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(205,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([195])).
** element(206,objective(solved,[determine,when,[ls138,1,y1_],low]),
                     concept(equivalent_goal,1,[object(device)])).
** element(207,objective(solved,[determine,when,[and,1,in2],low]),
                     concept(equivalent_goal,1,[object(device)])).
   element(208,equivalent([determine,when,[or,1,in2],low],
                       [determine,when,[and,1,in2],low]),
                       concept(equivalent_goal,1,[object(device)])).
   element(209,equivalent([determine,when,[or,1,in2],low],
                       [determine,when,[ls138,1,y1_],low]),
                       concept(equivalent_goal,1,[object(device)])).
   element(210,equivalent([determine,when,[or,1,in2],low],
                       [determine,when,[or,2,in2],low]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(211,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([193])).
** element(212,objective(solved,[determine,when,[cpu,1,lds_],low]),
                     concept(equivalent_goal,1,[object(device)])).
   element(213,equivalent([determine,when,[or,2,in1],low],
                       [determine,when,[cpu,1,lds_],low]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(214,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([192])).
   element(215,equivalent([determine,when,[or,2,in2],low],
                       [determine,when,[and,1,in2],low]),
                       concept(equivalent_goal,1,[object(device)])).
   element(216,equivalent([determine,when,[or,2,in2],low],
                       [determine,when,[ls138,1,y1_],low]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(217,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([212])).
```

```
>> element(218,rule(concept(equivalent_goal,1,[object(device)])),
                    elements([207])).
   element(219,equivalent([determine,when,[and,1,in2],low],
                          [determine,when,[ls138,1,y1_],low]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(220,rule(concept(equivalent_goal,1,[object(device)])),
                    elements([206])).
>> element(221,rule(concept(equivalent_goal,1,[object(device)])),
                    elements([203])).
>> element(222,rule(concept(equivalent_goal,3,[object(device)])),
                    elements([212])).
** element(223,objective(solved,[determine,when,[cpu,1,lds_],asserted]),
                    concept(equivalent_goal,3,[object(device)])).
   element(224,equivalent([determine,when,[cpu,1,lds_],low],
                          [determine,when,[cpu,1,lds_],asserted]),
                          concept(equivalent_goal,3,[object(device)])).
>> element(225,rule(concept(equivalent_goal,3,[object(device)])),
                    elements([206])).
** element(226,objective(solved,[determine,when,[ls138,1,y1_],asserted]),
                    concept(equivalent_goal,3,[object(device)])).
   element(227,equivalent([determine,when,[ls138,1,y1_],low],
                          [determine,when,[ls138,1,y1_],asserted]),
                          concept(equivalent_goal,3,[object(device)])).
>> element(228,rule(concept(equivalent_goal,3,[object(device)])),
                    elements([203])).
** element(229,objective(solved,[determine,when,[cpu,1,uds_],asserted]),
                    concept(equivalent_goal,3,[object(device)])).
   element(230,equivalent([determine,when,[cpu,1,uds_],low],
                          [determine,when,[cpu,1,uds_],asserted]),
                          concept(equivalent_goal,3,[object(device)])).
>> element(231,rule(concept(equivalent_goal,4,[object(mc68000)])),
                    elements([212])).
** element(232,objective(solved,[determine,when,[cpu,1,ds_],low]),
                    concept(equivalent_goal,4,[object(mc68000)])).
   element(233,equivalent([determine,when,[cpu,1,lds_],low],
                          [determine,when,[cpu,1,ds_],low]),
                          concept(equivalent_goal,4,[object(mc68000)])).
>> element(234,rule(concept(equivalent_goal,5,[object(mc68000)])),
                    elements([203])).
   element(235,equivalent([determine,when,[cpu,1,uds_],_233],
                          [determine,when,[cpu,1,ds_],low]),
                          concept(equivalent_goal,5,[object(mc68000)])).
>> element(236,rule(concept(ls138,2,[object(device)])),elements([226])).
** element(237,objective(solved,[determine,when,[ls138,1,e2],high]),
                    concept(ls138,2,[object(device)])).
** element(238,objective(solved,[determine,when,[ls138,1,e1_],low]),
                    concept(ls138,2,[object(device)])).
** element(239,objective(solved,[determine,when,[ls138,1,e0_],low]),
                    concept(ls138,2,[object(device)])).
>> element(240,rule(concept(equivalent_goal,1,[object(device)])),
                    elements([229])).
** element(241,objective(solved,[determine,when,[or,1,in1],asserted]),
                    concept(equivalent_goal,1,[object(device)])).
   element(242,equivalent([determine,when,[cpu,1,uds_],asserted],
                          [determine,when,[or,1,in1],asserted]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(243,rule(concept(equivalent_goal,1,[object(device)])),
                    elements([226])).
** element(244,objective(solved,[determine,when,[or,2,in2],asserted]),
                    concept(equivalent_goal,1,[object(device)])).
```

```
** element(245,objective(solved,[determine,when,[or,1,in2],asserted]),
                     concept(equivalent_goal,1,[object(device)])).
** element(246,objective(solved,[determine,when,[and,1,in2],asserted]),
                     concept(equivalent_goal,1,[object(device)])).
   element(247,equivalent([determine,when,[ls138,1,y1_],asserted],
                        [determine,when,[and,1,in2],asserted]),
                        concept(equivalent_goal,1,[object(device)])).
   element(248,equivalent([determine,when,[ls138,1,y1_],asserted],
                        [determine,when,[or,1,in2],asserted]),
                        concept(equivalent_goal,1,[object(device)])).
   element(249,equivalent([determine,when,[ls138,1,y1_],asserted],
                        [determine,when,[or,2,in2],asserted]),
                        concept(equivalent_goal,1,[object(device)])).
>> element(250,rule(concept(equivalent_goal,1,[object(device)])),
                     elements([223])).
** element(251,objective(solved,[determine,when,[or,2,in1],asserted]),
                     concept(equivalent_goal,1,[object(device)])).
   element(252,equivalent([determine,when,[cpu,1,lds_],asserted],
                        [determine,when,[or,2,in1],asserted]),
                        concept(equivalent_goal,1,[object(device)])).
>> element(253,rule(concept(equivalent_goal,2,[object(device)])),
                     elements([229])).
   element(254,equivalent([determine,when,[cpu,1,uds_],asserted],
                        [determine,when,[cpu,1,uds_],low]),
                        concept(equivalent_goal,2,[object(device)])).
>> element(255,rule(concept(equivalent_goal,2,[object(device)])),
                     elements([226])).
   element(256,equivalent([determine,when,[ls138,1,y1_],asserted],
                        [determine,when,[ls138,1,y1_],low]),
                        concept(equivalent_goal,2,[object(device)])).
>> element(257,rule(concept(equivalent_goal,2,[object(device)])),
                     elements([223])).
   element(258,equivalent([determine,when,[cpu,1,lds_],asserted],
                        [determine,when,[cpu,1,lds_],low]),
                        concept(equivalent_goal,2,[object(device)])).
>> element(259,rule(concept(equivalent_goal,3,[object(device)])),
                     elements([232])).
** element(260,objective(solved,[determine,when,[cpu,1,ds_],asserted]),
                     concept(equivalent_goal,3,[object(device)])).
   element(261,equivalent([determine,when,[cpu,1,ds_],low],
                        [determine,when,[cpu,1,ds_],asserted]),
                        concept(equivalent_goal,3,[object(device)])).
>> element(262,rule(concept(equivalent_goal,4,[object(mc68000)])),
                     elements([223])).
   element(263,equivalent([determine,when,[cpu,1,lds_],asserted],
                        [determine,when,[cpu,1,ds_],asserted]),
                        concept(equivalent_goal,4,[object(mc68000)])).
>> element(264,rule(concept(equivalent_goal,5,[object(mc68000)])),
                     elements([229])).
   element(265,equivalent([determine,when,[cpu,1,uds_],_233],
                        [determine,when,[cpu,1,ds_],asserted]),
                        concept(equivalent_goal,5,[object(mc68000)])).
>> element(266,rule(concept(tCHSL,2,[object(mc68000)])),elements([5,4,260])).
** element(267,objective(solved,[propagate,
                     event([state([cpu,1,clk],s2),tCHSL],
                     state([cpu,1,uds_],asserted))]),
                     concept(tCHSL,2,[object(mc68000)])).
** element(268,objective(solved,[propagate,
                     event([state([cpu,1,clk],s2),tCHSL],
                     state([cpu,1,lds_],asserted))]),
                    concept(tCHSL,2,[object(mc68000)])).
   element(269,event([state([cpu,1,clk],s2),tCHSL],
                       state([cpu,1,lds_],asserted)),
                       concept(tCHSL,2,[object(mc68000)])).
   element(270,event([state([cpu,1,clk],s2),tCHSL],
                       state([cpu,1,uds_],asserted)),
                       concept(tCHSL,2,[object(mc68000)])).
   element(271,constraint([state([cpu,1,clk],s2),tCHSL],[3,60]),
                       concept(tCHSL,2,[object(mc68000)])).
   element(272,value([state([cpu,1,clk],s2),tCHSL],[3,60]),
                       concept(tCHSL,2,[object(mc68000)])).
>> element(273,rule(concept(tAVSL,2,[object(mc68000)])),elements([4,260])).
** element(274,objective(solved,[propagate,
                    event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL]
                    ,state([cpu,1,uds_],asserted))]),
                    concept(tAVSL,2,[object(mc68000)])).
** element(275,objective(solved,[propagate,
                    event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL]
                    ,state([cpu,1,lds_],asserted))]),
                    concept(tAVSL,2,[object(mc68000)])).
   element(277,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                       state([cpu,1,lds_],asserted)),
                       concept(tAVSL,2,[object(mc68000)])).
   element(278,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                       state([cpu,1,uds_],asserted)),
                       concept(tAVSL,2,[object(mc68000)])).
   element(279,constraint([state([cpu,1,addressbus([_233,_234])],valid),
                       tAVSL],[30,undefined]),
                       concept(tAVSL,2,[object(mc68000)])).
   element(280,value([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                              [30,undefined]),
                              concept(tAVSL,2,[object(mc68000)])).
>> element(281,rule(concept(equivalent_goal,1,[object(device)])),
           elements([251])).
>> element(283,rule(concept(equivalent_goal,1,[object(device)])),
           elements([246])).
   element(284,equivalent([determine,when,[and,1,in2],asserted],
                       [determine,when,[or,1,in2],asserted]),
                       concept(equivalent_goal,1,[object(device)])).
   element(285,equivalent([determine,when,[and,1,in2],asserted],
                       [determine,when,[or,2,in2],asserted]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(286,rule(concept(equivalent_goal,1,[object(device)])),
           elements([245])).
   element(287,equivalent([determine,when,[or,1,in2],asserted],
                       [determine,when,[or,2,in2],asserted]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(288,rule(concept(equivalent_goal,1,[object(device)])),
           elements([244])).
>> element(289,rule(concept(equivalent_goal,1,[object(device)])),
           elements([241])).
>> element(291,rule(concept(equivalent_goal,1,[object(device)])),
           elements([239])).
** element(292,objective(solved,[determine,when,[cpu,1,as_],low]),
           concept(equivalent_goal,1,[object(device)])).
   element(293,equivalent([determine,when,[ls138,1,e0_],low],
                       [determine,when,[cpu,1,as_],low]),
                       concept(equivalent_goal,1,[object(device)])).
>> element(294,rule(concept(equivalent_goal,1,[object(device)])),
           elements([238])).
```

```
** element(295,objective(solved,[determine,when,[power,1,gnd],low]),
          concept(equivalent_goal,1,[object(device)])).
   element(296,equivalent([determine,when,[ls138,1,e1_],low],
                          [determine,when,[power,1,gnd],low]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(297,rule(concept(equivalent_goal,1,[object(device)])),
          elements([237])).
** element(298,objective(solved,[determine,when,[power,1,vcc],high]),
          concept(equivalent_goal,1,[object(device)])).
   element(299,equivalent([determine,when,[ls138,1,e2],high],
                          [determine,when,[power,1,vcc],high]),
                          concept(equivalent_goal,1,[object(device)])).
>> element(300,rule(concept(equivalent_goal,2,[object(device)])),
          elements([260])).
   element(302,equivalent([determine,when,[cpu,1,ds_],asserted],
                          [determine,when,[cpu,1,ds_],low]),
                          concept(equivalent_goal,2,[object(device)])).
>> element(303,rule(concept(equivalent_goal,3,[object(device)])),
          elements([239])).
** element(304,objective(solved,[determine,when,[ls138,1,e0_],asserted]),
          concept(equivalent_goal,3,[object(device)])).
   element(305,equivalent([determine,when,[ls138,1,e0_],low],
                          [determine,when,[ls138,1,e0_],asserted]),
                          concept(equivalent_goal,3,[object(device)])).
>> element(306,rule(concept(equivalent_goal,3,[object(device)])),
          elements([238])).
** element(307,objective(solved,[determine,when,[ls138,1,e1_],asserted]),
          concept(equivalent_goal,3,[object(device)])).
   element(308,equivalent([determine,when,[ls138,1,e1_],low],
                          [determine,when,[ls138,1,e1_],asserted]),
                          concept(equivalent_goal,3,[object(device)])).
>> element(309,rule(concept(equivalent_goal,3,[object(device)])),
          elements([237])).
** element(310,objective(solved,[determine,when,[ls138,1,e2],asserted]),
          concept(equivalent_goal,3,[object(device)])).
   element(311,equivalent([determine,when,[ls138,1,e2],high],
                          [determine,when,[ls138,1,e2],asserted]),
                          concept(equivalent_goal,3,[object(device)])).
>> element(312,rule(concept(solved_goal,1,[type(achieve)])),elements([67,238])).
>> element(313,rule(concept(solved_goal,1,[type(achieve)])),elements([66,237])).
>> element(314,rule(concept(equivalent_goal,1,[object(device)])),
          elements([304])).
** element(315,objective(solved,[determine,when,[cpu,1,as_],asserted]),
          concept(equivalent_goal,1,[object(device)])).
   element(316,equivalent([determine,when,[ls138,1,e0_],asserted],
                              [determine,when,[cpu,1,as_],asserted]),
                              concept(equivalent_goal,1,[object(device)])).
>> element(317,rule(concept(equivalent_goal,1,[object(device)])),
          elements([292])).
>> element(318,rule(concept(equivalent_goal,2,[object(device)])),
          elements([304])).
   element(319,equivalent([determine,when,[ls138,1,e0_],asserted],
                              [determine,when,[ls138,1,e0_],low]),
                              concept(equivalent_goal,2,[object(device)])).
>> element(320,rule(concept(equivalent_goal,3,[object(device)])),
          elements([292])).
   element(321,equivalent([determine,when,[cpu,1,as_],low],
                              [determine,when,[cpu,1,as_],asserted]),
                              concept(equivalent_goal,3,[object(device)])).
>> element(322,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
         elements([275])).
   element(323,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                      state([or,2,in1],asserted)),
                      connected([or,2,in1],[cpu,1,lds_])).
   element(324,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                      state([or,2,in1],low)),
                      connected([or,2,in1],[cpu,1,lds_])).
>> element(325,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
         elements([274])).
   element(326,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                      state([or,1,in1],asserted)),
                      connected([or,1,in1],[cpu,1,uds_])).
   element(327,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                      state([or,1,in1],low)),
                      connected([or,1,in1],[cpu,1,uds_])).
>> element(328,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
         elements([268])).
   element(329,event([state([cpu,1,clk],s2),tCHSL],
                      state([or,2,in1],asserted)),
                      connected([or,2,in1],[cpu,1,lds_])).
   element(330,event([state([cpu,1,clk],s2),tCHSL],state([or,2,in1],low)),
         connected([or,2,in1],[cpu,1,lds_])).
>> element(331,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
         elements([267])).
   element(332,event([state([cpu,1,clk],s2),tCHSL],
                      state([or,1,in1],asserted)),
                      connected([or,1,in1],[cpu,1,uds_])).
   element(333,event([state([cpu,1,clk],s2),tCHSL],state([or,1,in1],low)),
             connected([or,1,in1],[cpu,1,uds_])).
>> element(334,rule(concept(tCHSL,1,[object(mc68000)])),elements([4,315])).
** element(335,objective(solved,[propagate,
             event([state([cpu,1,clk],s2),tCHSL],
             state([cpu,1,as_],asserted))]),
             concept(tCHSL,1,[object(mc68000)])).
   element(336,event([state([cpu,1,clk],s2),tCHSL],
                      state([cpu,1,as_],asserted)),
                      concept(tCHSL,1,[object(mc68000)])).
>> element(337,rule(concept(tAVSL,1,[object(mc68000)])),elements([4,315])).
** element(338,objective(solved,
             [propagate,
              event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
              state([cpu,1,as_],asserted))]),
             concept(tAVSL,1,[object(mc68000)])).
   element(340,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                      state([cpu,1,as_],asserted)),
                      concept(tAVSL,1,[object(mc68000)])).
>> element(341,rule(concept(equivalent_goal,1,[object(device)])),
         elements([315])).
>> element(343,rule(concept(equivalent_goal,2,[object(device)])),
         elements([315])).
   element(345,equivalent([determine,when,[cpu,1,as_],asserted],
                      [determine,when,[cpu,1,as_],low]),
                      concept(equivalent_goal,2,[object(device)])).
>> element(346,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
         elements([338])).
   element(347,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                      state([ls138,1,e0_],asserted)),
                      connected([ls138,1,e0_],[cpu,1,as_])).
   element(348,event([state([cpu,1,addressbus([_233,_234])],valid),tAVSL],
                      state([ls138,1,e0_],low)),
                        connected([ls138,1,e0_],[cpu,1,as_])).
>> element(349,rule(concept(propagate,1,[type(achieve),object(mc68000)])),
          elements([335])).
   element(350,event([state([cpu,1,clk],s2),tCHSL],
                        state([ls138,1,e0_],asserted)),
                        connected([ls138,1,e0_],[cpu,1,as_])).
   element(351,event([state([cpu,1,clk],s2),tCHSL],state([ls138,1,e0_],low)),
          connected([ls138,1,e0_],[cpu,1,as_])).
>> element(352,rule(concept(ls138,1,[object(device)])),elements([226])).
   element(353,event([state([ls138,1,e0_],low),tgate_delay],
                        state([ls138,1,y1_],asserted)),
                        concept(ls138,1,[object(device)])).
   element(354,event([state([ls138,1,e0_],low),tgate_delay],
                        state([and,1,in2],asserted)),
                        connected([and,1,in2],[ls138,1,y1_])).
   element(355,event([state([ls138,1,e0_],low),tgate_delay],
                        state([and,1,in2],low)),
                        connected([and,1,in2],[ls138,1,y1_])).
   element(356,event([state([ls138,1,e0_],low),tgate_delay],
                        state([or,2,in2],asserted)),
                        connected([or,2,in2],[ls138,1,y1_])).
   element(357,event([state([ls138,1,e0_],low),tgate_delay],
                        state([or,2,in2],low)),
                        connected([or,2,in2],[ls138,1,y1_])).
   element(358,event([state([ls138,1,e0_],low),tgate_delay],
                        state([or,1,in2],asserted)),
                        connected([or,1,in2],[ls138,1,y1_])).
   element(359,event([state([ls138,1,e0_],low),tgate_delay],
                        state([or,1,in2],low)),
                        connected([or,1,in2],[ls138,1,y1_])).
>> element(360,rule(concept(or_gate,4,[object(logic_gate)])),elements([169])).
   element(361,event([state([or,1,in2],low),tgate_delay],
                        state([or,2,out],low)),
                        concept(or_gate,4,[object(logic_gate)])).
   element(362,event([state([or,1,in2],low),tgate_delay],
                        state([ram,2,oe_],low)),
                        connected([ram,2,oe_],[or,2,out])).
   element(363,event([state([or,1,in2],low),tgate_delay],
                        state([ram,2,oe_],asserted)),
                        connected([ram,2,oe_],[or,2,out])).
   element(364,event([state([or,1,in2],low),tgate_delay],
                        state([ram,2,cs_],low)),
                        connected([ram,2,cs_],[or,2,out])).
   element(365,event([state([or,1,in2],low),tgate_delay],
                        state([ram,2,cs_],asserted)),
                        connected([ram,2,cs_],[or,2,out])).
>> element(366,rule(concept(or_gate,4,[object(logic_gate)])),elements([165])).
   element(367,event([state([or,1,in2],low),tgate_delay],
                        state([or,1,out],low)),
                        concept(or_gate,4,[object(logic_gate)])).
   element(368,event([state([or,1,in2],low),tgate_delay],
                        state([ram,1,oe_],low)),
                        connected([ram,1,oe_],[or,1,out])).
   element(369,event([state([or,1,in2],low),tgate_delay],
                        state([ram,1,oe_],asserted)),
                        connected([ram,1,oe_],[or,1,out])).
   element(370,event([state([or,1,in2],low),tgate_delay],
                        state([ram,1,cs_],low)),
                        connected([ram,1,cs_],[or,1,out])).
   element(371,event([state([or,1,in2],low),tgate_delay],
                           state([ram,1,cs_],asserted)),
                           connected([ram,1,cs_],[or,1,out])).
>> element(372,rule(concept(resolve,1,[])),elements([94,365,200,81])).
   element(373,equivalent([state([ram,2,addressbus([a0,a14])],valid),
                                state([ram,2,cs_],asserted)],
                                state([ram,2,cs_],asserted)),
                                concept(resolve,1,[])).
   element(374,value([state([ram,2,cs_],asserted),tACS],[undefined,100]),
           concept(resolve,1,[])).
>> element(375,rule(concept(resolve,1,[])),elements([90,371,201,74])).
   element(376,equivalent([state([ram,1,addressbus([a0,a14])],valid),
                                state([ram,1,cs_],asserted)],
                                state([ram,1,cs_],asserted)),
                                concept(resolve,1,[])).
   element(377,value([state([ram,1,cs_],asserted),tACS],[undefined,100]),
           concept(resolve,1,[])).
>> element(378,rule(concept(common_ref,3,[])),elements([190])).
   element(379,common_ref([state([cpu,1,clk],s7),
                               state([cpu,1,databus([d0,d7])],valid)],
                               state([cpu,1,clk],s2)),concept(common_ref,3,[])).
>> element(380,rule(concept(determine_value,2,[])),elements([379,63,61])).
   element(381,value([state([cpu,1,databus([d0,d7])],valid),tDICL],
ÿ                      [132.500000,undefined]),concept(determine_value,2,[])).
```

Bibliography

[1] Allen, James F., "Towards a General Theory of Action and Time", *Artificial Intelligence*, vol. 23(2), 1984, pp. 123 - 54.

[2] Anderson, J. R., *The Architecture of Cognition*, Cambridge, Mass.:Harvard University Press, 1983.

[3] Anderson, John R., "The Expert Module", in *Foundations of Intelligent Tutoring Systems*, Martha C. Polson, J. Jeffrey Richardson (eds), 1989, pp. 21 - 53.

[4] Anderson, J. R., C. F. Boyle, B. J. Reiser, "Intelligent Tutoring Systems", *Science*, vol. 228, Apr. 1985, pp. 456 - 62.

[5] Beetz, Michael, "Specifying Meta-Level Architectures for Rule-Based Systems", *GWAI-87, 11th German Workshop on Artificial Intelligence*, K. Morik (ed.), 1987, pp.149 - 59.

[6] Boulet, M.-M., et al, "A Design Task Advisor", in *Proceedings of the 4th International Conference on AI and Education*, D. Bierman, J. Brahan, J. Sandberg (eds), May 1989, pp. 25 - 31.

[7] Brecht, Barbara, Marlene Jones, "Student Models: The Genetic Graph Approach", Research Report 88-2, ARIES lab, U. of Saskatchewan, 1988.

[8] Brown, John Seely, Kurt VanLehn, "Repair Theory: A Generative Theory of Bugs in Procedural Skills", *Cognitive Science*, vol. 4(4), Oct-Dec 1980, pp. 379 - 426.

[9] Brown, John Seely, Richard R. Burton, Johan de Kleer, "Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III", in *Intelligent Tutoring Systems*, D. Sleeman & J S. Brown (eds), 1982, pp. 227 - 82.

[10] Burton, Richard R., "Diagnosing Bugs in a Simple Procedural Skill",in *Intelligent Tutoring Systems*, D. Sleeman & J S. Brown (eds), 1982, pp.157 - 83.

[11] Burton, Richard R., John Seely Brown, "An Investigation of Computer Coaching for Informal Learning Activities", in *Intelligent Tutoring Systems*, D. Sleeman & J S. Brown (eds), 1982, pp.79 - 98.

[12] Charniak, Eugene, C. K. Riesbeck, D. V. McDermott, J. R. Meehan, *Artificial Intelligence Programming*, (2nd ed.), New Jersey:Lawrence Erlbaum Associates, 1987.

[13] Clancey, William J., "Tutoring Rules For Guiding a Case Method Dialogue", in *Intelligent Tutoring Systems*, D. Sleeman & J S. Brown (eds), 1982, pp.201 - 25.

[14] Clancey, William J., "The Epistemology of a Rule-Based Expert System: A Framework for Explanation", *Artificial Intelligence*, vol. 23(2), 1984, pp. 123 - 54.

[15] Clements, Alan, *Microprocessor Systems Design*, Boston:PWS Publishers, 1987.

[16] Clocksin, William F., Christopher S. Mellish, *Programming in Prolog*, 3rd ed., Berline:Springer-Verlag, 1987.

[17] De Kleer, Johan, "How Circuits Work", *Artificial Intelligence*, vol 24, 1984, pp. 205 - 80.

[18] De Kleer, Johan, "Choices Without Backtracking", *Proceedings of the National Conference on Artifical Intelligence*, 1984, pp. 79 - 85

[19] De Kleer, Johan, John Seely Brown, "A Qualitative Physics Based on Confluences", *Artificial Intelligence*, vol 24, 1984, pp. 7 – 83.

[20] Engelmore, Robert, Tony Morgan (eds), *Blackboard Systems*, Workingham, England:Addison-Wesley Publishing Co., 1988.

[21] Fogel, Earl, "Teaching Prolog Using ICAI and a Graphical Trace", M. Sc Thesis, University of British Columbia, 1988.

[22] Forbus, Kenneth D., "Qualitative Process Theory", *Artificial Intelligence*, vol 24(1), 1984, pp. 85 – 168.

[23] Frederiksen, John R., et al., "Intelligent Tutoring Systems for Electronic Troubleshooting", in *Intelligent Tutoring Systems: Lessons Learned*, 1988, pp.351 – 68.

[24] Genesereth, Michael R., "The Role of Plans in Intelligent Teaching Systems", in *Intelligent Tutoring Systems*, D. Sleeman, & J. S. Brown (eds), 1982, pp. 79 – 98.

[25] Goldstein, Ira P., "The Genetic Graph: a Representation for the Evolution of Procedural Knowledge", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 51 – 77.

[26] Goodman,
Danny, *Danny Goodman's HyperCard Developer's Guide*, Toronto:Bantam Books, 1988.

[27] Halasz, Frank G., "Reflictions on Notecards: Seven Issues for the Next Generation of Hypermedia System", *Communications of the ACM*, vol. 31(7), July 1988, pp. 836 – 52.

[28] Hayes-Roth, Barbara, Frederick Hayes-Roth, "A Cognitive Model of Planning", *Cognitive Science*, vol 3,1979, pp. 275 - 310.

[29] Hayes-Roth, Barbara, "A Blackboard Architecture for Control", *Artificial Intelligences*, vol 26, 1985, pp. 251 - 321.

[30] Kimball, Ralph, "A Self-Improving Tutor for Symbolic Integration", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 283 - 307.

[31] Kuipers, B. ,"Qualitative Simulation", *Artificial Intelligence*, vol 29, 1986, pp. 289 - 338.

[32] Kuipers, Benjamin, "Qualitative Simulation As Causal Explanation", *IEEE Transactions on Systems, Man, and Cybernetics*, vol SMC-17(3), May/June 1987, pp. 432 - 444.

[33] Mandl, Heinz, Alan Lesgold (eds), *Learning Issues for Intelligent Tutoring Systems*, New York:Springer-Verlag, 1988.

[34] Matz, M., "Towards a Process Model for High School Algebra Errors", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 25 - 50.

[35] McCalla, Gordon I., Jim E. Greer, "Intelligent Advising in Problem Solving Domains: The SCENT-3 Architecture", Research Report 88-1, ARIES Lab, University of Saskatchewan, 1988.

[36] McCune, B. P., et al., "RUBRIC : A System for Rule-based Information Retrieval", *IEEE Transactions on Software Engineering*, SE-11(9), 1985, pp. 939 - 45.

[37] McDermott, Drew V., "A Temporal Logic for Reasoning About Processes and Plans", *Cognitive Science*, vol. 6, 1982, pp. 101 - 55.

[38] Miller, Mark L., "A Structured Planning and Debugging Environment for Elementary Programming", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 119 - 35.

[39] *M68000 Microprocessors User's Manual*, 6th eds., Englewood Cliffs, N.J.:Prentice Hall, 1989.

[40] O'Shea, Tim, "A Self-Improving Quadratic Tutor", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 309 - 336.

[41] Polson, Martha C., J. Jeffrey Richardson (eds), *Foundations of Intelligent Tutoring Systems*, New Jersey:Lawrence Erlbaum Associates,1988.

[42] Psotka, Joseph, L. Dan Massey, Sharon A. Mutter (eds), *Intelligent Tutoring Systems: Lessons Learned*, New Jersey:Lawrence Erlbaum Associates, 1988.

[43] Reiser, B. J., J. R. Anderson, R. G. Farrell, "Dynamic Student Modelling in an Intelligent Tutor for Lisp Programming", *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 1985, pp. 8 - 14.

[44] Reiser, B. J., et al., "Knowledge Representation and Explanation in GIL, an Intelligent Tutor for Programming", CSL Report 37, Princeton University, Feb. 1989.

[45] Reiser, B. J., et al., "Facilitating Students' Reasoning with Causal Explanations and Visual Representations", in *Proceedings of the 4th International Conference on AI and Education*, D.Bierman, J. Breuker, J. Sandbery (eds), 1989.

[46] Shoham, Yoav, "Temporal Logics in AI: Semantical and Ontological Considerations", *Artificial Intelligence*, vol 33(1), Sep. 1987, pp. 89 - 104.

[47] Shoham, Yoav, *Reasoning About Change*, Cambridge, Mass.:MIT Press, 1988.

[48] Sleeman, D., "Assessing Aspects of Competence in Basic Algebra", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 185 - 99.

[49] Sleeman, D., "An Attempt to Understand Students' Understanding of Basic Algebra", *Cognitive Science*, vol 8(4) Oct-Dec 1984, pp.387 - 412.

[50] Sleeman,D., J. S. Brown (eds), *Intelligent Tutoring Systems*, New York:Academic Press, 1982.

[51] Sleeman, D., R. J. Hendley, "ACE: A System which Analyses Complex Explanations", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 99 - 118.

[52] Smith, John B., Stephen F. Weiss, "Hypertext", *Communications of the ACM*, vol. 31(7), July, 1988, pp. 816 - 19.

[53] Sterling, Leon, Ehud Shapiro, *The Art of Prolog*, Cambridge, Mass.:MIT Press, 1986.

[54] Stevens, Albert, Allan Collins, & Sarah E. Goldin, "Misconceptions in Students' Understanding", in *Intelligent Tutoring Systems*, D. Sleeman & J. S. Brown (eds), 1982, pp. 13 - 24.

[55] Tenney, Yvette J., "Issues in Developing an Intelligent Tutor for a Real-World Domain: Training in Radar Mechanics", in *Intelligent Tutoring Systems:Lessons Learned*, Joseph Psotka, L. Dan Massey, Sharon A. Mutter (eds), 1988, pp. 119 - 80.

[56] VanLehn, Kurt, "Student Modeling", in *Foundations of Intelligent Tutoring Systems*, Martha C. Polson, J. Jeffrey Richardson (eds), 1989, pp. 55 - 78.

[57] Wachsmuth, Ipke, "Modeling the Knowledge Base of Mathematics Learners: Situation-Specific and Situation-Nonspecific Knowledge", in *Learning Issues in Intelligent Tutoring Systems*, Heinz Mandl, Alan Lesgold (eds), 1988, pp. 63 - 79.

[58] Wenger, Etiene, *Artificial Intelligence and Tutoring Systems*, Los Altos:Morgan Kaufmann Publishers, Inc., 1987.