ICOT Technical Report: TR-0791

TR-0791

Analog and Digital Treatments for Quantities Based on Qualitative Reasoning

by
M. Ohki, E. Oohira, H. Shinjo
& M. Abe (Hitashi)

August, 1992

© 1992, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
(03)3456-3191～5
Telex ICOT J32964
Minato-ku Tokyo 108 Japan

Institute for New Generation Computer Technology

Analog and Digital Treatments for Quantities Based on Qualitative Reasoning

Masaru Ohki, Eiji Oohira, Hiroshi Shinjo, and Masahiro Abe

Central Research Laboratory, Hitachi, Ltd.,
Higashi-Koigakubo,
Kokubunji, Tokyo 185, Japan

Abstract

Expert systems using experimental knowledge are widely used, but they cannot solve unexpected problems and it is difficult to acquire experimental knowledge from human experts. One approach to solving the problem is to use qualitative reasoning that uses deep knowledge. Qualitative reasoning can estimate the value of quantities that change continuously during dynamic behaviors, but the cost of executing qualitative reasoning for large scale systems becomes huge.

One approach to solving this problem is to use qualitative reasoning for the only parts that qualitative reasoning is necessary. In this paper, we try to use qualitative reasoning for the parts that deal with analog treatments and to use event reasoning for the parts that deal with digital treatments. Event reasoning is usually faster than qualitative reasoning because it does not evaluate continuous changes of quantities. In qualitative reasoning, on the other hand, even discontinuous changes are dealt with as continuous changes. Here we apply this approach to circuits consisting of a DTL circuit and two d-flipflops. The two d-flipflops are regarded as digital circuits and, as such, their internal structures are ignored. The DTL circuit is dealt with as an analog circuit consisting of a transistor and some resistors and diodes.

keyword

Qualitative reasoning, Knowledge representation, Deep knowledge, Electronic circuit, Design support

1. Introduction

Expert systems are widely used, but because most of them use only experimental knowledge [5], they cannot solve unexpected problems. They lack the fundamental knowledge needed to analyze problems. In addition, it is difficult to acquire consistent and comprehensive experimental knowledge from human experts, because this kind of knowledge tends not to be systematic.

The use of deep knowledge is one approach overcoming these shortcomings, and qualitative reasoning is one of the reasoning mechanisms that use deep knowledge [1), 6~8), 14), 17)]. This kind of reasoning allows quantities to be dealt with qualitatively and determines dynamic behaviors, which are the states and state changes of a dynamic system, by using deep knowledge about the system. The use of qualitative reasoning helps expert systems "understand" the principle behind the problem. But, in qualitative reasoning, the costs of calculating all the physical quantities in analyzing the behaviors of large scale systems - such as very large electronic circuits - would be huge.

One way to solve this problem is to use simpler or rougher model so that fewer physical quantities need be calculated. Another way is to use qualitative reasoning for the only parts that qualitative reasoning is necessary, and to use other kind of reasoning, which is faster than qualitative reasoning, for the rest parts. In the electronic circuits including digital behaviors, for example, both analog and digital treatments may become necessary. Where the voltages and currents of the components are dealt with, analog treatment of the physical quantities should be used. And where the digital level of signals in the components are dealt with and these levels change discontinuously, digital treatments should be used instead. There are several methods available for the digital treatment, the method like STRIPS and temporal logic, and digital treatment based on symbolic calculation is generally faster than analog treatment based on qualitative reasoning. This is because qualitative reasoning treats discontinuous changes as continuous changes.

In this paper, we show a way of using both analog and digital treatments. We use Desq (Design support system based on qualitative reasoning) [12), 15)] based on an earlier qualitative reasoning system Qupras (Qualitative physical reasoning system) [9~11)]. In Desq, the qualitative reasoning performs the analog treatment, and an event reasoning performs the digital treatment. The main difference between the two treatments is that the changes of quantities in the analog treatment are continuous but the changes of quantities in the digital treatment are allowed to be discontinuous.

Desq is a prototype of a design support system for determining the ranges of design parameters. When Desq is given the structure of an electronic circuit with indeterminate design parameters and a specification of how the circuit should behave, Desq suggests the ranges of the design parameters by using deep knowledge that consists of Ohm's law and the basic function of such components as diodes and transistors. In this paper we will not use Desq for design support, but we will use its qualitative reasoning and event reasoning. When we want to use Desq to determine some design parameters for large circuits and we are able to separate the circuits to two parts, one dealt with as analog circuits and the other dealt with as digital circuits, we should use the method described in this paper.

Desq uses knowledge of physical rules and objects for its qualitative reasoning. When given the initial state, which consists of the connections of components and the initial values of attributes for the components, Desq reasons the following:

(1) Relations between objects that are components of physical systems.

(2) The state transitions of the system.

The event reasoning of Desq, which is also similar to that of Qupras, uses definitions of events and reasons the discontinuous changes caused by events. The definition of events is similar to the one used by STRIPS. It has a condition part and an action part. The condition part specifies the conditions that trigger the event, and the action part specifies the actions of the event. It is possible to add and remove expressions as action.

Desq's (or Qupras'es) event reasoning is similar to Forbus'es work [3], but Forbus deals with operators for actions as action hypotheses. Desq deal with events like physics or objects. If the conditions of an event definition are satisfied, the action is executed. Events are usually used to change the initial relations (which are the initial values) and the relations described by terms (relations that are not inequalities) defined in the initial state and added by the actions of other events.

In Desq, the event reasoning takes precedence over the qualitative reasoning. Whenever any definitions of events are triggered, the event reasoning is executed to modify the initial relations first, and then the qualitative reasoning is executed to determine the values of other analog quantities by using the modified initial relations.

One point to consider when using both analog and digital treatments is how to connect them. We use the event reasoning for these connections. For example, if the voltage of one analog terminal that is connected to the digital terminal increases and becomes 1.4 volts, it is interpreted in the digital treatment as though the level of the digital terminal changes from the "off" level to the "on" level discontinuously. We add one feature to the event reasoning. It refers to previous values of physical quantities and is used to detect the changes of levels in the condition parts of the definitions for events and physical rules.

Section 2 shows the organization of Desq and gives an example of its use in suggesting ranges for a design parameter for a DTL circuit, Section 3 describes the way that analog and digital treatments are connected by using an enhanced feature, Section 4 shows examples of the connection between DTL and d-flipflop circuits, and Section 5 summarizes this paper.

2. Desq

2.1 Qualitative Reasoning and Event reasoning of Desq

Desq, which uses knowledge from physics and engineering textbooks, has the following characteristics:

(1) Desq has three primitive representations: physical rules (laws of physics), objects, and events.

(2) Desq determines the dynamic behaviors of a system by using knowledge of physical rules, objects, and events to build all the equations for the system. The user need not enter these equations.

(3) Desq deals with equations that describe relations between quantities both qualitatively and quantitatively.

(4) Desq does not require quantity spaces to be given in advance. It finds the quantity spaces by itself during reasoning.

(5) Objects in Desq can inherit definitions from their superobjects. Physical rules and events can therefore be defined generally by using superobjects to specify the definitions of object classes.

(6) Desq can determine the ranges of design parameters that are indeterminate in an initial state

The qualitative reasoning of Desq is similar to QPT [2] but does not use influence. The representations describing relations of values in Desq are only equations. The laws of physics given in physics textbooks and engineering textbooks are usually described not by using influences but by using equations. Desq therefore uses only equations.

The representation of objects mainly consists of existential conditions and relations. Existential conditions are the conditions needed for the objects to exist. Objects satisfying these conditions are called active objects. The relations are expressed as relative equations that include physical quantities. If existential conditions are satisfied, their relations become known as relative equations that hold for physical quantities of the objects specified in the physical rule definition.

The representation of physical rules mainly consists of objects, applied conditions, and relations. The objects are those necessary to apply a physical rule. The representations of applied conditions and relations are similar to those of objects. Applied conditions are those required to activate a physical rule, and relations correspond to the laws of physics. Physical rules whose necessary objects are activated and whose conditions are satisfied are called active physical rules. If a given physical rule is active, its relations become known as in the case of objects.

The representation of events is similar to that of physical rules, and it consists of objects, applied conditions, and actions. If the objects are active and applied conditions are satisfied, the actions are executed. "Add" actions and "remove" actions are described in the actions part. The "add" action adds a specified relation to the set of the initial relations, and the "remove" action removes a specified relation from the set.

Qualitative reasoning in Desq involves two forms of reasoning: propagation reasoning and prediction reasoning. Propagation reasoning determines the state of the physical system at a given moment (or during a given time interval). Prediction reasoning determines the physical quantities that change with time, and it predicts their values at the next given time. Moreover, the propagation reasoning determines the subsequent states of the physical system by using the results from the prediction reasoning.

Before the reasoning, all initial relations defined in the initial state are set as known relations, which are used to evaluate the conditions of objects, physical rules, and events. The initial relations are used mainly to set the initial values of the physical quantities. If there is no explicit change to an initial relation, the initial relation is held. Explicit changes are the prediction of the next value in the prediction reasoning and the modification of the values in the event reasoning.

Propagation reasoning finds active objects and physical rules whose conditions are satisfied by the known relations. If a contradiction is detected after passing relations of the active objects and physical rules to the constraint solver in Desq, the propagation reasoning is stopped. If any condition of physical rules and objects cannot be evaluated, the reasoning process is split by the envisioning mechanism into two process: one process hypothesizing that the condition is satisfied and the other hypothesizing that it is not.

Prediction reasoning first uses the known relations resulting from the propagation reasoning to find the physical quantities changing with time. Then it searches for the new values or the new intervals of the changing quantities at the next specified time or during the next time interval. Desq updates the quantities according to the sought values or intervals in the same way that Qupras does. The updated values are used as the initial relations at the beginning of the next propagation reasoning.

The event reasoning in Desq is similar to prediction reasoning. The conditions of events and initial events are checked after propagation reasoning. If the conditions of the definitions for events are satisfied, the actions in the definitions are performed. After the process for the event finished, propagation reasoning starts with the result of the event. It is assumed that actions of events take precedence over changing quantities. If there are any activated events, the process for the prediction reasoning is not performed. The equations describing the initial relations are discontinuously changed by event reasoning, and then propagation reasoning is executed with the modified quantities.

2.2 System organization

Desq mainly consists of two subsystems:

(1) Behavior reasoner

Performs qualitative reasoning and event reasoning.

(2) Parallel constraint solver (Consort)

Solves simultaneous nonlinear inequalities.

Figure 1 shows how Desq is organized. If there are indeterminate design parameters in the initial state, Desq reasons the behavior of the system and determines the ranges of the design parameters. If there are no design parameters, Desq only reasons the behavior.

When Desq gets an initial state, the inequalities in the initial state are passed to Consort and are used to check the conditions of the physical rules, objects, and events. If the conditions of a physical rule or an object are satisfied, the inequalities in its consequences are added to the simultaneous inequalities in Consort. If the conditions of an event are satisfied, its actions are executed.

Consort [13] is written in KL1 and can be executed on the parallel inference machines supported by ICOT, which are PIM, Multi-PSI, or Pseudo Multi-PSI. It is used to test whether the conditions written in the definitions of physical rules, objects, and events are proved by the known relations obtained from active objects, active physical rules, and from initial relations. And if the conditions of active objects and active physical rules are satisfied, the constraint solver receives those relations and checks their consistency.

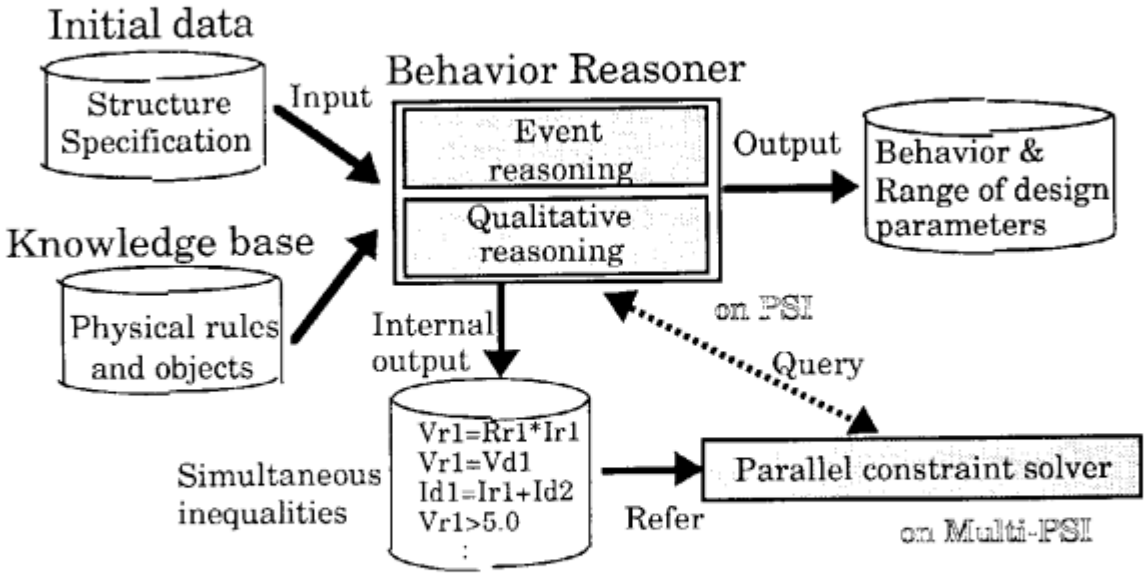


Fig. 1 System organization.

2.3 Design support by Desq

A designer often only changes or improves an old device, and sometimes a designer need only change the parameters of components in a device so as to satisfy the new requirements. In such cases, the designer already knows the structure of the device and needs only to determine the new values of the components. Desq supports these design decisions as follows:

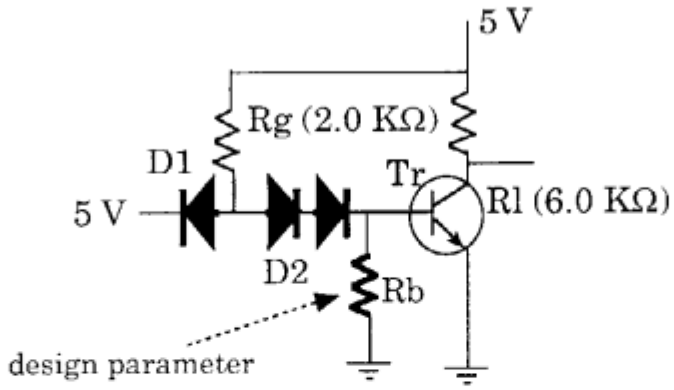
(1) All possible behaviors are found by envisioning with design parameters whose values are initially indeterminate and with specifications if a designer has clear specifications.

(2) If several behaviors are found in step (1), the designer selects preferable behaviors.

If a condition in the definitions of a physical rule or an object cannot be evaluated, Desq hypothesizes one case where the condition is valid and another where it is not valid and then separately searches each case to find all possible behaviors. This method is called envisioning, and is the same as the method described in Ref. 4. If a contradiction is detected, the reasoning is abandoned. If no contradiction is detected, the reasoning is valid. Finally Desq finds several possible behaviors of a device. The specification a designer gives is used to prune behaviors that the designer definitely does not wish.

2.3.1 Example of a DTL circuit

This is an example of determining a design parameter for a DTL circuit. The structure of the DTL circuit is shown in Figure 2a. The design parameter is the resistance value of the resistor Rb. In this example, the values of the other resistors are given, but the input voltage is indeterminate.



(a) Structure of a DTL circuit.

```
initial_state  dtl
  objects
     Rl-resistor ;
     Rg-resistor ;
     Rb-resistor ;
     Tr-transistor ;
     D1-diode ;
     D2-diode2 ;
  initial_relations
     connect(t1!Rl,t1!Rg) ;
     connect(t2!Rg,t1!D1,t1!D2) ;
     connect(t2!D3,t1!Rb,tb!Tr) ;
     connect(t2!Rl,tc!Tr) ;
     resistance@Rl=6000.0 ;
     resistance@Rg=2000.0 ;
     resistance@Rb >= 0.0;
     v@t1!Rl = 5.0 ;
     v@t2!D1 >= 0.0 ;
     v@t2!D1 =< 10.0 ;
     v@tc!Tr = 0.0 ;
     v@t2!Rb = 0.0 ;
end.
```

(b) Initial state for DTL.

Fig. 2 A DTL circuit and its initial state.

```
object  terminal:Terminal
  attributes
    v ;
    i ;
    state_transition - variable(initial_value(0.0));
end.

object  two_terminal_device:TTD
  parts_of
    t1-terminal ;
    t2-terminal ;
end.

object  diode:Di
  supers
    two_terminal_device;
  attributes
    v ;
    i ;
    resistance-constant ;
  initial_relations
    v@Di=v@t1!Di-v@t2!Di ;
  state  on
    conditions
      v@Di >= 0.7 ;
    relations
      v@Di= 0.7 ;
      i@Di >= 0.0 ;
  state  off
    condition
      v@Di < 0.7 ;
    relations
      resistance@Di=100000.0 ;
      v@Di=resistance@Di*i@Di ;
end.
```

Fig. 3 Definition of a diode.

The initial data is shown in Figure 2b. The "objects" field specifies the components in the DTL circuit and their classes. The first line in the "objects" field specifies that Rl is an instance of the class "resistor". The "initial_relations" field specifies the relations holding in the initial state. For example, "connect(t2!Rg, t1!D1, t1!D2)" specifies that the terminal t2 of the resistor Rg, the terminal t1 of the diode D1, and the terminal t1 of the diode D2 are all connected to each other. The "!" is a symbol specifying a part. The "t2!Rg" expresses the terminal t2 which is part of Rg. Rg is specified as a resistor in the "objects" definition. The "@" indicates a parameter, "resistance@Rl" represents the resistance value of Rl, and "resistance@Rl = 6000.0" specifies that Rl is 6000.0 ohms. The resistor Rb is constrained to be positive, and the input voltage, which is the voltage of the terminal t2 in the diode D1, is constrained to be between 0.0 and

10.0 volts. The values of Rb and the input voltage are both indeterminate, and Rb is a design parameter.

Figure 3 shows the definition of a diode. Its superobject is a two_terminal_device, so the diode inherits the properties of two_terminal_devices, i.e., it has two parts, both of which are terminals. Each terminal has three attributes: "v" for voltage, "i" for current, and "state_transition". "state_transition" is used to connect the analog and digital treatments, and it will be explained in more detail in the next section. The diode has an initial relation that specifies the voltage difference between its terminals. The diode also has two states: the "on" state, where the voltage difference is greater than 0.7 volts; and the "off" state, where the voltage difference is less than 0.7 volts. If the diode is in the "on" state, it behaves like a conductor, and if it is in the "off" state, it behaves like a resistor. A transistor is defined like a diode, but it has three states: "off", "on", and "saturated".

Figure 4 shows the definition of a physical rule. This rule shows Kirchhoff's law when the terminals t1 of three two_terminal_devices are connected. It is assumed that the current into the t1 of a two_terminal_device flows to the terminal t2. Three two_terminal_devices can in fact be connected in eight ways depending on how the terminals are connected.

```
physics three_connect_1
  objects
    TTD1 - two_terminal_device ;
    TTD2 - two_terminal_device ;
    TTD3 - two_terminal_device ;
    T1-terminal partname t1 part_of TTD1 ;
    T2-terminal partname t1 part_of TTD2 ;
    T3-terminal partname t1 part_of TTD3 ;
  conditions
    connect(T1,T2,T3);
  relations
    v@T1 = v@T2 ;
    v@T2 = v@T3 ;
    i@T1 + i@T2 + i@T3 = 0.0 ;
end.
```

Fig. 4 Definition of a physical rule.

Figure 5 shows the relationship between the resistance Rb and the input voltage. A designer can decide by looking at Figure 5 what the value of resistor Rb must be for the DTL circuit to behave as a NOT circuit: Rb must be greater than about 0.5 kilo ohms and less than about 100 kilo ohms, so that the DTL circuit can output a low voltage (nearly 0 volts) when the input is greater than about 1.5 volts

and output a high voltage (nearly 5 volts) when the input is less than about 1.5 volts. The range of this design parameter is shown by the area enclosed by the dotted lines.

We determined design parameters for a DTL circuit and a Schmitt trigger circuit, but we predicted that we would need a very long time to determine design parameters for larger circuits. We therefore need to reduce the time-cost of determining design parameters for large circuits.

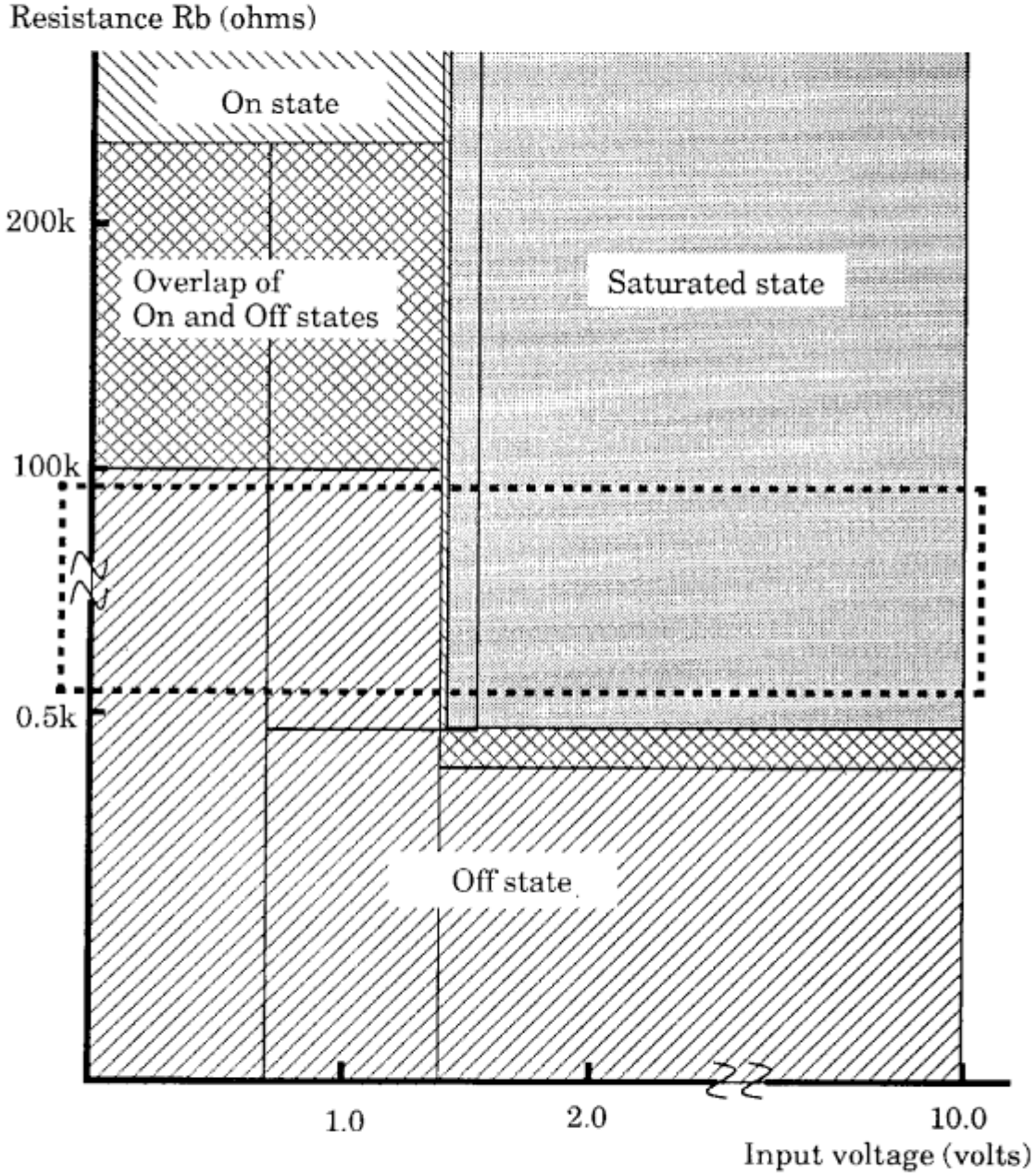


Fig. 5 Relationship between Resistance and Input voltage.

3. Connection between Analog and Digital Treatments for Quantities

3.1 Enhanced feature

In digital treatments, there are several cases in which triggers (which change from the high level to the low level or vice versa) are important. To describe knowledge about a trigger, such as if a trigger happens then some actions occur, it is necessary to refer the values of physical quantities at a previous time.

The "previous" function is used to refer to the value of a physical quantities at a previous time - which may be an interval or an instant. A trigger is described as a physical quantity whose value changes discontinuously. For example, if the previous value of a physical quantity is 0 and its current value is 1, that quantity is considered as a trigger. Figure 6a is an example using the "previous" function. It correctly describes that if a trigger happens at a digital terminal - that is, the level changes from low to high - the action part is executed. Figure 6b is an incorrect definition. It describes that if the trigger of a terminal is 1, its action part is executed. The event is triggered whenever the level is 1 - regardless of the previous level. The conditions of the incorrect definition do not specify that the level changes from the low level to the high level. The conditions of Figure 6a, on the other hand, describe a change as a trigger.

```
event trigger_on
  objects
    DT - digital_terminal ;
  conditions
    level@DT=1.0 ;
    prev(level@DT) =0.0 ;
  actions
    :
    ;
end.
```

(a) Correct definition.

```
event trigger_on
  objects
    DT - digital_terminal ;
  conditions
    level@DT=1.0 ;
  actions
    :
    :
end.
```

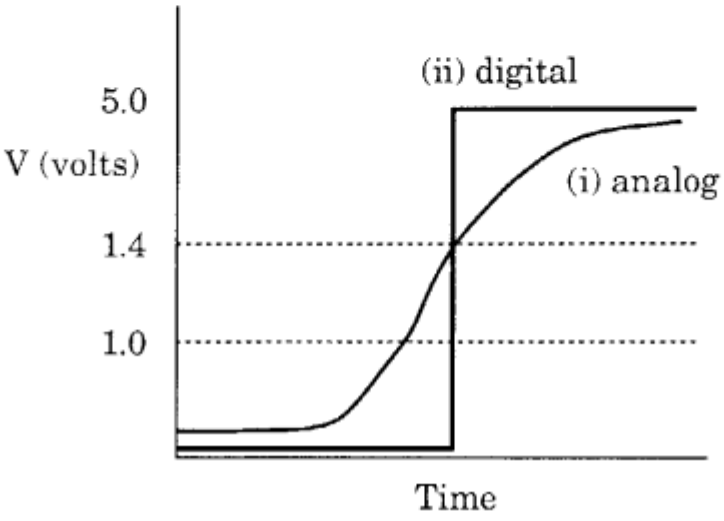
(b) Incorrect definition.

Fig. 6 Example of the "previous" function.

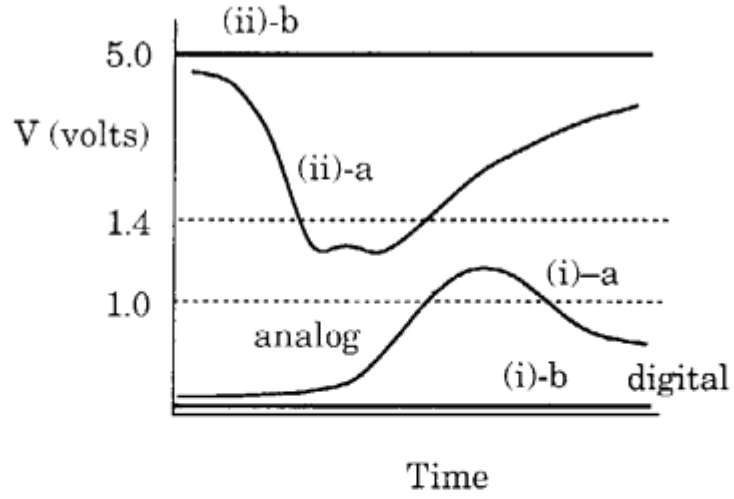
3.2 Connection of analog and digital treatments

3.2.1 Connection from analog treatments to digital treatments

The connection from analog treatments to digital treatments happens in a status where information transfers from analog equipment to digital equipment. Here the difference between analog equipments and digital equipments may be due to subjectivity: if one consider a piece of equipment as a digital equipment and if one wish to deal with it as a digital equipment, it is dealt with as digital equipment.



(a) Digital level change.



(b) No digital level change.

Fig. 7 Digital change induced by analog change.

```
event analog_to_digital_on_start
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(T,DT) ;
    state_transition@T = 0.0 ;
    v@T = 1.0 ;
    ddt(v@T) > 0.0 ;
  actions
    remove(state_transition@T = 0.0) ;
    add(state_transition@T = 1.0) ;
    add(intermediate_level@DT = 0.0) ;
end.

event analog_to_digital_on_end
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(T,DT) ;
    state_transition@T = 1.0 ;
    v@T = 1.4 ;
    intermediate_level@DT = 0.0 ;
  actions
    remove(state_transition@T = 1.0) ;
    add(state_transition@T = 0.0) ;
    remove(intermediate_level@DT = 0.0) ;
end.

event analog_to_digital_on_reset
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(T,DT) ;
    state_transition@DT = 1.0 ;
    v@T < 1.0 ;
    intermediate_level@DT = 0.0 ;
  actions
    remove(state_transition@DT = 1.0) ;
    add(state_transition@DT = 0.0) ;
    remove(intermediate_level@DT = 0.0) ;
end.
```

Fig. 8 Definition of digital change induced by analog change.

```
physics analog_to_digital_stay_off
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(T,DT) ;
    v@T =< 1.0 ;
  relations
    level@DT=0.0 ;
end.

physics analog_to_digital_stay_on
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(T,DT) ;
    v@T >= 1.4 ;
  relations
    level@DT=1.0 ;
end.

physics analog_to_digital_intermediate
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(T,DT) ;
    v@T < 1.4 ;
    v@T > 1.0 ;
  relations
    level@DT=intermediate_level@DT ;
end.
```

Fig. 9 Definition of digital unchange.

As an example, we consider a status in which analog equipment and digital equipment connect and informations transfers from the analog equipment to the digital equipment. Such a typical status is a case in which analog circuits connect with digital circuits. The curve (i) in Figure 7a, for example, shows an analog change of an increasing voltage. If the change is regarded as a digital change, and its threshold is 1.4 volts, then the change is described by the curve (ii) in Figure 7b. If the voltage is lower than 1.4 volts, it is digitally regarded as 0, which means that the voltage is 0.0 volts. If the voltage is higher than 1.4 volts, it is digitally regarded as 1, which means that the voltage is 5.0 volts. Because an analog voltage may change dynamically, we specify that the change of the digital

voltage from 0 to 1 happens only when the analog voltage goes up to 1.0 volts and also goes up to 1.4 volts. A change of an analog voltage described by the curve (i)-a in Figure 7b is regarded as the digital change described by the curve (i)-b because the analog voltage goes up to 1.0 volts but does not go up to 1.4 volts. We also specify that the change of the digital voltage from 1 to 0 happens only when the analog voltage goes down to 1.4 volts and also goes down to 1.0 volts. A change of an analog voltage described by the curve (ii)-a in Figure 7b is regarded as the digital change described by the curve (ii)-b because the analog voltage does not go down to 1.0 volts but does go down to 1.4 volts.

To connect analog treatments and digital treatments, we use the definitions of event and physical rules shown in Figures 8 and 9. They describe that if the voltage of the terminal increases from below 1.0 volts, and if the voltage becomes 1.4 volts, then the level of the digital terminal that the terminal is connected with is changed from low(0.0) to high(1.0).

The event definition of analog_to_digital_on_start in Figure 8 specifies that if DT is a digital terminal and T is an analog terminal, if the two terminals connect, if the voltage of the analog terminal is 1.0 volts, if its derivative is positive, and if the state of the terminal is not a transition state (specified as state_transition = 0.0), then the state of the terminal is set to a transition state (state_transition changes from 0.0 to 1.0) and the level of the intermediate state is set to 0.0 (that is, low). The value of the intermediate_level is used to set the level of the digital terminal in the physical rule analog_to_digital_intermediate in Figure 9.

The second event definition in Figure 8, analog_to_digital_on_end, specifies that if a digital terminal and a terminal connect, if the state of the terminal is a transition state, if the voltage of the terminal is 1.4 volts, and if the value of the intermediate_level is 0.0, then the state of the terminal is set to be no transition state (state_transition is changed from 1.0 to 0.0) and the equation that the level of the intermediate_level is set to 0.0 is removed. In the condition part of the event definition, the conditions that the state of the terminal is a transition state and that the value of the intermediate_level is 0.0 specify that the event definition analog_to_digital_on_start had been fired.

The last event definition, analog_to_digital_on_reset, is used to reset the state of the terminal and the intermediate_level after the event analog_to_digital_on_start had been triggered, because the voltage of the terminal became lower than 1.0 volts. This event definition is for the status indicated by the curve (i)-a in Figure 7b. The status is that though the voltage exceeds 1.0 volts, it does not exceed 1.4 volts and it becomes lower than 1.0 volts.

The three event definitions are for the status in which the voltage increases. For the status in which the voltage of the terminal decreases, another three event definitions similar to those in Figure 8 are necessary. The description of these definitions are omitted.

The three definitions of physical rules in Figure 9 are used to set the level of the digital terminal. The first physical rule, analog_to_digital_off, specifies that if a digital terminal and a terminal connect, and if the voltage of the terminal is lower than 1.0 volts, then the level of the digital terminal is 0.0. The second physical rule, analog_to_digital_on, is for the status in which the voltage is greater than 1.4 volts. The last physical rule, analog_to_digital_intermediate, is for the transition state of the terminal. In that rule, the level of the digital terminal is set

to the value of the intermediate_level that is set by the event definition analog_to_digital_on_start.

3.2.2 Connection from digital treatments to analog treatments

The connection from digital treatments to analog treatments happens in a status where informations transfers from digital equipments to analog equipments. This connection is the opposite of the connection from analog treatments to digital treatments.

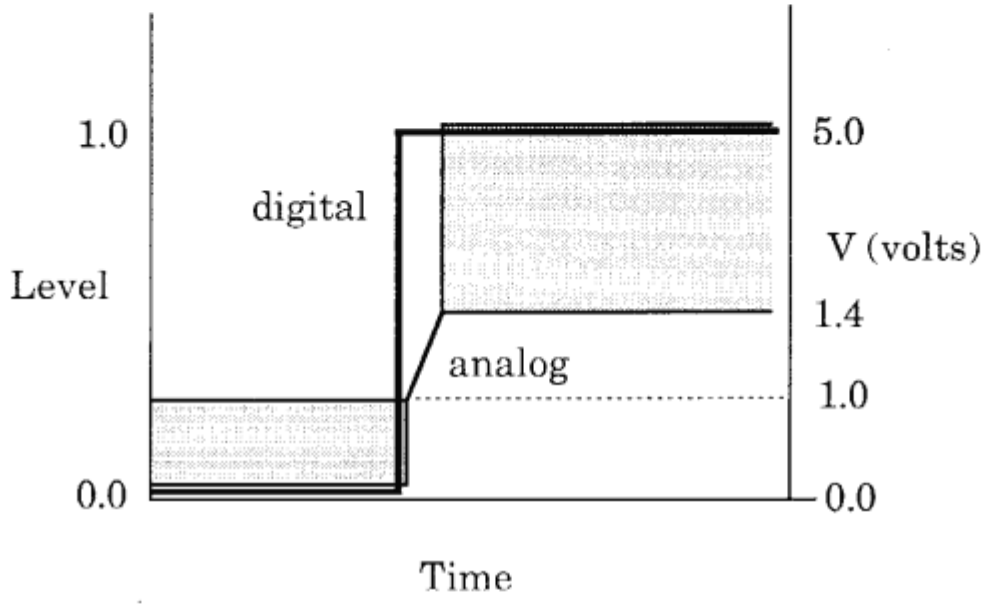


Fig. 10 Analog change induced by digital change.

A status in which a digital circuit and an analog circuit connect and a digital change in a digital circuit causes an analog change in an analog circuit is a typical example. In general, a digital change happens in an instant, but an analog change is a continuous change. A digital change happening in a digital circuit must therefore be translated to an analog change in an analog circuit. One example of this is shown in Figure 10. If the level of the digital terminal is 0.0, then the voltage of the analog terminal is less than 1.0 volts. If the level of the digital terminal is 1.0, that the voltage is greater than 1.4 volts.

We use two definitions of events in Figure 11 and two definitions of physical rules in Figure 12. The event definitions specify that the voltage of an analog terminal changes from less than 1.0 volts to greater than 1.4 volts because the level of the digital terminal changes from 0.0 to 1.0. The physical rule definitions specify the relations between the level of the digital terminal and the voltage of the analog terminal.

The first event definition in Figure 11 specifies that if a digital terminal and an analog terminal connect, if the state of the terminal is not a transition state, if the current level of the digital terminal is 1.0, and if the previous level is 0.0, then the voltage of the terminal is less than 1.0 volts, its derivative is positive, and the state

```
event digital_to_analog_on_start
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(DT,T) ;
    state_transition@T = 0.0 ;
    level@DT=1.0 ;
    prev(level@DT)=0.0 ;
  actions
    add(v@T=<1.0) ;
    add(ddt(v@T)>0.0) ;
    remove(state_transition@T = 0.0) ;
    add(state_transition@T = 1.0) ;
end.

event digital_to_analog_on_end
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(DT,T) ;
    state_transition@T = 1.0 ;
    level@DT=1.0 ;
    v@T = 5.0 ;
  actions
    remove(ddt(v@T)>0.0) ;
    remove(state_transition@T = 1.0) ;
    add(state_transition@T = 0.0) ;
end.
```

Fig. 11 Definition of analog change induced by digital change.

of the terminal changes to a transition state. The second event definition specifies that if a digital terminal and an analog terminal connect, if the state of the terminal is a transition state, if the current level of the digital terminal is 1.0, and if the voltage of the terminal is equal to 5.0 volts, then the inequality specifying that the derivative of the terminal is positive is removed and the state of the terminal changes to a no transition state.

The first physical rule in Figure 12, digital_to_analog_high, specifies that if a digital terminal and an analog terminal connect, if the state of the terminal is not a transition state, and if the current and previous levels of the digital terminal are 1.0 (that is, the level of the digital terminal stays 1.0), then the voltage of the terminal is greater than 1.4 volts. The second physical rule, digital_to_analog_high_to_low, specifies that if a digital terminal and an analog terminal connect, if the state of the terminal is not a transition state, if the current

```
physics digital_to_analog_high
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(DT,T) ;
    state_transition@DT = 0.0 ;
    level@DT = 1.0 ;
    prev(level@DT) = 1.0 ;
  relations
    v@T >= 1.4 ;
end.

physics digital_to_analog_high_to_low
  objects
    DT - digital_terminal ;
    T - terminal ;
  conditions
    connect(DT,T) ;
    state_transition@DT = 0.0 ;
    level@DT = 0.0 ;
    prev(level@DT) = 1.0 ;
  relations
    v@T >= 1.4 ;
end.
```

Fig. 12 Definition of analog unchange.

level of the digital terminal is 0.0, and if the previous level is 1.0, then the voltage of the terminal is greater than 1.4 volts. This rule is for the instant in which the level has changed from 0.0 to 1.0 but the event, digital_to_analog_on_start, has not yet been triggered.

4. Examples

We show two examples of connections: one of a connection from digital treatment to analog treatment, and the other of a connection from analog treatment to digital treatment.

4.1 Connection from analog treatment to digital treatment

This example shown in Figure 13 consists of a DTL circuit, a wire, and two d-flipflops. The DTL circuit and the wire are dealt with as analog circuits, and the DTL circuit consists of resistors, diodes, and a transistor. The wire is a component for connecting terminals, and it has two terminals, t1 and t2, as parts. The ranges of values for the voltages and currents used in the DTL circuit and the wire are analog quantities, that is, from the minus infinite to the infinite. The two d-flipflops, however, are dealt with as a digital circuit, and the ranges of values for the signals used in the digital circuit are 0 and 1. In the definitions, 0 and 1 are described 0.0 and 1.0.

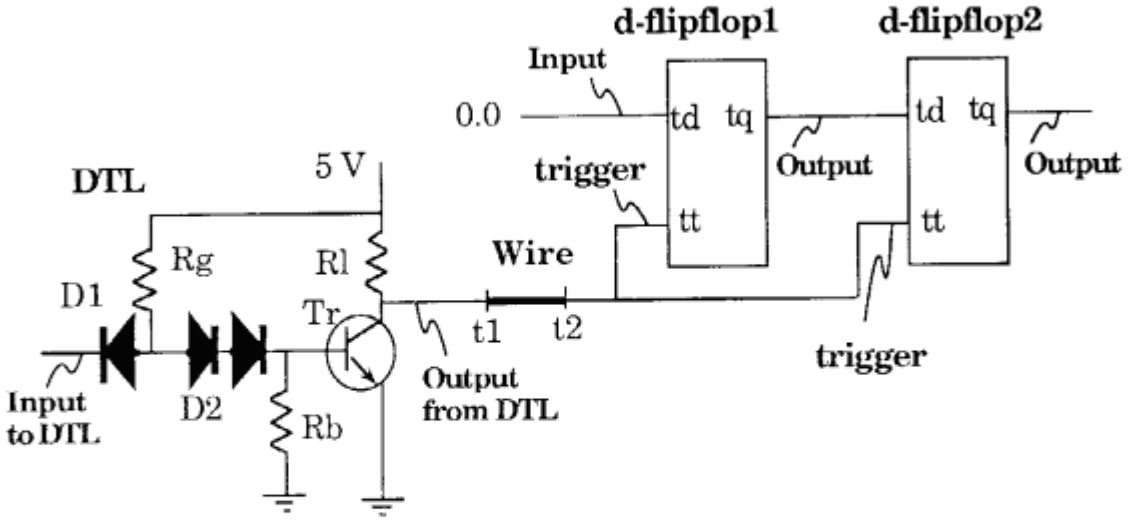


Fig. 13 Structure of a DTL circuit and two d-flipflops.

The connection between analog treatment and digital treatment happens between the terminal t2 of the wire and the two tt terminals of d-flipflops 1 and 2. The terminal t2 in the wire is dealt with as an analog circuit, but the tt terminal in d-flipflops are dealt with as digital circuits. In this example, the input to DTL decreases from 5.0 volts, and the input to the terminal td of the d-flipflop 1 is 0.0. And the initial outputs from the tq terminals of d-flipflops 1 and 2 are respectively 1.0 and 0.0. If the level of the tt terminal in a d-flipflop changes from 0 to 1, the level of the output from the tq terminal becomes equal to the level of the input to the td terminal.

The definition of the initial state for this example is shown in Figure 14. The objects part defines the components for the example in the figure 13. For example, the first line in the objects part

DFF1 - dflipflop ;

specifies that a component called DFF1 is an instance of the class d-flipflop (it is specified as dflipflop in the definition). That is, DFF1 has all properties of d-flipflops. The initial_relations part defines the initial values and the initial relations between components. The second line in the initial_relations part

connect(t2!Wire,tt!DFF1) ;

specifies that the terminal t2 of Wire and the terminal tt of DFF1 connect with each other. This connect relation shows the connection between the analog circuit and the digital circuit. The terminal t2 is an instance of the class terminal, whose definition is shown in Figure 3, and it has the voltage "v", the current "i", and "state_transition" as attributes. The terminal tt, on the other hand, is an instance of the class digital_terminal, whose definition is shown in Figure 15. It does not have the voltage and the current as attributes, but it does have the level and the intermediate_level as attributes. Another connection relation, for example, the fifth line in the initial_relation part

connect(t2!Vcc,t2!Inp,e!Tr,t2!RB);

```
initial_state  analog_to_digital
 objects
        % d_flip_flop
      DFF1 - dflipflop ;
      DFF2 - dflipflop ;
        % DTL
      RL - resistor ;
      RG - resistor ;
      RB - resistor ;
      Zd1 - diode ;
      Zd2 - diode2 ;
      Tr - transistor ;
      Inp - input_voltage ;
      Vcc - electric_power ;
        % Wire
      Wire - wire ;
 initial_relations
        % Connect two d_flip_flops
      connect(td!DFF2,tq!DFF1) ;
        % Connect DFF1 & DFF2 and Wire
      connect(t2!Wire,tt!DFF1) ;
      connect(t2!Wire,tt!DFF2) ;
        % Connect terminals in DTL
      connect(t2!Vcc,t2!Inp,e!Tr,t2!RB);
      connect(t1!Vcc,t1!RL,t1!RG);
      connect(t1!Inp,t2!Zd1);
      connect(t2!RG,t1!Zd1,t1!Zd2);
      connect(t2!Zd2,b!Tr,t1!RB);
      connect(t2!RL,c!Tr,t1!Wire);

        % Initial values for d_flip_flop
      level@td!DFF1 = 0.0 ;
      output@DFF1 = 1.0 ;
      output@DFF2 = 0.0 ;

        % Initial values for DTL
      resistance@RL  = 6000.0;
      resistance@RG  = 2000.0;
      resistance@RB  = 2000.0;

        % Power
      v@Vcc = 5.0;
        % Ground
      v@c!Tr = 0.0;

        % Initial values for input of DTL
      v@Inp = 5.0;
      ddt(v@Inp) < 0.0;

end.
```

Fig. 14 Definition for two d-flipflops and the initial state of a DTL circuit.

shows that four analog terminals - namely, the terminal t2 of electric power Vcc, the terminal t2 of the input voltage Inp, the emitter terminal "e" of the transistor Tr, and the terminal t2 of the resistor RB - connect at one point. The rest of the equations set the initial values. The input to the DTL circuit is described as follows in Figure 14:

```
v@Inp= 5.0;
ddt(v@Inp) < 0.0;
```

The first line specifies that the the initial value of the input is equal to 5.0 volts, and the second line specifies that the input decreases. "ddt" is the symbol for the time derivative.

```
object digital_terminal:TT
  attributes
     level - variable;
     intermediate_level - variable;
end.

object dflipflop:DF
  parts_of
     td - digital_terminal ;
     tq - digital_terminal ;
     tt - digital_terminal ;
  attributes
     input;
     output;
  relations
     input@DF = level@td!DF ;
     level@tq!DF = output@DF ;
end.
```

Fig. 15 Definition for digital terminals and d-flipflops.

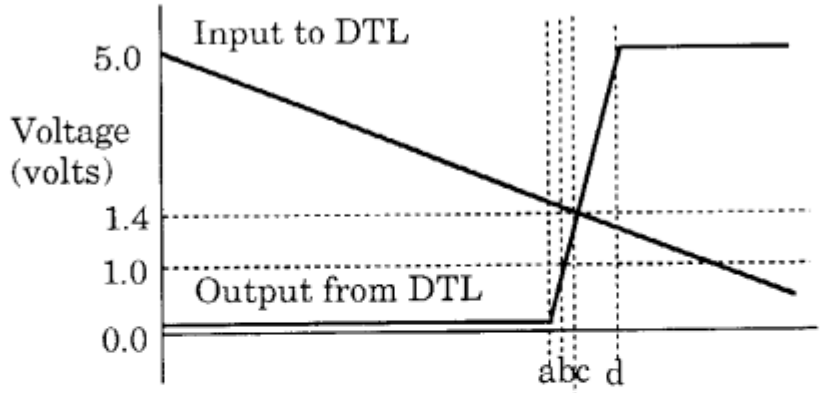
Figure 15 shows the definitions for digital terminals and d-flipflops. The definition for digital terminals specifies that the attributes of the digital terminals are level and intermediate_level. The attribute intermediate_level is used to specify the level of the digital terminal during the analog terminal connected with the digital terminal increases or decreases. The attribute is used in the three definitions of the events in Figure 8 and in the definition of the physical rule analog_to_digital_intermediate in Figure 9. The definition for d-flipflops specifies that a d-flipflop has three digital terminal, td, tq, and tt, as parts. A d-flipflop has two attributes, input and output, and their relations are specified in the relations part.

Figure 16 defines two events for d-flipflops. The event dflipflop_trigger_on is one that makes a d-flipflop on, and it specifies that if the level of the terminal tt change from 0.0 to 1.0, if the input of the d-flipflop is 1.0, and if the output is 0.0, then the output of the d-flipflop changes from 0.0 to 1.0. The event dflipflop_trigger_off is one that makes a d-flipflop off, and it is the opposite of the event dflipflop_trigger_on.

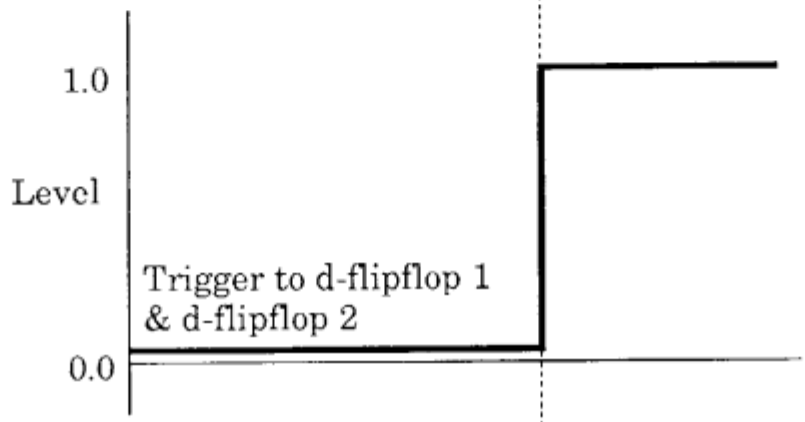
```
event dflipflop_trigger_on
  objects
    DF-dflipflop ;
  conditions
    prev(level@tt!DF) = 0.0 ;
    level@tt!DF = 1.0 ;
    input@DF = 1.0 ;
    output@DF = 0.0 ;
  actions
    remove(output@DF = 0.0) ;
    add(output@DF = 1.0) ;
end.

event dflipflop_trigger_off
  objects
    DF-dflipflop ;
  conditions
    prev(level@tt!DF) = 0.0 ;
    level@tt!DF = 1.0 ;
    input@DF = 0.0 ;
    output@DF = 1.0 ;
  actions
    remove(output@DF = 1.0) ;
    add(output@DF = 0.0) ;
end.
```

Fig. 16 Definition of events for d-flipflops.

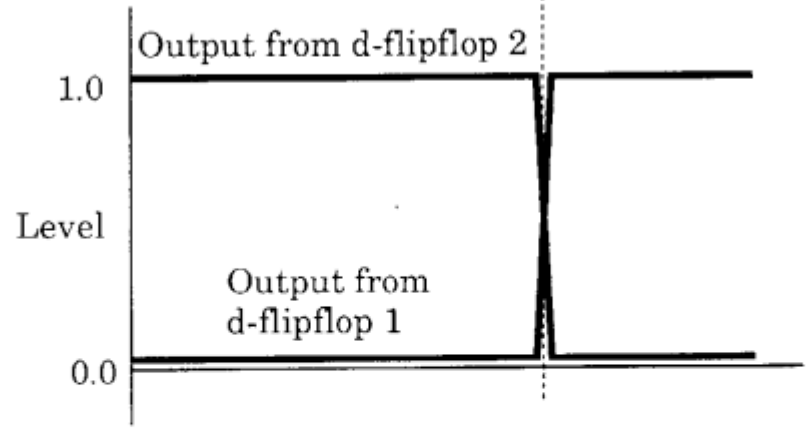
The result of Desq executing the initial state shown in Figure 14 is shown in Figure 17. Figure 17a shows the relation between the input voltage to the DTL circuit and the output voltage from it. The initial state is that the input voltage to the DTL circuit is 5.0 volts and it decreases. When the input voltage is greater than 1.400787 volts, the output voltage of the DTL circuit remains at about 0.2 volts. When the input voltage becomes 1.400787 volts, the output voltage begins to increase. The input voltage is shown in Figure 17a as "a". At the point "b", the output voltage is 1.0 volts and the input voltage is 1.400652 volts. When the output voltage becomes 1.4 volts, the input voltage is 1.400585 volts and it is indicated by "c". Finally, when the output voltage becomes 5.0 volts, the input voltage is 1.399978 volts at "d", and thereafter the output voltage stays at 5.0 volts as the input voltage decreases. Table 1 shows the input voltage to DTL and the output voltage from DTL at "a", "b", "c", and "d". Figure 17b shows the input trigger to the tt terminals in d-flipflops 1 and 2. The trigger is a digital signal to which the analog voltage that is the output voltage from the DTL circuit is translated. When the output voltage from the DTL circuit becomes 1.4 volts at "c", the trigger to the two d-flipflops changes from 0.0 to 1.0. The translation is executed by using the definitions shown in Figures 8 and 9. Because the trigger of the d-flipflops changes, the outputs of the two d-flipflops change by taking in the input to the terminal td in the d-flipflops.



(a) Input and output of DTL.



(b) Trigger to d-flipflops 1 & 2.



(c) Output from d-flipflops 1 & 2.

Fig. 17 Changes in a DTL circuit and two d-flipflops.

This example shows that by using the definitions for events and physical rules to translate from analog change to digital change, we can get the behavior of a circuit including the connection from analog treatment to digital treatment.

Table 1 Relations between input and output of the DTL circuit.

Symbols	Input (volts)	Output (volts)
a	1.400787	0.200808
b	1.400652	1.0
c	1.400585	1.4
d	1.399978	5.0

4.2 Connection from digital treatment to analog treatment

Figure 18 shows an example of a connection from digital treatment to analog treatment. This example consists of the same components as the previous example: a wire, two d-flipflops, and a DTL circuit. Moreover, their treatment is the same as in the previous example. Their connection, however, is different. Two connections from analog treatment to digital treatment happen between the terminal t2 of the wire and the two trigger terminals of the d-flipflops. The definitions of in Figure 8 and 9 are used for the connection as in the previous example. A connection from digital treatment to analog treatment happens between the output of the d-flipflop 2 and the input to the DTL circuit. The definitions for events in Figure 11 and the definitions for the physical rules in Figure 12 are used for the connection.

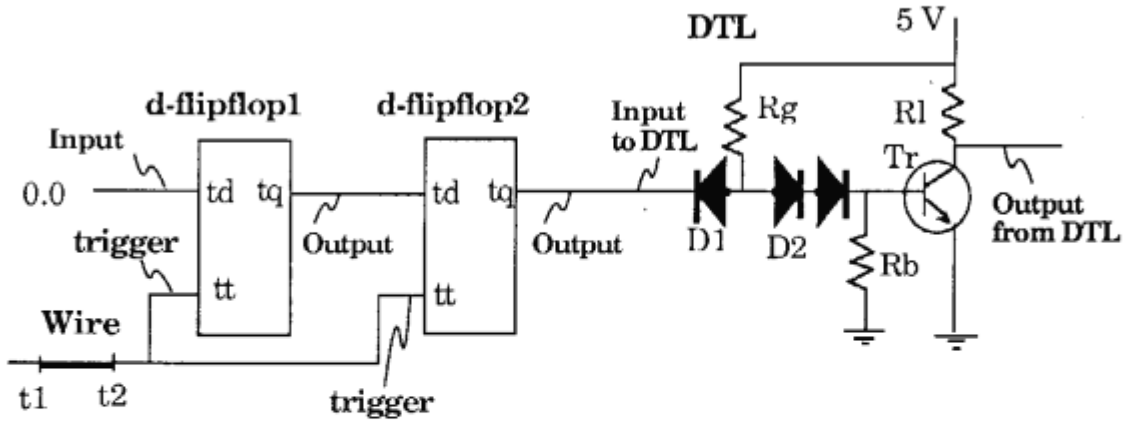


Fig. 18 Structure of a DTL circuit and two d-flipflops.

In this example, the voltage to the terminal t1 in a wire increases from 0.0 volts, and the change of the voltage is translated to the input to the trigger terminals of the two d-flipflops as a digital change. Because the two trigger terminals change from 0.0 to 1.0, the outputs of the two d-flipflop change from 1.0 to 0.0 and from 0.0

to 1.0. The change of output from d-flipflop 2, from 1.0 to 0.0, causes the change of the input voltage to the DTL circuit. As a result, the input voltage changes from 5.0 volts to 0.0 volts and the output of the DTL circuit changes from 0.0 volts to 5.0 volts.

The definition of the initial state for this example is shown in Figure 19. This definition is similar to that of Figure 14. The input to the wire is described at the end of the initial_relation part:

```
v@t1!Wire = 0.0 ;
ddt(v@t1!Wire) > 0.0 ;
```

The first line specifies the initial value of the voltage for the terminal t1 of the wire, and the second specifies that its derivative is positive, that is, the voltage increases.

The result of executing the initial state shown in Figure 19 is shown in Figure 20. The Figure 20a shows that when the input voltage to the terminal t1 in the wire increases and becomes 1.4 volts, the levels of trigger terminals in the two d-flipflops change from 0.0 to 1.0 as digital changes. Because the level of the trigger terminals changed, the output of the two d-flipflops changes from 0.0 to 1.0 and 1.0 to 0.0 (Figure 20b). The change of the digital output from d-flipflop 2 is translated to the change of the analog input voltage to the DTL circuit according to the definitions shown in Figures 11 and 12, and the translation is shown in Figure 20c. Since the input voltage to the DTL circuit changes as shown in Figure 20c, the output voltage from the DTL circuit changes as shown in Figure 20d. The relations between the input voltage and the output voltage of the DTL circuit are listed in Table 2. At "a", the output from the d-flipflop instantly changes from 1.0 to 0.0 and the input voltage to the DTL circuit begins to decrease. When the input voltage to the DTL circuit reaches the point "b", the output voltage of the DTL circuit begins to increase. Then when the input voltage to the DTL circuit gets to the point "c", the output of the DTL stops increasing. At the point "d", the input voltage to the DTL circuit becomes 0.0 volts and the output of the DTL circuit remains at 5.0 volts. Between "a" and "b" the transistor is in a saturated state, and between "b" and "c" it is in an "on" state. Between "c" and "d" it is in an "off" state. The output of the DTL circuit changes if the transistor is in an "on" state, since the transistor behaves like an amplifier when the transistor is in an "on" state.

By using definitions like those shown Figures 8 and 12, we can simulate behaviors of systems including digital treatment and analog treatment. If we want to reason the behavior of a large system and to get the detailed behavior of some part of the system, then we may be able to use analog treatment to simulate the part to be analyzed in detail and use digital treatment to simulate the rest part. As a result, we can reduce the costs of analyzing the behaviors of large systems.

```
initial_state  digital_to_analog
  objects
        % d_flip_flop
      DFF1 - dflipflop ;
      DFF2 - dflipflop ;
        % DTL
      RL - resistor ;
      RG - resistor ;
      RB - resistor ;
      Zd1 - diode ;
      Zd2 - diode2 ;
      Tr - transistor ;
      Vcc - electric_power ;
        % Wire
      Wire - wire ;
  initial_relations
        % Connect two d_flip_flops
      connect(td!DFF2,tq!DFF1) ;
        % Connect DFF1 & DFF2 and Wire
      connect(t2!Wire,tt!DFF1) ;
      connect(t2!Wire,tt!DFF2) ;
        % Connect DTL and d_flip_flop
      connect(t2!Zd1,tq!DFF2) ;
        % Connect terminals in DTL
      connect(t2!Vcc,e!Tr,t2!RB);
      connect(t1!Vcc,t1!RL,t1!RG);
      connect(t2!RG,t1!Zd1,t1!Zd2);
      connect(t2!Zd2,b!Tr,t1!RB);
      connect(t2!RL,c!Tr);

        % Initial values for d_flip_flop
      level@td!DFF1 = 0.0 ;
      output@DFF1 = 1.0 ;
      output@DFF2 = 0.0 ;

        % Initial values for DTL
      resistance@RL  = 6000.0;
      resistance@RG  = 2000.0;
      resistance@RB  = 2000.0;

        % Power
      v@Vcc = 5.0;
        % Ground
      v@e!Tr = 0.0;

        % Initial values for d_flip_flop
      v@t1!Wire = 0.0 ;
      ddt(v@t1!Wire) > 0.0 ;
end.
```

Fig. 19 Definition for initial state of the DTL circuit and two d-flipflops.

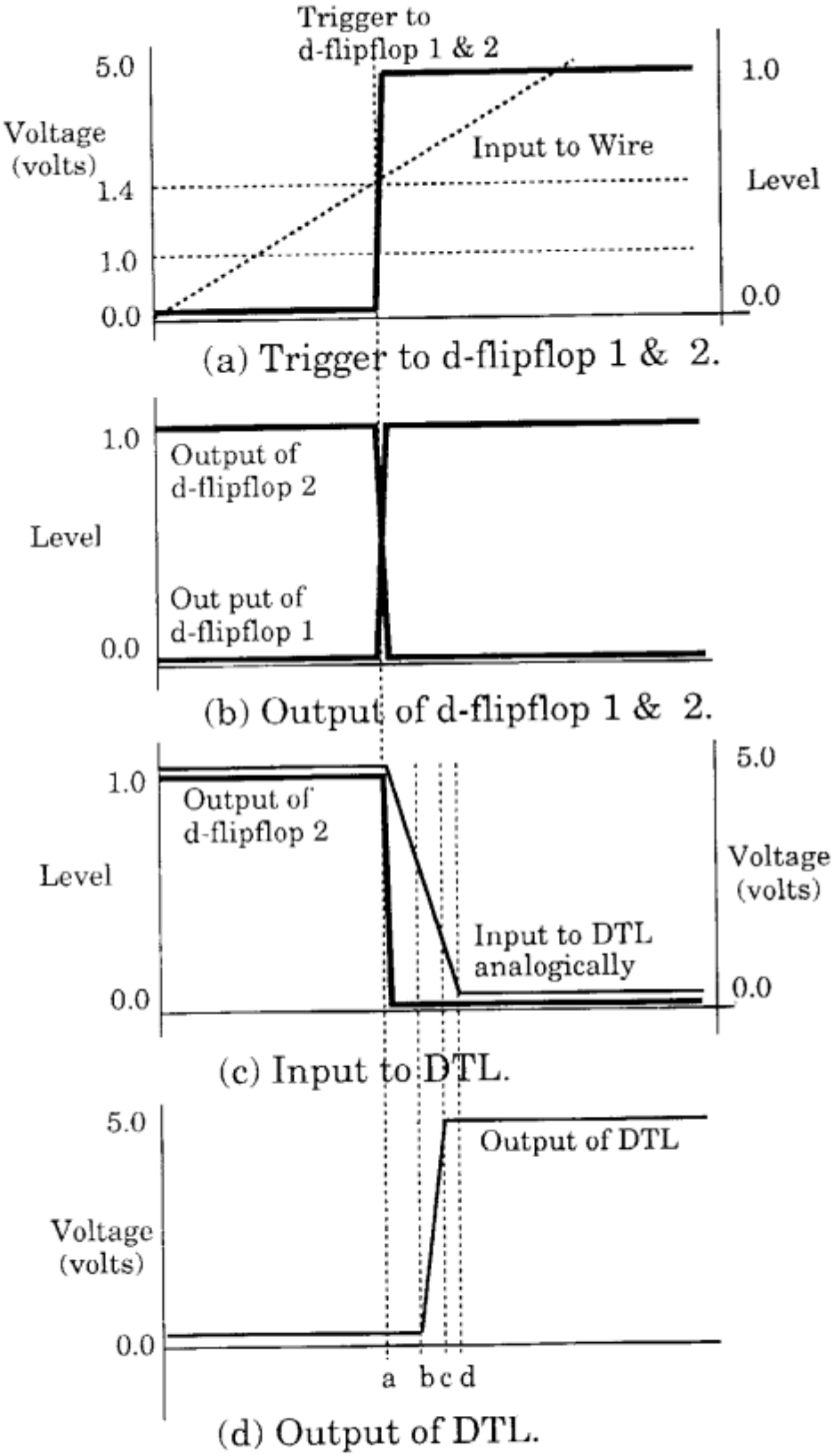


Fig. 20 Changes in the DTL circuit and two d-flipflops.

Table 2 Relations between Input and DFF and DTL, and Output of DTL.

Symbols	Input to DFF (volts)	Input to DTL (volts)	Out of DTL (volts)	State of Tr in DTL
a	1.4	5.0	0.2	saturated
b	>1.4	1.400787	0.200808	
c	>1.4	1.399978	5.0	on
d	>1.4	0.0	5.0	off

5. Conclusion

We have described a method for connecting digital treatment and analog treatment by using an event reasoning of Desq. We have applied the method to two examples. One lets information flow from a DTL circuit, which is dealt with as an analog circuit, to the two d-flipflops that are dealt with as digital circuits. The other example lets information flow in the opposite direction, from the two d-flipflops to the DTL circuit.

We think that we can apply Desq to larger circuits by using the method described in this paper. In the first example, the components of the DTL circuit are dealt with as analog circuits, and the behaviors of the quantities related to the components are reasoned by qualitative reasoning. But the components of the two d-flipflops are ignored, and the d-flipflop is regarded as a component and is dealt with as a digital circuit. If we want to analyze the detailed behavior of part of a large circuit by using qualitative reasoning and we can divide the circuit into analog and digital parts, then we can use qualitative reasoning for the analog parts and use event reasoning for the digital parts. This will reduce the costs of analyzing the behavior of the total circuit because event reasoning is usually faster than qualitative reasoning and we can ignore the details of the digital parts by using event reasoning.

This partitioning of analysis is especially suitable for modularizing the definition of components. We are now investigating a feature to modularize the definition of components by automatically building the definition of an object from definitions of components of the object. In our experiments that built the definition of a Schmitt trigger and a DTL circuit from their components [16], the CPU time for the simulation used to define of the Schmitt trigger circuit was about one the tenth of the simulation time used to define the components of the circuit. This modulalization feature is therefore expected to improve the performance of Desq. We are also investigating ways to make the definition more abstract - for example, to abstract from analog treatment to digital treatment by using time abstraction. The similar research is being done by Yoshida and Motoda[18]. If we can get abstract digital definitions automatically, we can use the method described in this paper to reduce the costs of calculating the total behavior of a large circuit.

6. Acknowledgements

This research was supported by the ICOT (Institute of New Generation Computer Technology). We wish to express our thanks to Dr. Fuchi, Director of the ICOT Research Center, who provided us with the opportunity to perform this research in the Fifth Generation Computer Systems Project. We also wish to thank Dr. Nitta and Mr. Ichiyoshi in the seventh research laboratory for their many comments and discussions regarding this study. And we wish to thank Mr. Kawaguti, Miss Toki, and Miss Isokawa of Hitachi Information Systems for their help in implementing of our system.

References

1) Bobrow, D. G.: Special Volume on Qualitative Reasoning about Physical Systems, Artificial Intelligence, 24, 1984.

2) Forbus, K. D.: Qualitative Process Theory, Artificial Intelligence, 24, pp. 85-168, 1984.

3) Forbus, K. D.: Introducing Actions into Qualitative Simulations, IJCAI-89, pp. 1273-1278, 1989.

4) Kuipers, B.: Commonsense Reasoning about Causality: Deriving Behavior from Structure, Artificial Intelligence, 24, pp. 169-203, 1984.

5) Mizoguchi, R.: Foundation of expert systems, Expert system - theory and application, Nikkei-McGraw-Hill, pp. 15, 1987 (in Japanese).

6) Nishida, T.: Recent Trend of Studies with Respect to Qualitative Reasoning (I) Progress of Fundamental Technology, pp. 1009-1022, 1988 (in Japanese).

7) Nishida, T.: Recent Trend of Studies with Respect to Qualitative Reasoning (II) New Research Area and Application, pp. 1322-1333, 1988 (in Japanese).

8) Nishida, T.: Qualitative Reasoning and its Application to Intelligent Problem Solving, pp. 105-117, 1991 (in Japanese).

9) Ohki, M. and Furukawa, K.: Toward Qualitative Reasoning, Proc. of Symposium of Japan Recognition Soc., 1986, or ICOT-TR 221, 1986.

10) Ohki, M., Fujii, Y., and Furukawa, K.: Qualitative Reasoning based on Physical Laws, Trans. Inf. Proc. Soc. Japan, 29, pp. 694-702, 1988 (in Japanese).

11) Ohki, M., Sakane, J., Sawamoto, K., and Fujii, Y.: Enhanced Qualitative Physical Reasoning System: Qupras, New Generation Computing, 10, pp.223-253, 1992, or ICOT TR-432, 1988.

12) Ohki, M., Oohira, E., Shinjo, H., and Abe, M.: Range Determination of Design Parameters by Qualitative Reasoning and its Application to Electric Circuits, pp. 1022-1029, FGCS'92, 1992.

13) Ohki, M., Toki, N., Isokawa, S., Oohira, E., Shinjo, H., Kawaguchi, T. and Abe, M.: Nonlinear inequalities constraint solver combined multiple constraint solvers, J. of Japanese Soc. for Artif. Intel., 1992 (submitted, in Japanese).

14) Ohwada, H., Mizoguchi, F., and Kitazawa, Y.: A Method for Developing Diagnostic Systems based on Qualitative Simulation, J. of Japanese Soc. for Artif. Intel., 3, pp. 617-626, 1988 (in Japanese).

15) Oohira, E., Ohki, M., Shinjo, H., and Abe, M.: A Reasoning Method for Computer-aided Circuit Design using Qualitative Reasoning for Circuits

involving Discontinuous Changes, Transactions of IEIC, 1992 (to appear, in Japanese).

16) Shinjo, H., Ohki, M., Oohira, E., and Abe, M.: Acquisition of hierarchical knowledge from deep knowledge in Qualitative Reasoning, SIG-FAI-9201-8, Japanese Soc. for Artif. Intel., (in Japanese), 1992.

17) Yamaguchi, T., Mizoguchi, R., Taoka, N., Kodaka, H., Nomura, Y., and Kakusho, O.: Basic Design of Knowledge Compiler Based on Deep Knowledge, J. of Japanese Soc. for Artif. Intel., 2, pp. 333-340, 1987 (in Japanese).

18) Yoshida, K. and Motoda, H.: Hierarchical Knowledge Representation Based on Approximations, J. of Japanese Soc. for Artif. Intel., 7, pp. 69-76, 1992 (in Japanese).