Constructing Functional Models of a Device from its Structural Description

By

Sunil Thadani, Ph.D.

The Ohio State University, 1994

B. Chandrasekaran, Adviser

This work proposes a solution for a version of the device understanding problem. Several tasks involving reasoning about devices, such as diagnosis, require an account, at multiple levels of abstractions, of how a specific function of the device comes about. Given the structural description of a device, hypotheses are generated about its functions and how they come about, in the form of function-specific device models at multiple levels of abstraction (FR models).

The proposed method uses the knowledge of frequently encountered abstract devices in the domain to derive functions and FR models from structure. Understanding of the abstract devices provides both the function-specific models of the parts and the abstractions required to build FR models. Two main strategies are used by the system. The first one is to try to understand the given device as an instance of already known abstract devices, and using the functions and FRs of the abstract devices to hypothesize functions and FRs for the given device. Additional methods are used to verify if the hypothesized functions are in fact applicable to the given device. If the first strategy does not work for a device, the second one is used. The second strategy is to use a decomposition-based method to view the given device as consisting of a combination of the instances of the already known abstract devices. The structural description of the given device is transformed by relabeling parts of the given device as instances of already known abstract devices, and the same two strategies may then be used to understand the new device. Recursive application of these two strategies is used to generate the functions and FRs of the given device.

Constructing Functional Models of a Device from its Structural Description

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the Graduate

School of The Ohio State University

By

Sunil Thadani, B.Tech. in EE, M.S. in CIS

* * * * *

The Ohio State University

1994

Dissertation Committee:

B. Chandrasekaran

John Josephson

James F. Davis

Approved by

Adviser
College of Engineering

Acknowledgements

I wish to thank B. Chandrasekaran for providing the framework, environment, and the guidance that made this work possible. I wish to thank John Josephson, Jim Davis, and Tom Bylander for their insightful comments on the earlier drafts of this thesis. I would also like to thank Jyothi and our family for their patience and support during this long haul.

This work was supported in part by DARPA research contract #F49620-89-C-0110 and a fellowship for graduate research from BP America.

VITA

1984	Bachelor of Technology, Electrical Engineering Indian Institute of Technology Delhi, India
1984-1986	Assitant Engineer Hindustan Computers Limited Delhi, India
1987	M.S., Computer Science The Ohio State University Columbus, Ohio
1988, Summer	Summer Intern British Petroleum Research Cetner Cleveland, Ohio
1989, Summer	Summer Intern NCR, Dayton, Ohio
1986-88, 1992-93	Teaching Assistant Computer Science Department The Ohio State University Columbus, Ohio
1988-89	Graduate Reserarch Assistant Department of Journalism The Ohio State University Columbus, Ohio
1990-92, 1993-Present	Graduate Research Assistant Computer Science Department The Ohio State University Columbus, Ohio

Fields of Study

Major Field: Computer and Information Sciences

Studies in:

Artificial Intelligence
Parallel Architectures
Distributed Databases

Publications

Sunil Thadani, B. Chandrasekaran "Structure-to-Function Reasoning". *Proceedings of the AAAI Workshop on Reasoning about Function*, July 11-16, 1993, Washington, D.C.

Sunil Thadani "Extending causal sequences to make teleological distinctions". *Proceedings of IEEE Conference on AI Applications*, March 3-5, 1993, Orlando, Florida.

Lindley Darden, Dale Moberg, Sunil Thadani "Scientific theory change as a domain for incremental knowledge improvement". *Proceedings of AAAI Spring Symposium on Incremental Modification and Reuse*, March 25-27, 1992, Stanford, California.

Lindley Darden, Dale Moberg, Sunil Thadani, John Josephson "A computational approach to scientific theory revision: The TRANSGENE experiments". *OSU CIS LAIR Technical Report 92-LD-TRANSGENE.*

Richard Fox, John Josephson, Sunil Thadani "A three level abduction machine for word recognition from articulation". *OSU CIS LAIR Technical Report 91-RF-WORD-RECOGNITION.*

Table of Contents

List of Figures

CHAPTER I

Introduction

1.1 The problem

The importance of the problem of understanding how devices work cannot be overstated. Understanding how devices work is a problem solving activity that is performed very frequently by most humans. Given an unfamiliar device one has to first go through the process of understanding the working of the device, that is, what the device does and how it does that, before one can perform tasks such as diagnosing the device, predicting its behavior, and explaining its working. In this paper we are interested in the process of understanding itself, that is, in how an agent acquires the understanding of the working of a device from its structural description. More specifically, by "understanding a device" we mean determining the functions of the device and for each function generating a causal account of how the device function arises from the functions of its components. This may also be referred to as the causal understanding of a device.

1.2 Situating this work

The importance of the problem of device understanding is also evident from the large body of work that has been done on device understanding in the field of AI. The motivations to address this problem have been quite varied ranging from building more flexible and robust expert systems to understanding the cognitive aspects of common sense reasoning. The version of the problem addressed by most of the earlier work in AI in this area [dK85, For84, Kui86a] was that given the structural description of the device and given an input perturbation, determine the resulting behaviors of the device. And some [dK85, IS86] also addressed the problem of generating an account of how the specific behaviors of the device come about from its structural description. The strategy used by all of them was to provide the system with the models of the parts. These parts could be physical components or could be processes. The system would then compose the models of the parts to generate the model of the whole device. Simulation of the device model for the given input perturbation was used as the method for going from structure to the possible behaviors of the device. All these approaches shared the following important characteristics. *Simulation was proposed as the primary method of going from structure to behavior. The device models and the behavior descriptions generated were all at a single level of abstraction.* Further, *the device models and the behavior description generated were general purpose, in the sense that they were independent of what specific aspects of the device behaviors the agent was interested in.* This characterization applies to a large part of the work done on device understanding in AI. Such an approach to reasoning about devices is useful for, say, discovering the behavioral implications of a new design. But there are several tasks involving reasoning about devices in which device models with the above characteristics are not adequate for the purpose [VIFC93, IFVC93, Cha94, CGI93, IC92].

For any complex device, most tasks require one to reason about the device at multiple levels of abstraction and also be able to smoothly switch levels while reasoning. For example, to diagnose a complex device one first reasons about the device at a high level of abstraction to localize the malfunctioning portions, and then reason at a more detailed level with the malfunctioning portions only to further narrow down the fault. Device models at multiple levels of abstraction improve efficiency of such tasks. Further, there are lot of scenarios in which one is interested only in specific behaviors of the device. In such cases there is no need to simulate the general purpose device model generating several irrelevant device behaviors, because for most complex devices simulating the general purpose models is computationally very expensive. To efficiently generate only the behaviors of interest we need to be able to generate function-specific device models. And finally, for most complex devices component-level behavior descriptions at a single level of abstraction are usually quite large and do not lend themselves to any efficient strategy to organize the behavior descriptions of a large number of devices in memory.

The problems mentioned above have been addressed in various forms in some of the recent work in this area. Depending on the specific task and the specific domain, different researchers have focused on different aspects of the issues mentioned above. For example, [KC87, Str88] use information about higher order derivatives to eliminate the generation of some spurious behaviors. [ACP91, NJA91] have proposed a mechanism for using context specific component models rather than using general purpose component models. [FF91a] builds device models at the right level of abstraction appropriate for answering specific questions about the behavior of the device. [Wel86] proposes an abstraction technique for repeating behaviors. [Kui87] proposes organizing the models of various mechanisms in a device by abstraction levels along the time dimension. Even though the various versions of the issues mentioned in the previous paragraph have been addressed by these researchers, most of them assume the context that makes specific commitments, indicated in italics earlier in this section, to what constitutes understanding of a device and how the understanding of devices is organized.

What distinguishes the work at our laboratory on device understanding are the specific commitments made about what constitutes understanding of a device[1]. There are at least three parts to the problem of understanding devices. The first one is determining what constitutes understanding of a device. That is, determining the form and content of the knowledge that constitutes understanding. In addition to satisfying other constraints, the form and content of device understanding should be such that it should be able to support the set of tasks that one is expected to perform if one claims to understand the device. For example, as mentioned earlier, if one claims to understand a device one is expected to be able to predict the behavior of the device, diagnose the device, and explain how the behavior of the device comes about. The second part of the problem of device understanding is to determine how this understanding can actually be generated starting from the structural description of the device. The third part is to determine how to use this understanding for various tasks mentioned earlier [VIFC93, IC92, IFVC93, CGI93, Keu89, All90, Goe90]. Naturally, the commitments made for the first part will determine the characteristics of the approaches used to do the second and third part.

The proposal on functional representation (FR) [SC88, Cha94, IC92] from our laboratory makes clear commitments to the content and the form of the knowledge that constitutes understanding of a device, *independent of the specific task or the specific domain*. These commitments have been motivated by larger concerns about how the understanding of a large number of complex devices may be organized in memory and how this understanding may be used to efficiently perform various tasks mentioned earlier. The form and content of what constitutes understanding of a device also provide the form and content of the output expected of the task addressed in this thesis, which is to generate the understanding of the device from its structural description. The distinguishing characteristics of the output expected provide both the motivation as well as the justification for some of the characteristics of the approach used to perform this task. Therefore a clear understanding of the commitments made in FR is essential to understand the work presented in this thesis.

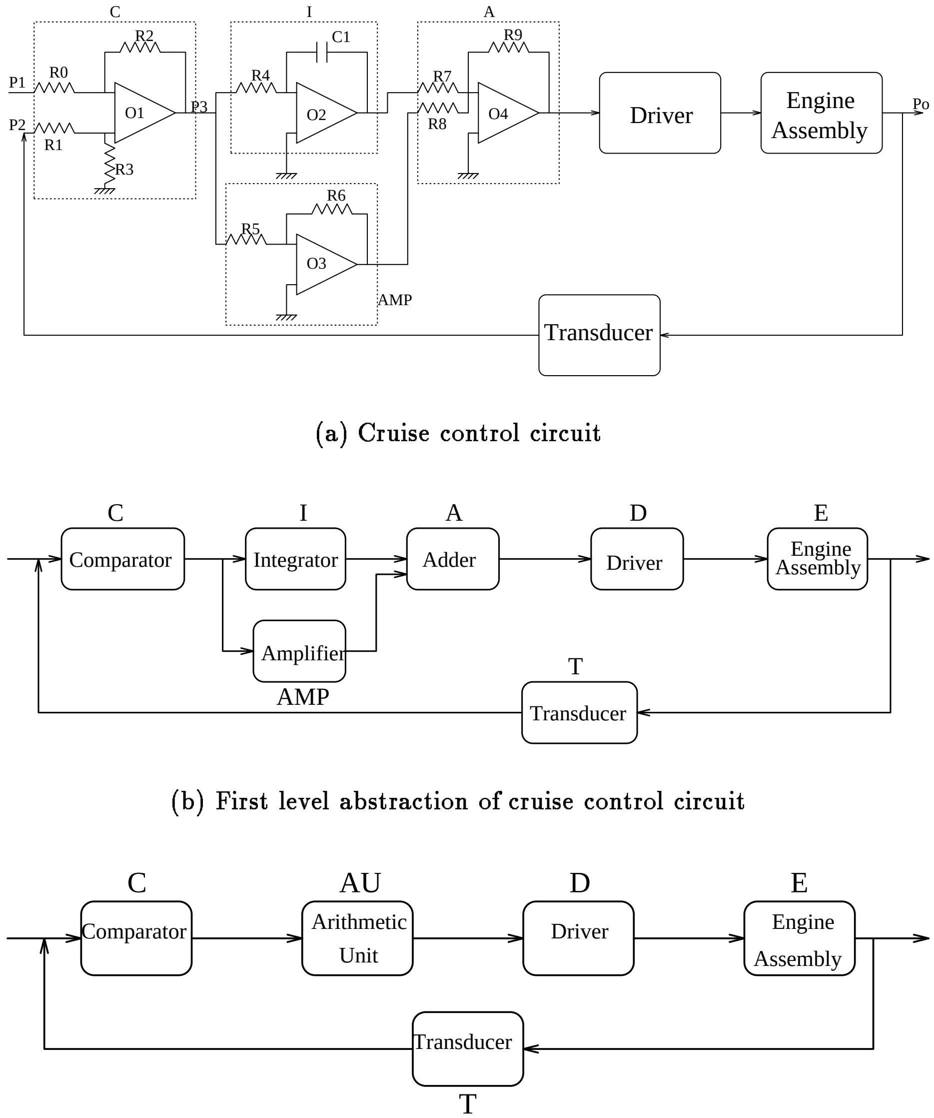
[1] Since in this work we are interested only in causal understanding of devices, "understanding" will be used to mean "causal understanding" of devices. We use the phrase "analyze the device" to refer to the process of understanding the device.

1.3 Characterizing the output

One of the important ideas stated in the proposal on FR is that the causal understanding of a device consists of a set of function-specific causal models of the device, also referred to as the functional representation (FR) models of the device, each of which is at multiple levels of abstraction enabling smooth transition between levels while reasoning.

Consider the circuit, shown in Figure 1(a), of a cruise control device used for controlling speed in automobiles. Even though the circuit is given in terms of components like op-amps, transistors, resistors, and capacitors, an expert describing how this circuit works uses several abstractions, such as comparator, adder, integrator, arithmetic-unit, and feedback loop. An expert's description of how this device works would be similar to the one as follows. The device shown in Figure 1(a) is understood in terms of the abstract structural descriptions shown in Figure 1(b) and Figure 1(c). The structural descriptions in Figure 1(b) and Figure 1(c) are the abstract descriptions of the circuit in Figure 1(a). The cruise control behavior of this device is described in terms of the control function of a feedback loop. The cruise control sets the speed of the engine proportional to the input voltage at P_1. Transducer T generates a voltage proportional to the output speed of the engine. This voltage is compared with the input voltage by the comparator C. Output of the comparator is an error voltage which is proportional to and opposite in sign to the error in the output speed. Arithmetic unit AU computes a function of the error voltage. The driver takes the output of the arithmetic unit and generates a driver voltage for the engine. If the output speed of the engine is higher than the speed set, a negative error voltage and a negative driver voltage is generated which reduces the speed of the engine. On the other hand, if the output speed is lower than the speed set, a positive error voltage and a positive driver voltage is generated which increases the speed of the engine. Such a description of the function of the cruise control device refers to several abstractions, such as feedback loop, comparator, and arithmetic unit. The function of the arithmetic-unit in turn is explained in terms of an adder, an integrator, and an amplifier. The functions of adder, integrator, and amplifier, in turn, are explained in terms of the functions of components like op-amp, resistors, and capacitors. Such a description of the function of the cruise control device is at multiple levels of abstractions and one can move between levels while reasoning with such a description.

Further, this description is a function-specific description of the cruise control device. That is, even though there are several other aspects of this device, but only the ones that are relevant to the cruise control function of this device are used in this description. Such function-specific models enable highly focused and computationally



(a) Cruise control circuit

(b) First level abstraction of cruise control circuit

(c) Final abstraction of cruise control circuit

Figure 1: Structural descriptions of cruise control circuit

efficient reasoning in tasks that use these models. These function-specific models achieve this efficiency by sacrificing generality, because these models can be used to simulate and explain only the modeled functions.

The description of the *cruise-control-func* given above is an English version of the FR model of the cruise control function of the device shown in Figure 1(a). According to the commitments made in the previous work this is the kind of description an agent is expected to have if the agent claims to understand the cruise control function of the device shown in Figure 1(a). This is also the kind of description we expect our system to generate as an output. These distinguishing characteristics of the target device model will explain some of the main features of the method proposed in this thesis to build such device models.

1.4 The task scenario

One of the key characteristics of our approach to understanding devices is that we assume that the agent already understands a lot of devices in the domain and uses that knowledge to understand a new device. And the understanding of the new device is in terms of the understanding of the devices the agent already knew about. For example, an expert working in the domain of electrical and electronic circuits already knows about a set of frequently encountered domain-specific and domain-independent devices, like voltage dividers, current dividers, filters, comparators, adders, integrators, and feedback loops. To understand an unfamiliar device in the same domain, say a cruise control circuit of Figure 1(a), the expert uses the understanding of some of the above devices. Parts of the new device may be recognized as instances of comparator, adder, integrator, and amplifier, and the understanding of the functions of these abstract devices may be used to understand the new device. Finally, the understanding of the new device is in terms of the same abstractions, that is, the function-specific causal models of the new device refer to the functions and causal models of the abstractions like comparator, integrator, adder, amplifier, and feedback loop.

A large class of device understanding problems correspond to this scenario, in which the agent already knows about a large number of abstract devices and uses that knowledge to understand an unfamiliar device. This typically corresponds to situations in which the agent is an expert in the domain. A technician diagnosing electronic audio equipment, a student analyzing electrical circuits after doing a basic course in circuit analysis, and a typical human being dealing with typical devices in day to day life - all have access to a large repertoire of abstractions that correspond to understanding of typical component configurations one encounters in the specific domain, that can be used to perform the reasoning outlined in the previous paragraph.

1.5 Outline of the thesis

Since we will be using the FR language to describe our causal models we will briefly outline the main characteristics of the FR language in chapter II. Also, a better understanding of FR will give a better idea about the form and content of the output that our method is aiming for. In chapter III we use the characteristics of the expected output, the FR model of the device function, to motivate our solution. Chapter IV defines the objects used by our system followed by a precise definition of the task. Chapter V describes the algorithm used to perform the task. Chapter VI gives example runs of the algorithm on two examples. This is followed by a search space analysis of the problem in chapter VII, giving a search space view of the task and also discussing the complexity of the various search spaces. Chapter VIII gives a comparison of our approach with others. The last chapter defines the limitations and the contributions of this work.

CHAPTER II

The Functional Representation (FR) language

By understanding of a device function we mean a description of how the device function is achieved using the functions of the components and the domain laws. Functions of most complex devices are understood at multiple levels of abstraction and the understanding at various levels is bridged so as to enable smooth transition between levels while reasoning. The FR language provides the primitives to represent such understanding of device functions. FR assumes that the function of a device is achieved by means of a causal process, that is, the device starting from some initial device state goes through a sequence of device states to achieve the state in which the function is achieved. An account of how such a function is achieved can be represented as a certain type of *causal process description* (cpd). A cpd represents the causal process underlying the function as a sequence of state transitions starting from some initial state to the final state in which the device function is achieved. And the state transitions of the causal process are explained either by appealing to component functions or domain laws.

In the FR of a device function, the device function is first described using a cpd that refers to functions of components and domain laws. The functions of components are in turn described using other cpds that refer to functions of subcomponents. That is, the functions of components at one level are described using cpds that refer to functions of subcomponents at the next lower level. An important intuition underlying such hierarchical functional descriptions is that a description of how a component achieves its function is irrelevant to understanding its role in another device. A component with a specific function in a device may be replaced by another component which has the same functionality but achieves its function differently.

2.1 Components of FR

The three main components of an FR are the structural descriptions, the functions, and the cpds.

2.1.1 Structure of the device

The structural description of a device specifies the components constituting the device and the relations between the components. An FR of a function of a device imposes a structure on the device in which it is described as being comprised of components, which supply various functionalities. As stated earlier, an FR of a device function would normally use several abstractions, that is, it will use functions of several abstract devices. For example, the FR for the cruise control device shown in Figure 1(a) uses several abstractions and refers to their structural descriptions. It refers to three structural descriptions one corresponding to each of the devices shown in Figure 1(a), 1(b), and 1(c). And to describe the functions of the abstract devices, like comparator and integrator, it also refers to the structural descriptions of these devices.

2.1.2 Functions

The function of a component is the role it plays in the containing device. But we also associate functions with stand-alone components, that is, components without a containing device. In such cases the functions of a component are the roles the component plays in the devices it typically occurs in. Function is given as the input-output behavioral specification of the device, that is, the function defines, given an input, what the output of the device would be. Input and output are specified in terms of the changes in the values of the state variables of the device. A function is represented by a frame with two slots, named *given* and *toachieve*. Both the *given* and the *toachieve* slots specify a predicate on the state variables. The predicate in the *given* slot corresponds to a value assignment to the independent state variables (also called exogenous state variables) of the device. These variables usually represent input to the device. The predicate in the *toachieve* slot corresponds to a value assignment, usually in terms of independent state variables, to the dependent state variables of the device. The dependent state variables of the device usually represent the output of the device. For example, for the function *amplify-func*, shown in Figure 2, of an *amplifier*, *given* is V_P, and *toachieve* is $(= V_Q \; KV_P)$, where V_P and V_Q are the input and output voltages of the *amplifier*.

Four different types of functions have been identified [Keu89, Cha94, IC92]. These are *toachieve*, *tomaintain*, *tocontrol*, and *toprevent*. All the four types of functions have the *given* and the *provided* slots, and the semantics of these two slots in all the four types of functions are the same as that defined for the *toachieve* function. In this work we will be using only the most commonly encountered function type, the *toachieve* type. The semantics of this function type is as if the goal of the device with *toachieve* type of function is to make the predicate in the *toachieve* slot true. In other

given: (= V_I V_P)

tomake: (= V_Q KV_P)

Figure 2: The *amplify-func*

words, the *toachieve* type of function of a device is said to have been achieved when the device gets into a state such that the predicate in the *toachieve* slot becomes true.

Also associated with each function of a component is the *Provided* slot that specifies the condition that should be true for the function to be applicable. That is, only if the *provided* condition is true will the device deliver the function specified by the *given* and *toachieve* slots. The *provided* condition can refer to anything, such as the values of the device state variables, values of component parameters, or some other global variables. In this work we will define a much more specific use of the *provided* slot.

2.1.3 Causal process descriptions

Associated with the function of a device may be a causal process description (cpd) describing how the function of the device comes about. As stated earlier, an FR imposes a structure on the device in which the device is described as being comprised of components, which supply various functionalities. A cpd describes how a particular arrangement of components in a device delivers the function of the device. A cpd is a directed acyclic graph [IC92, CGI93] with nodes corresponding to the partial states of the components and directed edges, also called links, implying transition from one partial state to another partial state. Annotations on the links explain the transition. Three kinds of annotations have been used on the links. *by-cpd* annotation points to another cpd that explains the transition. The cpd pointed to by the *by-cpd* annotation would usually represent a process whose details do not matter for the current purpose or can be explained separately. Using *by-cpd* annotation long process explanations can be hierarchically composed out of other process explanations, making explanations at each level shorter.

Using-func annotation refers to the function of the component that explains the state transition. This annotation may be used to construct cpds that explain the behavior of the device partly in terms of the functions of the components. The functions of the components in turn may be explained in terms of the functions of the subcomponents. As stated earlier, the description of how the function of the component comes about, is completely separate from the function of the component itself which is used in the cpd describing the behavior of the containing device. Cpd describing the function of the containing device may refer to the function of a component without even knowing how the component function arises from its subcomponents. This primitive enables one to describe device behaviors in terms of functional and structural hierarchies.

The *as-per* annotation refers to the name of the domain law that explains the state transition. Usually domain laws are used to compose the functions of subcomponents based on the interconnections between them.

Corresponding to each of the above link annotations, we have three kinds of links. The *by-cpd-link*, *using-func-link*, and *as-per-link*. For example, the cpd in Figure 5(a) has *using-func-link* type of link that refers to the function *compare* of component C, and the cpd in Figure 5(c) has *as-per-link* type of link that refers to a domain law *ohms-law*. In this work we use only the *using-func-link* and the *as-per-link*.

2.2 Extensions to FR

We have made a few extensions to the FR in this work. In the original FR proposal each link (or the state transition) connected exactly two states, the initial state and the final state. And the semantics of the state transition was that if the initial state is true then the final state will become true, and the explanation for the transition was provided by the annotation on the link. In the definition of the link, there is no reason to restrict the link to only two states. We have extended the semantics of the link to mean that if the set of initial partial states are true then the set of final partial states will become true and the annotation on the link explains the transition. Often there are situations when one needs to refer to several predecessor partial states to justify the transition to a particular state. With the extended semantics a link may have more than one initial states and more than one final states. Diagrammatically the new link is shown in Figure 3.

Another feature we have used that was not there in the original proposal on FR but has already been used extensively [Goe90, All90, dK85, Tha93], is the AND-branching in the cpd graph. That is, more than one link may converge into a state, or more than one link may go out of a state. The semantics of the links in Figure 4(a) would be that the partial state $S3$ would be true if and only if the partial states $S1$ AND $S2$ are true and if the functions $F1$ of $C1$ AND $F2$ of $C2$ are achieved. Semantics of the links in Figure 4(b) would be that if partial state $S1$ is true and if function $F1$ of $C1$ is achieved, then the partial state $S2$ is true AND if the function

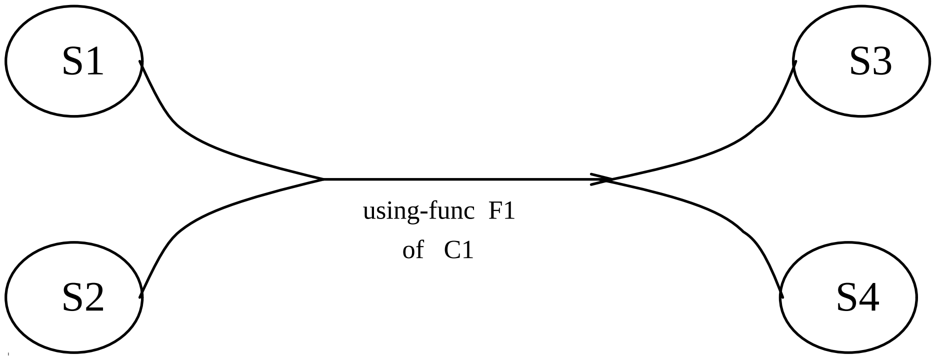
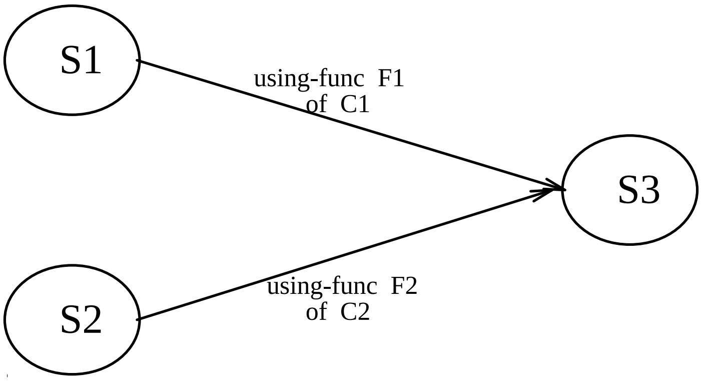
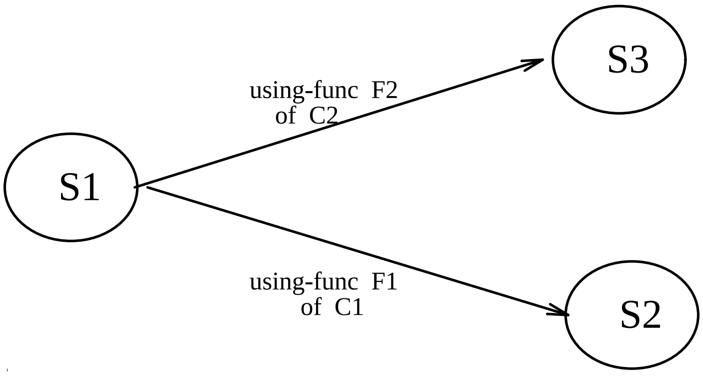


Figure 3: The new link

$F2$ of $C2$ is achieved, then the partial state $S3$ is true.



(a) The ANDed input



(b) The ANDed output

Figure 4: Cpd with ANDed links

CHAPTER III

Getting to the Solution

How can one generate the kind of FR models mentioned in the previous chapter starting from the structural description of the device? To describe our approach we shall start backwards from the characteristics of the output expected, and identify what kinds of knowledge are required to perform this task. We shall then propose the form in which these various knowledge types can be acquired and used to perform the task. This is followed by a high level description of the algorithm showing how the above mentioned knowledge can be used to perform the task.

3.1 What additional knowledge types are required?

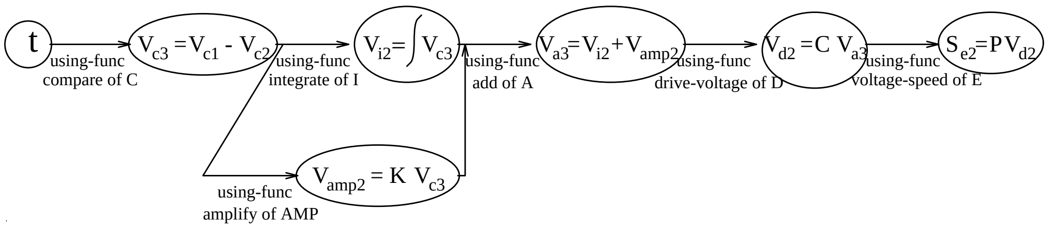
To identify the additional types of knowledge required to perform the task, let us look at the characteristics of the output of the task. One of the important characteristics of the output FR models is that for most non-trivial devices, the output FR models are at multiple levels of abstraction. And, the multiple levels of abstraction are so bridged that one can switch levels, without any significant overhead, while reasoning about the device. The input structural description does not have all these abstractions. For example, the FR of the cruise control uses abstractions such as comparator, adder, arithmetic unit, and feedback loop, none of which is given as part of the input structural description. FR uses both structural and behavioral abstractions. Components such as comparator and adder form the structural abstractions by hiding the structural details. Component *adder* in Figure 1(b) abstracts the structural description, in box A in Figure 1(a), that corresponds to the adder circuit. The component *adder* in turn becomes a part of another abstract component *arithmetic-unit*. Such recursive structural abstraction enables hierarchical structural organization of the device.

Behavioral abstractions in FR hide the behavioral details and provide new terms for behavior description. For example, the function *amplify* associated with the *amplifier* abstracts the cpd in Figure 5(c). The cpd *amplify-cpd* describes the amplification process in terms of voltages at various points in the given circuit. The function *amplify* abstracts from the voltage level description, hiding the details of the cpd and also introducing a new term "amplify" for the input-output behavior described by the corresponding cpd. Since these abstractions are not given as part of the input structural description, *the system needs knowledge to build structural and behavioral abstractions to build an FR* like the one shown in Figure 5.

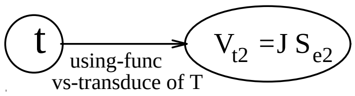
The other important characteristic of the output FR models is that the FR models are function-specific models of the device. That is, they model only a specific function of the device and hence can be used to simulate, explain, and diagnose only the modeled function of the device. An advantage of such function specific models is that they enable highly focused and computationally efficient reasoning in tasks that use these models. To build such a model, the agent has to select relevant aspects of the behaviors (subset of state variables and appropriate relations between them) for each abstract and primitive component, aspects that are relevant in describing the specific function of the device. For example, in the FR in Figure 5 (partial FR for the cruise-control function of the device shown in Figure 1(a)), for the part of the structural description that corresponds to the component *integrator*, only a small number of state variables corresponding to the function *integrate* are selected and an appropriate relation between them is defined. Even though there are man other state variables and parameters associated with the structural description of the component integrator, only a small subset of state variables is of interest in the context of the function *integrate*. *For each device function the system needs knowledge to derive appropriate relations between the relevant subset of state variables.*

3.2 Form of Knowledge

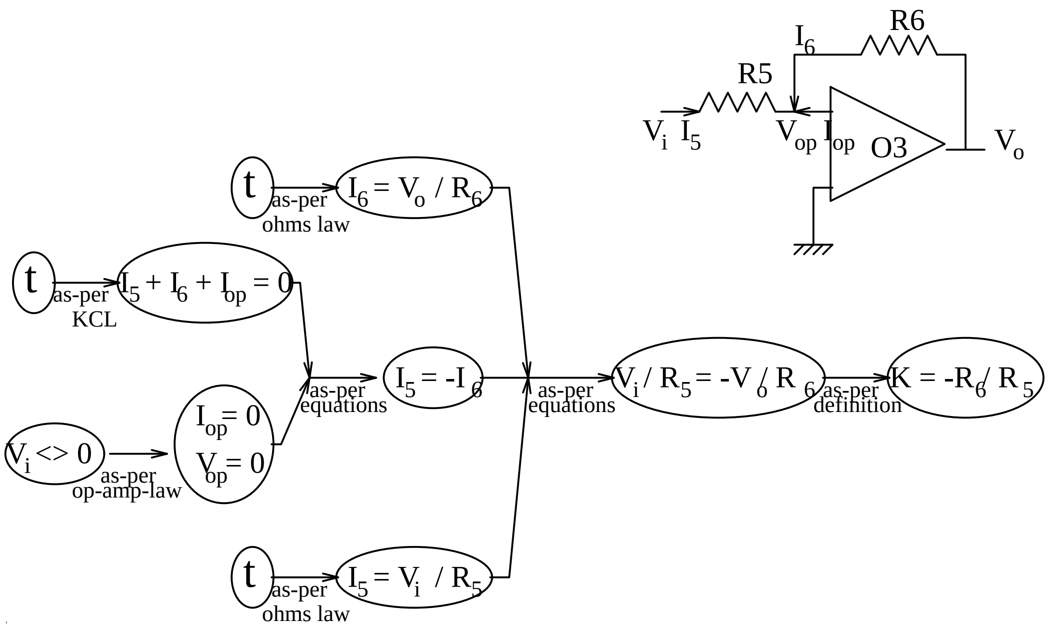
Having identified the kinds of knowledge required to solve the problem, we will now talk about the form in which this knowledge is provided to the agent. We represent the above mentioned knowledge types using structure-function-FR (SFF) templates. The reasoning system has access to a large number of SFF templates. Each SFF template represents the understanding of an abstract device in the domain. For example, an expert working with electronic circuits already knows about comparators, integrators, adders, amplifiers, and feedback loops. So a system reasoning about electronic circuits would be provided with SFFs for abstract devices like comparators, integrators, adders, amplifiers, and feedback loops. An SFF of an abstract device associates functions and FR templates of the device with the abstract structural description of the device. FR template and the abstract structural description are abstract descriptions in the sense that one needs to fill in additional details to get a specific instance of the FR or the structural description. For example, the SFF for the abstract device *comparator* associates the functions *compare-func* and *clipped-comparator-func*, and



(a) *Forward-path-cpd* of *feedback-loop*



(b) *Feedback-path-cpd* of *feedback-loop*



(c) *Amplify-cpd* of *amplifier*

Figure 5: Cpds for cruise control

the corresponding cpd templates *compare-cpd* and *clipped-compare-cpd* with the abstract structural description *comparator-sd* of the comparator. *compare-sd* defines an abstract structural description by defining the connections between sub-components of the comparator, where the sub-components are specified only by the component class they belong to. The corresponding functions and the FR templates referred to by the SFF also refer to the parameter names and quantity names of the sub-components specified in the abstract structural description *compare-sd*. This is how the SFF captures the understanding of the abstract device *comparator*, and as we shall show later, such an SFF can be used to generate the function and the FR for a specific instance of the comparator whose structural description matches the abstract structural description *comparator-sd*.

These SFFs provide both the abstraction terms as well as the knowledge required to map a structural description on to an abstraction. SFFs provide this knowledge for both the structural and behavioral abstractions. For example, given an instance of a structural description, like the one in box C in Figure 1(a), that matches the abstract structural description of the comparator, one can abstract the given structural description to the abstract component *comparator*. Thus, these SFFs can be used to form structural abstractions. SFFs also help form behavioral abstractions. Once the given structural description has been mapped on to an SFF, the function associated with the SFF, which abstracts a cpd of the abstract structural description, can be instantiated for, and associated with, the given structural description. For example, the cpd for the given structural description can be obtained by instantiating the *compare-cpd* template associated with the SFF. The resulting cpd can then be abstracted to the function *compare* associated with the comparator SFF. Thus, using an SFF a cpd associated with the given structural description can be abstracted to a function.

Further, each function associated with the SFF is defined in terms of only a small subset of state variables associated with the abstract structural description. In each SFF, possible functions of the abstract structural description have already been identified and each function has already been defined in terms of a small subset of state variables and appropriate relations between them. Once again consider the example of the SFF for the comparator. The function *compare*, associated with the comparator SFF, is defined in terms of voltages at points *P1, P2,* and *P3* only, even though there are so many other state variables associated with the given structural description of the comparator, like the voltages at the other two junctions, currents at various points, and power consumption in various parts. This is how SFFs capture the knowledge of what specific state variables and relations between them are relevant for a specific function of the given structural description. Thus, SFFs capture both the kinds of knowledge identified in the previous section that are required to perform the task.

3.3 How the knowledge is used

How are these SFFs used to generate the functions and the corresponding FRs of the given structural description? Our approach was primarily motivated by the way an expert in a domain would understand a new device in the same domain. An expert already understands a lot of abstract devices in the domain and uses the understanding of abstract devices to understand new devices. As stated earlier, the system has access to a large number of SFFs that correspond to the set of abstract devices an expert working in that domain already understands. Given a structural description, the system identifies parts of the structural description that match abstract structural descriptions associated with the SFFs. For example, the part of the circuit in box C in Figure 1(a) matches the abstract structural description of the SFF for the comparator. A part of the given structural description might match the abstract structural descriptions of more than one SFF. For each such part of the given structural description the system fetches the set of SFF candidates that might match the part of the structural description.

The SFF also points to the possible functions and the corresponding FR templates for the abstract structural description associated with the SFF. The functions associated with the matching SFF candidates become the hypothesized functions of the structural description in box C in Figure 1(a). For example, the functions associated with the comparator SFF, *compare-func* and *clipped-compare-func*, are hypothesized to be the possible functions for the structural description in box C. The system then verifies which of the SFF candidates and which of the hypothesized functions are actually applicable for the given structural description in the given context. Various methods may be used to verify the match between the given structural description and the SFF candidates and to verify if the hypothesized functions are applicable for the given structural description in the given context. Later in this section we will describe the methods we have used to do these verifications.

Once a matching SFF and a corresponding applicable function has been determined for a structural description, that structural description may be replaced by the abstract device. For example, after verifying the match between the comparator SFF and the structural description in box C, and after verifying that the hypothesized function *compare-func* is in fact a function of the given structural description in box C, the part of the circuit in box C can be replaced by an instance, C', of the abstract device *comparator*, with the function *compare-func* associated with the instance C'.

Similar things are done with other parts of the given structural description. This process results in a new structural description shown in Figure 1(b) consisting of abstract devices such as C'.

The above process is repeated on the new structural description generating more abstractions resulting in hierarchical FR models. The structural description in Figure 1(b) is further abstracted to the structural description in Figure 1(c), which is matched on to a feedback loop SFF and the function for the structural description in Figure 1(c) is obtained by verifying and instantiating the function and FR template associated with the feedback loop SFF. This is how the SFFs are used to go from structure to function and to generate structural and behavioral abstractions.

Of course, for a given structural description there may be several ways of decomposing the given structural description and for each decomposed part of the given structural description there may be several matching SFFs and several hypothesized functions. But only some of them would pass the verification and even fewer would result in a consistent FR of the whole device. Several heuristics can be used to control the search. Later in this section we will describe these heuristics. Depending on the knowledge available this search may be top down or bottom up. For example, if no additional knowledge is available, then starting from the given structural description one has to generate more and more abstract descriptions to generate the FR in a purely bottom up fashion. On the other hand, if the function and the matching SFF for a structural description can be hypothesized, then one can use that knowledge to appropriately decompose the given structural description and specifically look for some sub-components with specific functions, thus following a top-down approach.

3.3.1 Two main strategies

We would like to point out two important characteristics of the process described in the previous paragraph. The first characteristic is that the primary method of determining the functions and the FRs of a given structural description is by using the functions and FRs associated with the matching SFFs. This method is comprised of two main steps. First, for a given structural description a set of functions and FRs associated with the matching SFFs is hypothesized to be the possible functions for the given structural description. Second, additional methods are used to verify if the hypothesized functions are indeed the functions of the given structural description. This is shown diagrammatically in Figure 6(a).

The second characteristic of our method is the use of a decomposition technique to analyze the given structural description. Obviously the system cannot be provided with an SFF for every structural description that the system would encounter. So there are going to be structural descriptions for which the system does not have any matching SFFs. To be able to handle a large number of devices the system has to have some sort of a compositional method that enables the system to identify a given structural description as an instance of a combination of SFFs. The way our system achieves this is by decomposing the given structural description into parts and then analyzing each part separately using the method described in the previous paragraph. The analyzed parts are then combined to form a new structural description which is then analyzed again. This strategy can also be seen as a "structural description transformation" technique, because it takes in a structural description and generates a new structural description. It takes in a structural description, decomposes it, analyzes each part separately, and then composes the decomposed parts to get a new structural description. For example, applying this strategy once to the cruise-control circuit in Figure 1(a) yields a new structural description shown in Figure 1(b). This process is diagrammatically shown in Figure 7(a). If the new structural description cannot be analyzed using the method described in the previous paragraph, then this step can be applied again to generate yet another structural description. Our claim is that in several domains, given the right set of SFFs, a large class of devices can be analyzed using these two strategies.

The two strategies described above constitute the core of the algorithm used by our system to perform the task. In subsequent paragraphs we will discuss some of the other important features of our system.

3.3.2 Other important characteristics

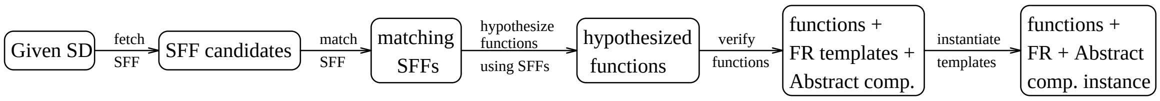
Organizing the SFFs

As pointed out earlier, various methods may be used to search for matching SFFs and to verify the hypothesized functions. Several strategies are used to control the search for the matching SFF. The SFFs provided to the system correspond to the set of abstractions an expert reasoning in that domain would have. The set of SFFs provided to the system would usually be quite large. To make the search efficient, the SFFs are organized into groups. We have used three different criteria to organize the SFFs:

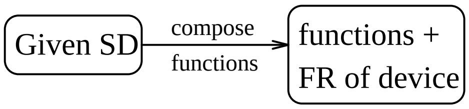
1. By the number of subcomponents in a component.

2. By the function of the component. Some functions are special in the sense that there are not too many different ways of achieving them. For each such function, components that achieve these functions are grouped together.

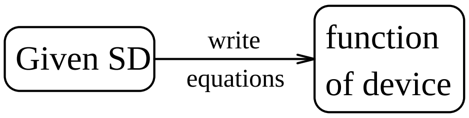
3. By special components. Some special components do not occur in too many devices in the domain of interest. For each such special component, devices



(a) The fetch-match-hypothesize-verify method

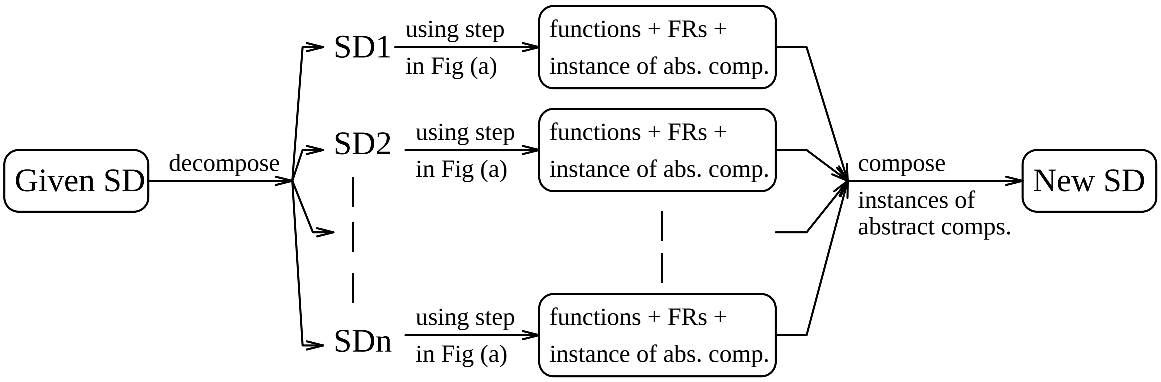


(b) The compose-func method

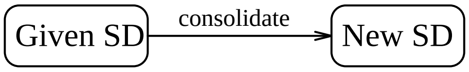


(c) The write-equations method

Figure 6: The three structure-to-function methods



(a) The decomposition method



(b) The consolidation method

Figure 7: The two structure-transformation methods

that refer to these components are grouped together.

After the possible SFF candidates for a given structural description have been fetched, additional criteria are used to evaluate and order the SFF candidates. Finally, the match between the SFF candidate and the given structural description is verified by matching the components and the connections of the given structural description with the components and connections of the abstract structural description associated with the SFF.

Verifying the function

The following criteria can be used to verify the hypothesized functions:

1. Does the given structural description satisfy certain structural constraints? These structural constraints would usually check if the various sub-components' parameters satisfy certain relations. This is the method that is used for function verification in all the examples described in this work.

2. Is the function of the structural description indeed the hypothesized function for all the inputs that are possible for the structural description in the given device? This question may sometimes be answered by simulating the relevant part of the device. This requires the capability to set up a simulation problem for the part of the device and then verifying if the output of the simulation corresponds to the functional specification for the structural description. [All90] uses this verification method in the domain of software program verification.

Additional structure-to-function methods

There might be structural descriptions that do not match any SFF and cannot be decomposed either. Two additional methods are used to analyze such structural descriptions. One is similar to the method proposed in [dKB84]. Using this method, the following equations are generated and used as the model of the given structural description:

1. equations in the models of the components of the given structural description.

2. equations written for the connections between the components. For example, in the domain of electrical circuits these will be the KVL and KCL equations.

The other method to get the functions and FRs for such structural descriptions is to directly compose the functions of the individual components following the connections between the components. For example, to get the functions and FRs of the arithmetic unit in the cruise control example we compose the functions of the adder, integrator, and the amplifier. An input goes to both the amplifier and the integrator. The outputs of the integrator and the amplifier then go into the adder. Following these input-output connections between components, we can compose the FRs of the individual components to get the functions and the FRs of the device. The final states of the cpds for the integrator and the adder are connected to the initial state of the cpd for the adder. Later on we describe under what circumstances such composition of individual FRs using the connectivity between components is possible.

Consolidation: Another structure-transformation method

Another simplification technique thats often used while understanding devices, particularly in the domain of electrical circuits, is to consolidate serial and parallel resistors and sources [BC85]. This may also be seen as one more way of transforming a structural description to another structural description. So if a given structural description does not match any SFF, one may try to consolidate serial and parallel resistors and sources generating a new structural description, and then try to match the modified structural description to one of the SFFs. This is diagrammatically shown in Figure 7(b). Figure 6 and Figure 7 summarize the main steps used by our system to analyze the given structural description in the domain of electrical circuits.

3.4 Summary

In this section we used the characteristics of the expected output, that is the FRs of the given structural description, to identify two additional knowledge types that are required to generate such an output.

1. The knowledge required to perform the structure-to-function abstractions.

2. The knowledge required to select the state variables and relations between them that are relevant to the specific function.

We then described the form, the SFF, in which this knowledge is provided to the system. Finally, we gave a high level description of how this knowledge is used to perform the task. Figure 6 diagrammatically shows the three methods used to go from structure to function and Figure 7 shows the two methods that are used to transform a structural description into another Further, it was claimed that given an appropriate set of SFFs, in most domains a combination of the two strategies shown in Figures 6(a) and 7(a) would be sufficient to perform the task.

Next we will define the objects used by the algorithm followed by a description of the task and the algorithm used to perform the task.

CHAPTER IV

Definitions of objects

This chapter defines the various objects used by our algorithm.

4.1 Primitive components

A *component* is defined by a set of ports and parameters. A component connects and interacts with other components through its *ports*. Three kinds of ports have been used in the examples described in this work - *electrical, mechanical, and thermal*. A component interacts with other components by exchanging substances through the ports. For example, electrical ports interact by exchanging the substance *electric-charge*. Associated with substances are quantities and qualities. *Voltage* is an intensive quality of the substance electric-charge, and *current* is the quantity *rate of flow of electric charge*. *Frequency* is the quality associated with the rate of flow of charge. Thermal ports interact by exchanging the substance *heat*. *Temperature* is an intensive quality of the substance heat, and *rate of flow of heat* is the quantity rate of flow of the substance heat. *Parameters* of a component is a set of parameter-name and parameter-value pairs. For example, a component called *rechargeable-battery* is defined by two ports, corresponding to the positive and the negative terminals, and parameter names such as emf, internal resistance, and charge-capacity each of which has a value associated with it. Associated with a component are *state variables*. The state of a component is defined by the values of its state variables. A *partial state* of a component is a wff of predicates on the state variables of the component. Also associated with a component may be functions. A *function* of a component relates the dependent state variables to the independent state variables of the component. *Independent state variables* of a component are the state variables whose values are assigned or changed by an agent external to the component and *dependent state variables* are the state variables whose values depend on the values of independent state variables. Often it is the case that the independent and the dependent state variables refer to quantities at the ports of the component. In such a case, the ports referred to by the independent state variables are called the *input ports* and the corresponding state variables are the inputs to the component. The ports referred to by the dependent state variables are called the *output ports* and the corresponding state variables are the outputs of the component. Some of the ports may be both, input as well as output ports.

4.2 Functions

We have already defined functions in Chapter II. Of the four different types of functions defined in Chapter II, only the *toachieve* type of function has been used in this work. This is the most commonly encountered function type and a large number of devices can be represented using just this function type. The *toachieve* type of function of a component is represented by a frame with two slots, the *given* and the *toachieve* slots. Both these slots have predicates on the values of the state variables of the device. When the predicate in the *given* slot is true, the predicate in the *toachieve* slot becomes true as a result of the execution of the function. See Chapter II for a more complete description of *function.*

For each function the component also has a condition, called *provided*. A function of the component is applicable only if the corresponding *provided* condition evaluates to true. These conditions may check anything ranging from the values of certain component parameters to some global variables used to capture the agent's interests. In this work we have used the *provided* condition for a more specific purpose. The *provided* slot defines the condition on the values of component parameters under which the corresponding function will be applicable for the given device. For example, for a rechargeable battery in a closed circuit the charge level of the battery has to be above a certain threshold value, Q_T, for the *deliver-voltage* function of the battery to be applicable. So the *provided* condition for *deliver-voltage* function of the rechargeable battery would be "actual-charge $> Q_T$."

4.3 Composite components

Components may be connected at the ports to form new components. *Structural description* of a component specifies the set of sub-components and the relationships between the sub-components. The *components* slot in the structural description gives the list of sub-components constituting the given structural description. For example, the *components* slot of the *comparator-sd* shown in Figure 10 lists the sub-components $R1$, $R2$, $R3$, $R4$, and $O1$ constituting the comparator. It also specifies the class for each of the sub-components.

In this work the relationships between the components are defined only in terms

```
ports:  ((electrical  nil  p1  port-ptr1)  (electrical  nil  p2  port-ptr2))
params:   ((voltage EMF nil "source voltage"
            (resistance  IR nil "internal resistance")
            (charge  CT  nil  "minimum-charge threshold")
            (charge  CC  nil  "charge capacity")
            (charge  ACTUAL-CHARGE  nil  "actual charge")
            (resistance  R  nil  "charging resistance"))
structural-description:  nil
beh-cpd-name:      (((and (equal (var-value  matchedas) 'voltage-source)
                         (> (param actual-charge)  (param  ct)))
                       deliver-voltage-beh  deliver-voltage-cpd  nil)
                    ((or  (subtypep  (var-value  matchedas)  'eload)
                       (subtypep  'eload  (var-value  matchedas)))
                     consume-power-beh  consume-power-cpd  nil)
                    ((or  (subtypep  (var-value  matchedas)  'eload)
                       (subtypep  'eload  (var-value  matchedas)))
                     accumulate-charge-beh  accumulate-charge-cpd  nil)))
hypothesized-beh:  nil
beh-context:  nil
```

Figure 8: Representation of the component *rechargeable-battery*

of the relations between their ports. In the domain of electrical circuits, one of the relations between component ports is *electrically-connected* that corresponds to an electrical connection between the ports. A relation between the ports also imposes certain constraints on the values of the quantities at the ports participating in that relation. For example, for *electrically-connected* ports, the voltage at all the ports participating in this relation is the same and the sum of the currents at all the ports is zero. Just like *electrically-connected*, we use the relation *mechanically-connected* for mechanical ports and the relation *thermally-connected* for thermal ports. For *mechanically-connected* ports, the sum of forces at all the ports is zero, and the displacement at all the ports is the same. For *thermally-connected* ports, the temperature at all the ports is the same and the sum of rate of heat flow at all the ports is zero. To use the vocabulary used by engineers the term *junction* is often used to refer to the connection between two or more ports. The *connections* slot in the structural description defines the connections between the sub-components specified in the *components* slot. *Connections* slot specifies connection by defining the junctions of the structural description. For each junction, it defines the name of the junction and the component ports that connect at that junction. For example, the *junction jc3* in the *connections* slot in the *comparator-sd* shown in Figure 10, connects the ports $p2$ of $R1$, $p1$ of $O1$, and $p1$ of $R3$.

A *primitive component* is one that does not have a structural description associated with it, that is, a primitive component is to be treated as consisting of just one sub-component. A *composite component* is one that has a structural description associated with it. The set of state variables of a composite component may have some or all of the state variables of its sub-components and may have some new state variables as well.

4.4 Functional Representation

We have already described FR in Chapter II. FR of a function of a device is comprised of the structural descriptions, functions, and cpds. Cpds are also defined in Chapter II. The structural and functional components have already been defined. The cpd is represented by a frame with five slots. The *states* and *links* slot list the states and links of the cpd. *Connectivity* describes the connectivity of the cpd graph, that is, connections between states and links. The *start-states* and *end-states* slots list the initial and final states of the cpd that are used for simulating the cpd. Start-states and end-states are automatically computed from the connectivity. States that do not have any links coming into them are identified as the start states and states that do not have any links going out of them are identified as end-states. As mentioned in

```
ports:  ((electrical input p1 port-ptr1) (electrical input p2 port-ptr2)
         (electrical output p3 port-ptr3))
params:  ((voltage IT nil "input voltage threshold"))
structural-description:   comparator-sd
beh-cpd-name:    (((and (< (voltage p1) (param IT))
                          (< (voltage p2) (param IT)))
                    compare-func compare-cpd default)
                  (( or (>= (voltage p1) (param IT))
                        (>= (voltage p2) (param IT)))
                    clipped-comparator-func clipped-comparator-cpd nil))
```

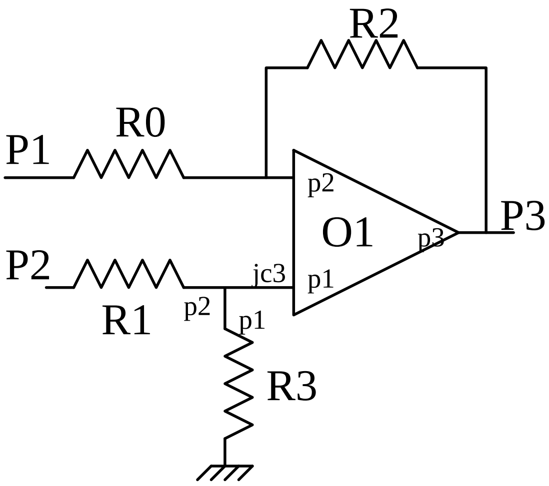
Figure 9: Comparator SFF

Chapter II, of the three types of links, only the *using-func-link* and *as-per-link* types have been used in this work. Further the method proposed here assumes one-to-one correspondence between the structural and the functional decomposition of the device. That is, each cpd in the resulting FR of a function of the device corresponds to the function of a component. Devices that satisfy this assumption can be represented using the two types of links described here. Our experience has been that a large number of devices belong to this class.

Components are organized in component classes. Each component instance is an instance of some component class. For example, all instances of rechargeable batteries in a given device would be instances of the component class *rechargeable-battery.*

4.5 SFF Templates

As mentioned earlier the agent has access to a set of structure-function-FR (SFF) templates. An SFF associates an abstract structural description, function templates, and the corresponding FR templates. For example, Figure 9 shows the comparator SFF. The structural description, *comparator-sd*, is the abstract structural description of a comparator. The beh-cpd-name slot points to the functions and the corresponding FR templates associated with the abstract structural description of the comparator. Each SFF corresponds to an abstract component.



components: ((resistor R1) (resistor R2) (resistor R3)
(resistor R4) (op-amp O1) (ground G1))

connections: ((jc1 (p1 (R1 p1)))
(jc2 (p2 (R2 p1)))
(jc3 ((R1 p2) (O1 p1) (R3 p1)))
(jc4 ((R2 p2) (O1 p2) (R4 p1)))
(jc5 ((O1 p3) (R3 p2) p3))
(jc6 ((R4 p2) (G1 p1))))

constraints: (and (equal (param R R1) (param R R2))
(equal (param R R3) (param R R4)))

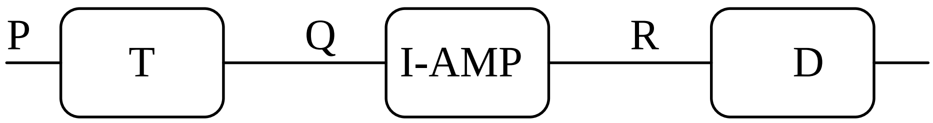
Figure 10: Structural description of *comparator*

4.6 Abstract structural description

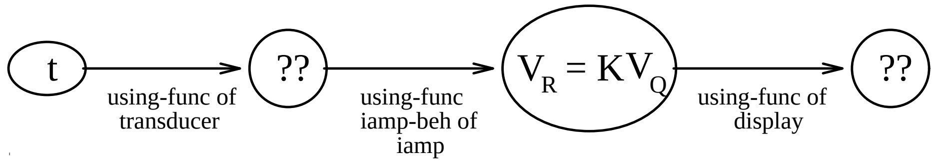
An *abstract structural description* is a template for a class of structural descriptions. It is just like the structural description, except that instead of referring to specific sub-components, the abstract structural description defines two types of constraints for each of its sub-components, the class of the sub-component and constraints on the values of the parameters of the sub-component. For example, the Figure 10 shows the *comparator-sd*.

4.7 FR templates

The function templates and the FR templates do not refer to real components but only refer to the quantities and components of the abstract structural description. Further, in *using-func-link* instead of mentioning the component name and the function name, one may mention only the type of function that is required to enable the state transition. For example, consider the *measuring-instrument* shown in Figure 11(a). It consists of three components, a transducer, an instrumentation amplifier, and a display. The transducer takes in some quantity and generates a proportional electrical signal, which is then amplified by the instrumentation amplifier. The output of the instrumentation amplifier is then displayed by the display. In the *measure-cpd* template for the *measuring-instrument*, we only need to state that it has some sort of a display that takes an electrical signal and generates an output that can be perceived by human beings. The output could be visual, or sound, or light etc. The way we represent this concept is by defining a component class *display* and making the components such as *display-meter* or *light-indicator* subtypes of *display*. And on the using-func-link we just mention the name of the component D which is of class *display* and do not mention the function. All components of subtype *display* would have some sort of a display function associated with them. A *display-meter*, for example, has the function *meter-deflection* that takes an electrical input and generates a proportional deflection in the needle. A cpd template, like the one shown in Figure 11(b), can be used to generate the cpd for any given instance of a measuring instrument. For example, for an instance of a measuring instrument having a *display-meter* for the display, the last *using-func-link* in the cpd shown in Figure 11(b) would have the component D instantiated to the component of type *display-meter* and the function slot would have the value *meter-deflection*. Once a specific function has been determined the *toachieve* predicate of the function is used to determine the predicate for the following state (the question mark in the state in the cpd-template indicates that the state predicate is unknown.) This is how an FR template represents the generic



(a) Measuring Instrument sd



(b) Measuring instrument cpd template

Figure 11: Measuring Instrument

FR of a generic device whose abstract structural description is pointed to by the SFF.

4.8 Component class hierarchy

As is evident from the description in the previous paragraph, each SFF can also be seen as representing a class of components. All the component classes, primitive as well as SFFs, are organized in a component class hierarchy. A component class inherits ports, parameters, functions, and FRs from its parent classes. A component can have more than one parent class. For example, the component *rechargeable-battery* is a subtype of the component classes *voltage-source* and *load*. The component classes provided to the system usually correspond to either off-the-shelf devices used in the domain or the abstract components used by the experts in the domain. In our system, there is no concept of function without the notion of an associated component with ports and parameters. Associated with each function there is a component. To define purely functional components one can create dummy components. For example, to represent the function *amplify*, one has to create a dummy component, *pure-amplifier* that has the appropriate ports and parameters and a single function *amplify*. A real amplifier component class would be a sub-class of *pure-amplifier* and some other component classes too.

4.9 Organizing SFFs

As mentioned in chapter III, the components in the library are organized by the following three criteria.

1. Organize the SFFs by functions. Some functions get used very often in some domains and there may be several ways of achieving each of these functions. That is, several SFFs may be associated with the same function. For example, in the domain of electronic circuits, amplification function may be obtained in several different ways. So SFFs corresponding to the different devices achieving the same function can be grouped together, and can be indexed by the common function they can all achieve. If the hypothesized function of a given structural description is given or can be determined, but the FR is yet to be determined, then one can use that hypothesized function to fetch a set of candidate SFFs that might match the given structural description. So one can identify a set of such frequently encountered functions and group the SFFs by these functions.

2. Another way we organize the SFFs is by the number of subcomponents their abstract structural descriptions contain. If the given structural description consists of, say, five subcomponents, then while looking for a matching SFF there is no need to consider SFFs that contain less than or more than five sub-components, unless other SFFs are suggested by other criteria.

3. In some domains there are some components that do not occur in very many different configurations. For example, in the field of analog computational circuits, a component like an op-amp is almost always used in one of a few known ways. Similarly, in the domain of say audio equipment circuits, again the op-amp is usually used in one of a few known ways, that are different from the ways in the domain of analog computational circuits. Depending on the kind of circuits an agent is expected to encounter, there may be 10 to 20 different SFFs whose abstract structural descriptions refer to an op-amp. Such SFFs can be grouped together and indexed by the special component they share. Given a structural description with a component *op-amp*, one need to check only the SFFs indexed by the component *op-amp* to get a match.

4.10 Additional component features

Additional information is associated with a component that may be used during the processing. Associated with each component is the *behavioral-context* of a component in a device. We use the set of cpds of the containing device as the behavioral context of the component. For example, in the *measuring-instrument* device, the behavioral context for the *instrumentation-amp* component is the cpd of the containing device, that is, *measure-cpd* of the *measuring-instrument*. If for a given structural description, the function *measure-func* is hypothesized, then while analyzing any part of the given structural description the functions referred to in the *measure-cpd* can be used to hypothesize functions for the part of the structural description being analyzed. Functions referred to in the cpd of the containing device may suggest possible functions for the components. Also associated with a component is the *hypothesized-function* of the component. This is a list of hypothesized functions of the component that are yet to be verified. This slot may be filled using knowledge of various kinds. For example, for the *measuring-instrument* device, the system may be provided with a rule that if the given structural description has a sub-component of type *transducer* and a sub-component of type *display*, then hypothesize the function of the given structural description to be *measure-func*. The information in the *hypothesized-function* slot is used while trying to determine the function of the structural description. Both the hypothesized function and the behavioral context of a component are generated and used during the process of understanding the device to prune the search space.

4.11 The Task Definition

Input:

1. Structural description of a device.
2. The ports of the given structural description, and their types. Optionally one may specify the quantities of interest at the input and output ports. If known, the hypothesized relation between the input and the output quantities may also be specified.
3. A library of component models, both primitive and abstract (SFFs), organized in component class hierarchies.

The first two inputs are problem-instance-specific inputs, where as third input is part of the setup.

Output:

1. Possible functions of the device.
2. For each hypothesized function in (1), an FR describing how the function arises from the structural description given as input.

CHAPTER V

The Algorithm

The algorithm is described in terms of goals and methods [Cha90]. Each task or sub-task corresponds to a goal or a sub-goal. Corresponding to each goal there is at least one method to achieve the goal. Also associated with each goal is the knowledge to select a method appropriate for the given situation. A method is just a sequence of steps some of which might set up new sub-goals. Some of the steps given here are specific to the domain of electrical circuits and may not apply to other domains. We will identify such steps whenever we come across them. The following is the method for the top level goal.

1. Set up a subgoal simplify-sd-g. The goal is to remove all the superfluous components. Superfluous components are the ones that do not contribute to the output of the device in any way. In some cases structural features can be used to detect such components whereas in other cases it may be necessary to simulate the behavior to see if a component contributes to the function of the device. For example, in an electrical circuit, if an active component is not connected to either the power or the ground, then it is definitely not serving any electrical function in the device. This goal will have domain-specific methods.
2. Set up a subgoal analyze-device-g. This step determines the functions and the corresponding FRs for the given structural description.
3. After the functions and the corresponding FRs for the device have been determined, the subgoal clean-up-frs-g is set up the for each function. This step removes the superfluous parts of the FR, that is, removes cpds that do not correspond to the function.

5.1 Simplify the structural description

Goal: simplify-sd-g
Method: simplify-sd

This is a domain specific method for the goal simplify-sd-g that is applicable only in the domain of electrical circuits. The following method removes superfluous components from a given structural description. This method uses only structural features to determine superfluous components and works only for electrical circuits. A component in the given structural description that does not satisfy all of the criteria is considered a superfluous component:

- If the device's input-output ports are specified, then for each component there should be a path from a device input port to each input port of the component and a path from each output port of the component to an output port of the device.
- All active components should have the power and ground connected.
- All sources should be in a closed loop with at least one non-source passive component. Or one of the ports of the source should be connected to either ground or power. Or both the ports may be connected to different ports of other non-superfluous components.
- All non-source passive components should be in a closed loop with at least one source. Or one of the ports of the component should be connected to either ground or power. Or both the ports may be connected to different ports of other non-superfluous components.

Since this method uses only structural features to detect superfluous components, it is possible that a superfluous component may not be detected by this method. The only consequence of this would be that a superfluous component may get analyzed, even though it does not get referenced by the FR of the device function.

5.2 Analyze the structural description

Goal: Analyze-device-g
Given a structural description and possibly the input-output specification, the goal is to determine the functions and the corresponding FRs of the device. There are six different methods associated with this goal, which are tried in the order in which they are given. If a method works, it returns the functions and FRs of the given structural description. If none of the methods works, then return with nil. Following are the six methods for this goal:

5.2.1 Special-method

Set up a goal special-method-g. The agent has access to a collection of special methods to analyze certain kind of devices. For example, the agent has a special method to analyze feedback loops. So if the given structural description is some sort of a feedback loop, one can use the special method for feedback loops to analyze it. Special methods, in some sense, capture the abstract structural description of the device procedurally. Hypothesized functions or the structural features of the given structural description may suggest some special methods. The special-method-g goal is to check if there are any special methods applicable for the given structural description. If yes, then invoke the appropriate special method and return its result, else return from (5.2.1) with nil.

Here is an example of a special method available for electro-mechanical feedback loops, shown in Figure 12. The abstract structural description of a feedback loop has a component, *arithmetic-unit*, whose abstract structural description cannot be specified. Arithmetic-unit is the component that computes a function of the error signal to generate the control signal. The structural description of the arithmetic-unit will depend on the function it computes. As a result, a feedback loop cannot be matched on to an SFF by just matching the structural descriptions. Also, for a feedback loop the partitioning, using the method in (5.2.4) is computationally very expensive. This special method procedurally captures the structural description of the feedback loop and also directs the partitioning algorithm. The method first isolates the feedback path from the forward path. This requires breaking two edges. Next the structural description for the comparator needs to be separated. This is done by partitioning the structural description for the forward path and retaining the partition with the device input-port. The partition with the device input port is recursively partitioned till it cannot be partitioned any further. This will yield the structural description for the comparator. Similarly separate the structural description for the mechanical part, in this case the engine part. Use the same strategy to isolate the driver part. The remaining structural description is to be analyzed as an arithmetic-unit. Here are the steps for this method. These steps refer to a specific type of feedback loop shown in Figure 12. This type of feedback loop is used for controlling electromechanical devices.

1. Break the edge going from the output of the given structural description to the input of a component.
2. Partition the resulting structural description by breaking only one edge such that both the device input and the output ports are in one partition. As will be described in (5.2.4) a structural description is partitioned by breaking only

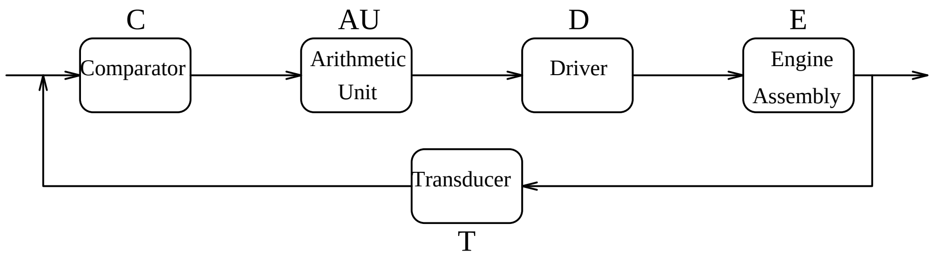


Figure 12: The feedback loop

a junction-junction edge. This will separate the transducer part. Set the hypothesized function of the structural description for the transducer part to the function *vs-transduce*. Set up the subgoal analyze-device-g for the transducer part to get the functions and FRs for the transducer part.

3. Partition the remaining structural description such that one of the partitions is the smallest possible partition containing the input port of the device. This corresponds to the comparator part. Set the hypothesized function of the structural description for the comparator part to the function *compare*. Set up the subgoal analyze-device-g for the comparator part to get the functions and FRs for the comparator part.

4. Partition the remaining structural description to get a partition that contains all the mechanical parts and the output port of the device. This partition corresponds to the mechanical device being controlled. Set up a subgoal analyze-device-g for this partition to get the functions and FRs for the mechanical part.

5. Partition the remaining structural description to get the smallest partition that contains the output port of the structural description. This part corresponds to the driver part. Set the hypothesized function for this partition to *drive*, and set up a subgoal analyze-device-g for this partition to get the functions and FRs for the driver part.

6. The remaining structural description is to be analyzed as an arithmetic unit. Set up a subgoal analyze-device-g for the remaining structural description to get the functions and FRs for the arithmetic unit. The resulting function is used as the function being computed by the arithmetic unit.

7. Compose the analyzed comparator, arithmetic unit, driver, the mechanical part to get a new structural description. Set up a subgoal analyze-device-g for the new structural description. The resulting functions and FRs would be the func-

tions and FRs for the forward path of the feedback loop in the feedback loop template.

8. The function and FR for the transducer part becomes the function and FR of the feedback-path of the feedback loop in the feedback loop template.

5.2.2 Fetch-match-sff-method

The method gets the functions and the corresponding FRs by mapping the given structural description on to an SFF. The applicable functions and FRs of the selected SFF are then instantiated for the given structural description.

1. Set up a subgoal fetch-sffs-g. The goal is to get a list of candidate SFFs from the library whose abstract structural descriptions might match the given structural description. A list of candidates sorted in the order of the likelihood of their match is returned.

2. Set up a subgoal match-comps-g. The goal is to try matching the components of the given structural description with the components of the abstract structural description of the SFF candidates in the order of their ratings. Based on the extent to which the components match the scores of the SFF candidates are modified. It returns a sorted list of SFF candidates along with a list of component matches for each SFF candidate that resulted in the component match. If no SFF matches the given structural description, then return from (5.2.2) with nil.

3. Starting with the highest rated SFF, say X, from the list of SFF candidates (sorted in the order of their ratings) obtained in (2) execute the following loop:

   (a) Set up a subgoal match-conns-g. The goal is to check if the connections of an SFF candidate, X, match with that of the given structural description. If X matches the given structural description, then return X and the set of component matchings that resulted in the match. If X does not match the given structural description, then go to the step (3f).

   (b) Set up a subgoal instantiate-comp-g. The goal is to make an instance of the abstract component associated with the matching SFF, X.

   (c) Set up a subgoal verify-select-func-g. The goal is to determine which of the functions hypothesized by X for the given structural description are in fact the functions of the given structural description. If none of the functions hypothesized by X are applicable, then go to step (3f).

(d) Set up a subgoal set-context-g. The goal is to set the context for the subcomponents of the given structural description. Once the functions for the given structural description have been determined, one can use this information to set the context information for the subcomponents of the given structural description.

(e) Set up a subgoal extend-refine-g. The goal is to refine the the FRs selected in step (3c) for the given structural description. The FRs obtained by instantiating the FR templates of the matching SFF may not be complete, in the sense that some information on the links or in the states may be missing. The goal is to get the missing information. Thus the missing information guides further reasoning. Return from (3) with the selected functions and FRs.

(f) If no more SFF candidates, then return from (3) else continue loop at (3a).

Fetch SFFs

Goal: fetch-sffs-g.
Given a structural description, the goal is to get a list of candidate SFFs from the library whose abstract structural descriptions might match the given structural description. Candidates are rated based on the likelihood of their match.
Method: num-func-special.
Three separate sets of SFF candidates are fetched from the library based on the number of subcomponents, hypothesized functions, and the special subcomponents. Larger the number of sets an SFF appears in the more likely it is to be the right candidate, because each set corresponds to an evidence in favor of selecting an SFF. Candidates are rated based on the likelihood of their matching. A score of 1, 2, or 3 is associated with each candidate in the final set depending on the number of sets it appears in. A list of candidates sorted in the order of their scores is returned.

Match SFF and component

Goal: match-comps-g.
Given an SFF, check if the subcomponents of the abstract structural description of the SFF match the subcomponents of the given structural description.
Method: type-subtype-match.
A component of the given structural description, say C_{real}, matches an abstract component, say C_{abs}, of an SFF, if C_{abs} is a subtype of C_{real}. The class of C_{abs} is expected to be the most specific component class required to get the desired function of the abstract component in that SFF. A score is added to each candidate based on the

result of the component match. If all the components of the given structural description match the abstract components of the SFF, then a score of two is added to the candidate. If there are any components in the SFF or the given structural description that are not fully specified and all the fully specified components match, then a score of one is added to the candidate. In case of a component mismatch, that is, if no component match is possible between the two structural descriptions, it returns nil. Otherwise it returns the SFF along with a score and a list of possible component matches.

Match connections

Goal: match-conns-g.
Given an SFF, check if the connections of the abstract structural description of the SFF match with that of the given structural description.
Method: match-conns
Connections of the two structural descriptions are flattened, that is, all the redundant junctions are removed. For each possible component match between the two structural descriptions, the interconnections are checked for a match. If found a match, the component match that resulted in the complete structural match is returned, else nil is returned.

Instantiate component

Goal: instantiate-comp-g.
Once a matching SFF has been found for the given structural description, the goal is to instantiate the abstract component, function and FR templates referred to by the matching SFF.
Method: instantiate-comp
To instantiate the abstract structural description to the given structural description, one has to find the correspondence between the components, junctions, and the device ports of the two structural descriptions. For example, assume that the abstract structural description, function, and FR templates of the SFF refer to a component R of class *resistor*. Further assume, the component match that resulted in matching the SFF to the given structural description, matched the component R of the abstract structural description to the component R_1 of the given structural description. Then all references to say, port p_1 of R, in the abstract structural description in the SFF would correspond to the real port instance p_1 of R_1. Similarly, all references to the voltage at port p_1 of R in the function and the FR template associated with the SFF, would now refer to the voltage at the real port p_1 of R_1. Just like the

components, the junctions and the device ports of the abstract structural description in the SFF are instantiated to the corresponding junctions and device ports of the given structural description. These component, junction, and port instantiations are used to instantiate the function and the FR templates associated with the SFF.

Verify and select functions

Goal: verify-select-func-g.
Functions associated with the matching SFF become the hypothesized functions of the given structural description. But in a given context usually only some of these hypothesized functions are applicable and even fewer may be relevant to the goals of the agent. The goal is to select the functions that are applicable to the given structural description and are relevant in the context of the functions of the containing device.
Method: provided-hyp-context.
First we verify each of the hypothesized functions. A hypothesized function is assumed to be applicable to the given structural description if the *provided* condition evaluates to true. If the *provided* condition cannot be evaluated due to the lack of information, then the corresponding function is assumed to be applicable if the *provided* condition can be assumed to be true without causing a conflict with any of the *provided* conditions already assumed to be true. The *provided* condition is usually a constraint on the values of the parameters of the sub-components of the given structural description.

After the applicable functions have been identified for the given structural description, we use the following two criteria to determine which of the functions are relevant in the context of the specific functions of the containing device.

1. If the hypothesized functions for the given structural description are known, then the verified functions of the given structural description that match the hypothesized functions are selected.
2. As described earlier, also associated with the components is a behavior-context that points to the cpds of the containing device. If a function of the component is not being referred to by any of the cpds of the containing device, then it is very likely that this function of the component is not relevant in this context. If the behavior context for the given structural description is known, then the verified functions of the given structural description that are referred to by the behavior context of the given structural description are selected.

If neither of the above criteria apply in a given situation, then the default functions associated with the component are selected. That is, if no hypothesized-function or behavior-context is given, then neither of the above criteria will apply. To take care of such cases some of the functions of components are marked as *default*. Consider a light bulb for example. One might be interested in the light generating function of the bulb, heat generating function of the bulb, or just the resistive characteristics of the bulb. Depending on the above information one may select the functions applicable in the given situation. But if nothing specific has been asked for, that is, if none of the above criteria apply, then it is reasonable to take the most commonly used function, the light generating function, to be the function of the bulb. For each component, some of the functions have been identified and marked as default functions. When none of the above criteria apply, the default functions of the component are selected by this method.

Set function context

Goal: set-context-g.
The abstract component to which the given structural description has been matched and the selected functions provide contextual information for the subcomponents of the given structural description. Such contextual information can be used to select specific functions for the components of the given structural description. Given the matching SFF, the goal is to fill in the contextual information for the subcomponents of the given structural description.
Method: matchtype-behcontext.
Two kinds of contextual information are stored for the components of the given structural description after it has been matched with an SFF.

1. For each subcomponent of the given structural description the type of the subcomponent that resulted in the match is noted. For example, if a rechargeable battery in the given structural description was matched on to a resistor in the abstract structural description, then one knows that the rechargeable battery is being treated as of type *resistor*, which means it is in recharge mode. Such information is useful for further analysis and is noted for each subcomponent.
2. The set of cpds corresponding to the functions selected in (3c) provide the behavioral context for the subcomponents of the given structural description. This set of cpds is copied into the beh-context slot of each subcomponent of the given structural description.

Fill FR template

Goal: extend-refine-g.
Since SFFs point to FR templates and not to FRs, some of the information in the

FR templates may be missing. The goal is to refine (that is obtain the missing information) the FRs selected in step (3c) for the given structural description.
Method: extend-refine-links.
One way in which the information may be missing in the FR template is that the component name (slot comp-name) in a using-func type of link may be given but the behavior name (slot beh-name) may not be given. In such cases the agent analyzes the subcomponent mentioned on the link to determine its functions, which are then used to fill up the beh-name slot of the using-func-link type of link. This is one of the places in our approach where the missing information is used to guide the problem solving.

5.2.3 Consolidate-analyze-method

This method transforms the given structural description into a new structural description by consolidating serial or parallel sources or resistors, and then tries to analyze the new structural description. Here are the steps for this method:

1. Set up a subgoal consolidate-g. The goal is to consolidate a pair of parallel or serial sources or resistors in the given structural description thus generating a new structural description. If no consolidation is possible, then return from (5.2.3) with nil.
2. Set up a subgoal analyze-device-g for the new structural description generated in step (1). If the goal analyze-device-g is successful, then return from (5.2.3) with the functions and FRs for the new structural description, else go to step (1).

Parallel/Serial consolidation

Goal: consolidate-g.
The goal is to consolidate a pair of parallel or serial sources or resistors in the given structural description thus generating a new structural description.
Method: par-ser-comb.
It first tries to consolidate parallel sources or resistors and then tries to consolidate serial sources or resistors. Here are the steps.

1. First flatten the given structural description, that is, remove all the redundant junctions, and then identify two components in parallel such that both are either subtypes of source or subtypes of resistor. If no such pair is found, then go to step (3).

2. The identified pair of parallel sources or resistors in the given structural description is replaced by a single composite component *parallel-sources* or *parallel-resistors.* Go to step (5).
3. Identify a pair of serial sources or resistors. If could not find any such pair then return from (5.2.3).
4. The identified pair of serial sources or resistors in the given structural description is replaced by a single composite component *serial-sources* or *serial-resistors.*
5. Set up a goal verify-select-func-g for the new component parallel/serial-source/resistor.
6. Set up a goal set-context-g for the two subcomponents consolidated, that is, the two sources or the two resistors.
7. Set up a subgoal extend-refine-g for the two subcomponents consolidated.
8. Return the new structural description obtained by replacing the consolidated pair by the new component in the given structural description.

5.2.4 Partition-method

Like the consolidation method, this method also transforms the given structural description into a new structural description, and then analyze the new structural description. The given structural description is partitioned into parts, each one of which is analyzed separately. Analyzed parts are composed to get a new structural description which is then analyzed to get the functions and the FRs of the given structural description. Here are the steps for this method:

1. Set up a subgoal decompose-g. The goal is to decompose the given structural description in to parts. Increasingly complex partition algorithms are tried starting from the simplest one. First it tries decomposing by breaking only the junction-junction (j-j) edges, that is, edges that connect two junctions. this is achieved by setting up the subgoal j-j-edges-g. If that does not work, then tries to partition the given structural description in to two parts by breaking a single non-j-j edge. This is achieved by setting up the goal single-edge-g. Finally, it tries to partition the given structural description by breaking two non-j-j edges. This is achieved by setting up the goal double-edge-g. Partitioning is successful if at least one of the partitions is not already analyzed. This criteria for successful partitioning is to avoid infinite looping on partitioning and composing a given structural description. If no partitions are possible, then return from (5.2.4).

2. Set up a subgoal analyze-device-g for each partition obtained in step (1). If could not analyze any of the partitions, then return nil.
3. Set up a subgoal compose-subsets-g. The input is the set of analyzed partitions, say $S1$, and the set of edges that were broken to get the partitions. The goal is to compose the largest possible subset, say $S2$, of the analyzed partitions at the previously broken edges to get a new structural description that would match an SFF in the library. Return the new structural description composing the components in $S2$. If no such subset exists, then return nil.
4. Set up the subgoal analyze-device-g for the new structural description obtained in step (3). This new structural description will get analyzed by the fetch-match-sff-method. Return the new analyzed part, say $C1$.
5. Set up the subgoal compose-partitions-g. Compose the analyzed part, $C1$, obtained in step (4) with the analyzed partitions not used in step (3) to obtain $S2$. That is compose, $C1$ and components in $(S1 - S2)$ to get a new structural description.
6. Set up a goal analyze-device-g for the new structural description obtained in step (5). If analyze-device-g is successful, then return from (5.2.4 with functions and FRs for the new structural description. If cannot analyze the new structural description, then return nil.

To execute the partition-method, the above steps are executed in the given order. If any of the steps fails, the previous step is retried to get a new result. And if the previous step cannot get a new result, it will go back one step more and retry that step and so on. For example, if the last step fails, that is, it fails to analyze the new structural description, then it will go back to step (4) and try again to analyze the new structural description to get a new result. If this step fails to get a new result, then it will retry (3). It will try to compose a different subset of components. If that also does not get a new result, then it will retry (2) to get a different result for at least one of the partitions. If this also does not help, then it will retry (1) and try to get a different partition of the given structural description.

Partition: j-j edge

Goal: j-j-edge-g.
Partition the given structural description by breaking all j-j edges.
Method: j-j-edge.
When a human reasoner decomposes a given structural description in to parts by looking at its schematic, the decomposition is strongly guided by the visual layout of

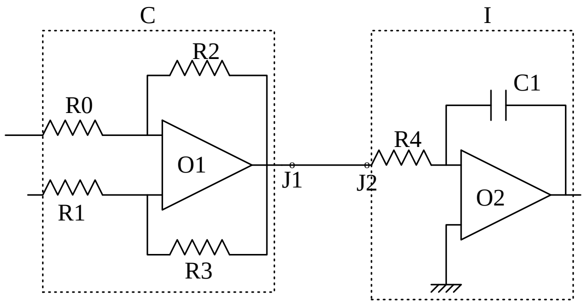
the schematic. The structural description is decomposed to separate the parts that visually look relatively independent. We have tried to capture the visual layout aspect of the schematic in our structural description by inserting some redundant junctions. Parts of the structural description that look fairly independent in the visual layout are connected by junction-to-junction (j-j) edges with other parts. For example, for the cruise control circuit the edge that connects the comparator to the integrator is a j-j edge as shown in Figure 13(a). One could have defined that connection with just one junction as shown in Figure 13(b). But an additional junction is added to get a j-j edge between the comparator and the integrator part. The partitioning algorithm partitions the given structural description by breaking j-j edges only. This limits the scope of the partitioning algorithm, in the sense that this method will not be able to try all the possible partitions. But it is a useful heuristic to control the computational complexity. This method breaks all the j-j edges in the given structural description and then separates the partitions. If it is able to partition the structural description, it returns the partitions and the list of edges that were broken to obtain the partitions. If it cannot partition, it will return nil.

Partition: single edge

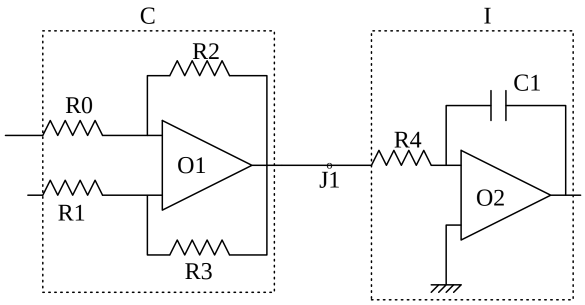
Goal: single-edge-g.
Partition the given structural description into two parts by breaking exactly one edge.
Method: single-edge.
This partitioning algorithm tries to partition the given structural description by breaking only one edge. Again this heuristic is used to control the computational complexity of partitioning. If it is possible to partition the given structural description in to two by breaking an edge, then it will returns the two partitions and the edge that was broken to get the partitions. Otherwise it returns nil.

Partition: double edge

Goal: double-edge-g.
Partition the given structural description in to parts by breaking two j-j edges.
Method: double-edge.
We partition the given structural description by breaking either one or two edges only. Single-edge partitioning works for cases where the signal flow is through a linear structure (linear combination of components.) Where as, double-edge partitioning takes care of cases where there are loops in the structural description. Obviously, the partitioning based on one or two edges only cannot partition all the structural descriptions. For example, a linear structure with three or more signals flowing between



(a) j-j edge connection



(b) Single junction connection

Figure 13: Two kinds of edges

components cannot be decomposed by breaking one or two edges only. These are some of the heuristic simplifications used to avoid a the general purpose graph partitioning algorithms which have exponential complexity. More heuristics, domain independent or domain specific, may be added to improve the efficiency or the range of applicability of the partition method.

Compose partitions

Goal: compose-partitions-g.
The goal is to compose a given set of analyzed partitions at the given edges to get a new structural description.
Method: structural-composition.
Here are the steps to construct the new structural description:

1. Components for the new structural description is the union of the components of all the partitions.
2. Connections for the new structural description is the union of connections of all the partitions plus the new connections between the partitions corresponding to the given set of edges.
3. The set of ports for the new structural description consists of the following:
    - The ports of the partitions that do not connect to any other port.
    - At those junctions of the new structural description that have only input (or only output) ports of subcomponents connected to them, an additional port is created. This new port serves as an input (or output) port of the new structural description.

5.2.5 Compose-cpds-method

This method gets the functions and FRs of a given structural description by composing the functions of the components. As mentioned in section 3.3, some functions can be composed by simply concatenating them following the structural description. But its not possible to do so for all functions. More work needs to be done to come up with general criteria to identify what kind of functions can be composed in such a manner. For the range of components and functions we have used in the the domain of electrical circuits, we have developed some criteria to determine whether a certain component function is composable, that is, can be composed with other component functions in the above manner.

A function of a component is composable if it satisfies the following conditions:

1. Independent variables of the function are quantities at the input ports only.
2. Dependent variables are quantities at the output ports only.
3. Sets of ports used by independent variables and dependent variables are mutually exclusive.

Given a structural description the following method is used to compose the functions of the components to generate a new cpd and a new function:

1. Check if each component of the structural description has at least one composable function. If not, then return from (5.2.5) with nil.
2. For each possible set S, where S is a set of composable functions one for each component, try to generate a new function and the corresponding FR as follows:
   (a) For each function in S generate a cpd with two nodes corresponding to the *given* and *toachieve* predicates of the function.
   (b) Based on the connections between the components, create new links of as-per type combining nodes in the cpds generated in the last step. For each new link created verify that the start states of the link imply the final states of the link. If for any link the ANDing of start states does not imply the final states, then the corresponding cpds (whose nodes are being linked together) are not composable. In such a case go to step (2e).
   (c) Generate a function definition for the new cpd generated in the previous step.
   (d) Determine independent and dependent variables for the new function. Set of independent variables is the union of independent variables of all the functions whose *given* state appears as the start-state of the new cpd generated in the previous step. Set of dependent variables for the new functions is the union of dependent variables of all the functions whose *toachieve* state appears as the end-state of the new cpd.
   Return from (5.2.5) with the new function and the new cpd.
   (e) Continue with the loop at (2(a)).

5.2.6 Write-equations-method

If none of the above methods works for a structural description, then ask the user if it should use this method. If the user says no, then it will return from (5.2.6)with nil. Else it will return the set of equations modeling the given structural description. The following set of predicates and equations is used to model the given structural description:

- For each function of each component appearing in the structural description, generate a predicate by ORing the following three predicates:
  - Predicate appearing in the *toachieve* slot of the function.
  - Not of the predicate appearing in the *given* slot of the function.
  - Not of the *provided* condition associated with the function.
- Equations written for the connections between the components. For example, in the domain of electrical circuits these will be the KVL and KCL equations.

5.3 Clean-up-frs-g

While analyzing the device, information about the behavior context of the components, that is, functions and FRs of the containing device, may not be known. This will often be the case when the device analysis is bottom up. Once the containing device has been analyzed, the information about the behavior context of a component can be used to reject some of the functions and FRs of the component. After the whole device has been analyzed, the selected functions and FRs of the device can be used to select specific functions and FRs for the components. The goal is to remove functions and FRs of the components that are not relevant to the device functions. Also, if there are any superfluous components in the structural description that do not contribute to the function in any way, then such components will be removed from the FR model.

Method: context-cleanup.
For each device function, it goes through the corresponding FR. For all the *using-function* type of links in the FR, the functions of components referred to in these links are collected. For each component in the given structural description, the functions of components that do not appear in the above set, that is, do not get used in the FR, are rejected. The set of functions and the cleaned up FRs are returned.

CHAPTER VI

Examples

In this section we will show the execution steps of this algorithm on two examples. These two examples together demonstrate most of the features of our algorithm.

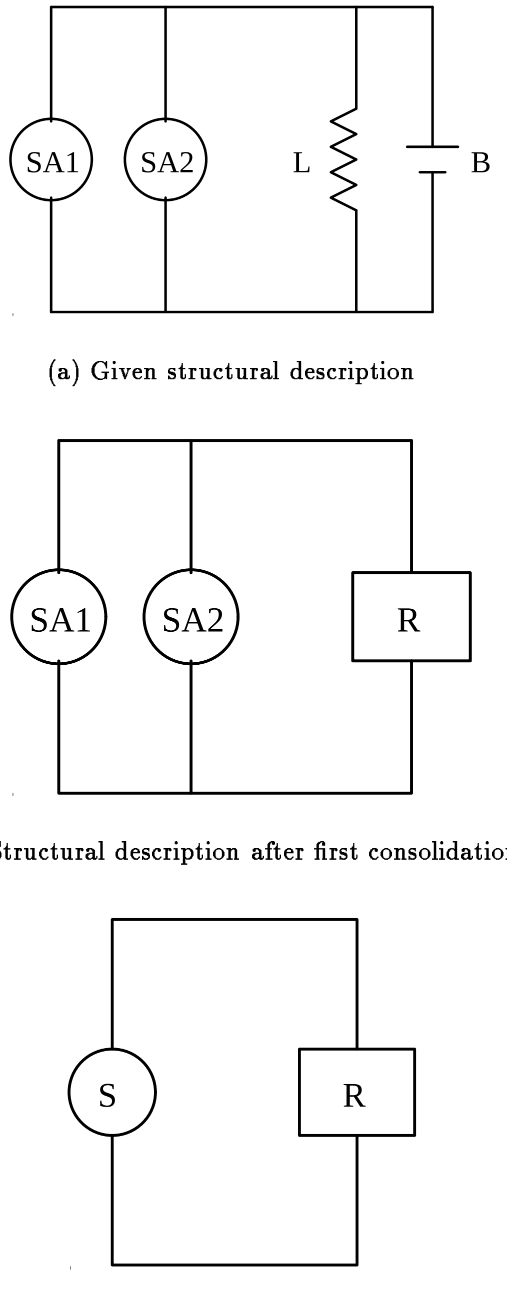
6.1 Battery recharging circuit

The circuit to be analyzed is the one given in Figure 14(a). *SA1* and *SA2* are solar cells, B is a rechargeable battery, and L is a load. The structural description of this circuit shown in Figure 15 is provided as an input to the system. Given the structural description of this circuit, an output that captures the following is expected:

> The function is *deliver-power*. This function is achieved as follows:
>
> *SA1* and *SA2* in parallel are equivalent to a *parallel-sources* S. Rechargeable battery, B, is probably acting as a load. B and L in parallel are equivalent to *parallel-resistors* R. Given structural description is equivalent to the structural description shown in Figure 14(c). New structural description is a *closed-circuit* with a source, S, and a resistor, R. By KVL for the *closed-circuit* the voltage across R is equal to the voltage across S. Using *parallel-load* function of R, power is delivered to L and battery B is recharged. The *parallel-load* function of R is achieved as follows. As-per *parallel-resistor-voltage* domain law, the voltage across R would result in voltage across L and B. By *ohms-law* voltage across L would cause current through L. By *consume-power* function of L, current through L will deliver power to the load L. Voltage across B will cause current through B. By *consume-power* and *accumulate-charge* functions of the battery B, current through B will deliver power to B and recharge B.

6.1.1 Components used

For the circuit in Figure 14(a), the system uses the following components from the library to perform the task:



(a) Given structural description

(b) Structural description after first consolidation

(c) Final structural description

Figure 14: Given circuit and its abstractions

components: ((solar-array SA1) (solar-array SA2)
(rechargeable-battery B) (resistor L))

connections: ((jh1 ((SA1 p1) (SA2 p1) jh3))
(jh2 ((SA1 p2) (SA2 p2) jh4))
(jh3 ((L p1) (B p1) jh1))
(jh4 ((L p2) (B p2) jh2)))

Figure 15: Representation of the structural description of the given circuit

1. Primitive components:

   (a) Resistor
   parameters: resistance
   cpds: resistor-v-i-cpd (voltage-story?), resistor-i-v-cpd (current-story?), consume-power-cpd (power-story?)

   (b) Electrical-load
   parameters: resistance
   cpds: inherited from resistor. Consume-power-cpd is the default.

   (c) Voltage-source
   parameters: emf, internal-resistance, current-capacity
   cpds: deliver-voltage-cpd

   (d) Solar-array
   parameters: emf, internal-resistance, current-capacity
   cpds: deliver-voltage-cpd (sun-shining?)

   (e) Rechargeable-battery
   parameters: emf, internal-resistance, charge-capacity, charge-threshold, actual-charge, charging-resistance
   cpds: deliver-voltage-cpd (actual-charge > CT), accumulate-charge-cpd (actual-charge < CT), consume-power-cpd (actual-charge < CT)

2. Abstract components:

   (a) Parallel-resistors
   parameters: resistance
   cpds: parallel-load-cpd (parallel-load-p), shunt-resistor-cpd (shunt-resistor-p), resistor-i-v-cpd (current-story?), resistor-v-i-cpd (voltage-story?), consume-power-cpd (power-story?)

(b) Parallel-sources
parameters: emf, current-capacity;; inherited
cpds: deliver-voltage-cpd (inherited)

(c) closed-circuit
parameters: nil
cpds: closed-ckt-cpd

In addition to the structural description, two values are also specified. *sunlight?* is set to true and the parameter *charge-level* for battery B is less than C_0, the charge threshold for the battery. If the charge in the battery falls below C_0 it needs to be recharged and is unlikely to be used as a source.

6.1.2 Main steps in execution

Following are the main steps in the analysis of the circuit given in Figure 14(a). Complete trace of the execution of this example is given in Appendix A.

- *Simplify-sd* does not modify the circuit.
- Set up the goal *analyze-device-g* for the given structural description.
- No special methods are applicable.
- *Fetch-match-sff-method* returns nil, because the given structural description does not match any SFF.
- *Consolidate-analyze-method*: Since the charge level of the battery is below the charge threshold C_0, it is probably in the recharge mode and not in the source mode. Two consolidations are possible. *SA1* and *SA2* in parallel, or L and B in parallel. The L and B pair is randomly selected for consolidation.
- The parallel combination of L and B is replaced by an instance, R, of a *parallel-resistances* component class generating a new structural description given in Figure 14(b).
- *Provided-hyp-context* for the parallel-resistor R. Of the three cpds associated with the component *parallel-resistors*, conditions for the cpds *resistor-v-i-cpd, resistor-i-v-cpd*, and the *power-cpd* are unknown. The *provided* condition for *shunt-resistor-cpd* evaluates to nil, and the condition for *parallel-load-cpd* evaluates to true. All cpds, except the *shunt-resistor-cpd* are selected at this time.
- *Extend-refine* cpds of R. The *parallel-load-cpd* shown in Figure 16 has partially specified links of type using-function. The functions of the two subcomponents, L and B, have been left unspecified. Determine the functions of L and B and fill up the missing information in the links.

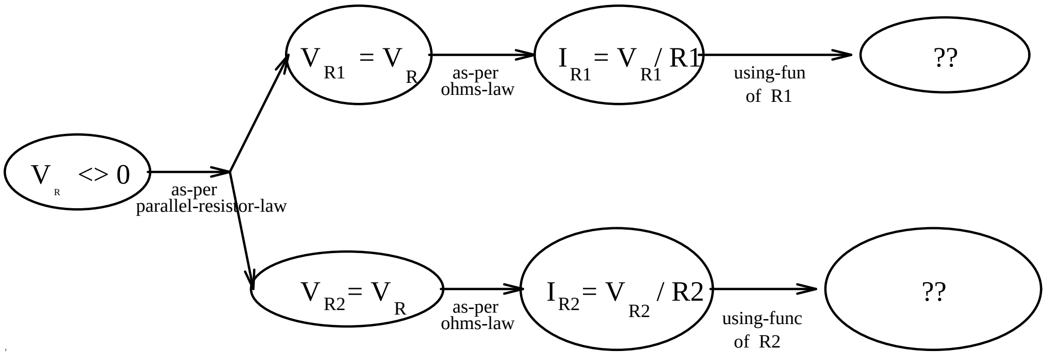
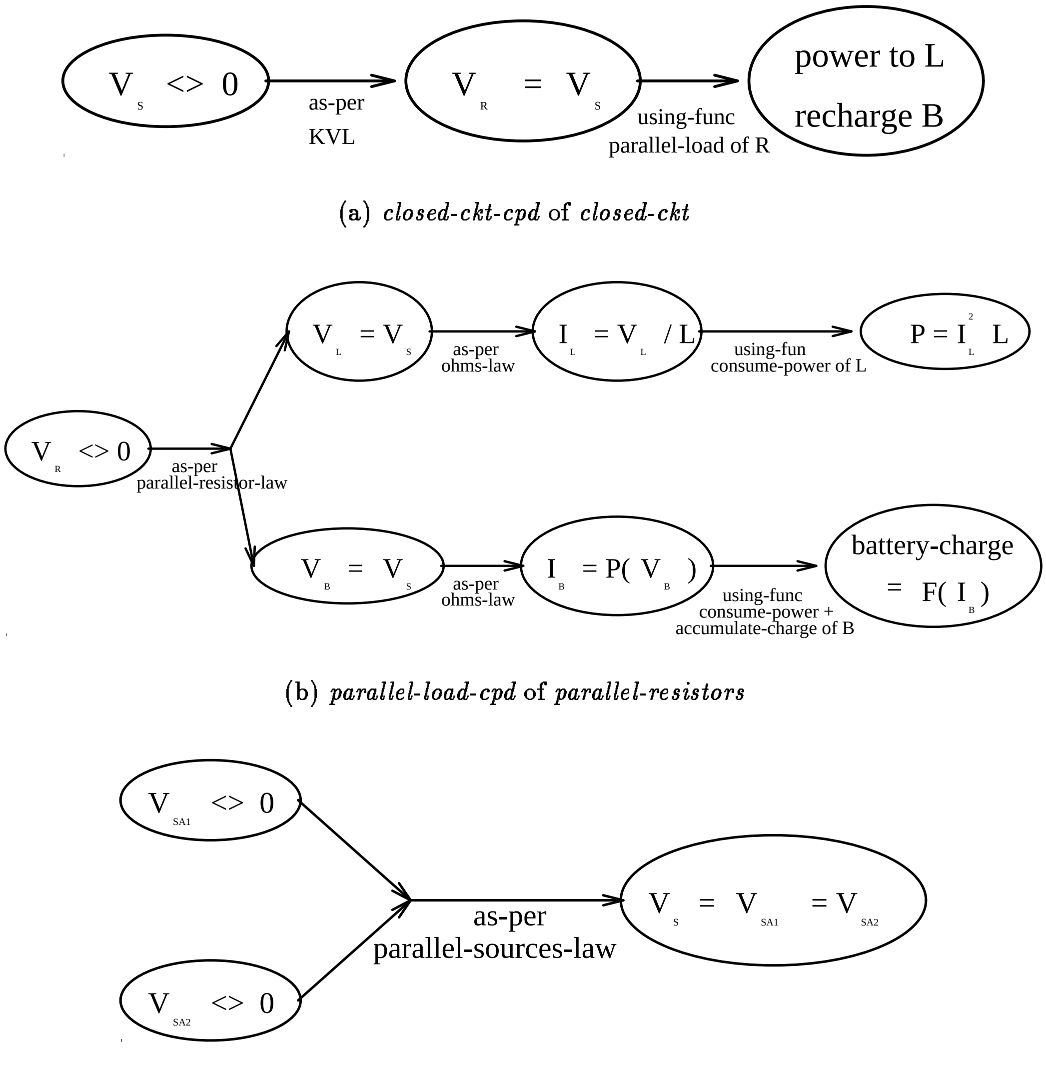


Figure 16: *parallel-load-cpd* template

- *L* is of type *electrical-load* which has three cpds associated with it. Since the conditions for all three cpds are unknown, only the default cpd *consume-power-cpd* is selected.
- *B* is of type *rechargeable-battery* with three cpds associated with it. Since the battery has already been assumed to be acting as a load, the condition for the cpd *deliver-voltage-cpd* evaluates to false. As the conditions for *consume-power-cpd* and *accumulate-charge-cpd* evaluate to true, both are selected.

This yields the parallel-load-cpd shown in Figure 17(b).

- Set up the goal *analyze-device-g* for the new structural description in Figure 14(b).
- No special methods are applicable.
- *Fetch-match-sff-method* returns nil, because the given structural description does not match any SFF.
- *Consolidate-analyze-method*: Picks the pair of parallel sources *SA1* and *SA2* for consolidation.
- Replace the pair *SA1* and *SA2* by an instance, *S*, of the component class *parallel-sources*, to get a new structural description shown in Figure 14(c).
- *Provided-hyp-context* for *S*. There is only one cpd *deliver-voltage-cpd*, shown in Figure 17(c) associated with the *parallel-sources* which is selected, because its *provided* condition is always true.
- *Extend-refine* cpds of *S*. Since the only cpd of *S* is fully specified nothing is done.



(a) *closed-ckt-cpd* of *closed-ckt*

(b) *parallel-load-cpd* of *parallel-resistors*

(c) *deliver-voltage-cpd* of *parallel-sources*

Figure 17: Final cpds for the given circuit

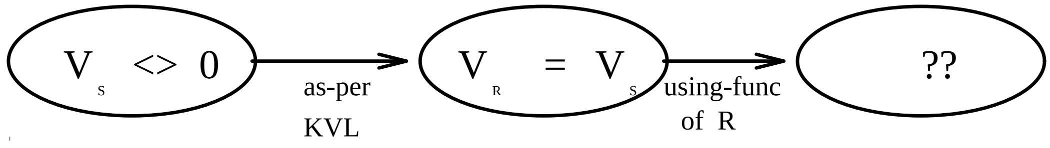


Figure 18: *closed-ckt-cpd* template

- Set up the goal *analyze-device-g* for the new structural description in Figure 14(c).
- No special methods are applicable.
- *Fetch-match-sff-method*: All SFFs with two subcomponents are fetched as potential candidates. Only the closed-ckt SFF actually matches.
- Create an instance of the *closed-ckt* and instantiate it to the structural description in Figure 14(c).
- *Provided-hyp-context* for the *closed-ckt*. The only cpd, *closed-ckt-cpd* shown in Figure 18, associated with it is selected, because its *provided* condition is always true.
- *Extend-refine closed-ckt-cpd*. There is one *using-func-link* in *closed-ckt-cpd* that is partially specified. The function of *R* has been left unspecified.
- Since *R* has already been analyzed, the selected function for *R*, *parallel-load-func*, is used to fill up the link and the subsequent partial state in the *closed-ckt-cpd*.
- Return from all the *analyze-device-g's*.
- *Context-cleanup*. Of all the selected functions and cpds for the components, remove the functions and cpds of components that are not used by the selected functions and cpds of the containing device. For example, *resistor-v-i-cpd, resistor-i-v-cpd*, and *power-cpd* of *parallel-resistor R* are rejected because these cpds are not relevant in the context of the function of the containing device. This leaves the FR shown in Figure 17. Return this FR along with the top level function corresponding to the *closed-ckt-cpd* as the output of the task.

The set of cpds selected in this analysis, shown in Figure 17, describes how the given structural description in Figure 14(a) achieves the function of delivering power to the load and recharging the battery.

6.2 Temperature measuring device

The circuit to be analyzed is the one shown in Figure 20(a). This is a circuit of a temperature measuring device. To explain how it works, consider the equivalent structural description of the same device shown in Figure 20(b). The thermocouple generates a voltage signal proportional to the temperature on the input port. The instrumentation amplifier filters and amplifies the voltage signal generated by the thermocouple. The analog-display takes the output voltage signal of the instrumentation amplifier and generates a deflection proportional to the input voltage. In the instrumentation amplifier, the signal first passes through a buffer for isolation. The output of the buffer passes through a low pass filter to filter out noise in the input signal. The filtered signal is then amplified by the amplifier. The driver takes the amplified signal from the amplifier and generates a proportional higher power signal. The functions of the buffer, lp-filter, and the amplifier are further explained in terms of the components op-amp, resistors, and capacitors.

The structural description, shown in Figure 19, of the temperature measuring device is given as an input to the system. In addition to some of the components defined in the previous example, the system uses the following component models from the library.

6.2.1 Components used

1. Primitive components:

   (a) Op-amp
   parameters: beta, input-impedance, output-impedance
   cpds: zero-ip-voltage-cpd, zero-ip-current-cpd

   (b) Capacitor
   parameters: capacitance, leakage-resistance
   cpds: charging-cpd, leakage-cpd (leakage-beh?), cap-v-i-cpd (voltage-story?), cap-i-v-cpd (current-story?)

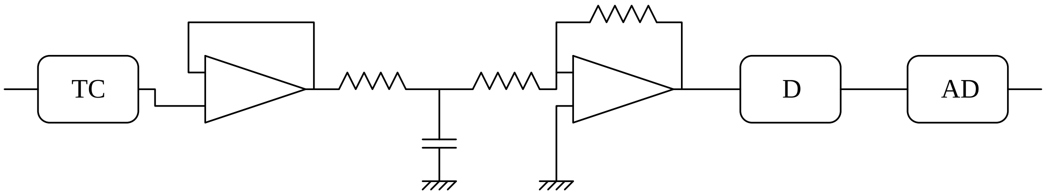
   (c) Transducer
   parameters: K (scaling factor)
   cpds: nil (the type of the output port is not specified)

   (d) Thermocouple
   parameters: K (scaling factor), Vl , Vu (voltage range), Tl, Tu (temperature range)
   cpds: temp-voltage-cpd, clipped-temp-volt-cpd (when temp > Tu), temp-zero-volt-cpd (when temp < Tl)

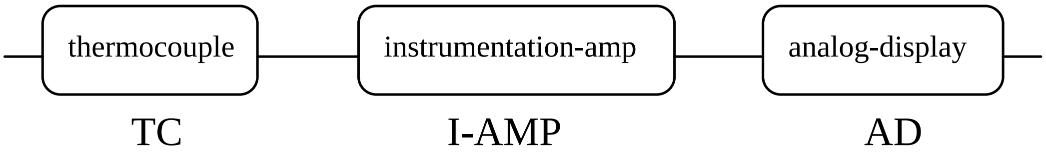
```
components:   ((thermocouple  TC)
                (op-amp  OB1)  (capacitor  CB1)  (ground  GB1)
                (resistor RF1)  (capacitor  CF1)  (ground  GF1)
                (op-amp  OA1)  (resistor  RA1)  (resistor  RA2)
                (ground  GA1))

connections:  ((jx1  (p1  (TC  p1)))
                (jx2  ((TC  p2)  jb1))
                (jb1  (jx2  (OB1  p2)  (CB1  p1)))
                (jb2  ((CB1  p2)  (GB1  p1)))
                (jb3  ((OB1  p1)  (OB1 p3)  jx3))
                (jx3  (jb3  (RF1  p1)))
                (jf1  ((RF1  p2)  (CF1  p1)  jx4))
                (jf2  ((CF1 p2)  (GF1 p1)))
                (jx4  (jf1  (RA1  p1)))
                (ja1  ((RA1  p2)  (OA1  p1)  (RA2  p1)))
                (ja2  ((OA1  p2)  (GA1  p1)))
                (ja3  ((OA1  p3)  (RA2  p2)  jx5))
                (jx5  (ja3  (DR  p1)))
                (jx6  ((DR  p2)  jx7))
                (jx7  (jx6  (AD  p1)))
                (jx8  ((AD  p2)  p2)))
```

Figure 19: Representation of the structural description of the given device



(a) The circuit of the temperature measuring device



(b) Abstract structural description of the same device

Figure 20: Structural descriptions of the temperature measuring device

(e) Driver
parameters: VG (voltage gain), CG (current gain), IVT (input voltage threshold), ICT (input current threshold)
cpds: drive-voltage-cpd, drive-current-cpd

(f) Display
parameters: K (scaling factor), Vl, Vu (input voltage range)
cpds: nil (the type of the output port is not specified)

(g) Analog-display
parameters: K (scaling factor), Vl, Vu (input voltage range)
cpds: meter-deflection-cpd, no-deflection-cpd (when input voltage $<$ Vl), max-deflection-cpd (when input voltage $>$ Vu)

2. Composite components:

(a) Buffer
parameters: Ri (input resistance), Ro (output resistance), K (scaling factor), IT (input voltage threshold)
cpds: buffer-cpd, clipped-buffer-cpd

(b) Lp-filter
parameters: Fo (cutoff frequency), IT (input threshold voltage)

cpds: lp-filter-cpd

(c) Integrator
parameters: Vo (output voltage), t (time at which Vo was registered)
cpds: integrator-cpd

(d) Amplifier
parameters: x (amplification factor), VT (input voltage threshold)
cpds: amplifier-cpd, clipped-amp-cpd

(e) Inverter
parameters: IT (input voltage threshold)
cpds: inverter-cpd, clipped-inverter-cpd

(f) Instrumentation-amp
parameters: x (amplification factor), IT (input threshold voltage)
cpds: instrumentation-amp-cpd, clipped-ia-cpd

(g) Sweep-generator
parameters: S (slope factor), IT (input threshold voltage) cpds: sweep-ckt-cpd

(h) Measuring-instrument
parameters: K (scaling factor)
cpds: measuring-instrument-cpd, clipped-mi-cpd

6.2.2 Main steps in execution

We will show two different executions of this example. The two executions differ in the knowledge provided to the system. In the first execution, as we will see, the system considers a large number of hypotheses to generate the function and the FR of the whole device. In the second execution, the system is provided with the additional knowledge that if the given structural description contains a sub-component of type *transducer* and a sub-component of type *display*, then hypothesize the given structural description to be an instance of the *measuring-instrument*. Once a function has been hypothesized for the given structural description, the given structural description is analyzed in a top-down fashion using a special method, resulting in a highly focused reasoning.

Execution 1: without the additional rule

The complete trace of the first execution is given in Appendix B. Here are the main steps the algorithm goes through to analyze the circuit in Figure 20(a).

- *Simplify-sd* does not change the given structural description.

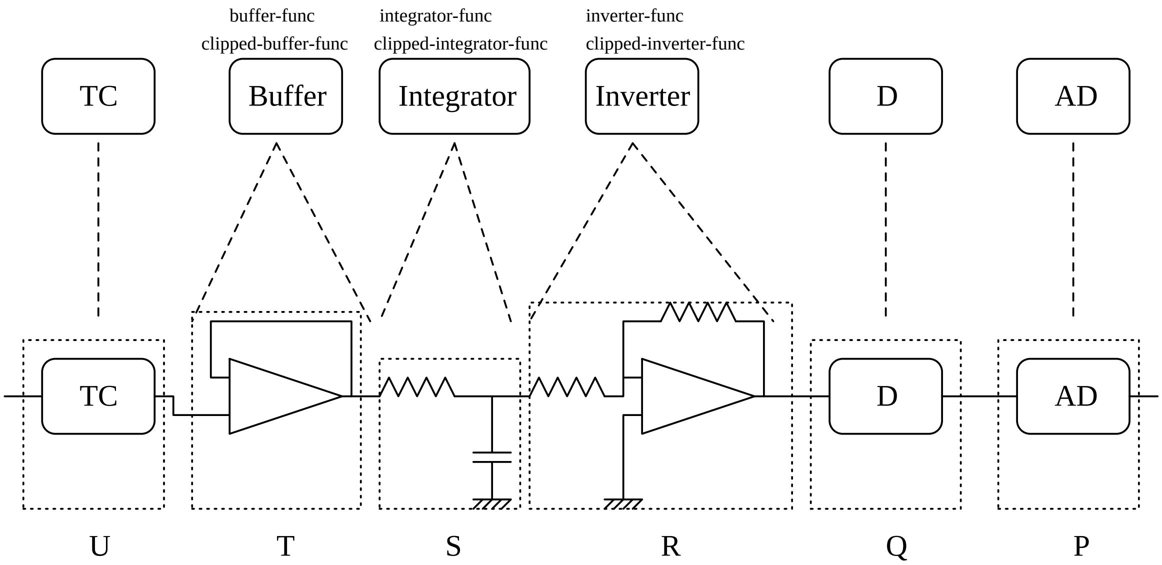


Figure 21: Analyzing the given structural description

- Set up the goal *analyze-device-g* for the given structural description.
- No special method is applicable.
- *Fetch-match-sff-method* returns nil, because no matching SFF could be found.
- *Consolidate-analyze-method* returns nil, because no consolidation is possible.
- *Partition-method*: Partitions the given structural description by breaking all the j-j edges in the given structural description. Six partitions, each one corresponding to the structural description in the box P, Q, R, S, T, and U in Figure 21, are generated.
- Analyze each partition: set up the goal *analyze-device-g* for each partition.
- *Analyze-device-g* for partition P: P is an instance of *analog-display*. Applicable functions and cpds are selected for P.
- *Analyze-device-g* for Q: Q is an instance of *driver*. Applicable functions and cpds are selected for Q.
- *Analyze-device-g* for R: *Fetch-match-sff-method* matches this structural description on to the inverter SFF. An instance of inverter is created and applicable functions and FRs are selected.
- *Analyze-device-g* for S: *Fetch-match-sff-method* matches this structural description on to an integrator SFF. An instance of integrator is created and applicable

functions and FRs are selected.

- *Analyze-device-g* for T: *Fetch-match-sff-method* matches this structural description on to a buffer SFF. An instance of buffer is created and applicable functions and FRs are selected.
- *Analyze-device-g* for U: U is an instance of a *thermocouple*. Applicable functions and cpds are selected for U.
- The result of analyzing all the partitions is shown in Figure 21. Set up the goal *compose-subsets-g* for the analyzed partitions P, Q, R, S, T, U. Try composing various subsets of the analyzed partitions. No subset of the analyzed partitions matches a known SFF. Return from *structural-composition* with nil.
- Go back and try to redo the previous step in the *partition-method*, that is, the step "analyze each partition."
- *Analyze-device-g* for P and Q returns the same result as before.
- *Analyze-device-g* for R: Last time R was analyzed as an inverter. This time *fetch-match-sff-method* matches this structural description on to the amplifier SFF. An instance of amplifier is created and applicable functions and FRs are selected.
- Since a new result has been obtained for R, old results are used for the remaining components S, T, and U. This is shown in Figure 22.
- Set up the subgoal *compose-subsets-g*. This time it identifies a subset that can be composed and the resulting structural description matched on to an SFF. Results for R, S, and T, that is, buffer, integrator, and the amplifier are composed to form a new structural description, $SG1$.
- Set up the goal *analyze-device-g* for $SG1$. *Fetch-match-sff-method* matches it to the *sweep-generator* SFF. Applicable functions and SFFs are selected. See Figure 22.
- Set up the goal *compose-partitions-g* to compose the thermocouple, the sweep-generator, the driver, and the analog-display.
- Set up the goal *analyze-device-g* for the new structural description obtained in the previous step. *Fetch-match-sff-method, consolidate-analyze-method,* and the *partition-method* all of them return nil for this structural description.
- Try redoing the previous step. That is, redo the *analyze-device-g* for the the structural description $SG1$. No new result is possible. So go back one more step.

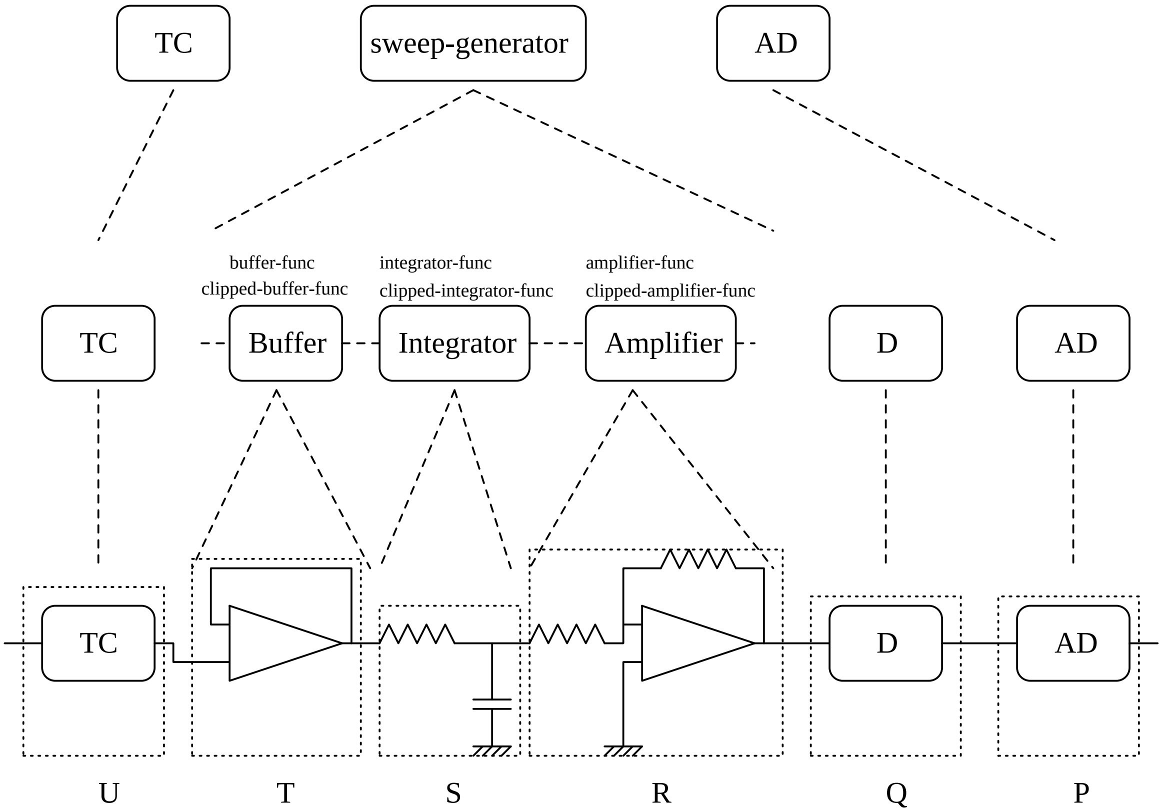


Figure 22: Analyzing the given structural description

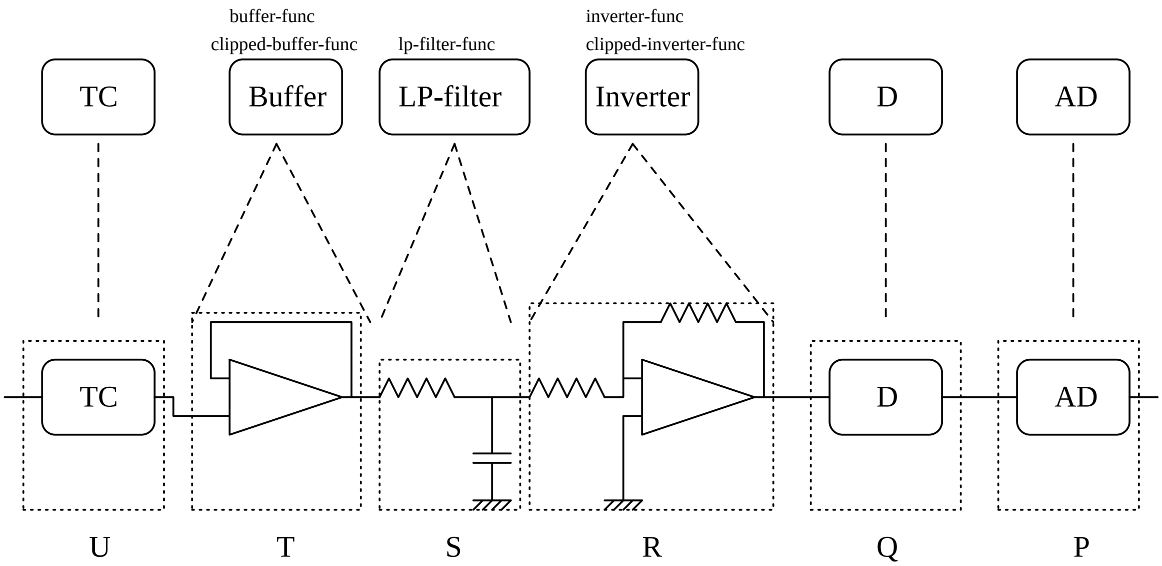


Figure 23: Analyzing the given structural description

- Try redoing the *compose-subset-g* with P, Q, R, S, T, and U. Again no new result is possible. So go back one more step.
- Try redoing the step "analyze each partition." Set up *analyze-device-g* for each partition.
- Same results are obtained for partitions P and Q.
- Partition R is analyzed by *fetch-match-sff-method* to get an instance of an *amplifier*.
- Partition S is analyzed by *fetch-match-sff-method* to get an instance of *lp-filter*. Applicable functions and cpds are selected.
- Since a new result is obtained for S, old results are used for the remaining components T and U. This is shown in Figure 23.
- Set up the goal *compose-subset-g* for the new analyzed components. This goal returns nil, because it fails to find a subset of components that would match an SFF.
- Once again try redoing the last step. That is, try the step "analyze each partition" one more time.
- Same results are obtained for the partitions P and Q.
- Partition R is reanalyzed as an inverter by *fetch-match-sff-method*.

- Old results are used for the remaining partitions S, T, and U. This is shown in Figure 24.
- Set up the goal *compose-subset-g* for the analyzed partitions.
- It identifies a subset consisting of the components buffer, lp-filter, amplifier, and driver. Compose these components to get a new structural description IA.
- Set up the subgoal *analyze-device-g* for IA. *Fetch-match-sff-method* is used to analyze this structural description. Its identified as an instance of *instrumentation-amp*. Applicable functions and cpds are selected.
- Set up the goal *compose-partitions-g* for the thermocouple, instrumentation-amp, and the analog-display. It composes these three components and returns a new structural description MI. This is shown in Figure 24.
- Set up the goal *analyze-device-g* for MI. Again *fetch-match-sff-method* is used to match it to the *measuring-instrument* SFF. Applicable function, *measuring-instrument-func*, and the corresponding cpd, shown in Figure 11(b) are selected. The other function associated with this SFF, *clipped-mi-func*, is rejected, because the *provided* condition evaluates to false. The provided condition for the *clipped-mi-func* checks if the maximum output voltage of the transducer is more than the input voltage threshold of the *instrumentation-amp*. Since for the given structural description this condition is not true, the corresponding function is not applicable.
- Set up the goal *extend-refine-g* for the *measuring-instrument-cpd*. Since the functions of the thermocouple and the analog-display are already known, these functions are used to fill up the missing information in the *measuring-instrument-cpd*.
- Return from *partition-method*. Return from top level *analyze-device-g*.
- Set up *context-cleanup*. Functions of components that are not relevant in the context of the *measuring-instrument-func* function of the containing device are rejected. For example, the function *clipped-ia-func* of the instrumentation amplifier, and the function *clipped-buffer-func* of the buffer are rejected, because these functions do not contribute to any selected function of the containing device.

Figure 26 shows the resulting FR of the temperature measuring device, which has been understood as an instance of the abstract component measuring-instrument.

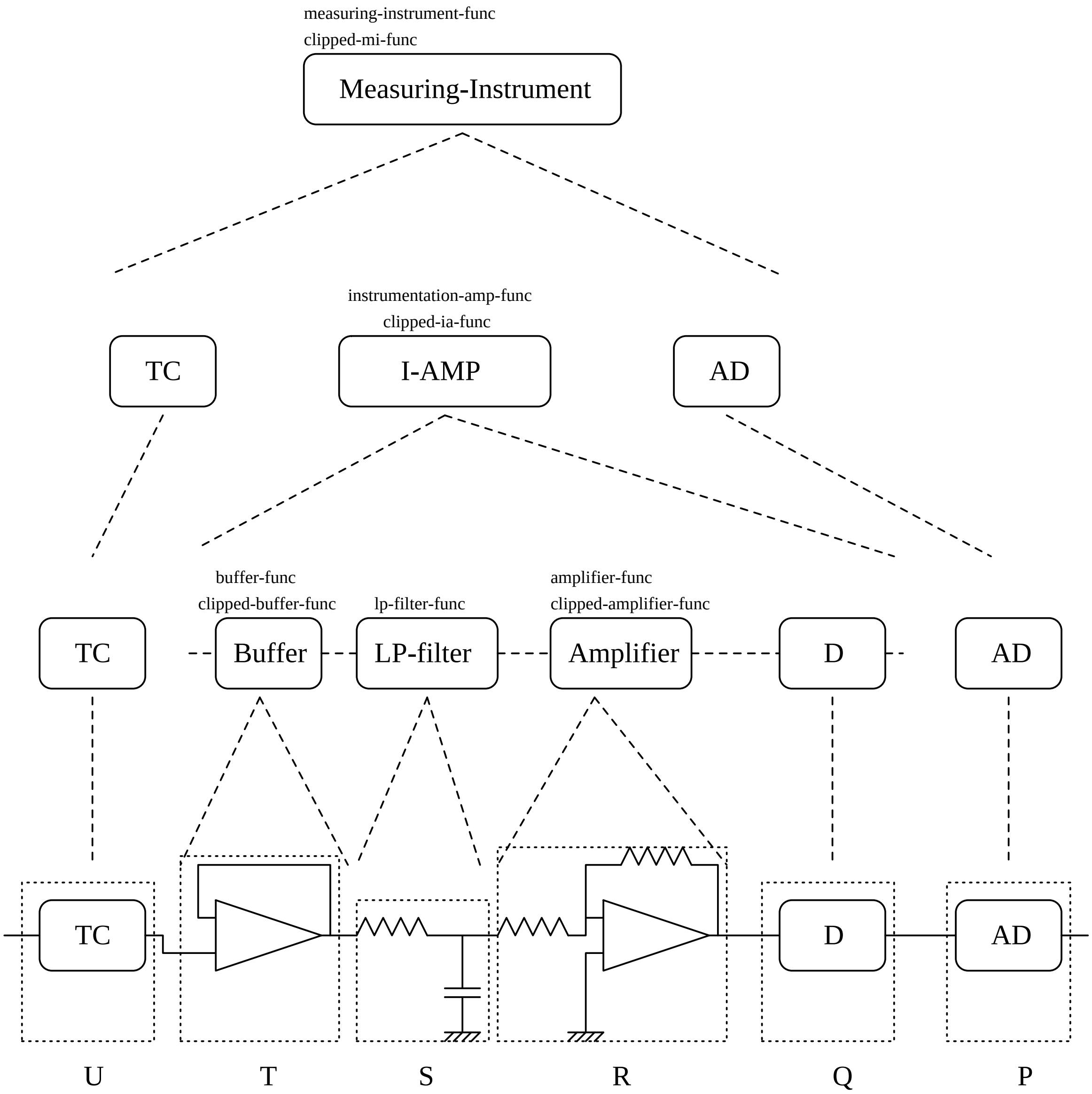
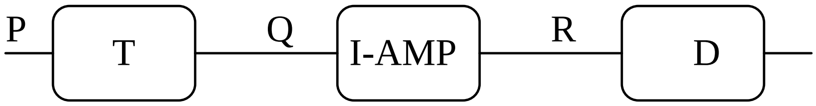
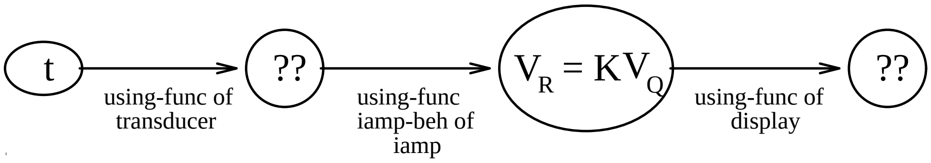


Figure 24: Analyzing the given structural description



(a) Structural description of the *measuring-instrument*



(b) *measuring-instrument-cpd* template

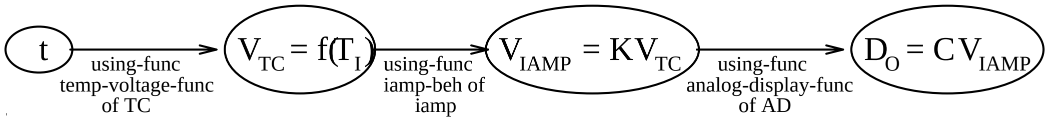
Figure 25: Measuring instrument

Execution 2: with an additional rule

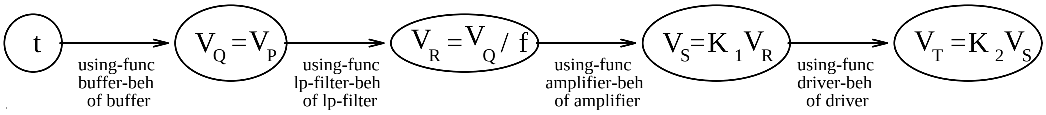
As mentioned earlier, this time the system is provided with an additional rule that if the given structural description contains a sub-component of type transducer and a sub-component of type display, then hypothesize the the given structural description to be a *measuring-instrument* with the function *measuring-instrument-func*. With this additional knowledge the special method for analyzing measuring instruments gets invoked which is able to analyze the given structural description in a very focused manner. Since the system already knows from the measuring-instrument SFF that a measuring instrument is supposed to contain an instrumentation-amp, and that an instrumentation-amp contains an amplifier and a low pass filter, the partitions R and S are directly mapped on to the *amplifier* SFF and the *lp-filter* SFF respectively. So the information about the function of the device is used to disambiguate the selection of functions for the components at the lower levels. This is unlike the previous execution in which the system was following a purely bottom up approach and had to backtrack several times because not enough information was available locally to disambiguate between the various possible functions for the partitions R and S.

Again the detailed execution of this example is given in Appendix C. Here are the main steps the algorithm goes through during the execution of this example using the additional knowledge.

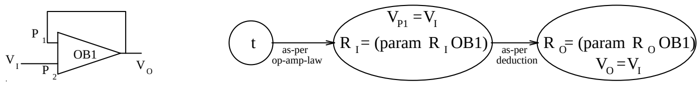
- Special-method for measuring-instruments is applied. This method first partitions the given structural description to isolate the transducer part. This is



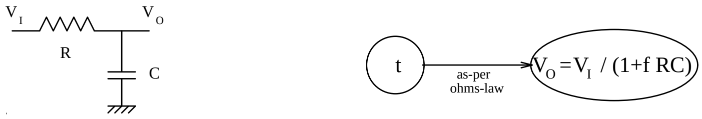
(a) *measuring-instrument-cpd*



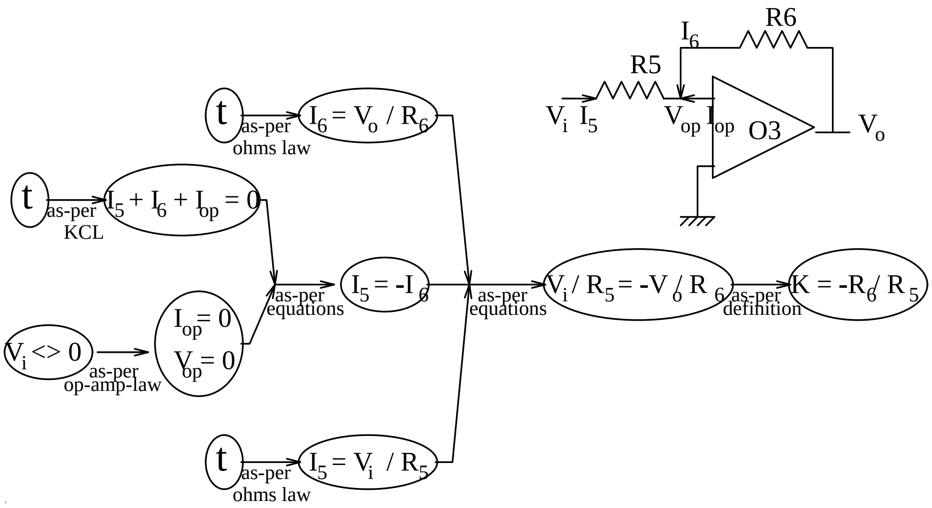
(b) *instrumentation-amp-cpd*



(c) *buffer-cpd*



(d) *lp-filter-cpd*



(e) *amplifier-cpd*

Figure 26: Cpds for the temperature measuring device

done by partitioning the given structural description into two parts, and selecting the part with the input port. This part is partitioned again. This process is repeated till the part with the input port cannot be partitioned further by breaking the j-j edge. In this case it will separate the partition U.

- Set up the goal *analyze-device-g* for the transducer part. This partition is an instance of the thermocouple. Applicable functions and cpds are selected.
- Partition the remaining structural description to separate the display part. This is done by repeatedly partitioning the structural description till the part with the output port cannot be partitioned further by breaking a j-j edge. In this case, this will separate the partition P.
- Set up the goal *analyze-device-g* for the display part. This partition is an instance of the analog-display. Applicable functions and cpds are selected.
- The remaining structural description corresponds to the instrumentation amplifier. The remaining structural description is partitioned by breaking all j-j edges generating the partitions Q, R, S, and T. The slot *beh-context* is set to the cpd *instrumentation-amp-cpd* for each of the partitions. Each partition is analyzed separately.
- Set up the goal *analyze-device-g* for partition Q. This is an instance of driver. Applicable functions and cpds are selected.
- Set up the goal *analyze-device-g* for partition R. *Fetch-match-sff-method* fetches the SFF candidates and hypothesizes functions for this structural description. One of the hypothesized functions, *amplifier-func*, appears in the function context, that is, in the *instrumentation-amp-cpd*. So the function *amplifier-func* is selected as against the function *inverter-func*.
- Set up the goal *analyze-device-g* for partition S. *Fetch-match-sff-method* fetches the SFF candidates and hypothesizes functions for this structural description. One of the hypothesized functions, *lp-filter-func*, appears in the behavior context, that is, in the instrumentation-amp-cpd. So the function *lp-filter-func* is selected as against the function *integrator-func*.
- Set up the goal *analyze-device-g* for partition T. *Fetch-match-sff-method* matches this partition on to the buffer SFF. The *buffer-func* is selected for this partition.
- The cpd template, *measuring-instrument-cpd*, for the measuring instrument is filled to get the cpd for the temperature measuring device. This completes the FR for the given structural description.

CHAPTER VII

Search spaces and complexity

In this section we define the various search spaces in which the algorithm searches, and then for each search space we discuss the complexity of the search and how the search is constrained.

The task is to go from the given structural description to the functions and FRs of the given structural description. As mentioned earlier the algorithm uses three different methods to go from structure to functions and FRs. These are *fetch-match-sff-method, compose-cpds-method, and write-equations-method*. Each of these methods searches in a different search space.

7.1 Fetch-sff-method

In the *fetch-match-sff-method* three operators *num-func-special, type-subtype-match*, and *match-conns* are used to search in the space of SFFs to find the matching SFF. The following strategies have been used to control the search in this search space.

1. The step *num-func-special* searches in the space of SFFs (or abstractions) to fetch the SFF candidates for the given structural description. The size of the space of abstractions is usually determined by the domain and the range of device functions the system is expected to be able to analyze. As described earlier the SFFs are organized by the functions associated with the SFF, by the number of subcomponents in the corresponding abstract structural description, and by the special subcomponents these abstract devices might contain. Given a structural description, each of the above three criteria can be used to index into a set of SFF candidates.
2. Search is made more efficient by ordering the search operators in the order of their computational complexity. For example, while trying to verify the match between the given structural description and the SFF candidates, simple operators like *type-subtype-match* (to match the subcomponents of the structural descriptions) are used to reject as many SFF candidates as possible before

trying the computationally expensive operators like *match-conns* (to match the connections of the structural descriptions).

3. The operator *type-subtype-match* searches in the space of SFF candidates identified by the *num-func-special* operator. *type-subtype-match* operator checks if the subcomponents of the given structural description match the subcomponents of the abstract structural description of the SFF. Matching the subcomponents of the two structural descriptions is like matching the elements of two sets. The worst case complexity for this step is $O(n^2)$, where n is the number of subcomponents. Our experience is that most abstractions used by the domain experts have only a small number of subcomponents. Rarely would an abstraction have more than eight subcomponents. If an abstraction has more than eight or so subcomponents, then usually it is not only possible but also more efficient to introduce additional abstractions and reduce the number of subcomponents in each abstraction. Thus by restricting the number of subcomponents in each abstraction, one can further reduce the computational complexity of the *type-subtype-match* operator.

4. If the subcomponents of the two structural descriptions match, then for the given subcomponent-match the operator *match-conns* checks if the connections of the two structural descriptions match. Connections are matched by matching junctions.
   Assume the structural description has n subcomponents and each subcomponent has at the most c ports, where c is a constant.
   Since each junction can have at most nc entries, checking if two junctions match takes $O(n^2)$ time (matching two sets each with at most nc elements).
   Further, each port can participate in at the most one junction, and each junction has to have at least two ports. $\Rightarrow$ The maximum number of junctions possible in each structural description is $nc/2$.
   To match the connections of the two structural descriptions requires one to match their junctions. At the most $(nc/2)^2$ junction comparisons are required. $\Rightarrow$ Time complexity of *match-conns* is $O(n^4)$.

   As stated earlier, the number of subcomponents in the abstract structural description of an SFF is usually less than eight or so. That keeps the complexity of *match-conns* operator under control.

After an SFF has been matched, the *provided-hyp-context* operator verifies the hypothesized functions and selects the functions that are relevant in the given context. Functions associated with the matching SFF become the hypothesized functions of the given structural description. So *provided-hyp-context* searches in the space of hypothesized functions. Since the number of matching SFFs and hence the functions hypothesized for a structural description is usually not too large, each hypothesized function is checked for its applicability. In this work, we use only structural constraints to verify the applicability of a function. This is done by checking, for each function, the truth of a predicate on the values of component parameters. So the complexity of the verification step is determined by complexity of determining the truth of a predicate, the *provided* condition, associated with the functions.

The verified functions are tested for relevancy by matching them against the hypothesized functions (*hypothesized-func*) or the behavior context (*beh-context*). The relevancy test is done only for the verified functions and each test takes constant time.

7.2 Compose-cpds-method and write-equation-method

The complexity of the compose-cpds method is linear in the number of components whose functions are to be composed. The operator write-equations goes through two steps to get the equations for the given structural description. The first is to write equations for the subcomponents, and second step is to write equations for the connections. The first step is linear in the number of subcomponents. The second step is linear in the number of junctions and hence in the number of subcomponents. So the overall worst case complexity for this method is $O(n)$, where n is the number of subcomponents.

7.3 Consolidate-analyze-method

For a given structural description, the *par-ser-comb* step of the *consolidate-analyze-method* operator generates an equivalent structural description. Equivalent structural descriptions are determined by consolidating pairs of parallel or serial sources or resistors. For parallel consolidation it picks up a pair of parallel sources (or resistors) that are connected at a junction, and then checks all other junctions to see if the other ports of these two sources (or resistors) are also connected together.

Let the number of subcomponents be n, and let each subcomponent have at the most c ports.

We can assume that at each junction there are at the most k component ports, where k is a constant.

For each junction number of possible pairs of parallel sources (or resistors) $= C_2^k$

There are at the most $nc/2$ junctions.

$\Rightarrow$ Maximum number of total possible pairs $= nc/2.C_2^k$

For each pair, one has to check each junction to see if their other ports are also

connected together. Checking each entry at each junction means checking all the ports.
$\Rightarrow$ Maximum number of comparisons for each pair $= 2nc$.
$\Rightarrow$ Maximum number of total comparisons for all the possible pairs $= nc/2.C_2^k.2nc = O(n^2)$.

For serial sources (or resistors) it checks if two and only two sources (or resistors) are connected at a junction. It checks each junction to identify serial sources (or resistors).
Complexity of this step $= O(\text{number of junctions}) = O(n)$.

Rest of the steps in *par-ser-comb* operator take constant time.

7.4 Partition-method

The *partition* method also generates a new structural description. It generates new structural descriptions by relabeling parts of the given structural description. It is this operator that helps restrict the size of the abstraction space searched by the *fetch-match-sff* operator. The *partition-method* first partitions the given structural description into parts. The general purpose partition algorithm is intractable. We have used the following heuristics to control the search by the *partition* operator:

1. The partition algorithm tries to decompose a given structural description by breaking junction-junction (j-j) edges only. This significantly reduces the size of the search space for the partition algorithm.
2. The partition algorithm tries to partition the given structural description by breaking only one or two edges.

Following is a complexity analysis of the various steps in the partition algorithm that breaks j-j edges only.

- Break j-j edges: = O(number of j-j edges)
  Maximum number of edges in a structural description with n subcomponents $< (nc)^2$, where constant c is the maximum number of ports per subcomponent. Usually a port is connected to at the most two or three other ports.
  $\Rightarrow$ Usually the number of edges $= O(n)$
  $\Rightarrow$ Number of j-j edges (which is a subset of all the edges) $= O(n)$
  $\Rightarrow$ The complexity of this step $= O(n)$.
- Separate the partitions:
  - Start with one of the subcomponents in set S.

- Loop for each element, c, in S:
  Go through each junction j of the given structural description. If a port of element c occurs in j, then set
  $S = S \cup$ all components whose ports meet at j, and
  restart the loop.

Let the number of junctions be k.
Let each junction have at the most p subcomponent ports.
$\Rightarrow$ Number of comparisons in the loop for each element in $S = pk$

Let the number of subcomponents in any partition be $\leq x$.
$\Rightarrow$ Above loop is repeated at the most x times for each partition.
$\Rightarrow$ Complexity of the above loop for each partition $= O(pkx)$
Usually $x < 8$, and p is typically less than 5
$\Rightarrow$ Complexity of the above loop $= O(k) = O(n)$

The above loop for separating the partitions is executed once for each partition.
$\Rightarrow$ Complexity of this step $= O(n^2)$

- Analyze each partition:
  Let the complexity of analyzing each part be $O(T)$.
  Number of partitions be $= y = O(n)$
  Complexity for coming up with one possible analysis for all the partitions $= O(Tn)$.

  Since it may be possible to analyze a partition in more than one way, a given set of partitions may be analyzed in several ways. But for most structural descriptions there would be only one, and in rare cases two, matching SFFs. Hence in most cases the given structural description can be analyzed in only one way. So for a given set of partitions, the number of ways the set of partitions can be analyzed is usually a very small number if at all greater than one.

- Compose-subsets: At the time of partitioning the structural description an adjacency list representation of the graph corresponding to the structural description consisting of the partitioned components is generated.

  To generate a subset graph of size r, that is with r nodes, by traversing the adjacency list in a depth first manner, one has to traverse Cr links, where C is constant.
  To generate all the possible subsets of size r, do the above n times, each time starting with a different node. This will take $= nCr$ steps.

Complexity of generating all the possible subsets $= \sum_{r=1}^{y} nCr$
Assuming there are no abstractions with size larger than eight, we do not need to try and compose subsets of size larger than eight.
$\Rightarrow$ The complexity of generating all the subsets $= \sum_{r=1}^{8} nCr = O(n)$.

- Composing the subset obtained in the previous step to get a new structural description, S_1, is linearly proportional to the number of components in the subset. Since the number of components in the subset is less than 8 or so, the complexity of this step is constant.
- Analyze S_1 takes $O(T)$ time.
- Composing S_1 with the remaining partitions to get a new structural description S_2 takes $O(n)$ time, because the number of partitions is proportional to n.

Thus, the complexity of all the operators used in this algorithm in almost all the situations is linear in the size of the given structural description, that is, in the number of subcomponents. The algorithm scales up pretty well. The cruise control device described here has about twenty components and our system could analyze this device in about a minute and to analyze the temperature-measuring-device without the additional rule it took about 90 seconds. Scalability of the representation and the reasoning methods has been one of the main driving motivations for the design of the FR language and also for this algorithm. Using single-edge or double edge partition methods significantly increase the complexity of the algorithm.

Assuming the FR templates associated with the SFFs are all correct, then the soundness of the algorithm in Figure 6(a) is determined by the soundness of the verification step. In this work we use only structural criteria to verify hypothesized functions. If appropriate *provided* conditions can be written to check the applicability of the function and enough information is provided to compute these conditions, then the algorithm will generate sound results. In case enough information is not provided such that all the relevant *provided* conditions cannot be evaluated, then other criteria are used to select functions. *Provided* conditions associated with the selected functions are assumed to be true. For every *provided* condition that is assumed to be true, it is checked if it is consistent with the previous assumptions. This would be best handled by incorporating an ATMS with the system, but right now this done by asking the user. In such cases, there exists a set of parameter values such that all the *provided* conditions that were assumed to be true would evaluate to true. Thus the algorithm in Figure 6(a) is sound. The other two methods of going from structure-to-function are sound by definition.

With respect to the knowledge provided to the system, the algorithm is not complete. This is because the partition heuristic does not try all the possible partitions, as a result there could be possible decompositions of the given structural description that never get considered.

CHAPTER VIII

Comparison with other work

Most of the work on reasoning about devices did not pay much attention to the problem of device modeling till very recently. Most of the work in the area of reasoning about the physical world and more specifically about devices focused on the behavior generation part which involved simulating the device model to generate the behaviors and identifying means to control the simulation. Not much attention was paid to the device modeling part itself.

[dK85] proposes methods to generate functions and their causal accounts from the structure of the device. The device is first modeled using component models and domain laws for the interconnections. For example, an agent reasoning about electrical circuits would have component models for a resistor, capacitor, transistor etc. The models are in the form of qualitative equations. These component models are kept as general as possible to improve their range of applicability. The model of a device consists of the model of the components and the equations written using the domain laws for connections between the components. For example, in the domain of electrical circuits KVL and KCL are used to write the equations for the connections.

Causal account of how a behavior comes about is generated by doing causal analysis. The input disturbance is propagated through the device model by doing constraint propagation and the causal ordering between the device's state variables is determined by the order in which the values are assigned to the variables in the device model. The output of causal analysis is a mechanism graph which is somewhat like a cpd. A node in the mechanism graph corresponds to a value assignment to a state variable, and the directed edge represents causal ordering between the variables referred to in the nodes. Edges are annotated either by a component name or the domain law whose equation was used to determine the causal dependency represented by the edge.

The hierarchical functional decomposition of the device is generated by doing teleological analysis. Teleological analysis parses the mechanism graph, the output of the causal analysis, and replaces fragments of the causal graph by equivalent functional components. Parts of the resulting graph, called the fragment graph, are abstracted or transformed using rewrite-rules. Rewrite rules are applied recursively till the fragment graph matches on to a single function. The result is a hierarchical functional model of the device.

The goals of [dK85] are very different from the goals of the work described in this thesis. [dK85] aims to study the causal analysis and teleological analysis separately to gain an insight into these processes. The teleological knowledge is deliberately not used to control, direct, or focus the causal analysis. This work on the other hand uses teleological knowledge to build functional models of devices. The goal of this work is to study how the knowledge of the function of the device can be used to control and direct the causal analysis. Further, this work proposes how a large number of domain specific function templates, capturing teleological knowledge, may be represented, organized, and used to build function specific causal models.

Another consequence of the above difference in the goals is that the device models generated in [dK85] are general purpose models at a single level of abstraction. Further the device model refers to all the state variables that appear in its component models. Several state variables and behaviors of the components may not even be relevant in the context of the given device. Even more variables and component behaviors would be irrelevant if one is interested only in specific aspects of the behavior of the device. Such general purpose models are useful for, say, discovering behavioral implications of a new design or for reasoning about devices in a completely new domain. This work on the other hand generates function specific device models at multiple levels of abstraction, like an FR model. In an FR model only those functions of components are modeled that are relevant in the context of the function of interest of the device.

Another important difference between [dK85] and the work presented here is that the device model and the mechanism graph generated by [dK85] are at a single level of abstraction and refer only to component level state variables. The fragment graph also refers only to the component level state variables at all levels of abstraction. This work on the other hand models the device at multiple levels of abstraction using cpds at multiple levels of abstraction and at each level new state variables may be introduced. The causal accounts at various levels of abstraction are related.

[JS85] describes a template-based approach to understanding software programs that is somewhat similar to the template-based teleological parse proposed in [dK85]. Given a computer program and a specification of the problem the program is supposed to solve, [JS85] tries to verify if the given program solves the problem. [JS85] uses a library of programming plans that are indexed by goals. A programming plan is a code template that may refer to other goals. Goals that match the given problem specification are used to fetch the programming plans. Parts of the given program are then checked to see if they match the corresponding code templates. Like the template-based teleological parse of [dK85], this approach is also based purely on structural matching and no behavioral descriptions like cpds are used to drive function from structure. It is the use of the behavioral descriptions (descriptions of how a specific behavior of the device arises from the functions of the sub-components) that enables our approach to use other techniques, such as simulation-based technique, to drive function from structure for those parts of the given structural description for which no matching SFF template can be found. Where as in [JS85] if a part of the given program cannot be matched with any code template, then some other proof of correctness cannot be used for that part of the code template because there is no representation for the role a part of the code template plays in achieving the specific goals of the whole code template.

The approach used in [For84] is very similar to the one in [dKB84] except that the building blocks for modeling are process models instead of the component models. A process model requires certain structural constraints to be fulfilled and defines a set of equations relating the relevant state variables of the given situation. A situation, at a given time, is modeled by the set of processes active at that time. Based on the given initial conditions, the set of applicable processes models the given situation. The initial set of processes is simulated to determine the changes in the values of state variables which might make other processes active. All the arguments given in the previous paragraph for [dKB84] also hold for this process based approach.

[NJA91] proposes a method to build somewhat more goal-specific device models. This approach is also component based. Each component is associated with multiple models capturing different aspects of the component's behaviors. Structural, behavioral, and functional context are used to select specific component models for a given device. For structural context [NJA91] uses component's physical properties and the kind of connections (like electrical connection) it has with other components. For example, for a piece of wire with temperature sensitive resistive properties appearing in an electrical circuit one may choose the model of a thermal resistance. This approach enables model selection only at a very coarse level, that is, this approach cannot be used to select between models with fine distinctions. This is because it does not take in to account a very important aspect of structural context - the specific component configuration of which the component is a part. For example, a transistor in different component configurations (all in the domain of electrical circuits) may need to be modeled differently. For the behavior context of a component, [NJA91] uses the values of certain state variables of the component itself. This is only one aspect of the behavioral context of a component. Its not clear how other aspects of the behavioral context can be used using this mechanism without violating the principle of modularity for the component models. For example, how to account for the non-local behavioral context, that is, values of state variables that are not part of the component under consideration, which play an important role in selecting specific component models. The biggest limitation of this approach is that this also models the device at a single level of abstraction. Further, to capture fine distinctions between models, enumerating all the possible models for each component at a single low level of abstraction may not be feasible.

[FF91a] proposes a method for goal-specific model composition. The task addressed in [FF91a] is to build an appropriate device model to answer a query about the behavior of the device. Depending on the features of the given situation and the terms of the query, appropriate ontology, granularity level, and the approximation assumptions are made to build a model to answer the query. Ontology assumptions determine the global modeling perspective like, contained stuff ontology or the energy flow ontology. Granularity assumptions determine what components of the device are to be modeled explicitly and which ones can be modeled together as a subsystem. Approximations assumptions select between the alternate models for the same situation or component.

Ontology assumptions help in model selection but such global commitments can be used to make only coarse modeling distinctions. Approximation assumptions may be used to select between models with finer distinctions, but they have not proposed any principled way, domain specific or domain independent, of mapping the top level goal, in this case the given query, to the appropriate approximation conditions. As in the modeling approaches mentioned in the above paragraphs, this approach also does not take in to account the specific component configuration the component is a part of, to select the component model. Here also, the final model generated is at a single level of abstraction, and it is not clear how this method can be extended to generate models at multiple levels of abstractions.

In addition to the ontology assumptions behavior context is also captured by operating assumptions, which like [NJA91] are constraints on the values of the state variables of the component. But they have not proposed any principled way of mapping the device behavior on to the behavior assumptions of the components. Thus the method doesnt really support developing behavior specific device models.

Another recent work in the area of modeling is [ACP91]. Even though the problem addressed in [ACP91] is a modeling problem, the way it is formulated makes it very different from the kind of modeling problems we have been talking about. [ACP91] assumes that the behavior of the device is known. Given a structural description and a device behavior, the goal is to generate a model of the device that predicts the given behavior. Rules based on the differences between the values of the behavioral parameters are used to direct the search. The differences between the given behavior and that generated by the model are traced back to the modeling assumptions that might fix these differences. For large devices, a difference in the value of a parameter may be traced back to several modeling assumptions. Its not clear how the search is kept under control. Also, since the search for the appropriate model is guided completely by the differences between the actual and the expected behaviors, structural context is never taken in to account for selected component models. Here also the models are generated at a single level of abstraction, and it is not clear how can this method be used to generate device models at multiple levels of abstraction.

CHAPTER IX

Contributions and Future Work

9.1 Contributions

Following are the main contributions of this work:

1. We identified and proposed a solution for a specific version of the structure-to-function reasoning problem in which one is interested in generating function specific device models at multiple levels of abstraction. This is a useful version of the general structure-to-function reasoning problem because several tasks, such as diagnosing the device and explaining how a function of the device comes about, require device models with these characteristics.

   The solution proposed here generates such device models. The characteristic "function specific" or "the multiple levels of abstraction" is not something that is added on after generating general purpose device model at a single level of abstraction. Instead, these characteristics are an integral part of the device modeling process itself. Abstractions are used by our system to understand parts of the given device, and these same abstractions then become a part of the output device model. And the function specific models associated with these abstractions become part of the function specific model of the whole device.

   The solution proposed uses both the top-down and the bottom-up strategies to constrain the reasoning. Methods like *special-methods*, *extend-refine-links*, using hypothesized functions and the behavior context in *provided-hyp-context* are examples of top-down strategies. Where as, *fetch-match-method* and *write-equations* are examples of bottom-up strategies. As shown in the chapter on search space analysis, the method proposed here scales up for a large class of devices.

2. In contrast to the most past approaches to structure-to-function reasoning problem, the approach proposed here assumes a knowledge rich environment. Most past approaches to the device understanding problem use only the knowledge

of component models and domain laws to derive functions from structure of devices. The solution proposed here represents and uses additional domain specific knowledge that corresponds to the understanding of frequently encountered abstractions in the domain. In fact the solution approach was motivated by how the domain experts understand an unfamiliar device in the same domain.

3. We also proposed strategies to organize the knowledge of various abstract devices in memory. The knowledge of abstract devices is captured by SFFs. These SFFs may be organized by function, special components, or the number of subcomponents they contain. We also showed how such organization helps control the search during the task of device understanding.

4. Another important contribution of this work is that it proposes a way to represent and use certain kinds of non-local knowledge to understand devices. By non-local knowledge we mean the knowledge that cannot be associated with any specific component. In particular two kinds of non-local knowledge have been represented in our system. One is the user's interest in specific functions of devices which is captured by the specific set of abstractions (or SFFs) provided to the system. The other is that SFFs capture structural and behavioral context and the context-specific models of the subcomponents referred to by the SFFs. The function specific model associated with an SFF refers to specific aspects of the behaviors of a subcomponent of the SFF, aspects that are relevant in the structural context of the abstract structural description associated with the SFF and aspects that are relevant in the behavioral context of the specific function of the SFF being modeled. SFFs capture these two kinds of non-local knowledge and this knowledge is used in the process of understanding devices.

5. We have proposed a way to represent and use abstract functions, cpds, and structural descriptions. These abstractions are represented using the function, cpd, and structural description templates. An abstract structural description defines a class of structural descriptions using a set of constraints on the structural description. Function (and FR) templates associated with an abstract structural description define a class of functions (and FRs) that are functions (and FRs) of the instances of the corresponding structural description class.

6. We also proposed heuristics for matching structural descriptions and for partitioning structural descriptions.

9.2 Research issues and future work

Currently our method does not use simulation at all during the modeling process. As mentioned earlier, simulation may be used locally on parts of the given structural description for various purposes. It may be used to verify hypothesized functions for parts of the device. Or it may be used to functionally match a given structural description to an abstract structural description. To use simulation for either of these two purposes one has to be able to set up simulation for a part of the structural description and compute the values of the relevant state variables to the desired degree of approximations. There are two main problems with setting up such simulations. The first is that the input to the part of the device to be simulated is often not known, and trying to get this information by simulating the rest of the device may be computationally very expensive or may not even be possible because of lack of enough information. The second problem is that to be efficient the simulation needs to be highly goal directed, because the purpose of such simulations is often to get very specific results just to be able to verify or disambiguate some predicates on state variables. It is not clear how to carry out such focused and goal directed simulations. We need to develop mechanisms to set up such simulations for parts of the device as and when the need arises. In addition to developing such a simulation mechanism, we also need to develop a control strategy that will allow seamless integration of such simulation-based method with the rest of the steps. The goal-subgoal architecture that we have used should help in developing such control strategy.

So far we have considered devices with only the *toachieve* type, the most common type, of function. Additional work needs to be done to extend this method to handle other kinds of functions, *tomaintain, tocontrol,* and *toprevent* [Keu89].

Further, we assume the FRs to be strictly hierarchical, and expect a one-to-one correspondence to exist between structural and functional decompositions. If the FR is not strictly hierarchical, say when the cpds of two subsystems interact, the current partition method or the compose-cpd method will not be able to handle such devices. Techniques to handle such devices need to be developed.

Here we have proposed three different ways to organize the SFFs in the library. As the number of SFFs becomes quite large, these three techniques alone may not be enough to keep the size of the search space for *fetch-comps* step under control. We need to develop more such domain independent or domain dependent techniques to organize SFFs.

Additional work may be done to improve the fetch, match, and partition algorithms. Both to improve their efficiency and to improve their range of applicability.

Currently we use a domain specific criteria to determine if the functions and the cpds of a given set of components are composable. As mentioned in [CGI93], to develop more general criteria to determine composability, further work is required to clearly understand the notion of ports in a component.

Currently we do not have any way of responding to the varying degrees of approximation required for models of various parts of the device. At best we can associate multiple models with components and SFFs, each one with different degrees of approximation and each one having a set of conditions associated with it to determine when that model would be applicable. But methods like these, which involve explicitly enumerating models with different approximation assumptions, cannot be used to achieve a fine variation in the degree of approximation, which is so characteristic of human reasoning. We need to develop techniques to perform computation with parts of the device model only up to the required degree of approximation as determined by the current goals of the agent, thus minimizing the computation effort.

Another dimension along which more work may be done is to develop mechanisms for automatically generating SFFs. When a structural description does not match any of the given SFFs, then some abstract form of this structural description should become a potential candidate for a new SFF. If similar structural description is encountered often enough one should generate a new SFF corresponding to these structural descriptions. This would require some generalization techniques.

Currently we define an abstract structural description for an SFF by specifying only a subcomponent's class and parameter range. We need to identify other dimensions along which a component class may be abstracted.

The inference strategy implicit in the algorithm described in this thesis is a version of abductive inference [JJ94]. Assuming the given device was designed to achieve specific functions, the task can be viewed as hypothesizing functions of the device that best explain the structural description of the device. Viewed like this the task is a version of abduction problem. Since the primary intention in this thesis was to highlight the use of cpd templates for deriving function from structure, the problem was not stated as an abduction problem. By explicitly framing the proposed approach as an abductive approach one may be able to use some of the already existing frameworks for abductive reasoning.

Another interesting dimension along which this work may be extended is to handle situations where the structural description of the device may not be completely specified. In such cases the structural description itself needs to be hypothesized before the behavior of interest of the device can be explained in terms of the functions of its subcomponents. Consider the example (this example is due to Prof. B. Chandrasekaran) of a recorded video cassette whose recording gets erased when it is left on top of a speaker of an audio amplifier. When one tries to explain the phenomenon of the recording on the video cassette getting erased, one may not initially realize the connection between the the subcomponents *speaker* and the *video cassette*. Such partial structural descriptions are very common social or economic situations, where relations between parts or sometimes even some of the parts themselves may not be obvious. Before one can try to explain the behavior of interest in the given situation, one has to hypothesize the structural description of the situation that results in the behavior of interest. Cpd templates of various processes may be used to hypothesize the structural description itself. In this case the understanding of the two processes (represented using cpds), how something is recorded on a video cassette and how the audio amplifier speaker produces sound from electrical signals, can be used to hypothesize the magnetic field connection between the speaker and the video cassette. Once the appropriate structural description has been hypothesized the cpd templates of the same two processes may be used generate an explanation for the behavior of interest using the approach described in this thesis.

Appendix A

Execution trace of Example 1

Following is the execution trace for the structural description given in Figure 11(a). Comments, statements preceded by ";;", have been inserted to make it easier to understand the trace.

```
;; Top level goal is to analyze the device given in Figure 11(a).
 0: (ANALYZE-DEVICE #<HUBBLE @ #x7e03e6> NIL)

;; Try fetch-match-sff-method.
   1: (FETCH-COMP-METHOD #<HUBBLE @ #x7e03e6> NIL)

;; Fetch SFF candidates from memory.
     2: (FETCH-COMPONENTS #<HUBBLE @ #x7e03e6> NIL)
     2: returned (((INTEGRATOR INTEGRATOR-I) 1) ((AMPLIFIER
  AMPLI FIER-I) 1))
     2: (INSTANCE-GENERIC-P #<INTEGRATOR @ #x7d89fe> #<HUBBLE @
#x7e03e6>)
     2: returned NIL
     2: (INSTANCE-GENERIC-P #<AMPLIFIER @ #x7d8956> #<HUBBLE @
#x7e03e6>)
     2: returned NIL
   1: returned NIL
;; Fetch-match-sff-method failed.

;; Try consolidating using par-ser-comb-method.
   1: (PAR-SER-COMB-METHOD #<HUBBLE @ #x7e03e6> NIL)
     2: (PARALLEL-COMP-DECOMPOSITION #<HUBBLE @ #x7e03e6>)
     2: returned (T2601 T2598)
;; Parallel consolidation successful.  Consolidates L and B.

;; Select funcs for the new parallel comp.
     2: (SELECT-CPD #<PARALLEL-RESISTOR @ #x85bb6e>)
       3: (EVALUATE-COND (VAR-VALUE CURRENT-STORY-P)
                          #<PARALLEL-RESISTOR @ #x85bb6e>)
       3: returned ?
...
```

```
       3: (EVALUATE-COND (PARALLEL-LOAD-P) #<PARALLEL-RESISTOR @
#x85bb6e>)
       3: returned T
       3: (EVALUATE-COND (SHUNT-RESISTOR-P) #<PARALLEL-RESISTOR @
 #x85bb6e>)
       3: returned NIL
     2: returned #<PARALLEL-RESISTOR @ #x85bb6e>

;; Extend-refine selected functions of the parallel-resistor comp
     2: (EXTEND-REFINE #<CPD @ #x7d976e> #<PARALLEL-RESISTOR @
#x85bb6e>)
     2: returned #<CPD @ #x7d976e>
...

;; Parallel-load function can be refined further
     2: (EXTEND-REFINE #<CPD @ #x7d8836> #<PARALLEL-RESISTOR @
#x85bb6e>)
;; Get applicable functions of the rechargeable battery B
       3: (EVALUATE-COND (AND (EQUAL (VAR-VALUE MATCHEDAS)
'VOLTAGE-SOURCE)
                              (> (PARAM ACTUAL-CHARGE) (PARAM CT)
))
                         #<RECHARGEABLE-BATTERY @ #x7e0396>)
       3: returned NIL
       3: (EVALUATE-COND (OR (SUBTYPEP (VAR-VALUE MATCHEDAS)
'ELOAD)
                             (SUBTYPEP 'ELOAD (VAR-VALUE
MATCHEDAS)))
                         #<RECHARGEABLE-BATTERY @ #x7e0396>)
       3: returned T T
...
;; Get applicable functions of the load L
       3: (EVALUATE-COND (VAR-VALUE CURRENT-STORY-P) #<ELOAD @
#x7e03be>)
       3: returned ?
...
     2: returned #<CPD @ #x7d8836>

;; Analyze the new structural description obtained after
;; consolidating L and B.  The new structural description is
;; shown in Figure 11(b).
     2: (ANALYZE-DEVICE #<COMPONENT @ #x85d10e> NIL)

;; Try fetch-match-sff-method
       3: (FETCH-COMP-METHOD #<COMPONENT @ #x85d10e> NIL)
         4: (FETCH-COMPONENTS #<COMPONENT @ #x85d10e> NIL)
         4: returned NIL
       3: returned NIL
```

```
;; Fetch-match-sff-method failed

;; Try consolidating using par-ser-comb-method
       3: (PAR-SER-COMB-METHOD #<COMPONENT @ #x85d10e> NIL)
         4: (PARALLEL-COMP-DECOMPOSITION #<COMPONENT @ #x8ef83e>)
         4: returned (T2619 T2616)
;; Consolidation successful.  Consolidated SA1 and SA2

;; Select applicable functions of the new parallel-sources
;; component
         4: (SELECT-CPD #<PARALLEL-SOURCE @ #x947fae>)
           5: (EVALUATE-COND T #<PARALLEL-SOURCE @ #x947fae>)
           5: returned T
         4: returned #<PARALLEL-SOURCE @ #x947fae>
         4: (EXTEND-REFINE #<CPD @ #x8ff9b6> #<PARALLEL-SOURCE @
#x947fae>)
         4: returned #<CPD @ #x8ff9b6>

;; Analyze the new structural description, shown in Figure 11(c),
;; obtained after consolidating SA1 and SA2.
         4: (ANALYZE-DEVICE #<COMPONENT @ #x948f8e> NIL)

;; Try fetch-match-sff-method.
           5: (FETCH-COMP-METHOD #<COMPONENT @ #x948f8e> NIL)
;; Get possible candidates.
             6: (FETCH-COMPONENTS #<COMPONENT @ #x948f8e> NIL)
             6: returned
                  (((CLOSED-CIRCUIT CLOSED-CIRCUIT-I) 1)
                   ((PARALLEL-RESISTOR PARALLEL-RESISTOR-I) 1)
                   ((PARALLEL-SOURCES PARALLEL-SOURCE-I) 1))
             6: (INSTANCE-GENERIC-P #<CLOSED-CIRCUIT @ #x8fee7e>
                                    #<COMPONENT @ #x948f8e>)
             6: returned
                  (((S T2616) (R T2598))
                   (((JCC1) (JH1 JH3)) ((JCC2) (JH2 JH4))) NIL
     NIL)
;; Matched with closed-circuit component, and returned an
;; instance of it.

;; Select applicable functions of the closed-circuit comp.
             6: (SELECT-CPD #<CLOSED-CIRCUIT @ #x94c366> NIL)
               7: (EVALUATE-COND T #<CLOSED-CIRCUIT @ #x94c366>)
               7: returned T
             6: returned #<CLOSED-CIRCUIT @ #x94c366>

;; Extend-refine the selected functions.
             6: (EXTEND-REFINE #<CPD @ #x8ffb5e>
#<CLOSED-CIRCUIT @ #x94c366>)
```

```
            6: returned #<CPD @ #x8ffb5e>
          5: returned (T2621 (T2621))
        4: returned (T2621 (T2621))
      3: returned (T2621 (T2616 T2621))
    2: returned (T2621 (T2616 T2621))
  1: returned (T2621 (T2598 T2616 T2621))
 0: returned (T2621 (T2598 T2616 T2621))

;; Remove redundant functions/cpds of components based on the
;; selected functions of the containing device.
 0: (CLEAN-UP-ALL-BEHS T2621)

    .....
 0: returned NIL
;; Completed analyzing the given structural description.

;;---------------------------------------------------------------

;; Printing out the output.  The output consists of the
;; hierarchical structural descriptions generated and the
;; corresponding functions and cpds.

;; The hierarchical structural description.  Component T2621 is
;; the highest level component.


Top level component

T2621 of class CLOSED-CIRCUIT
((VOLTAGE-SOURCE S) (RESISTOR R))          ;;Subcomponents
((JCC1 ((S P1) (R P1))) (JCC2 ((S P2) (R P2))))   ;;Connections
;; Instantiation.
(((S T2616) (R T2598)) (((JCC1) (JH1 JH3)) ((JCC2) (JH2 JH4)))
NIL NIL)


Next level

T2616 of class PARALLEL-SOURCE
((VOLTAGE-SOURCE S1) (VOLTAGE-SOURCE S2))        ;;Subcomponents
  ;;Connections
((JPS1 (P1 (S1 P1) (S2 P1))) (JPS2 (P2 (S1 P2) (S2 P2))))
(((S1 SA1) (S2 SA2)) NIL NIL NIL)   ;;Instantiation

T2598 of class PARALLEL-RESISTOR
((RESISTOR R1) (RESISTOR R2))       ;;Subcomponents
  ;;Connections
((JPR1 (P1 (R1 P1) (R2 P1))) (JPR2 (P2 (R1 P2) (R2 P2))))
```

```
(((R1 B) (R2 L)) NIL NIL NIL)   ;;Instantiation


Next level

((SOLAR-ARRAY SA1) (SOLAR-ARRAY SA2) (RECHARGEABLE-BATTERY B)
(RESISTOR L))

--------------------------------------------------------


;; Functions and cpds of components.


Component T2598 of class PARALLEL-RESISTOR

PORTS
((ELECTRICAL NIL P1 #<PORT @ #x93dfee>) (ELECTRICAL NIL P2
#<PORT @ #x93e006>))

BEH-CPD-NAME
(((PARALLEL-LOAD-P) PARALLEL-LOAD-BEH PARALLEL-LOAD-CPD DEFAULT))

GIVEN OF BEH
T

TOMAKE OF BEH
(AND (AND (EQUAL (PARAM P) (* (CURRENT P1)
      (CURRENT P1) (PARAM R)))
          (EQUAL (PARAM ACTUAL-CHARGE)
                 (+ (PARAM INITIAL-CHARGE) (* (PARAM K)
      (CURRENT P1) T))))
     (AND (EQUAL (PARAM P) (* (CURRENT P1) (CURRENT P1)
   (PARAM R)))))

INDEPENDENT VARS OF BEH
((VOLTAGE P1) (VOLTAGE P2))

DEPENDENT VARS OF BEH
((PARAM P) (PARAM ACTUAL-CHARGE) (PARAM P))

STATES OF CPD
(PARALLEL-LOAD-S1 T)
(PARALLEL-LOAD-S2
 (AND (EQUAL (VOLTAGE P1) (VOLTAGE P1 R1))
      (EQUAL (VOLTAGE P2) (VOLTAGE P2 R1))))
(PARALLEL-LOAD-S3
 (AND (EQUAL (PARAM P) (* (CURRENT P1) (CURRENT P1) (PARAM R)))
```

```
      (EQUAL (PARAM ACTUAL-CHARGE)
             (+ (PARAM INITIAL-CHARGE) (* (PARAM K) (CURRENT P1)
  T)))))
(PARALLEL-LOAD-S4
 (AND (EQUAL (VOLTAGE P1) (VOLTAGE P1 R2))
      (EQUAL (VOLTAGE P2) (VOLTAGE P2 R2))))
(PARALLEL-LOAD-S5
 (AND (EQUAL (PARAM P) (* (CURRENT P1) (CURRENT P1) (PARAM R)))))

LINKS OF THE CPD
(PARALLEL-LOAD-L1 USING-FUNC-LINK NIL (PARALLEL-LOAD-FUNC))
(PARALLEL-LOAD-L2 USING-FUNC-LINK R1 (CONSUME-POWER-BEH
      ACCUMULATE-CHARGE-BEH))
(PARALLEL-LOAD-L3 USING-FUNC-LINK NIL (PARALLEL-LOAD-FUNC))
(PARALLEL-LOAD-L4 USING-FUNC-LINK R2 (CONSUME-POWER-BEH))

CONNECTIVITY, START-STATES, END-STATES
((PARALLEL-LOAD-L1 (PARALLEL-LOAD-S1) (PARALLEL-LOAD-S2))
 (PARALLEL-LOAD-L2 (PARALLEL-LOAD-S2) (PARALLEL-LOAD-S3))
 (PARALLEL-LOAD-L3 (PARALLEL-LOAD-S1) (PARALLEL-LOAD-S4))
 (PARALLEL-LOAD-L4 (PARALLEL-LOAD-S4) (PARALLEL-LOAD-S5)))
(PARALLEL-LOAD-S1)
(PARALLEL-LOAD-S3 PARALLEL-LOAD-S5)

;;----

Component T2616 of class PARALLEL-SOURCE

PORTS
((ELECTRICAL NIL P1 #<PORT @ #x948d86>) (ELECTRICAL NIL P2
 #<PORT @ #x948ef6>))

BEH_CPD_NAME
((T DELIVER-VOLTAGE-BEH PARALLEL-VOLTAGE-CPD NIL))

GIVEN OF BEH
T

TOMAKE OF BEH
(EQUAL (PARAM EMF) (- (VOLTAGE P1) (VOLTAGE P2)))

INDEPENDENT VARS OF BEH
((PARAM EMF))

DEPENDENT VARS OF BEH
((VOLTAGE P1) (VOLTAGE P2))

STATES OF CPD
```

```
(PARALLEL-VOLTAGE-S1 T)
(PARALLEL-VOLTAGE-S2
 (EQUAL (PARAM EMF S1) (- (VOLTAGE P1 S1) (VOLTAGE P2 S1))))
(PARALLEL-VOLTAGE-S3 (EQUAL (PARAM EMF S1) (- (VOLTAGE P1)
      (VOLTAGE P2))))
(PARALLEL-VOLTAGE-S4 T)
(PARALLEL-VOLTAGE-S5
 (EQUAL (PARAM EMF S2) (- (VOLTAGE P1 S2) (VOLTAGE P2 S2))))

LINKS OF THE CPD
(PARALLEL-VOLTAGE-L1 USING-FUNC-LINK S1 (DELIVER-VOLTAGE-BEH))
(PARALLEL-VOLTAGE-L2 USING-FUNC-LINK NIL (PARALLEL-VOLTAGE-FUNC))
(PARALLEL-VOLTAGE-L3 USING-FUNC-LINK S2 (DELIVER-VOLTAGE-BEH))
(PARALLEL-VOLTAGE-L4 USING-FUNC-LINK NIL (PARALLEL-VOLTAGE-FUNC))

CONNECTIVITY, START-STATES, END-STATES
((PARALLEL-VOLTAGE-L1 (PARALLEL-VOLTAGE-S1)
      (PARALLEL-VOLTAGE-S2))
 (PARALLEL-VOLTAGE-L2 (PARALLEL-VOLTAGE-S2)
      (PARALLEL-VOLTAGE-S3))
 (PARALLEL-VOLTAGE-L3 (PARALLEL-VOLTAGE-S4)
      (PARALLEL-VOLTAGE-S5))
 (PARALLEL-VOLTAGE-L4 (PARALLEL-VOLTAGE-S5)
      (PARALLEL-VOLTAGE-S3)))
(PARALLEL-VOLTAGE-S1 PARALLEL-VOLTAGE-S4)
(PARALLEL-VOLTAGE-S3)

;;----

Component T2621 of class CLOSED-CIRCUIT

PORTS
NIL

BEH_CPD_NAME
((T KVL-BEH KVL-CPD NIL))

GIVEN OF BEH
T

TOMAKE OF BEH
(EQUAL (- (VOLTAGE P1 S) (VOLTAGE P2 S))
       (- (VOLTAGE P1 R) (VOLTAGE P2 R)))

INDEPENDENT VARS OF BEH
((VOLTAGE P1 S) (VOLTAGE P2 S))

DEPENDENT VARS OF BEH
```

```
((VOLTAGE P1 R) (VOLTAGE P2 R))

STATES OF CPD
(KVL-S1 T)
(KVL-S2
 (EQUAL (- (VOLTAGE P1 S) (VOLTAGE P2 S)) (- (VOLTAGE P1 R)
     (VOLTAGE P2 R))))

LINKS OF THE CPD
(KVL-L1 AS-PER-LINK KVL-LAW)

CONNECTIVITY, START-STATES, END-STATES
((KVL-L1 (KVL-S1) (KVL-S2)))
(KVL-S1)
(KVL-S2)

;;----

;;PRIMITIVE COMPONENTS

Component (SOLAR-ARRAY SA1)

PORTS
((ELECTRICAL NIL P1 #<PORT @ #x94108e>) (ELECTRICAL NIL P2
 #<PORT @ #x9410a6>))

BEH_CPD_NAME
(((VAR-VALUE SUN-SHINING-P) DELIVER-VOLTAGE-BEH
    DELIVER-VOLTAGE-CPD DEFAULT))

;;----

Component (SOLAR-ARRAY SA2)
...
;;----

Component (RECHARGEABLE-BATTERY B)

PORTS
((ELECTRICAL NIL P1 #<PORT @ #x941016>) (ELECTRICAL NIL P2
 #<PORT @ #x94102e>))

BEH_CPD_NAME
(((OR (SUBTYPEP (VAR-VALUE MATCHEDAS) 'ELOAD)
      (SUBTYPEP 'ELOAD (VAR-VALUE MATCHEDAS)))
  CONSUME-POWER-BEH CONSUME-POWER-CPD NIL)
 ((OR (SUBTYPEP (VAR-VALUE MATCHEDAS) 'ELOAD)
      (SUBTYPEP 'ELOAD (VAR-VALUE MATCHEDAS)))
```

```
  ACCUMULATE-CHARGE-BEH ACCUMULATE-CHARGE-CPD NIL))

;;----

Component (RESISTOR L)

PORTS
((ELECTRICAL NIL P1 #<PORT @ #x94105e>) (ELECTRICAL NIL P2
 #<PORT @ #x941076>))

BEH_CPD_NAME
(((VAR-VALUE POWER-STORY-P) CONSUME-POWER-BEH CONSUME-POWER-CPD
      DEFAULT))

---------------------------------------------------------
```

Appendix B

Execution trace of Example 2

Following is the execution trace for the structural description given in Figure 20(a). Comments, statements preceded by ";;", have been inserted to make it easier to understand the trace.

```
;; Top level goal is to analyze the device given in Figure 20(a).
 0: (ANALYZE-DEVICE #<TEMP-MEASURE @ #x960946>
                   (((TEMPERATURE P1)) ((VOLTAGE P2)) NIL))

;; Try fetch-match-sff-method
   1: (FETCH-COMP-METHOD #<TEMP-MEASURE @ #x960946>
                        (((TEMPERATURE P1)) ((VOLTAGE P2)) NIL)
     NIL)
     2: (FETCH-COMPONENTS #<TEMP-MEASURE @ #x960946>
                         (((TEMPERATURE P1))((VOLTAGE P2)) NIL))
     2: returned NIL
   1: returned NIL
;; fetch-match-sff-method failed

;; try consolidate-analyze-method
   1: (PAR-SER-COMB-METHOD #<TEMP-MEASURE @ #x960946>
                          (((TEMPERATURE P1)) ((VOLTAGE P2))
    NIL) NIL)
     2: (PARALLEL-COMP-DECOMPOSITION #<TEMP-MEASURE @ #x960946>)
     2: returned NIL
   1: returned NIL
;; consolidate-analyze-method failed

;; try partition-method
   1: (PARTITION-METHOD #<TEMP-MEASURE @ #x960946>
                        (((TEMPERATURE P1)) ((VOLTAGE P2)) NIL)
NIL)

;; partition the given sd
     2: (PARTITION #<TEMP-MEASURE @ #x960946>)
       3: (GET-PARTITIONED-COMPS #<TEMP-MEASURE @ #x960946>
```

```
         ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3) (JF1 JX4)
  (JX4 JF1)
          (JA3 JX5) (JX5 JA3) (JX6 JX7) (JX7 JX6)))
    3: returned (T5205 T5202 T5198 T5194 T5190 T5186)
  2: returned
       ((T5205 T5202 T5198 T5194 T5190 T5186)
        ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3) (JF1 JX4)
 (JX4 JF1)
         (JA3 JX5) (JX5 JA3) (JX6 JX7) (JX7 JX6)))

;; analyze the partitions.  This corresponds to analysis in
;; Figure 21.
  2: (ANALYZE-PARTS ((T5205 NIL) (T5202 NIL) (T5198 NIL)
    (T5194 NIL) (T5190 NIL) (T5186 NIL)))

;; analyze first partition corresponding to P
    3: (ANALYZE-DEVICE #<COMPONENT @ #xae077e> NIL NIL)
      4: (SINGLE-COMP-METHOD #<COMPONENT @ #xae077e> NIL NIL)
...
      4: returned (AD (AD))
    3: returned (AD (AD))

;; analyze second partition corresponding to Q
    3: (ANALYZE-DEVICE #<COMPONENT @ #xadf926> NIL NIL)
      4: (SINGLE-COMP-METHOD #<COMPONENT @ #xadf926> NIL NIL)
...
      4: returned (DR (DR))
    3: returned (DR (DR))

;; analyze second partition corresponding to R
    3: (ANALYZE-DEVICE #<COMPONENT @ #xadea16> NIL NIL)

;; try fetch-match-sff-method to analyze the partition
;; corresponding to R
      4: (FETCH-COMP-METHOD #<COMPONENT @ #xadea16> NIL NIL)

;; fetch and match components.
        5: (FETCH-COMPONENTS #<COMPONENT @ #xadea16> NIL)
        5: returned
             (((INTEGRATOR INTEGRATOR-I) 2)
 ((INVERTER INVERTER-I) 2)
              ((AMPLIFIER AMPLIFIER-I) 2)
              ((INSTRUMENTATION-AMP INSTRUMENTATION-AMP-I) 1)
              ((SWEEP-GENERATOR SWEEP-GENERATOR-I) 1))
        5: (INSTANCE-GENERIC-P #<INTEGRATOR @ #xa8092e>
                               #<COMPONENT @ #xadea16>)
        5: returned NIL
        5: (INSTANCE-GENERIC-P #<INVERTER @ #xa7fea6>
```

```
                                  #<COMPONENT @ #xadea16>)
          5: returned
               (((R1 RA1) (R2 RA2) (O1 OA1) (G1 GA1))
                (((JA1) (JX4)) ((JA2) (JA1)) ((JA3) (JA2))
  ((JA4) (JA3)))
                ((P1 T5196) (P2 T5197)) ((P1 JX4) (P2 JA3)))

;; verify and select functions.
          5: (SELECT-CPD #<INVERTER @ #xaedb66> NIL)
            6: (EVALUATE-COND (AND (VOLTAGE P1) (PARAM VT)
                                   (< (VOLTAGE P1) (PARAM VT)))
                              #<INVERTER @ #xaedb66>)
            6: returned ?
            6: (EVALUATE-COND (AND (VOLTAGE P1) (PARAM VT)
                                   (>= (VOLTAGE P1) (PARAM VT)))
                              #<INVERTER @ #xaedb66>)
            6: returned ?
          5: returned #<INVERTER @ #xaedb66>

;; extend-refine cpds
          5: (EXTEND-REFINE #<CPD @ #xa802b6> #<INVERTER @
     #xaedb66>)
          5: returned #<CPD @ #xa802b6>
          5: (UPDATE-STATES INVERTER-BEH INVERTER-CPD)
          5: returned INVERTER-CPD
        4: returned (T5211 (T5211))
      3: returned (T5211 (T5211))

;; analyze second partition corresponding to S
      3: (ANALYZE-DEVICE #<COMPONENT @ #xadd066> NIL NIL)
        4: (FETCH-COMP-METHOD #<COMPONENT @ #xadd066> NIL NIL)
...
        4: returned (T5216 (T5216))
      3: returned (T5216 (T5216))

;; analyze second partition corresponding to T
      3: (ANALYZE-DEVICE #<COMPONENT @ #xadbb96> NIL NIL)
        4: (FETCH-COMP-METHOD #<COMPONENT @ #xadbb96> NIL NIL)
...
        4: returned (T5219 (T5219))
      3: returned (T5219 (T5219))

;; analyze second partition corresponding to U
      3: (ANALYZE-DEVICE #<COMPONENT @ #xad738e> NIL NIL)
        4: (SINGLE-COMP-METHOD #<COMPONENT @ #xad738e> NIL NIL)
...
        4: returned (TC (TC))
      3: returned (TC (TC))
```

```
     2: returned
          (((T5205 AD) (T5202 DR) (T5198 T5211) (T5194 T5216)
    (T5190 T5219) (T5186 TC))
           (TC T5219 T5216 T5211 DR AD))
;; analyzed all the partitions

;; now try to compose subsets of analyzed partitions
     2: (COMPOSE-SUBSETS (AD DR T5211 T5216 T5219 TC)
                         ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
  (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
  (JX6 JX7) (JX7 JX6))
                         NIL)
       3: (GET-SUBSETS 6 (AD DR T5211 T5216 T5219 TC))
       3: returned ((AD DR T5211 T5216 T5219 TC))
       3: (SFF-MATCHES (AD DR T5211 T5216 T5219 TC)
                       ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
(JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
(JX6 JX7) (JX7 JX6)))
         4: (MAKE-NEW-COMP (AD DR T5211 T5216 T5219 TC)
                           ((JX2 JB1) (JB1 JX2) (JB3 JX3)
    (JX3 JB3) (JF1 JX4) (JX4 JF1)
                            (JA3 JX5) (JX5 JA3) (JX6 JX7)
    (JX7 JX6)))
         4: returned T5235
         4: (FETCH-COMPONENTS #<COMPONENT @ #xb0780e> NIL)
         4: returned NIL
       3: returned NIL
       3: (GET-SUBSETS 5 (AD DR T5211 T5216 T5219 TC))
       3: returned
           ((AD DR T5211 T5216 T5219) (AD DR T5211 T5216 TC)
            (AD DR T5211 T5219 TC) (AD DR T5216 T5219 TC)
            (AD T5211 T5216 T5219 TC) (DR T5211 T5216 T5219 TC))
       3: (SFF-MATCHES (AD DR T5211 T5216 T5219)
                       ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
(JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
        (JX6 JX7) (JX7 JX6)))
         4: (MAKE-NEW-COMP (AD DR T5211 T5216 T5219)
                           ((JX2 JB1) (JB1 JX2) (JB3 JX3)
    (JX3 JB3) (JF1 JX4) (JX4 JF1)
    (JA3 JX5) (JX5 JA3) (JX6 JX7)
    (JX7 JX6)))
         4: returned T5245
         4: (FETCH-COMPONENTS #<COMPONENT @ #xb0b1b6> NIL)
         4: returned (((COMPARATOR COMPARATOR-I) 1)
      ((ADDER ADDER-I) 1))
         4: (INSTANCE-GENERIC-P #<COMPARATOR @ #xa80d66>
                                #<COMPONENT @ #xb0b1b6>)
         4: returned NIL
```

```
          4: (INSTANCE-GENERIC-P #<ADDER @ #xa80b6e> #<COMPONENT @
         #xb0b1b6>)
          4: returned NIL
        3: returned NIL
...
        3: (GET-SUBSETS 4 (AD DR T5211 T5216 T5219 TC))
        3: returned
             ((AD DR T5211 T5216) (AD DR T5211 T5219)
      (AD DR T5211 TC) (AD DR T5216 T5219)(AD DR T5216 TC)
      (AD DR T5219 TC) (AD T5211 T5216 T5219)
      (AD T5211 T5216 TC) (AD T5211 T5219 TC)
              (AD T5216 T5219 TC) (DR T5211 T5216 T5219)
      (DR T5211 T5216 TC) (DR T5211 T5219 TC)
      (DR T5216 T5219 TC) (T5211 T5216 T5219 TC))
        3: (SFF-MATCHES (AD DR T5211 T5216)
                        ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
         (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
         (JX6 JX7) (JX7 JX6)))
...
        3: returned NIL
...
        3: (GET-SUBSETS 3 (AD DR T5211 T5216 T5219 TC))
        3: returned
             ((AD DR T5211) (AD DR T5216) (AD DR T5219) (AD DR TC)
              (AD T5211 T5216) (AD T5211 T5219) (AD T5211 TC)
      (AD T5216 T5219) (AD T5216 TC) (AD T5219 TC)
      (DR T5211 T5216) (DR T5211 T5219) (DR T5211 TC)
      (DR T5216 T5219) (DR T5216 TC) (DR T5219 TC)
              (T5211 T5216 T5219) (T5211 T5216 TC)
      (T5211 T5219 TC) (T5216 T5219 TC))
...
        3: (GET-SUBSETS 2 (AD DR T5211 T5216 T5219 TC))
        3: returned
             ((AD DR) (AD T5211) (AD T5216) (AD T5219) (AD TC)
      (DR T5211) (DR T5216) (DR T5219) (DR TC)
      (T5211 T5216) (T5211 T5219) (T5211 TC) (T5216 T5219)
      (T5216 TC) (T5219 TC))
        3: (SFF-MATCHES (AD DR)
                        ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
         (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
         (JX6 JX7) (JX7 JX6)))
        3: returned NIL
...
      2: returned NIL
;; No subset of components could be composed.

;; Step back one step and try to analyze the partitions again.
;; This corresponds to the analysis shown in Figure 22.
```

```
     2: (ANALYZE-PARTS ((T5205 AD) (T5202 DR) (T5198 T5211)
(T5194 T5216) (T5190 T5219) (T5186 TC)))

;; try to reanalyze the partition corresponding to part P
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa826ae> NIL AD)
         4: (SINGLE-COMP-METHOD #<COMPONENT @ #xa826ae> NIL AD)
         4: returned NIL
       3: returned NIL
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa826ae> NIL NIL)
         4: (SINGLE-COMP-METHOD #<COMPONENT @ #xa826ae> NIL NIL)
...
         4: returned (AD (AD))
       3: returned (AD (AD))

;; try to reanalyze the partition corresponding to part Q
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa826de> NIL DR)
         4: (SINGLE-COMP-METHOD #<COMPONENT @ #xa826de> NIL DR)
         4: returned NIL
       3: returned NIL
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa826de> NIL NIL)
         4: (SINGLE-COMP-METHOD #<COMPONENT @ #xa826de> NIL NIL)
...
         4: returned (DR (DR))
       3: returned (DR (DR))

;; try to reanalyze the partition corresponding to part R.
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa8270e> NIL T5211)
         4: (FETCH-COMP-METHOD #<COMPONENT @ #xa8270e> NIL T5211)
           5: (FETCH-COMPONENTS #<COMPONENT @ #xa8270e> NIL)
           5: returned
                (((INTEGRATOR INTEGRATOR-I) 2)
 ((INVERTER INVERTER-I) 2)
                 ((AMPLIFIER AMPLIFIER-I) 2)
                 ((INSTRUMENTATION-AMP INSTRUMENTATION-AMP-I) 1)
                 ((SWEEP-GENERATOR SWEEP-GENERATOR-I) 1))
           5: (INSTANCE-GENERIC-P #<AMPLIFIER @ #x8a261e>
                                  #<COMPONENT @ #xa8270e>)
...
           5: returned #<AMPLIFIER @ #xb0066e>
...
         4: returned (T5764 (T5764))
       3: returned (T5764 (T5764))
;; Got a new result for this partition.  Partition analyzed as
;; amplifier.  Return the new set of results for the partitions.
     2: returned
          (((T5205 AD) (T5202 DR) (T5198 T5764) (T5194 T5216)
    (T5190 T5219) (T5186 TC))
           (TC T5219 T5216 T5764 DR AD))
```

```
;; Try to compose subsets.
     2: (COMPOSE-SUBSETS (AD DR T5764 T5216 T5219 TC)
                         ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
  (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
  (JX6 JX7) (JX7 JX6))
                         NIL)
......
;; The following subset of components can be composed.
       3: (SFF-MATCHES (DR T5764 T5216 T5219)
                       ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
(JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
        (JX6 JX7) (JX7 JX6)))
         4: (MAKE-NEW-COMP (DR T5764 T5216 T5219)
                           ((JX2 JB1) (JB1 JX2) (JB3 JX3)
    (JX3 JB3) (JF1 JX4) (JX4 JF1)
    (JA3 JX5) (JX5 JA3) (JX6 JX7)
    (JX7 JX6)))
         4: returned T5959
...
       3: returned T5959
     2: returned ((DR T5764 T5216 T5219) T5959)
;; Returned the new component obtained by composing the subset
;; of comps.  Analyze the new component.
     2: (ANALYZE-DEVICE #<COMPONENT @ #x98d5f6> NIL NIL)

;; Try fetch-match-sff-method
       3: (FETCH-COMP-METHOD #<COMPONENT @ #x98d5f6> NIL NIL)
...
       3: returned (T5961 (T5961))
     2: returned (T5961 (T5961))
;; Analyzed the new component as an instance of sweep-generator.
;; Compose an instance of sweep generator with the remaining
;; partitions to get a new sd.
     2: (MAKE-NEW-COMP (T5961 TC AD)
                       ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
(JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
        (JX6 JX7) (JX7 JX6)))
     2: returned T5970

;; Analyze the new sd.
     2: (ANALYZE-DEVICE #<COMPONENT @ #x99886e> NIL NIL)
;; Try fetch-match-sff-method
       3: (FETCH-COMP-METHOD #<COMPONENT @ #x99886e> NIL NIL)
...
       3: returned NIL
;; Try consolidate-analyze-method
       3: (PAR-SER-COMB-METHOD #<COMPONENT @ #x99886e> NIL NIL)
```

```
...
       3: returned NIL
;; Try partition-method
       3: (PARTITION-METHOD #<COMPONENT @ #x99886e> NIL NIL)
...
       3: returned NIL
     2: returned NIL
;; could not analyze the new sd.

;; Once again step back one step and try to reanalyze the
;; component obtained by composing a subset of partitions.
     2: (ANALYZE-DEVICE #<COMPONENT @ #x98d5f6> NIL T5961)
...
     2: returned NIL
;; Could not get a different result for this component

;; Step back one more step.  Try to compose another subset of
;; partitions.
     2: (COMPOSE-SUBSETS (AD DR T5764 T5216 T5219 TC)
                         ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
  (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
  (JX6 JX7) (JX7 JX6))
                         (DR T5764 T5216 T5219))
       3: (GET-SUBSETS 4 (AD DR T5764 T5216 T5219 TC))
       3: returned
            ((AD DR T5764 T5216) (AD DR T5764 T5219)
     (AD DR T5764 TC) (AD DR T5216 T5219)
     (AD DR T5216 TC) (AD DR T5219 TC)
             (AD T5764 T5216 T5219) (AD T5764 T5216 TC)
     (AD T5764 T5219 TC) (AD T5216 T5219 TC)
     (DR T5764 T5216 T5219) (DR T5764 T5216 TC)
             (DR T5764 T5219 TC) (DR T5216 T5219 TC)
     (T5764 T5216 T5219 TC))
       3: (SFF-MATCHES (DR T5764 T5216 TC)
                       ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
(JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
        (JX6 JX7) (JX7 JX6)))
         4: (MAKE-NEW-COMP (DR T5764 T5216 TC)
                           ((JX2 JB1) (JB1 JX2) (JB3 JX3)
    (JX3 JB3) (JF1 JX4) (JX4 JF1)
    (JA3 JX5) (JX5 JA3) (JX6 JX7)
    (JX7 JX6)))
         4: returned NIL
       3: returned NIL
...
     2: returned NIL
;; Could not compose another subset of components.
```

```
;; Step back one more step.  Try to reanalyze the partitions.
;; This analysis corresponds to Figure 23.
     2: (ANALYZE-PARTS ((T5205 AD) (T5202 DR) (T5198 T5764)
         (T5194 T5216) (T5190 T5219) (T5186 TC)))
;; Try to reanalyze the partition for part P
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa7acb6> NIL AD)
...
       3: returned (AD (AD))
;; Try to reanalyze the partition for part Q
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa7ace6> NIL DR)
...
       3: returned (DR (DR))
;; Try to reanalyze the partition for part R
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa7ad16> NIL T5764)
...
       3: returned (T6324 (T6324))
;; Try to reanalyze the partition for part S
       3: (ANALYZE-DEVICE #<COMPONENT @ #xa7ad8e> NIL T5216)
         4: (FETCH-COMP-METHOD #<COMPONENT @ #xa7ad8e> NIL T5216)
...
         4: returned (T6329 (T6329))
       3: returned (T6329 (T6329))
;; Got a new result for partition S.  Analyzed it as lp-filter.

     2: returned
          (((T5205 AD) (T5202 DR) (T5198 T6324) (T5194 T6329)
    (T5190 T5219) (T5186 TC))
           (TC T5219 T6329 T6324 DR AD))
;; Try composing subsets of components.
     2: (COMPOSE-SUBSETS (AD DR T6324 T6329 T5219 TC)
                        ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
  (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
  (JX6 JX7) (JX7 JX6))
                        NIL)
....
     2: returned NIL
;; Could not compose any subset.

;; Step back one step.  Try to reanalyze the partitions again.
;; This analysis corresponds to the Figure 24.
     2: (ANALYZE-PARTS ((T5205 AD) (T5202 DR) (T5198 T6324)
         (T5194 T6329) (T5190 T5219) (T5186 TC)))

;; Try to reanalyze the partition P
       3: (ANALYZE-DEVICE #<COMPONENT @ #x8f6d16> NIL AD)
...
       3: returned (AD (AD))
```

```
;; Try to reanalyze the partition Q
       3: (ANALYZE-DEVICE #<COMPONENT @ #x8f6d36> NIL DR)
...
       3: returned (DR (DR))

;; Try to reanalyze the partition R
       3: (ANALYZE-DEVICE #<COMPONENT @ #x8f6d56> NIL T6324)
         4: (FETCH-COMP-METHOD #<COMPONENT @ #x8f6d56> NIL T6324)
         4: returned (T6870 (T6870))
       3: returned (T6870 (T6870))
;; Got a new result for part R.  Analyzed as amplifier.

     2: returned
          (((T5205 AD) (T5202 DR) (T5198 T6870) (T5194 T6329)
    (T5190 T5219) (T5186 TC))
           (TC T5219 T6329 T6870 DR AD))

;; Try to compose subset of components.
     2: (COMPOSE-SUBSETS (AD DR T6870 T6329 T5219 TC)
                         ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
  (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
  (JX6 JX7) (JX7 JX6))
                         NIL)
...
;; Composed the following subset of components.
       3: (SFF-MATCHES (DR T6870 T6329 T5219)
                       ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
(JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
        (JX6 JX7) (JX7 JX6)))
         4: (MAKE-NEW-COMP (DR T6870 T6329 T5219)
                           ((JX2 JB1) (JB1 JX2) (JB3 JX3)
    (JX3 JB3) (JF1 JX4) (JX4 JF1)
    (JA3 JX5) (JX5 JA3) (JX6 JX7)
    (JX7 JX6)))
         4: returned T7065
...
       3: returned T7065
     2: returned ((DR T6870 T6329 T5219) T7065)
;; Returned the new composed component.

;; analyze the new component.
     2: (ANALYZE-DEVICE #<COMPONENT @ #xae91b6> NIL NIL)

;; Try fetch-match-sff-method
       3: (FETCH-COMP-METHOD #<COMPONENT @ #xae91b6> NIL NIL)
...
       3: returned (T7067 (T7067))
     2: returned (T7067 (T7067))
```

```
;; analyzed the new component as an instance of an
;; instrumentation-amplifier.

;; Compose the instance of the instrumentation-amp with the
;;  remaining partitions.
     2: (MAKE-NEW-COMP (T7067 TC AD)
                       ((JX2 JB1) (JB1 JX2) (JB3 JX3) (JX3 JB3)
        (JF1 JX4) (JX4 JF1) (JA3 JX5) (JX5 JA3)
        (JX6 JX7) (JX7 JX6)))
     2: returned T7077

;; Analyze the new sd.
     2: (ANALYZE-DEVICE #<COMPONENT @ #x96a63e> NIL NIL)
       3: (FETCH-COMP-METHOD #<COMPONENT @ #x96a63e> NIL NIL)
...
       3: returned (T7079 (T7079))
     2: returned (T7079 (T7079))
;; Analyzed the new sd as an instance of a measuring-instrument.

   1: returned (T7079 (T7079 T7067 TC T5219 T6329 T6870 DR AD))
 0: returned (T7079 (T7079 T7067 TC T5219 T6329 T6870 DR AD))

;; Get rid of the unnecessary cpds.
 0: (CLEAN-UP-ALL-BEHS T7079)
 0: returned NIL
;; Completed analyzing the given structural description.

;;-----------------------------------------------------------------

;; Printing out the output.  The output consists of the
;; hierarchical structural descriptions generated and the
;; corresponding functions and cpds.

;; The hierarchical structural description.  Component T7079 is
;; the highest level component.

The top level component

T7079 of class MEASURING-INSTRUMENT
;Subcomponents
((TRANSDUCER T1) (INSTRUMENTATION-AMP IA1) (DISPLAY DS1))
;Connections
((JX1 (P1 (T1 P1))) (JX2 ((T1 P2) (IA1 P1))) (JX3 ((IA1 P2) (DS1 P1)))
 (JX4 ((DS1 P2) P2)))
(((T1 TC) (IA1 T7067) (DS1 AD)) ;instantiation
 (((JX1) (T7072)) ((JX2) (T7075)) ((JX3) (T7076)) ((JX4) (T7074)))
 ((P1 T7071) (P2 T7073)) ((P1 JX1) (P2 JX8)))
```

```
Next level

T7067 of class INSTRUMENTATION-AMP
;Subcomponents
((BUFFER B1) (LP-FILTER F1) (AMPLIFIER A1) (DRIVER D1))
;Connections
((JX1 (P1 (B1 P1)))(JX2 ((B1 P2) (F1 P1)))(JX3 ((F1 P2) (A1 P1)))
 (JX4 ((A1 P2) (D1 P1))) (JX5 ((D1 P2) P2)))
(((B1 T5219) (F1 T6329) (A1 T6870) (D1 DR)) ;Instantiation
 (((JX1) (T7060)) ((JX2) (T7059)) ((JX3) (T7058)) ((JX4) (T7062))
  ((JX5) (T7061)))
 ((P1 T7063) (P2 T7064)) ((P1 JB1) (P2 JX6)))


Next level

T5219 of class BUFFER
((OP-AMP OB1) (CAPACITOR CB1) (GROUND GB1)) ;Subcomponents
;Connections
((JB1 (P1 (OB1 P2) (CB1 P1))) (JB2 ((CB1 P2) (GB1 P1)))
 (JB3 ((OB1 P3) (OB1 P1) P2)))
;Instantiation
(((OB1 OB1) (CB1 CB1) (GB1 GB1))
 (((JB1) (JB1)) ((JB2) (JB2)) ((JB3) (JB3)))
 ((P1 T5188) (P2 T5189)) ((P1 JB1) (P2 JB3)))

T6329 of class LP-FILTER
((RESISTOR RF1) (CAPACITOR CF1) (GROUND GF1)) ;Subcomponents
;Connections
((JF1 (P1 (RF1 P1))) (JF2 (P2 (RF1 P2) (CF1 P1)))
 (JF3 ((CF1 P2) (GF1 P1))))
;Instantiation
(((RF1 RF1) (CF1 CF1) (GF1 GF1))
 (((JF1) (JX3)) ((JF2) (JF1)) ((JF3) (JF2)))
 ((P1 T5192) (P2 T5193)) ((P1 JX3) (P2 JF1)))

T6870 of class AMPLIFIER
;Subcomponents
((RESISTOR R1) (RESISTOR R2) (OP-AMP O1) (GROUND G1))
;Connections
((JA1 (P1 (R1 P1))) (JA2 ((R1 P2) (O1 P1) (R2 P1)))
 (JA3 ((G1 P1) (O1 P2)))
 (JA4 ((O1 P3) (R2 P2) P2)))
(((R1 RA1) (R2 RA2) (O1 OA1) (G1 GA1)) ;Instantiation
 (((JA1) (JX4)) ((JA2) (JA1)) ((JA3) (JA2)) ((JA4) (JA3)))
 ((P1 T5196) (P2 T5197)) ((P1 JX4) (P2 JA3)))
```

```
Next level

((THERMOCOUPLE TC) (OP-AMP OB1) (CAPACITOR CB1) (GROUND GB1)
 (RESISTOR RF1) (CAPACITOR CF1) (GROUND GF1) (OP-AMP OA1)
 (RESISTOR RA1) (RESISTOR RA2) (GROUND GA1) (DRIVER DR)
 (ANALOG-DISPLAY AD))
--------------------------------------------------------


;; Functions and cpds of components

Component T7079 of class MEASURING-INSTRUMENT

PORTS
((PORT INPUT P1 #<PORT @ #x96e45e>) (PORT OUTPUT P2 #<PORT @
 #x96e8ae>))

BEH_CPD_NAME
((T MEASURING-INSTRUMENT-BEH MEASURING-INSTRUMENT-CPD DEFAULT))

GIVEN OF BEH
T

TOMAKE OF BEH
(AND (AND (EQUAL (DEFLECTION P2) (* (PARAM K) (VOLTAGE P1)))))

INDEPENDENT VARS OF BEH
NIL

DEPENDENT VARS OF BEH
((DEFLECTION P2))

STATES OF CPD
(MEASURING-INST-S1 T)
(MEASURING-INST-S2 (AND (EQUAL (VOLTAGE P2)
       (* (PARAM K) (TEMPERATURE P1)))))
(MEASURING-INST-S3 (EQUAL (VOLTAGE P2 IA1)
  (* K (VOLTAGE P1 IA1))))
(MEASURING-INST-S4 (AND (EQUAL (DEFLECTION P2)
       (* (PARAM K) (VOLTAGE P1)))))

LINKS OF THE CPD
(MEASURING-INST-L1 USING-FUNC-LINK T1 (TEMP-VOLTAGE-BEH))
(MEASURING-INST-L2 USING-FUNC-LINK IA1 (INSTRUMENTATION-AMP-BEH))
(MEASURING-INST-L3 USING-FUNC-LINK DS1 (ANALOG-DISPLAY-BEH))

CONNECTIVITY, START-STATES, END-STATES
```

```
((MEASURING-INST-L1 (MEASURING-INST-S1) (MEASURING-INST-S2))
 (MEASURING-INST-L2 (MEASURING-INST-S2) (MEASURING-INST-S3))
 (MEASURING-INST-L3 (MEASURING-INST-S3) (MEASURING-INST-S4)))
(MEASURING-INST-S1)
(MEASURING-INST-S4)

;;----

Component T7067 of class INSTRUMENTATION-AMP

PORTS
((ELECTRICAL INPUT P1 #<ELECTRICAL @ #x955366>)
 (ELECTRICAL OUTPUT P2 #<ELECTRICAL @ #x95537e>))

BEH-CPD-NAME
((T INSTRUMENTATION-AMP-BEH INSTRUMENTATION-AMP-CPD DEFAULT)
 ((> (VOLTAGE P1) (PARAM TV)) CLIPPED-IA-BEH CLIPPED-IA-CPD NIL))
NIL
(MEASURING-INSTRUMENT-CPD CLIPPED-MI-CPD)

GIVEN OF BEH
T

TOMAKE OF BEH
(EQUAL (VOLTAGE P2) (* (PARAM K) (VOLTAGE P1)))

INDEPENDENT VARS OF BEH
((VOLTAGE P1) (PARAM K))

DEPENDENT VARS OF BEH
((VOLTAGE P2))

STATES OF CPD
(INSTRUMENTATION-AMP-S1 T)
(INSTRUMENTATION-AMP-S2 (EQUAL (VOLTAGE P2 B1)
      (* (PARAM K B1) (VOLTAGE P1))))
(INSTRUMENTATION-AMP-S3
 (EQUAL (VOLTAGE P2 F1)
        (* (PARAM K F1) (/ (VOLTAGE P1 F1) (FREQUENCY P1 F1)))))
(INSTRUMENTATION-AMP-S4
 (EQUAL (VOLTAGE P2 A1) (* (PARAM K A1) (VOLTAGE P1 A1))))
(INSTRUMENTATION-AMP-S5
 (EQUAL (VOLTAGE P2 D1) (* (PARAM K D1) (VOLTAGE P1 D1))))

LINKS OF THE CPD
(INSTRUMENTATION-AMP-L1 USING-FUNC-LINK B1 (BUFFER-BEH))
(INSTRUMENTATION-AMP-L2 USING-FUNC-LINK F1 (LP-FILTER-BEH))
(INSTRUMENTATION-AMP-L3 USING-FUNC-LINK A1 (AMPLIFIER-BEH))
```

```
(INSTRUMENTATION-AMP-L4 USING-FUNC-LINK D1 (DRIVER-BEH))

CONNECTIVITY, START-STATES, END-STATES
((INSTRUMENTATION-AMP-L1 (INSTRUMENTATION-AMP-S1)
 (INSTRUMENTATION-AMP-S2))
 (INSTRUMENTATION-AMP-L2 (INSTRUMENTATION-AMP-S2)
 (INSTRUMENTATION-AMP-S3))
 (INSTRUMENTATION-AMP-L3 (INSTRUMENTATION-AMP-S3)
 (INSTRUMENTATION-AMP-S4))
 (INSTRUMENTATION-AMP-L4 (INSTRUMENTATION-AMP-S4)
 (INSTRUMENTATION-AMP-S5)))
(INSTRUMENTATION-AMP-S1)
(INSTRUMENTATION-AMP-S5)

;;----


Components T5219 of class BUFFER

PORTS
((ELECTRICAL INPUT P1 #<ELECTRICAL @ #x8f9306>)
 (ELECTRICAL OUTPUT P2 #<ELECTRICAL @ #x8f9316>))

BEH-CPD-NAME
((T BUFFER-BEH BUFFER-CPD DEFAULT)
 ((> (VOLTAGE P1) (PARAM IT)) CLIPPED-BUFFER-BEH
  CLIPPED-BUFFER-CPD NIL))
NIL
(INSTRUMENTATION-AMP-CPD CLIPPED-IA-CPD)

GIVEN OF BEH
T

TOMAKE OF BEH
(EQUAL (VOLTAGE P2) (VOLTAGE P1))

INDEPENDENT VARS OF BEH
((VOLTAGE P1))

DEPENDENT VARS OF BEH
((VOLTAGE P2))

STATES OF CPD
(BUFFER-S1 T)
(BUFFER-S2 (EQUAL (VOLTAGE P1 OB1) (VOLTAGE P1)))
(BUFFER-S3 (EQUAL (VOLTAGE P2) (VOLTAGE P1)))

LINKS OF THE CPD
```

```
(BUFFER-L1 AS-PER-LINK OP-AMP-L2)
(BUFFER-L2 AS-PER-LINK SAME-NODE)

CONNECTIVITY, START-STATES, END-STATES
((BUFFER-L1 (BUFFER-S1) (BUFFER-S2)) (BUFFER-L2 (BUFFER-S2)
 (BUFFER-S3)))
(BUFFER-S1)
(BUFFER-S3)

;;----


Component T6329 of class LP-FILTER

PORTS
((ELECTRICAL INPUT P1 #<ELECTRICAL @ #x955306>)
 (ELECTRICAL OUTPUT P2 #<ELECTRICAL @ #x95531e>))

BEH-CPD-NAME
((T LP-FILTER-BEH LP-FILTER-CPD DEFAULT))
NIL
(INSTRUMENTATION-AMP-CPD CLIPPED-IA-CPD)

GIVEN OF BEH
T

TOMAKE OF BEH
(EQUAL (VOLTAGE P2)
       (/ (VOLTAGE P1) (+ 1 (* (PARAM R R1) (PARAM C C1)
        (FREQUENCY P1)))))

INDEPENDENT VARS OF BEH
((VOLTAGE P1) (FREQUENCY P1) (PARAM R R1) (PARAM C C1))

DEPENDENT VARS OF BEH
((VOLTAGE P2))

STATES OF CPD
(LP-FILTER-S1 T)
(LP-FILTER-S2
 (EQUAL (VOLTAGE P2) (+ 1 (* (PARAM R R1) (PARAM C C1)
 (FREQUENCY P1)))))

LINKS OF THE CPD
(LP-FILTER-L1 AS-PER-LINK OHMS-LAW)

CONNECTIVITY, START-STATES, END-STATES
((LP-FILTER-L1 (LP-FILTER-S1) (LP-FILTER-S2)))
```

```
(LP-FILTER-S1)
(LP-FILTER-S2)

;;----


Components T6870 of class AMPLIFIER

PORTS
((ELECTRICAL INPUT P1 #<ELECTRICAL @ #x9553c6>)
 (ELECTRICAL OUTPUT P2 #<ELECTRICAL @ #x9553de>))

BEH-CPD-NAME
(((AND (VOLTAGE P1) (PARAM VT) (< (VOLTAGE P1) (PARAM VT)))
  AMPLIFIER-BEH AMPLIFIER-CPD DEFAULT))
NIL
(INSTRUMENTATION-AMP-CPD CLIPPED-IA-CPD)

GIVEN OF BEH
T

TOMAKE OF BEH
(> (VOLTAGE P2) (* -1 (VOLTAGE P1)))

INDEPENDENT VARS OF BEH
((VOLTAGE P1) (PARAM X))

DEPENDENT VARS OF BEH
((VOLTAGE P2))

STATES OF CPD
(AMPLIFIER-S1 T)
(AMPLIFIER-S2 (EQUAL (CURRENT P1 O1) 0))
(AMPLIFIER-S3 (EQUAL (VOLTAGE P1 O1) (VOLTAGE P2 O1)))
(AMPLIFIER-S4
 (AND (EQUAL (CURRENT P1 R2) (/ (VOLTAGE P1) (PARAM R R1)))
      (EQUAL (VOLTAGE P2) (* -1 (CURRENT P1 R2) (PARAM R R2)))))
(AMPLIFIER-S5
 (EQUAL (VOLTAGE P2) (/ (* -1 (VOLTAGE P1) (PARAM R R2))
         (PARAM R R1))))

LINKS OF THE CPD
(AMPLIFIER-L1 AS-PER-LINK OP-AMP-L1)
(AMPLIFIER-L2 AS-PER-LINK OP-AMP-L2)
(AMPLIFIER-L3 AS-PER-LINK OHMS-PLUS-DEDUCTION)
(AMPLIFIER-L4 AS-PER-LINK DEDUCTION)

CONNECTIVITY, START-STATES, END-STATES
```

```
((AMPLIFIER-L1 (AMPLIFIER-S1) (AMPLIFIER-S2))
 (AMPLIFIER-L2 (AMPLIFIER-S1) (AMPLIFIER-S3))
 (AMPLIFIER-L3 (AMPLIFIER-S2 AMPLIFIER-S3) (AMPLIFIER-S4))
 (AMPLIFIER-L4 (AMPLIFIER-S4) (AMPLIFIER-S5)))
(AMPLIFIER-S1)
(AMPLIFIER-S5)

;;----

;; Primitive components

Component DR of class DRIVER

......

;;----------------------------------------------------------------
```

Appendix C

Execution trace of Example 2 with the added rule

Following is the execution trace for the structural description given in Figure 20(a). Comments, statements preceded by ";;", have been inserted to make it easier to understand the trace.

```
;; Top level goal is to analyze the device given in Figure 20(a).
 0: (ANALYZE-DEVICE #<TEMP-MEASURE @ #xa908a6>
                    (((TEMPERATURE P1)) ((VOLTAGE P2)) NIL))

;; Try fetch-match-sff-method
   1: (FETCH-COMP-METHOD #<TEMP-MEASURE @ #xa908a6>
                         (((TEMPERATURE P1)) ((VOLTAGE P2)) NIL)
 NIL)
...
   1: returned NIL

;; Try consolidate-analyze-method
   1: (PAR-SER-COMB-METHOD #<TEMP-MEASURE @ #xa908a6>
                           (((TEMPERATURE P1)) ((VOLTAGE P2))
     NIL) NIL)
...
   1: returned NIL

;; Try special method for measuring-instrument device
;;  Try to isolate the transducer part
   1: (GET-SINGLE-EDGE #<SD @ #xa8ef7e>)
   1: returned ((JX2 JB1))
   1: (GET-PARTITIONED-COMPS #<TEMP-MEASURE @ #xa908a6>
     ((JX2 JB1)))
   1: returned (T7177 T7174)
;; Separated the transducer part

;; Analyze the transducer part
   1: (ANALYZE-DEVICE #<COMPONENT @ #xaa074e> NIL)
     2: (SINGLE-COMP-METHOD #<COMPONENT @ #xaa074e> NIL NIL)
...
```

```
    2: returned (TC (TC))
  1: returned (TC (TC))

;; Try to separate the display part.  Done by repeatedly
;; partitioning the part with the output port.
  1: (GET-SINGLE-EDGE #<SD @ #xaa43f6>)
  1: returned ((JB3 JX3))
  1: (GET-PARTITIONED-COMPS #<COMPONENT @ #xaa42f6> ((JB3 JX3)))
  1: returned (T7185 T7182)
  1: (GET-SINGLE-EDGE #<SD @ #xaac5e6>)
    2: (PARTITION-POSSIBLE-P #<SD @ #xaac5e6> ((JF1 JX4)))
    2: returned T
  1: returned ((JF1 JX4))
  1: (GET-PARTITIONED-COMPS #<COMPONENT @ #xaac4e6> ((JF1 JX4)))
  1: returned (T7191 T7188)
  1: (GET-SINGLE-EDGE #<SD @ #xab0bde>)
    2: (PARTITION-POSSIBLE-P #<SD @ #xab0bde> ((JA3 JX5)))
    2: returned T
  1: returned ((JA3 JX5))
  1: (GET-PARTITIONED-COMPS #<COMPONENT @ #xab0ade> ((JA3 JX5)))
  1: returned (T7197 T7194)
  1: (GET-SINGLE-EDGE #<SD @ #xab46a6>)
    2: (PARTITION-POSSIBLE-P #<SD @ #xab46a6> ((JX6 JX7)))
    2: returned T
  1: returned ((JX6 JX7))
  1: (GET-PARTITIONED-COMPS #<COMPONENT @ #xab45a6> ((JX6 JX7)))
  1: returned (T7203 T7200)
  1: (GET-SINGLE-EDGE #<SD @ #xab70fe>)
  1: returned NIL
  1: (GET-PARTITIONED-COMPS #<COMPONENT @ #xaa42f6> ((JX6 JX7)))
  1: returned (T7209 T7206)
;; Got the display part

;; Analyze the display part.
  1: (ANALYZE-DEVICE #<COMPONENT @ #xabbc46> NIL)
    2: (SINGLE-COMP-METHOD #<COMPONENT @ #xabbc46> NIL NIL)
...
    2: returned (AD (AD))
  1: returned (AD (AD))

;; Try to analyze the rest of the device as an instrumentation
;; amp.
  1: (ANALYZE-DEVICE #<COMPONENT @ #xabae1e> NIL NIL)

;; Try fetch-match-sff-method
    2: (FETCH-COMP-METHOD #<COMPONENT @ #xabae1e> NIL NIL)
...
    2: returned NIL
```

```
;; Try consolidate-analyze-method
     2: (PAR-SER-COMB-METHOD #<COMPONENT @ #xabae1e> NIL NIL)
...
     2: returned NIL

;; Try partition-method
     2: (PARTITION-METHOD #<COMPONENT @ #xabae1e> NIL NIL)
       3: (PARTITION #<COMPONENT @ #xabae1e>)
...
       3: returned
            ((T7225 T7222 T7218 T7214)
             ((JB3 JX3) (JX3 JB3) (JF1 JX4) (JX4 JF1) (JA3 JX5)
      (JX5 JA3)))
;; Got the partitions

;; Analyze the partitions
       3: (ANALYZE-PARTS ((T7225 NIL) (T7222 NIL) (T7218 NIL)
  (T7214 NIL)))
         4: (ANALYZE-DEVICE #<COMPONENT @ #xac5d8e> NIL NIL)
           5: (SINGLE-COMP-METHOD #<COMPONENT @ #xac5d8e>
  NIL NIL)
...
           5: returned (DR (DR))
         4: returned (DR (DR))
         4: (ANALYZE-DEVICE #<COMPONENT @ #xac4f36> NIL NIL)
           5: (FETCH-COMP-METHOD #<COMPONENT @ #xac4f36> NIL NIL)
...
           5: returned (T7227 (T7227))
         4: returned (T7227 (T7227))
;; Analyzed as amplifier.

         4: (ANALYZE-DEVICE #<COMPONENT @ #xac3586> NIL NIL)
           5: (FETCH-COMP-METHOD #<COMPONENT @ #xac3586> NIL NIL)
...
           5: returned (T7231 (T7231))
         4: returned (T7231 (T7231))
;; analyzed as lp-filter

         4: (ANALYZE-DEVICE #<COMPONENT @ #xac20e6> NIL NIL)
           5: (FETCH-COMP-METHOD #<COMPONENT @ #xac20e6> NIL NIL)
...
           5: returned (T7234 (T7234))
         4: returned (T7234 (T7234))
;; Analyzed as buffer

       3: returned
            (((T7225 DR) (T7222 T7227) (T7218 T7231)
```

```
      (T7214 T7234)) (T7234 T7231 T7227 DR))

;; Compose subsets of analyzed partitions
       3: (COMPOSE-SUBSETS (DR T7227 T7231 T7234)
                           ((JB3 JX3) (JX3 JB3) (JF1 JX4)
    (JX4 JF1) (JA3 JX5) (JX5 JA3))
                           NIL)
         4: (GET-SUBSETS 4 (DR T7227 T7231 T7234))
         4: returned ((DR T7227 T7231 T7234))
         4: (SFF-MATCHES (DR T7227 T7231 T7234)
                         ((JB3 JX3) (JX3 JB3) (JF1 JX4) (JX4 JF1)
  (JA3 JX5) (JX5 JA3)))
           5: (MAKE-NEW-COMP (DR T7227 T7231 T7234)
                             ((JB3 JX3) (JX3 JB3) (JF1 JX4)
      (JX4 JF1) (JA3 JX5) (JX5 JA3)))
           5: returned T7244
...
         4: returned T7244
       3: returned ((DR T7227 T7231 T7234) T7244)
;; Compose all the partitions to get a new sd.

;; Analyze the new sd.
       3: (ANALYZE-DEVICE #<COMPONENT @ #xad948e> NIL NIL)
         4: (FETCH-COMP-METHOD #<COMPONENT @ #xad948e> NIL NIL)
...
         4: returned (T7246 (T7246))
       3: returned (T7246 (T7246))
;; Analyzed the new sd as an instance of an instrumentation-amp.

     2: returned (T7246 (T7246 T7234 T7231 T7227 DR))
   1: returned (T7246 (T7246 T7234 T7231 T7227 DR))

;; Compose the instance of instrumentation-amp with the
;; remaining sd.
   1: (MAKE-NEW-COMP (T7246 TC AD) ((JX2 JB1) (JX6 JX7)))
   1: returned T7255

;; Analyze the new sd.
   1: (ANALYZE-DEVICE #<COMPONENT @ #xae2726> NIL NIL)
     2: (FETCH-COMP-METHOD #<COMPONENT @ #xae2726> NIL NIL)
...
     2: returned (T7257 (T7257))
   1: returned (T7257 (T7257))
;; Analyzed the new sd as an instance of measuring-device.

 0: returned (T7257 (T7257 T7246 T7234 T7231 T7227 DR TC AD))

;; Remove redundant functions/cpds of components based on the
```

```
;; selected functions of the containing device.
 0: (CLEAN-UP-ALL-BEHS T7257)
 0: returned NIL

;; Completed analyzing the given structural description.
```

Bibliography

[ACP91] Sanjay Addanki, Roberto Cremonini, and J. S. Penberthy. Graphs of models. *Artificial Intelligence*, 51(1-3):145–177, October 1991.

[All90] D. Allemang. *Understanding Programs as Devices.* PhD thesis, The Ohio State University, 1990.

[BC85] T. Bylander and B. Chandrasekaran. Understanding behavior using consolidation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 450–454, 1985.

[CDR91] Steve Chien, Richard Doyle, and Nicolas Rouquette. Sensor placement for diagnosability in space-borne systems: A model-based reasoning approach. In *Proceedings of the second International workshop on the principles of diagnosis*, Milan, Italy, Oct 1991.

[CGI93] B. Chandrasekaran, Ashok Goel, and Yumi Iwasaki. Functional representation as design rationale. *IEEE Computer, Special issue on concurrent engineering*, pages 48–56, Jan 1993.

[Cha90] B. Chandrasekaran. Design problem solving: A task analysis. *AAAI magazine*, Winter:59–71, 1990.

[Cha94] B. Chandrasekaran. Functional representation and causal processes. *Advances in computers*, 38, 1994.

[CR90] Bei-Tseng Bill Chu and James Reggia. Modeling diagnostic problem-solving at multiple levels of abstraction. In *Proceedings of the conference on Artificial Intelligence Applications*, pages 55–61. IEEE, IEEE press, 1990.

[Dav84] Randall Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[dK85] J. de Kleer. How circuits work. In D. G. Bobrow, editor, *Qualitative Reasoning About Physical Systems.* MIT Press, 1985.

[dKB84] J. de Kleer and J. S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.

[dKW87] J. de Kleer and Brian Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[dKW89] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *International Joint Conference on Artificial Intelligence*, pages 1324–1330, Detroit, MI, August 1989.

[Doy86] Richard Doyle. Constructing and refining causal explanations from an incosistent domain theory. In *Proceedings of fifth National Conference on Artificial Intelligence*, 1986.

[DRD87] P. Dague, O. Raiman, and P. Denes. Troubleshooting when modeling is the trouble. In *Proceedings of the 6th National conferenc on AI*, Seatle, WA, Aug 87. AAAI.

[FF88] Brian Falkenhainer and Kenneth D. Forbus. Setting up large scale qualitative models. In *Proceedings of Seventh National Conference on Artificial Intelligence*, 1988.

[FF91a] B. Falkaenhainer and K. D. Forbus. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51(1-3):95–143, Oct 1991.

[FF91b] Hartmut Freitag and Gerhard Friedrich. Focusing on independent diagnosis problems. In *Proceedings of the workshop on Principles of diagnosis*, 1991.

[Fis88] P. A. Fishwick. The role of process abstraction in simulation. *IEEE Trnasaction on Systems, Man, and Cybernetics*, 18(1):18–39, 1988.

[For84] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[Fra89] D. W. Franke. Representing and acquiring teleological descriptions. In *Proceedings of the Workshop on Model-Based Reasoning*, 1989.

[Fri90] Gerhard Friedrich. Momo - model-based diagnosis for everyone. In *Proceedings of the conference on Artificial Intelligence applications*, pages 206–213. IEEE, IEEE press, 1990.

[Goe90] A. Goel. *Integration of Case-Based Reasoning and Adaptive Design Problem Solving.* PhD thesis, The Ohio State University, 1990.

[Ham90a] Walter Hamscher. Modeling digital circuits for troubleshooting: An overview. In *Proceedings of the IEEE conference on Artificial Intelligence Applications*, pages 2–8, Santa Barbara, CA, March 1990. IEEE, IEEE Computer Society Press.

[Ham90b] Wlater Hamscher. Xde: Diagnosing devices with hierarchic structure and known component failure modes. In *Proceedings of the IEEE conference on Artificial Intelligence Applications*, pages 48–54, Santa Barbara, CA, March 1990. IEEE, IEEE Computer Society Press.

[IB88] Y. Iwasaki and I. Bhandari. Formal basis for commonsense abstraction of dynamic systems. In *National Conference on Artificial Intelligence*, pages 307–312, 1988.

[IC92] Yumi Iwasaki and B. Chandrasekaran. Design verification through function- and behavior-oriented representations: Bridging the gap between function and behavior. In *Proceedings of AI in Design Conference*, pages 597–616, 1992.

[IFVC93] Y. Iwasaki, R. Fikes, M. Vescovi, and B. Chandrasekaran. How things are intended to work: Capturing functional knowledge in device design. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1516–1522. Morgan Kaufmann, 1993.

[IS86] Yumi Iwasaki and Herbert A. Simon. Causality in device behavior. *Artificial Intelligence*, 29, 1986.

[JJ94] John Josephson and Susan Josephson, editors. *Abductive Inference: Computation, Philosophy, Technology.* Cambridge University Press, 1994. forthcoming.

[JS85] W. Lewis Johnson and Elliot Soloway. Intention-based diagnosis of programming errors. *AAAI Magazine*, pages 162–168, 1985.

[KC87] Benjamin Kuipers and Charles Chiu. Taming intractable branching in qualitative simulation. In *Proceedings of the IJCAI*, pages 1079–1085, 1987.

[Keu89] A. Keuneke. *Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions.* PhD thesis, The Ohio State University, 1989.

[Kui86a] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–388, 1986.

[Kui86b] Benjamin Kuipers. Qualitative simulation as causal explanation. Technical Report AI-TR86-24, University of Texas at Austin, AI Lab, 1986.

[Kui87] B. Kuipers. Abstraction by time scale in qualitative simulation. In *Proceedings of AAAI-87*, 1987.

[LF90] Zheng-Yang Liu and Arthur M. Farley. Shifting ontological perspective in reasoning about physical systems. In *Proceedings of Ninth National conference on Artificial Intelligence*, pages 395–400. AAAI, 1990.

[LK88] Wood W Lee and Benjamin Kuipers. Non-intersection of trajectories in qualitative phase space: a global constraint for qualitative simulation. In *Proceedings of AAAI*, pages 286–290. AAAI, 1988.

[Mos85] Jack Mostow. Towards a better model of the design process. *AAAI Magazine*, 6.1, 1985.

[NJA91] P. Pandurang Nayak, Leo Jaskowicz, and Sanjay Addanki. Automated model selection using context-dependent behaviors. Technical Report RC 16906 (#74798), IBM, IBM Research division, T. J. Watson Research Center, Yorktown Heights, NY 10598, May 1991.

[Sac87] Elisha Sacks. Hierarchical reasoning about inequalities. In *Proceedings of the AAAI*, pages 649–654. AAAI, 1987.

[SC88] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Riesbeck, editors, *Experience, Memory, and Reasoning*, pages 47–73. Lawrence Erlbaum Associates, 1988.

[SD89] P. Struss and O. Dressler. Physical negation: Integrating fault models into the general diagnostic engine. In *International Joint Conference on Artificial Intelligence*, pages 1318–1323, Detroit, MI, Aug 1989.

[Sim91] H. Simon. Nonmonotonic reasoning and causation: comment. *Cognitive Science*, 15(2):293–300, 1991.

[SN87] J. Siekmann and Rolf T. Nossum, editors. *Lecture Notes in AI, 2nd Advanced course by ACAI*, chapter Qualitative Reasoning, pages 60–95. Springer-Verlag, 1987.

[Sti90] J. Sticklen. Paper in model based reasoning workshop (???). In *Proceedings of The Model Based Reasoning Workshop*, 1990.

[Str88] Peter Struss. Global filters for qualitative behaviors. In *Proceedings of the AAAI*, pages 275–279. AAAI, 1988.

[TC93] Sunil Thadani and B. Chandrasekaran. Structure-to-function reasoning. In *AAAI Workshop on Reasoning about function*, Washington, D.C., July 1993.

[Tha93] Sunil Thadani. Extending causal sequences to make teleological distinctions. In *Proceedings of The NinthConference on Artificial Intelligence for Applications*, Orlando, Florida, March 1993. IEEE, IEEE Computer Society Press.

[VIFC93] M. Vescovi, Y. Iwasaki, R. Fikes, and B. Chandrasekaran. Cfrl: A language for specifying the causal functionality of engineered devices. In *Proceedings of the Eleventh National Conference on AI*, pages 626–633. AAAI, July 1993.

[Wei91] M. Weintraub. *An Explanation-based Approach to Assigning Credit.* PhD thesis, The Ohio State University, 1991.

[Wel86] D. S. Weld. The use of aggregation in causal simulation. *Artificial Intelligence*, 30, 1986.

[Wil84] Brian C. Williams. Qualitative analysis of mos circuits. *Artificial Intelligence*, 24, 1984.

[Wil88] Brian C. Williams. Minima: a symbolic approach to qualitative algebraic reasoning. In *National Conference on Artificial Intelligence*, pages 264–269, 1988.