Reasoning with Time Dependent Data

Thomas Anton Russ

Reasoning with Time Dependent Data

by
Thomas Anton Russ

Originally submitted to the
Department of Electrical Engineering and Computer Science
on August 31, 1991, in partial fulfillment fo the
requirements for the degree of
Doctor of Philosophy

Abstract

Knowledge-based systems that perform monitoring and management must contend with information that changes over time. Information that is changing, is delayed and arrives out of order complicates the task of programming such systems. To simplify the construction of monitoring and management systems, a Temporal Control Structure (TCS) has been developed which manages data dependencies over time. The TCS simplifies the job of a system builder by decomposing reasoning into static and dynamic components. To the extent that the decomposition is successful, each part of the reasoning problem can be addressed separately and the overall task is easier to accomplish.

The Temporal Control Structure performs the bookkeeping tasks needed to assure that information is propagated and that the reasoning in the system is complete. Completeness means that all data entered into the system have been processed, and no more changes in the outputs of the system occur. To deal with the temporal complexities of the monitoring domain, TCS exploits two properties common in monitoring: exact knowledge of when events occur and a fixed plan for handling eventualities. These properties allow automatic scheduling of reasoning processes in response to data changes and also allow data dependencies (needed for change propagation) to be compiled into the program.

This report describes the design of the Temporal Control Structure and reports the results of several reasoning systems implemented using the formalism. The most ambitious system can track the progress of patients suffering from diabetic ketoacidosis over the course of several days. In a formal evaluation by an expert panel, the computer-generated advice was judged similar in quality to actual hospital treatment.

Thesis Supervisor: Peter Szolovits
Title: Professor of Computer Science

Acknowledgements

I would like to thank Dr. Michael Klein from Boston University–University Hospital for providing the example case used in the hindsight section.

Without the aid of my medical collaborators, Dr. Michael Hagen and Dr. Klemens Meyer from Tufts New England Medical Center, it would not have been possible to construct the expert system. In spite of their assistance, any errors in the medicine are my own. Klemens' calm demeanor at the battle of the Xerox machines minutes before the formal evaluation certainly steadied my nerves and demonstrated the value of having a collaborator to provide moral support.

Ramesh Patil helped me clarify a lot of thoughts by listening to what I had to say and then summarizing it in one quick phrase.

I would also like to extend thanks to William Long who has been an invaluable sounding board for ideas relating to the design of TCS. Bill had the gift of just sitting there and looking at me until I was able to figure out how to fix what at times seemed insurmountable problems.

My advisor Peter Szolovits has been a fine mentor and supporter during my graduate studies. He certainly lived up to the meaning of his German title, *Doktorvater*, for I have been able to develop a close friendship in the course of my years here at MIT

My wife Ellen has been understanding and supportive, particularly in the last agonizing stages of writing the thesis.

I would also like to thank Annette Ellis for her assistance in editing the thesis in the process of producing the technical report.

Finally I would like to acknowledge the research support from National Institutes of Health grant 5-R01 HL 33041 from the National Heart, Lung and Blood Institute and grant 5-R01 LM 04493 from the National Library of Medicine.

Contents

List of Figures

Chapter 1

Introduction

Expert systems have typically involved a single consultation in which all of the information is available for analysis in reaching a diagnosis or other conclusion. This is not always feasible in the real world. As systems evolve from one-shot consultation toward management, there will be a need to track information over time. The very concept of management implicitly assumes there will be changes. In medicine, for example, data that affect the medical decisions change over time, necessitating a revision of the description of the patient.

Programs that solve problems over time need to use data that are *time dependent*, i.e., information that changes with time. Typical examples would be a patient blood pressure in medicine, or wind direction and strength in handling a chemical plant accident. The ability to accept and react to changing data lies at the heart of the management task.

There are several complications that increase the difficulty of finding solutions. First, information may not be instantly available. Some laboratory analyses take time to perform. When the information becomes available, it refers to conditions at some time in the past. Because the delays are not uniform, information can also arrive out of sequence. In order for the data to be properly interpreted, updating of conclusions must take place. In this process care must be exercised because actions taken in the past cannot be undone. They must be accepted and compensated for in the future. Although it is legitimate to use all information when evaluating the correctness of decisions, any explanation of why certain actions were taken must use only information that was available at the time the decision was made.

I solve these problems using a model of temporal reasoning which divides continuous dynamic processes into static segments, coupled with a dependency-directed updating system that allows conclusions to be retracted as information changes. The static segmentation is used to simplify the reasoning process and is similar to techniques used in other engineering disciplines. Two examples are the piece-wise linear decomposition of more complex functions and the division of device characteristics into operating regions in electronics.

In this work, I supplement the static elements with data links to model dynamic processes. Finally, I use data-directed updating to ensure that conclusions have access to all relevant data, and to retract conclusions that are no longer supported by the data.

1.1 Motivation: Clinical Management

The initial impetus for this work was a need to improve the technology for designing intelligent monitoring systems. Management imposes the need to monitor data that change over time, the need to retract erroneous assumptions or refine initial assessments, and the need to act in spite of the incompleteness of the data. Many past Artificial Intelligence in Medicine (AIM) programs have concentrated on the diagnostic problem in the context of a single consultation. Even programs which evaluate the temporal course of an illness [45] assume all of the information that is known about the patient is available at the time of the consultation.

An expert system for cardiac intensive care monitoring (the Arrhythmia Advisor) [68, 73] demonstrated a need for better tools for expert system construction. In that environment, continuous heartbeat monitoring information is available, along with occasional input from clinical laboratory tests and bedside examinations [54]. The original expert system for therapy management was able to handle some time-varying data, but it did not have a general mechanism for dealing with time. Consequently, some of the time-varying data were not properly used, because special-purpose coding of the temporal aspects of the reasoning was needed.[1] Aside from being inelegant, the need to use special care in the system implementation increased the likelihood that programming errors would be introduced, particularly the omission of updating in response to data changes.

In the case of laboratory test data, changes in patient state or clinical interventions often occur between the time samples are sent to be tested and the time the results are reported. The state of the patient disease can change, and interim therapy can be instituted without waiting for all of the data. Figure 1.1 illustrates the interaction of these events in a clinical setting. Acting before all of the data is gathered is especially common in emergency rooms and intensive care units, where the urgency of life-threatening problems precludes a strategy of waiting for complete data collection and a definitive diagnosis.

Even if there were no problem caused by data that arrive out of temporal sequence, some capability to change past information entered into the system is also required of a real-world system, because it must be possible to make changes in the data in order to correct data that are later determined to have been in error. This problem appears in cardiac intensive care, because one of the primary sources of data, automated

[1] An example was the effect of changes in weight or cardiac output on models used to predict drug concentration. An example showing how TCS successfully handles this problem is presented in chapter 3.

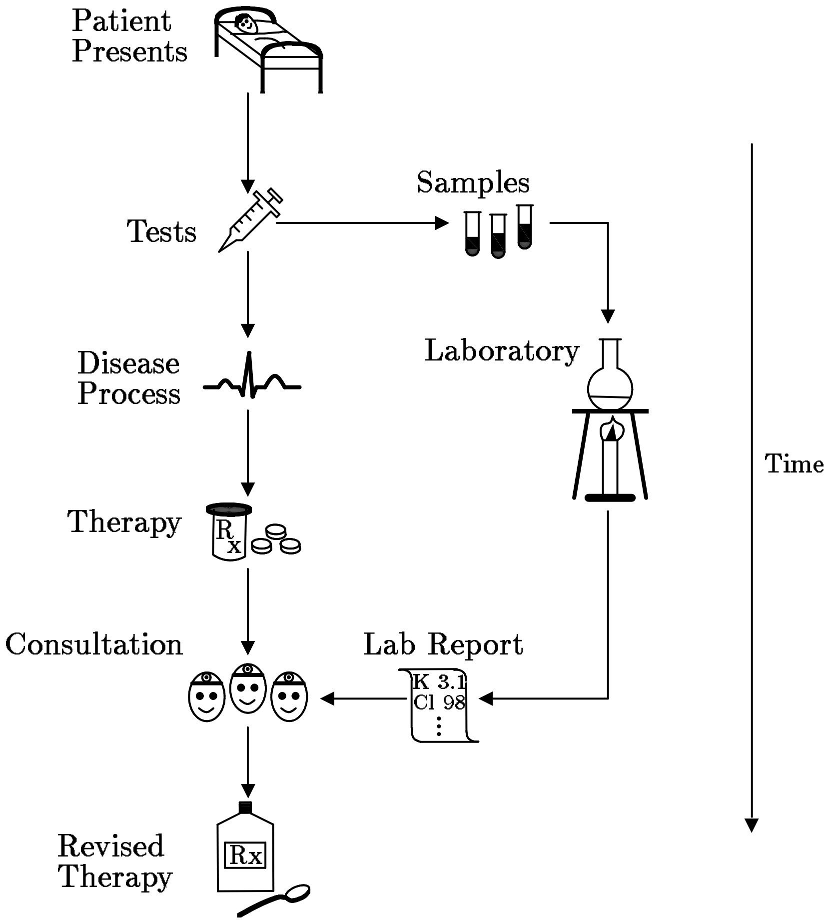


Figure 1.1: Decision making Over Time in Intensive Care.

electrocardiographic monitoring, is prone to a small but significant number of errors that can be detected during post-editing by trained clinical personnel. If such post-editing indicates an error in the data, then the program should be able to take that into account. Since events occurring in the past can affect the current assessment and treatment of a patient, one cannot simply ignore changes to "old" data.

These problems led to the invention of a mechanism to support the programming of monitoring systems. In addition to the development of a software architecture for such expert systems, I report the results of applying the approach to a clinical problem: the management of patients in acute diabetic ketoacidosis. I demonstrate the effectiveness of the program by a formal evaluation of the computer-generated advice by a panel of physicians in a blind study. In the remainder of this chapter I discuss the design considerations and sketch the function of the system architecture. A more detailed discussion of the implementation of the programming shell follows. I then present the design and evaluation of a Ketoacidosis Advisor, followed by a comparison of this approach with previous work in the field. Finally, I present a summary of the work and directions for further research.

1.2 Constraints: System Design Considerations

In this section I describe the problems that must be solved by a tool for creating systems that use time-dependent data. I first present the requirements that make the solution of the reasoning problem more difficult; then I examine domain features that can be exploited to simplify the programming task. The design challenge is to use the latter to make coping with the complicating factors work reasonably.

A system with a management component interacts with its environment in pursuit of some goal such as "stabilizing a patient" or "protecting citizens from a chemical cloud." To solve the task, the system has actions that can affect the environment, but action is complicated because the environment can change autonomously, and reports concerning the state of the environment may be delayed or arrive out of sequence. These characteristics require an ability to modify conclusions which depend on the newly available data. A thesis of this work is that such updating should be incremental and opportunistic, rather than global. This hypothesis is based on the presumed locality of effect. In a complicated system, it is rare that one piece of information will radically change the interpretation of all of the other information in the system. This premise justifies using a dependency-based updating scheme.

In sum, the complicating features that characterize management and monitoring tasks are:

1. Data change over time. Since a single measurement or variable can have different values at different times, a model of time is required.

2. Information does not arrive in order. A mechanism for generating the correct conclusions using data that arrives after a time delay and out of order must

be present. To the greatest extent, this function should be transparent to the program.

3. Reasoning in the past and in the future are different. Conclusions and interpretations can change in the past, while actions cannot. This dichotomy must be supported.

4. Complete updating is required. The system must insure that all decisions that rely on a particular datum are updated when changes to that datum occur.

5. The system is not in complete control. Management is complicated because the environment can change autonomously. In addition, the actions that can be taken do not always have the desired effect (predictability is limited). This aspect of the domain needs to be considered in the system design.

These features make the job of constructing an expert system more difficult. To ease the task, a programming system can exploit three characteristics of the monitoring domain:

1. Good temporal resolution. Since the environment is under surveillance, the system knows when observations of the environment take place and when actions are carried out. This eliminates the uncertainty in the time of data and actions. This is a powerful feature that can be exploited to limit the work that the system must perform.

2. Protocol-driven management. In the medical domain, much of the treatment of specific problems is driven by a protocol, a set plan of action. Although the plan itself is flexible enough to be adapted to individual patient circumstances, it is not necessary to generate a complete plan from scratch.

3. Parameters and actions are known in advance. The range of input variables the system can observe and the type of actions that can be performed can be determined in advance. Since the structure of the problem-solving task is not in flux, compilation of the reasoning and, crucially, of the dependency structure is possible.

I developed a computational model that exploits these general domain characteristics to meet the requirements listed earlier. A system coupling dependency-directed updating with temporal data can effectively deal with the problems encountered in the patient management problem. The resulting techniques can be applied beyond the intensive care domain that provided the initial motivation. The computational model of temporal reasoning also provides a formalization of the data and reasoning processes that gives insight to different fundamental types of reasoning with temporal data. This insight can identify which processes are computationally more expensive than others. In addition, this approach has the following advantages:

1. There is no system-mandated reasoning paradigm. Decisions are specified in Lisp, which is a powerful general-purpose programming language. Applications can make use of disparate sources of information as well as heterogeneous decision procedures.

2. The algorithms for the actual decision making and the combination of evidence are specified by the programmer. The programmer can make an informed decision about the algorithms used in the system.

3. Programming is simplified by an abstraction that separates static and dynamic analysis. A static environment for reasoning is established with a series of such static environments linked together to form a dynamic process.

4. The dichotomy between the static and dynamic components simplifies the transition from existing non-temporal expert systems to temporal ones. Existing systems can be encapsulated in the static abstraction and the dynamics can be implemented with the control structure.

The problems encountered in the arrhythmia monitoring project led to the design and construction of a programming framework: one that could schedule reasoning processes and update conclusions in the face of changes [53, 69]. Although the motivation for this approach arose in a medical setting, the mechanism itself does not contain domain-specific knowledge, so it can be applied in other domains with similar requirements.

1.3 Solution: Temporal Control Structure

I have designed and implemented a Temporal Control Structure (TCS) which applies the method of dependency-directed updating to the problem of data that change over time. This extends and generalizes previous work on truth- (or reason-) maintenance systems. My extension removes restrictions on the reasoning method and inverts the system architecture. The basic reasoning unit is a temporal *process instance*, which is treated as a "black box" by the system. The inputs and outputs of the "black box" are monitored by the system and drive the updating process. I present a detailed comparison of TCS and truth maintenance systems in section 8.2. The first fundamental design decision was to use dependency-directed updating.

A second fundamental design decision was to separate the problem of reasoning over time into two components: a static component and a dynamic component. A continuous process is divided into a series of "segments," each of which is a static context. Temporal change is handled by the dynamic component, which links successive static components. A series of static states is used to model a dynamic process. The segmentation is illustrated in figure 1.2.

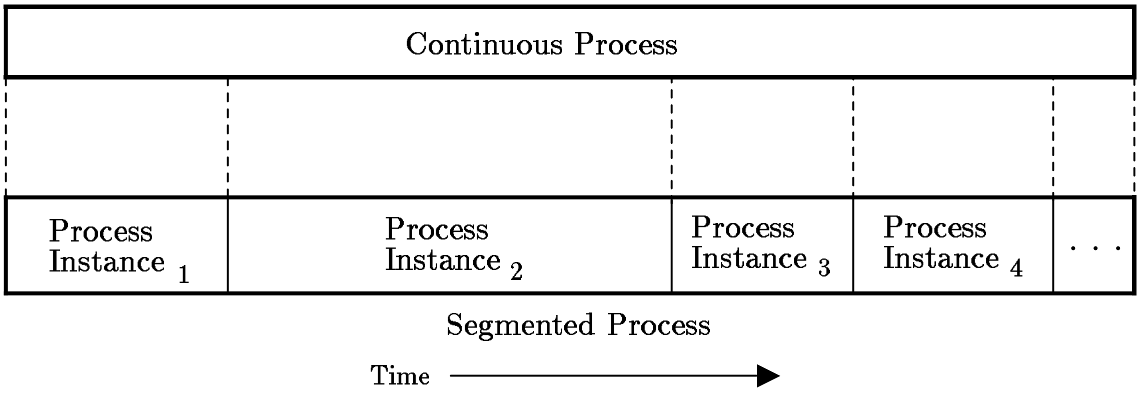


Figure 1.2: Segmenting a Temporal Process

Segmentation corresponds to the description of the problem by domain experts in terms of *states* and *changes of state*. The usefulness of the state abstraction explains the success and popularity of the rule-based paradigm of expert systems design. In a classic rule-based system, each rule has a set of antecedents which refer to a static situation, one in which each antecedent has a single value. The rule clauses do not consider the possibility of changes in the values of the antecedents that are being checked.[2] By being clever, we can use this state abstraction to simplify decision making. TCS supports this static abstraction as a principal technique in expert system design. The TCS framework handles the bookkeeping details needed to schedule the static reasoning units in an environment with changing data. This allows one to divide reasoning in time into nearly orthogonal components: one which examines current, unchanging values, and another which handles the changes in values.

By segmenting the reasoning, a designer can divide the task of reasoning over all of time into smaller pieces that can be more readily handled. This division of the timeline through segmentation creates distinct temporal regions. As figure 1.3 shows, each process instance has the time divided into a *current period*, a (relative) *past* and a (relative) *future*. By appropriately choosing the starting and ending times of the process instance, the values of inputs to a decision can be held constant during the current period, with all changes in value taking place either in the future or the past. It is important to make a distinction between the absolute past (or future) and the relative past (or future). The absolute times are measured against the real-world time. Maintaining this distinction is important because no system can take actions in the past. The point of reference for absolute time is *now*, a TCS variable which represents the time in the real world. Relative time is the timeline seen from the point of view of an executing process instance. Parts of the reasoning process that occur earlier on the timeline than the current process are in the relative past, while parts of the

[2] Exceptions to this are VM and TOPAZ which I describe later (see sections 8.3.2 and 8.3.3).

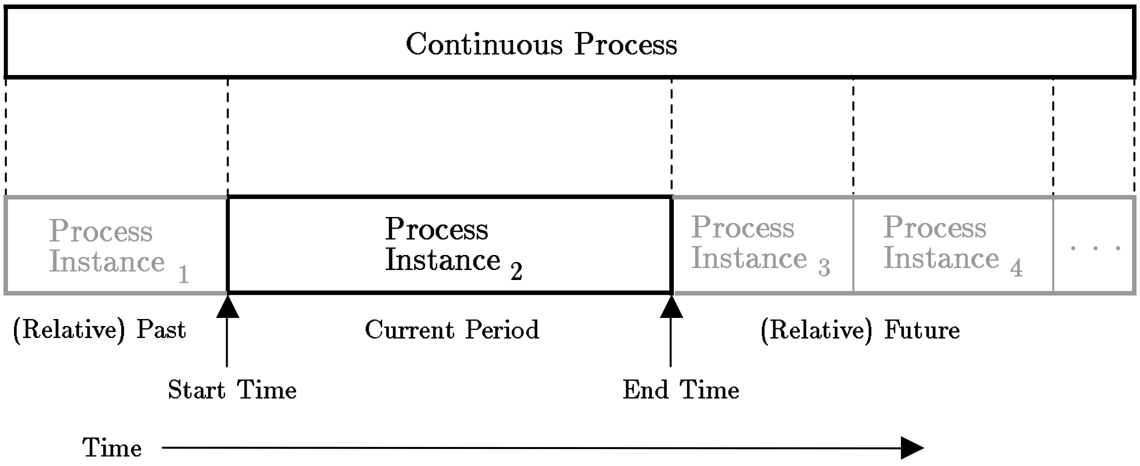


Figure 1.3: The View from a Single Process Instance

decision that occur later are in the relative future. The point of reference for relative time is the execution interval for the current process instance. For reasoning that processes data as it arrives, the relative and absolute reference points may coincide. In reasoning about changed data or speculating about the future, the relative and absolute reference points will typically be different. In general, the meaning of the terms "past" and "future" will be clear from the context. When necessary, I use the qualifying terms "relative" and "absolute".

In order to provide the updating services, the Temporal Control Structure must know the data dependencies of each decision. Since complete updating is required, access to data must be limited to that which has been declared to the system. At the same time, however, the system should impose the minimum restrictions on the calculation performed by the reasoning elements. To accomplish this, each reasoning unit, called a *module*, declares the time-varying information upon which its conclusion depends. A module is the static description of a process instance, and corresponds to a procedure definition in a programming language; the process instance corresponds to a procedure invocation. The interface declared to the TCS consists of the inputs, outputs and internal state variables. The inputs and outputs link different modules whereas the state variables link successive process instances of the same module. Figure 1.4 shows the interfaces between process instances. The details of the algorithm or reasoning method used inside the module are of no interest to the TCS. It is only required that the algorithm be a deterministic function of the inputs and internal state. The reasoning modules themselves serve as black boxes, whose interface to the world is only through variables that have been declared to the TCS. The updating is accomplished by having the TCS monitor the value of all of the temporal variables and then schedule the appropriate reasoning modules whenever the input values change.

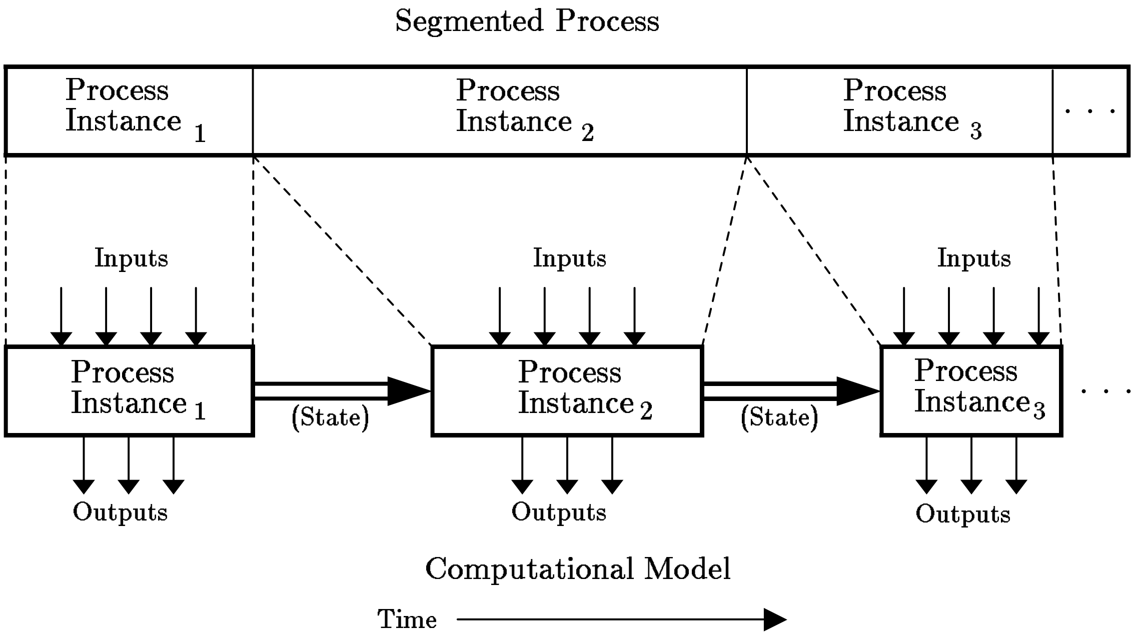


Figure 1.4: Interfaces Between Process Instances in the TCS.

The TCS need only be able to establish equality of value for the variables.

TCS restricts the data availability of each process instance to the values of its input variables during the interval of execution. The only access to data from outside this interval is through the use of internal state variables (which are made known to TCS). This restriction allows a clean implementation of the updating mechanism without sacrificing data access. It also means that the need to update a process instance can always be determined on the basis of locally available information.

The TCS is able to support the static abstraction and guarantee complete updating by imposing only minimal restrictions on the data values and on the types of reasoning that can be implemented in the modules. Variable values are limited in only two ways:

- Times on variable values must be exact.
- Equality of variable values must be computable.

The first restriction allows automatic scheduling of process instances in response to changing values without ambiguity or the need to have a branching model of time. The second restriction is used to stop the propagation of values when they are no longer changing. Since the only system-imposed requirement is that the equality of values be computable, the data representation remains largely unconstrained. Reasoning modules also have very few restrictions:

- Functions must be deterministic. If invoked with the same inputs, the same output must be produced.

- There can be no hidden state. The only communication between process instances must be through system-declared variables, or state variables known to the system.

These restrictions give the TCS the freedom to execute the process instances in any order, and even to evaluate them more than once without affecting the outcome. Within the restrictions noted above, each reasoning module can use any type of reasoning. Arithmetic calculations, formal logical inference, statistical analyses and heuristic decision rules are examples of the range of reasoning types that can be accommodated. Since all decisions must be made in the black box modules, the system cannot be responsible for combining the effects of several different reasoning modules on any one value, so any given variable can be the output of at most one reasoning module. Any combination of values from different sources must be explicitly programmed by the application designer in a module. This flexibility allows the programmer to specify any method he wishes: a dominance relation, Bayesian combination of probabilities, Dempster-Shafer evidence combination, an *ad hoc* method, etc. The decision-making will be organized around the individual output variables. All immediate influences on a particular value must be concentrated together in one module. The information dependence of any variable's value can therefore be readily determined simply by examining the inputs of the reasoning module. Of course, a complex computation may be decomposed into meaningful subparts by introducing additional variables and corresponding reasoning processes.

The fundamental thesis of this work is that *the division of the reasoning into static and dynamic components is a suitable abstraction for simplifying the construction of time-dependent expert systems.* Techniques developed for constructing current non-temporal expert systems can be incorporated into TCS-based systems through the static context. Additional programming necessary for recognizing the important dynamic aspects of the application domain can be added separately, resulting in a natural decomposition of the development effort.

1.4 Overview of the Thesis

The remainder of the thesis is divided into three major sections. In the first section (chapters 2–4) I describe the Temporal Control Structure design. In chapter 2 I describe the low-level function of the TCS. Chapter 3 contains an extended example showing the implementation and control of a mathematical model, and in chapter 4 I present higher-level abstractions built on underlying TCS structures. In the second section (chapters 5 and 6) I report on the design and evaluation of a major application: a therapy advisor for handling diabetic ketoacidosis. The construction of an effective expert system demonstrates the practicality of using TCS in a real world domain. Finally, in chapters 7 and 8 I relate TCS to other work in the field and discuss the future development of the Temporal Control System.

Chapter 2

Reasoning Model

The core of TCS is a simple but low-level system that assures complete updating. A programmer working at this level has the greatest flexibility, but at the cost of having to write the most detailed code. More abstract reasoning tasks can be built on top of this substrate. Such reasoning abstractions exploit the combination of common temporal reasoning tasks to provide higher-level functions which hide some of the detailed calculations. The updating guarantees are preserved at the higher level because they are derived from the underlying substrate. This approach has the further advantage of allowing TCS to acquire more powerful tools as experience in using the system grows. Most of the higher-level tools were developed as generalizations of low-level inference procedures that solved commonly occuring problems. One example is a module which calculates the persistence (for reasoning purposes) of information derived from a point sample.

In this chapter I describe the basic components of the TCS. I show in the next chapter how TCS can be used to implement a simple time-dependent mathematical model. In chapters 4 and 5, I introduce the higher-level abstractions.

The control structure embodies a model of temporal reasoning that facilitates the efficient updating of the state of knowledge of the system. TCS updating uses data dependency to limit the work performed to that which is absolutely required. This requires that the control system be aware of the data dependencies among the individual reasoning elements that implement the decision-making. For example, a weather module that uses temperature, wind speed and humidity in producing a forecast must declare the dependence of its reasoning on those variables. This enables TCS to monitor variables and update conclusions that depend on the variable values. I describe the exact requirements and their effects below in section 2.2

All data in the system are associated with points or intervals on a time line. These data are stored in point or interval variables. The variables collectively provide a central repository for all of the information used by a TCS application. The database is as complete a description of the world as possible. The description becomes progressively more complete as inaccuracies are corrected and more conclusions are made.

```
  ;; Syntax for defining TCS variables
(defmodvar name type options...) ; type is one of :point or :interval

  ;; Syntax for defining TCS reasoning modules
(defmodule name (inputs...) (outputs...)
           ;; process state declarations:
     ((type name options...) ; type is one of :history or :oracle
        ⋮
        )

     ;; implementation of the reasoning. In the body of the code, the input, output,
     ;; and process state variables are referred to as ordinary Lisp variables.
  program code
  )
```

Figure 2.1: Basic TCS Constructs

Based on this interpretation, incorrect past conclusions should be changed at their moment in history. The "mistakes" are removed from the database whenever they are discovered. (If it should be desirable to remember that things have changed, then an audit trail of these changes can be maintained. Indeed, this allows a system to explain why a change in values occurred. This is particularly useful in explaining some action taken on the basis of information later determined to be false.)

Reasoning is performed by user-specified functions that act on information contained in the temporal variables. As part of the definition of such a function, the user must also declare the data dependencies. The syntax for defining variables and reasoning modules is shown in figure 2.1. The program fragments in this thesis give a general impression of TCS as a programming language. For a complete description of the syntax and a discussion of the options the reader should consult the reference manual [70].

2.1 Time and Data Model

A distinction can be made between several types of time-related databases. This distinction, following terminology developed by Snodgrass [82] is made on the basis of the types of questions that can be answered. A conventional database storing only a set of facts without special support for time is called a *snapshot* database, because is holds only the currently believed value for each entry. For example, one can only ask "Do we know anyone who has broken the Enigma code?" When time is added, there are two separate time axes that define the information in a database. The first time axis is *transaction time*, which describes when information is entered into a database. By keeping track of transaction time, one can answer questions about the state of a database as of any particular time: "*As of* February 1945, do we know anyone who has broken the Enigma code?" At any moment, however, there is only

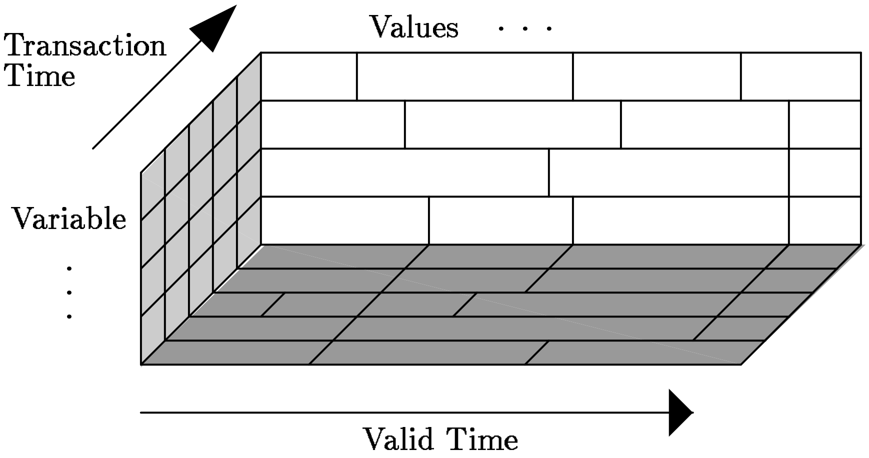


Figure 2.2: Time and Databases

one entry for each variable. Such a database is called a *rollback* database, because we can always roll back the contents of the database to their values at a previous time. The second, orthogonal axis is the *valid time* axis, which represents the time (in the world) when a particular fact holds. Such a database is called *historical*. The database can contain more than one value for each entry, as long as the entries occur at different times. With this time axis, a different question is possible: "Do we know anyone who has broken the Enigma code *on* January 1, 1942?" Since the value of the basic question—people who have broken the Enigma code—can change over time, multiple entries are possible. With a historical database, one can ask for any particular value. Since the two time axes are independent (see figure 2.2), one can combine them and implement a *temporal* database, combining the features of the rollback and the historical databases. This allows a combined question such as "*As of* February 1945, do we know anyone who has broken the Enigma code *on* January 1, 1942?" Such a question addresses both the time when an entry had a particular value (valid time) and the time we found out the value of that entry (transaction time).

The database used in TCS is fundamentally a historical database, but in the implementation I made provision for storing the additional information needed to make it a temporal database. In order to conserve memory space, I have not exploited this capability in any of the experiments I have conducted so far.

What distinguishes TCS from a conventional historical or temporal database is the close link between the contents of the data base and the inference methods. This link is analogous to the connection between the database in a rule-based system and the rules, insofar as both systems relieve the application creator of the need to program explicitly the connection between information being entered into the database and the execution of inferences. By automatically scheduling inferences in response to data as they become available, TCS allows an implementor to ignore the effects of the

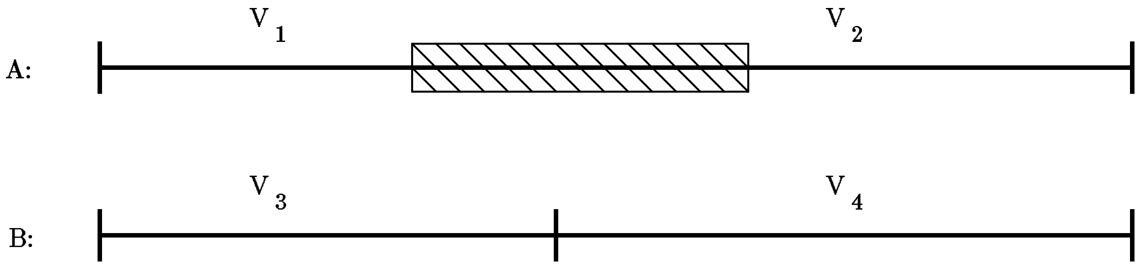


Figure 2.3: Inputs without Exact Endpoints

transaction time axis on the program. Since all relevant decisions will be reconsidered if the data change, the order in which individual pieces of information arrive is irrelevant. One benefit of using the TCS is that it frees the implementor from the need to handle the transaction time axis. Since this axis is an artifact of when information is reported to the system, rather than a feature of the domain itself, transaction time should not directly influence the decision-making.

2.1.1 Model of Time

The time model that I have implemented is a discrete time model. The time points are the integers from negative to positive infinity. The code itself does not require integer values, but some computational complexity results rely on the fact that the temporal model is discrete rather than continuous. Aside from the computational issues, the underlying model of time, whether discrete or continuous, is largely irrelevant to the discussion in this thesis. It is important that all time points are precisely specified.

Time ranges (except as intervals with exactly fixed endpoints) are not permitted. In other words, all times and extents must be explicitly specified. It is not possible, for example, to have "fuzzy" endpoints on intervals. The problem with allowing inexact time points is that it places the system's scheduling algorithm in an awkward position. Given the input situation in figure 2.3, where an inexact endpoint overlaps another input's endpoint, there are three qualitatively different orderings:

1. A's value changes before B's.

2. Both variables change values at the same time.

3. A's value changes after B's.

In order to schedule this situation while maintaining the static abstraction that we find so useful, the system would need to create three future branches. It is easy to see that this would greatly increase the computational load on the system. The real situation could be even worse than the qualitative analysis suggests. Since the

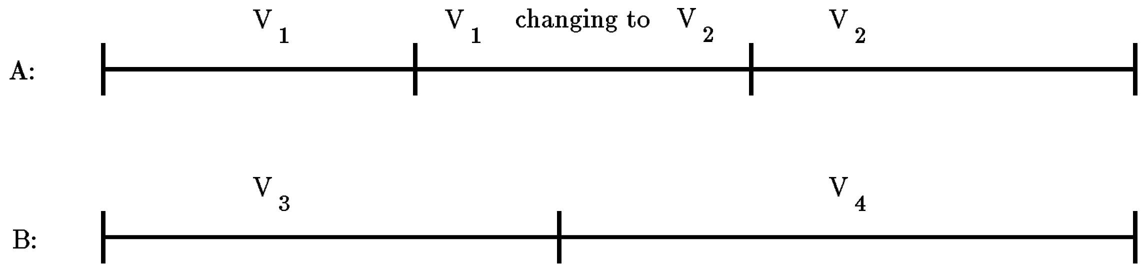


Figure 2.4: Inputs without Single Values

duration of the intervals might also be of interest, the number of possible points at which A's new value might begin is the same as the length of the fuzzy interval (using a discrete time model). The number of branching futures that are required could be equal to the duration of the uncertain interval. The TCS therefore does not support fuzzy endpoints on intervals.

This restriction is not as limiting as it seems, however. This is because of the freedom in the choice of value that is assigned to any particular interval. Unlike logic-based systems that allow only one of two choices for an interval's value, the TCS allows arbitrary values. Although in logic one could specify that either A or not-A is true over an interval, one could not specify that A changes to not-A over some interval. The notion of change is absent from logic-based systems, since they assume universal truth values. The example in figure 2.3 could be transformed into the situation shown in figure 2.4, where three fixed intervals are used instead of two fuzzy intervals. The three intervals correspond to two intervals with definite values and one with an uncertain or mixed value. It is then up to the user-specified decision procedure to handle this situation. I do not feel that the programming language should dictate an inescapable solution which has potentially disastrous effects on the efficiency of the problem solution. In section 8.1.2, I take issue with the approach of having a system-imposed and, of necessity, syntactic approach to the resolution of conflicts.

2.1.2 Model of Data

Since the data are time dependent, variables will need to represent not just data values, but also the time over which those values are valid. Each value has associated with it an expression denoting its temporal validity. For simplicity, this validity must be completely and unambiguously given.

The model of data used in this system divides temporal data into two distinct classes, that of points and intervals. Point variables contain values that are associated with a single instant of time. They have zero duration. Interval variables, on the other hand, consist of a series of values with a non-zero duration. Point and interval values cannot be mixed in the same variable. Point variables provide a natural representation for a data sample or a discrete action: each can be described by a value which has a single time associated with it. Each interval variable value has a datum and a beginning and ending time associated with it. They form a natural representation for states or continuous actions, namely a period of time in which certain values hold. The value of an interval variable must hold throughout each subinterval. This is a basic premise underlying the scheduling algorithm and the static abstraction.[1] It also allows TCS to combine adjacent intervals with the same value, reducing the amount of computation that must be done by process instances that use that interval as an input.

In theory it is not necessary to have both point and interval variables as base types, since they can be defined in terms of each other: points can be degenerate intervals, or intervals can be defined by pairs of points. In practice, however, the data manipulations and the ways values are used in calculations depend on the type of data they represent. Since this division into two classes has a practical significance, TCS supports the dichotomy with primitive elements.

The question of whether intervals or interval variables are open or closed will not arise in use, because reasoning is done only within intervals chosen so that interval variables will have only one value throughout the interval. By using the same interval determination in the variables and in the processes that reason about the variables, I finesse the problem of open or closed intervals, as far as interval variables are concerned.[2] The value of an interval value on the boundary between two values is undefined and inaccessible to any reasoning module. A query to the data base will return either the earlier or the later value at the user's option.

Unfortunately, when point variables are used, the question of whether intervals should be open or closed is unavoidable. It would be naive to imagine that one could avoid this issue entirely. One place where the problem surfaces is when considering whether intervals should be open or closed with regard to the point variable values that are accessible within a reasoning interval. In TCS all input intervals will be closed. This is the most general case, because reasoning programs can enforce an open-ended interval policy on themselves, by not examining point values at the extremes of an interval. Since all point variables have times associated with them, and a process instance knows its beginning and ending time, this is a trivial operation.

When point variables are the output of a module, the problem is trickier. A decision must be made and communicated to the control structure so that the value for a point variable on the boundary between two process instances can be set by the appropriate process. For instance, assume a point data value was located at time t and two process instances which produce values for this variable meet at time t as well. If both process instances will produce the same answer, then it does not matter

[1] This corresponds to the notion of a *property* in Allen's interval calculus [4].

[2] This is not a new idea. Allen [3, 4] and Ladkin [50, 49] restrict their temporal algebras to consider only intervals in order to finesse this problem.

which one is chosen to update the variable. In some cases, though, the answer will differ. A common example is in the context-dependent interpretation of a data value. If a data sample occurs at the same time the context changes, then the interpretation of that data value can change as well. The appropriate behavior at such a point depends heavily on the semantics of the variable and the reasoning process. With no naturally favored best choice, the desired behavior must be specified by the expert system designer.[3]

2.1.3 Past, Future and Now

The current time has an influence on the reasoning produced because of the difference between actions and interpretations. Actions can only be taken in the future. Past actions cannot be changed. Modules whose results are affected by the current time need a method of determining that time. TCS provides the current time as a point variable (`now`) as well as past and future as the interval variables `past?` and `future?`. The current time is a point value at the current time on the timeline whose value is the current time. (The value and time parts are the same). Past and future are intervals with boolean values of true over the past or future, respectively. They are accessible in the same manner as user-defined variables and cause reasoning to be updated as the time changes.

2.2 Model of Inference

The foundation of this approach is imagining an agent reasoning about things that change with time. Over any suitably chosen interval in time, the agent will have the inputs of its sensors available for inspection. This provides a means for dealing with data that are temporally current. This is represented by the diagram in figure 2.5. In addition to the current inputs, an agent is able to remember information, both raw input and inferred conclusions, from the relative past. The TCS provides a method for storing information that will always (subsequently) be available to the reasoning process. Such storage implements the *process state* of a reasoning process over time. Because the information concerns the relative past, such a process state variable is known as a *history* variable. This model is shown in figure 2.6. The history mechanism allows information to flow forward in time. In some cases, it is useful to have information flow backward in time as well. In this way, a process instance can have access to information from the relative future. Such information is stored in an *oracle* variable. An oracle is the dual of a history variable. The complete

[3]It *is* important for this to be specified, since process instances can be executed in any order. The underlying assumption derived from the determinism condition is that the order in which processes are executed does not matter. If adjacent process instances can each set a different value on the boundary, then a race condition occurs and the results can be unpredictable.

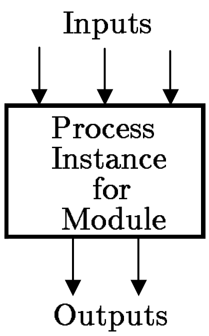


Figure 2.5: Schematic of a Process.

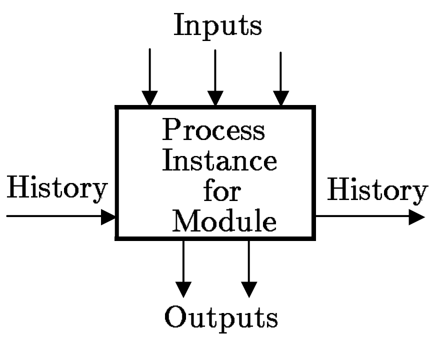


Figure 2.6: Schematic of a Process with History.

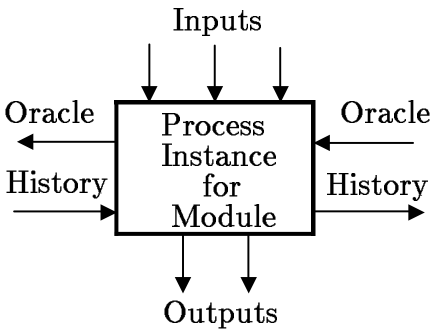


Figure 2.7: Schematic of a Process with Oracle.

model is shown in figure 2.7. The process state of a reasoning process is completely characterized by the contents of its history and oracle variables.

Inference takes place inside process instances, which take data from input values and transform them into output values. This transformation may also be influenced by the values of the process state variables. The access of a process to data in its input variables is restricted to the period of time over which it is instantiated. A module defines the interface and reasoning code used by processes. The mechanism by which modules are instantiated and scheduled is the topic of the next section.

2.2.1 Process Instances and Updating

A process instance is a module being executed in a specific time interval. The interval is chosen so that all of the interval variables in the input list have a single value. This enforces a state abstraction to make the reasoning easier. The only input variable values made available to the reasoning code are the ones current during the interval in which the process is being executed. Information from intervals other than the current one must be explicitly recorded in a process state variable by the module's code. The simplest method of using information from other intervals is to "remember" data from past time periods. Any information placed in one of these *history* variables will be available to the next process in the chain. This is shown graphically in figure 2.8. By programming the module to continue the propagation of remembered data, the information can be available at any arbitrary time in the future. Similarly, it is also possible to propagate information backwards in time through the use of *oracle* variables. This opens the area of reasoning by hindsight, which I discuss in the next section. Histories and oracles comprise the process state of a process or module.

Because the contents of the process state variables are under the direct control of a reasoning procedure, one can implement both selective memory and forgetting (the purging of memory) by deciding what to store in the histories. Although it would be possible to retain all of the information, it generally should not be done in the

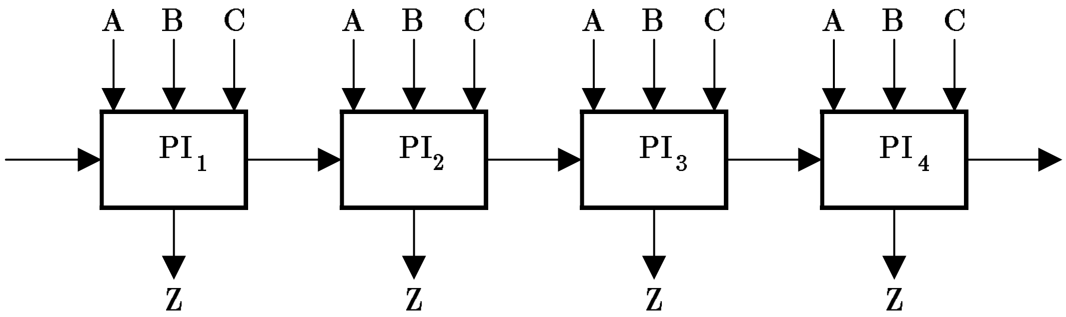


Figure 2.8: A Chain of Reasoning Processes.

interests of efficiency of reasoning.[4]

The contents of both the memory and the oracle variables that comprise the process' state are monitored by the control structure. This ensures that any information that changes the process state of a process causes any successor (or predecessor) process that (potentially) depends on that information to be reexecuted. This is needed to guarantee the correctness of the updating scheme. The ability to send information in both directions along the timeline allows a programmer to construct loops in the data-dependency and also non-terminating programs. As long as these loops have a fixed point, the processing will always terminate.

The final information that is made available to a process instance is the beginning and ending times of the interval over which it is being executed. These data are contained in the variables `begin_time` and `end_time`.

For example, consider a pharmacokinetic model which is used to estimate drug concentration in the blood. A typical model with two compartments is shown in figure 2.9. The difference equations that specify the behavior of a model of this type are given below:

$$\mathrm{A}_1(t+1) = \mathrm{A}_1(t) + \mathrm{D}(t+1) + \mathrm{k}_{21}\mathrm{A}_2(t) - \mathrm{k}_{12}\mathrm{A}_1(t) - \mathrm{k}_{\mathrm{el}}\mathrm{A}_1(t) \tag{2.1}$$

$$\mathrm{A}_2(t+1) = \mathrm{A}_2(t) + \mathrm{k}_{12}\mathrm{A}_1(t) - \mathrm{k}_{21}\mathrm{A}_2(t) \tag{2.2}$$

These equations can form the basis of the simulation used to calculate concentration values. Those values are the absolute amount of drug in the first compartment divided by its volume. Starting with no drug in the system, one can use the history of drug administration (D) to calculate the serum concentration at any point in time. This level is governed by the distribution between compartments (controlled by (k_{12} and k_{21}) as well as the elimination of drug from the system (k_{el}). The iterative formulas used depend on past information about the amount of the drug in the system. This

[4]Selective updating of the memory contents when a need for this only becomes apparent in the future is more difficult. It could be modeled using oracles as control variables to propagate the request for information back in time.

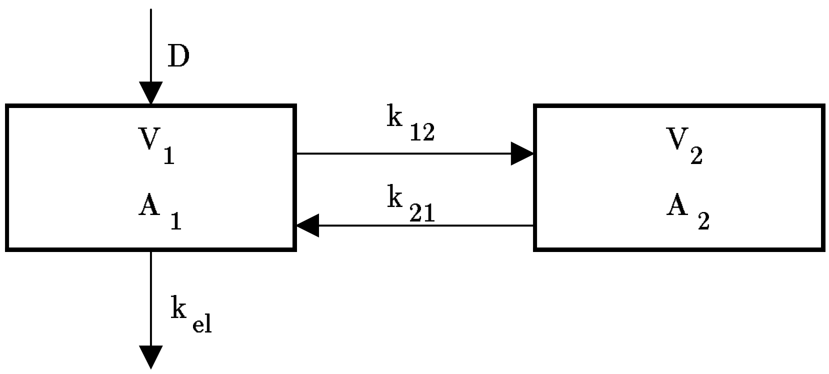


Figure 2.9: Two Compartment Pharmacokinetic Model

In this model "D" is the amount of drug entering the system, "V_i" is the volume of the i-th compartment, "A_i" is the absolute amount of drug in the compartment, and "k_i" is a rate constant describing the flow between compartments.

can be read directly from the equations above because $A_i(t+1)$ depends on the values of $A_j(t)$. The previous value of the amount in the system is needed to calculate the current amount.

If we wish to interrupt the process performing this calculation and later continue the calculation, it is necessary to record the values of A_i. A process that embodied this simulation would only need to keep the previous value of A_i to enable it to continue the calculation of the simulation. Only the most recent value need be remembered.

But why split this calculation? There are two reasons: First, some of the rate constants in a pharmacokinetic model are dependent upon outside factors such as patient weight, cardiac output or kidney function. These influences can be conveniently represented as interval variables. If these values are different over the course of the simulation, then it is most convenient to have the continuous simulation segmented into individual processes in which the parameters are constant. This relieves the simulation program of the need to check for the current values at each step of the iteration. This type of scheduling is done automatically by the TCS. The second reason to split a calculation is to limit its temporal extent. I address this concern in section 2.2.3.

2.2.2 Process Scheduling

The TCS system schedules processes to be run using heuristic methods to determine the intervals over which separate process instances should be created. It is possible

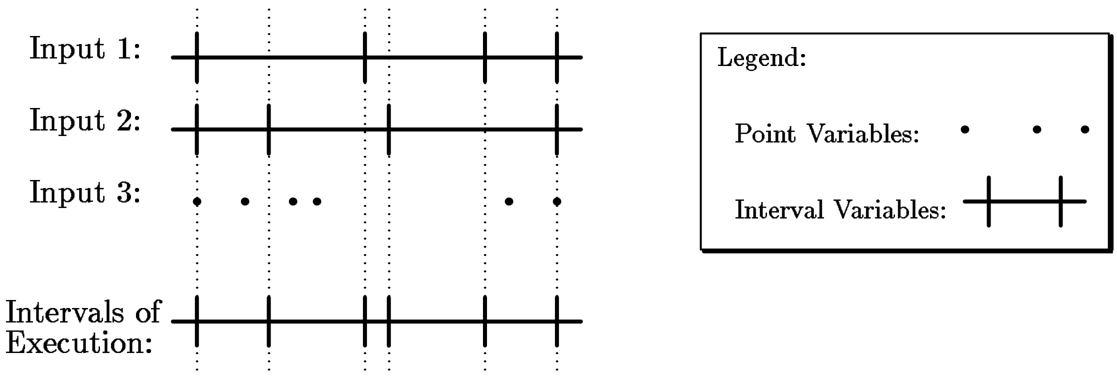


Figure 2.10: Scheduling Modules with Interval Variable Inputs

for the process instance itself to override this decision by changing the endpoints of its execution interval. So that data remain consistent, process instances can only decrease the length of the interval in which they are running. Any time not covered will cause the TCS to schedule a new process instance for the gap.

With Interval Variables Inputs

The simplest case is a process instance that has interval variables as inputs. The interval of execution is found by intersecting the intervals of all the interval input variables as illustrated in figure 2.10. TCS thereby assures the module function that the input variables will only have a single value. The intersection of data intervals is quite common in other systems as well, either implicitly in the definition of rule application (see [24]) or as an explicit calculation in rules (for example [32]). This approach can also be used for the case of mixed interval and point value inputs. Because the point variables do not have specific time extents associated with them, they do not influence the scheduling. This lets each process be executed in the longest interval in which the static abstraction of interval variables having a single value is valid. This method of process scheduling is very useful in practice. As a bonus, it allows one to add a temporal component easily to a static expert system simply by wrapping the static decision making units in modules with the appropriate inputs and outputs.

Without Interval Variable Inputs

If only point variables are inputs, there is no natural way of splitting the timeline. In this case a process instance is created which covers the entire timeline. If it is necessary to produce interval output, the process instance can adjust the length of its execution interval so that it covers the appropriate time span. The part of the timeline not covered is detected by TCS, which schedules one or more new process instances to handle that part of the original execution interval that has not yet been taken care of. For subsequent updates, the scheduling intervals are determined by examining the process intervals that were created by the user function.

This is a heuristic which attempts to minimize the amount of recalculation that is done. It is assumed that the user's process had some reason for shortening its execution interval and that the old process interval is a good first approximation to what is correct if there is a change in the data. This strategy can lead to a fragmentation of processes, but this fragmentation is limited to one level of the system. If the output values of the sequence of process instances can be combined, TCS automatically does this before the next level of reasoning processes are executed. Because interval values are combined at the next level, this approach minimizes the period of time over which new process instances need to be run. Without analyzing the user-supplied functions (an impossible task), the TCS has no way of knowing *a priori* whether adjacent process instances will calculate the same result. I feel that the cost of storing extra process instances when fragmentation occurs is worth the savings in not having to create multiple process instances if TCS combines intervals which turn out to have different values. Informal testing with both strategies indicates that this approach results in fewer process instances being created and run.

The decision to use this scheduling heuristic stems from a practical consideration encountered in the Arrhythmia Advisor project. Some of the data came regularly from an automated arrhythmia monitor. Since it was expected that many similar values could arrive in sequence, TCS needed an ability to "batch process" many point values in a row without incurring the overhead of process creation and scheduling. For that reason, multiple point values are available in each process instance. This was also the most flexible choice, since any modules which wished to handle data in smaller parts could be programmed to restrict their execution intervals.

2.2.3 Process Control Issues

In addition to the scheduling that is done automatically by the control structure itself, there are some additional control issues that the TCS user must address. One that was alluded to above was the need to adjust process duration, particularly when dealing with point input data. If point data are being abstracted into intervals, a decision must be made as to the extent of those intervals. Since this is a common procedure and the programming details are tedious, TCS provides a higher-level module which extends the value part of a point variable either until a new value occurs, or until a user-specified maximum duration is reached. The use of such persistence modules is quite common in TCS-based systems. It also forms the core of reasoning in Dean's Time Map Manager [17, 19].

Returning to the pharmacokinetic example used above, it is clear that the simulation program, once started, could in principle (and very easily in programming practice as well!) calculate the serum concentration out to infinity. This would, unfortunately, take a rather long time to execute, so some form of control would be more reasonable. This could either be a restriction on how long into the future (relative to the variable `now` which holds the current time) to calculate, or it could be a separate control variable (such as some interval variable) that is set either by a TCS module or by some external program using the TCS. The ability to stop the simulation while retaining sufficient state information to restart it later allows one to project the values into the future on an as-needed basis. For instance, the model could be set up so that it always had available predictions for the next eight hours. As the time of day advanced, the model would be restarted to maintain the eight hour forecast.[5]

2.3 Reasoning Units

As noted earlier, the TCS treats the calculation in a module as a black box. The user provides the reasoning function; TCS handles the job of applying the function as information changes and maintaining the database with the external values and the process state of the function. The function itself is not allowed to have any variable value storage that persists beyond its execution. All such information must be placed either in the external variables or in the process state of the reasoning function.

The module's function can exist in a static environment and not pay any attention to time. The environment is set up so that the current input values are passed to the function. When invoked the function receives the following information (see also figure 2.11):

1. The bounds of the time interval chosen by the scheduler.
2. The values of its input variables over that time interval. For interval variables, this is a single value. For point variables it is all of the points that occur during that interval (inclusive of endpoints).
3. The value of the history variables from the earlier invocation of the function.
4. The value of the oracle variables from the later invocation of the function.

The user function returns the following values:

1. The bounds of the time interval actually handled by the function.
2. The values of its output variables over that time interval. For interval variables, this is a single value. For point variables it is all of the points that occur during that interval. Endpoints are included or not according to declarations made at function definition time.

[5]Even in the Arrhythmia Advisor, with its lack of ability to take changing patient influences into account, some control over the execution window for the prediction was still necessary, or that implementation of the model could also run to infinity.

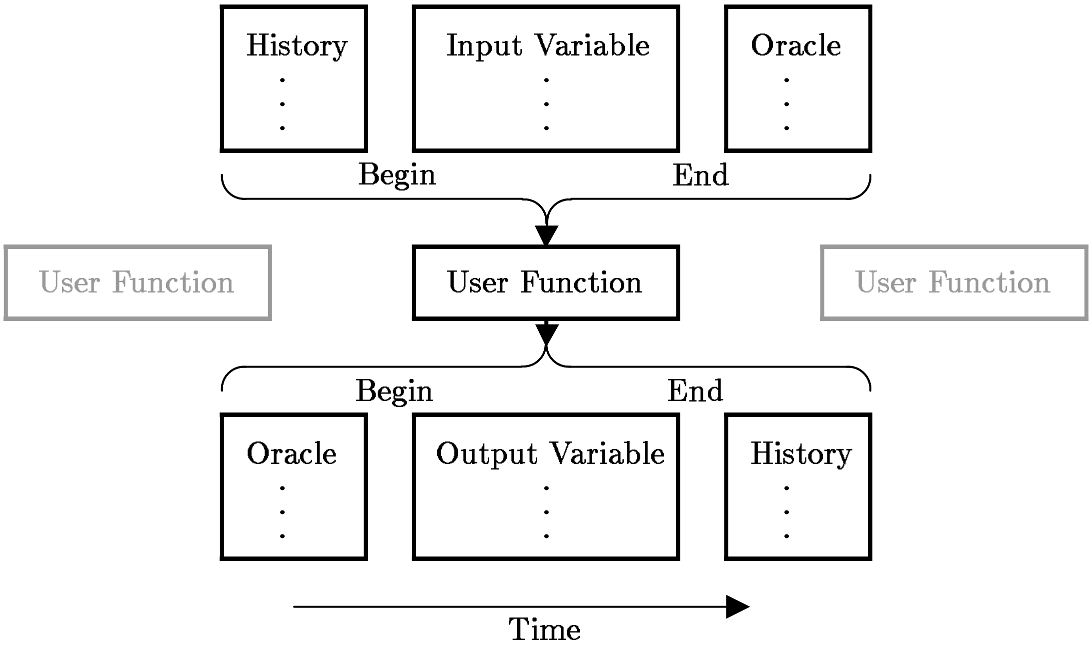


Figure 2.11: Inputs and Outputs of the Module Function

3. The value of the history variables to be passed to the following invocation of the function.

4. The value of the oracle variables to be passed to the preceding invocation of the function.

Of these, only the output variables *must* be set by the function. By default the execution interval and process state variables retain their values. The execution interval is further restricted to be less than or equal to the initially scheduled interval; i.e., the execution interval can only shrink, not expand. Expansion is ruled out because process instances are only given access to the data covering their initial execution interval. They are not allowed to set a value over an interval for which they have not received data.

2.3.1 Percent Example

As a concrete example of the use of histories and oracles, consider a module that calculates what percentage a given value is relative to the maximum value ever seen. This example is not particularly efficient, but it will serve as a simple introduction to the updating and scheduling behavior of the TCS.

The example module (figure 2.12) sets its output variable `percent` to be the current value of `value` as a fraction of the maximum value of `value` over all time. In order to do this, the largest previous value must be remembered in a process

```
(defmodvar value   :interval :initial-value 0)  ; An input value (≥ 0).
(defmodvar percent :interval :initial-value 0)  ; Output: Percent of maximum value.

(defmodule percent (value) (percent)               ; name (input) (output)
       ((:history past-max  :initial-value -1000)  ; for the past
        (:oracle future-max :initial-value -1000)) ; for the future

  (setq past-max   (max value past-max))                 ; update history variable
  (setq future-max (max value future-max))               ; update oracle variable
  (setq percent    (/ value (max future-max past-max)))  ; set output variable
  )
```

Figure 2.12: Histories and Oracles: Percent Program

history variable (called `past-max`) and the largest "future" value in a process oracle variable (called `future-max`). This allows the information from other time periods to influence the value in this period. It is the programmer's responsibility to see that these variables are properly handled.

The history value will initialize `past-max` for the process scheduled to run in the time interval *after* the current interval, and the oracle value will initialize the variable `future-max` for the process scheduled to run in the time interval *before* the current one. The initial value of -1000 is arbitrary. A sample run of this module is shown in figure 2.14, with the input, output, history and oracle values identified. Understanding this figure is the key to understanding the data-directed updating scheme using histories and oracles. As each process is run, the history and oracle values change, with the values being propagated along the timeline to temporally adjacent process instances. These changes cause the outputs of most process instances to change. In line 8, process instance P8 is run because the incoming oracle changed from "10" to "8." Since the function implemented by the module is a black box, the change in any incoming value leads to the recalculation of the function. Note that P8 does not change its output or the values of `future-max` or `past-max`. The ever-vigilant TCS then notices that there are no changes, and the updating process ends.

In the example, only P1, P2, P4 and P7 are triggered by the arrival of the external data. The other four process instances are created in response to changes in the history and oracle values. The creation of process instances for modules which depend on the output of the module `percent` is not shown, but they would be queued by the change in the output variable `percent` resulting from the execution of process instances P1–P7.

A total of eight process instances are created in order to calculate the value for the four input variable values. Using information about the history and oracle value propagation behavior of this module, it would be possible to solve the problem using fewer process instances (seven in this case[6]), by running process instances from left

[6]The process instance with input eight need only run once, since it is at the end of a propagation

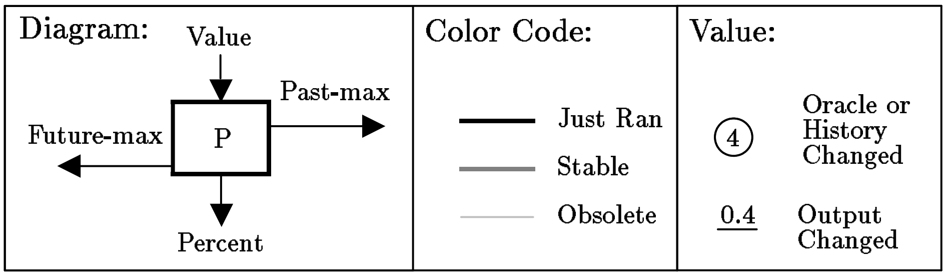


Figure 2.13: Histories and Oracles: Percent Process Key

to right and then back again. Although more efficient in terms of number of process instances run, this type of clever scheduling requires more knowledge of the behavioral characteristics of the system than is available to TCS. It also requires a global plan for the scheduling of the module.

In TCS I opted for the simpler solution of making all the scheduling decisions locally, at the cost of more updating than strictly necessary in certain cases.[7] In a realistic situation where the data is being added over time, there is always the possibility that a new value larger than all others will be added, necessarily causing recalculation over the entire timeline. This is inherent in the function programmed in the module. (A more realistic application would limit the time over which the percentage calculation is being performed.) One of the strengths of TCS is that the resolution of efficiency issues is left in the hands of the implementor, who can best determine how to trade computational effort for more precise answers.

2.4 Using the TCS

The history and oracle features allow the functions to maintain state information over time. They provide a communication path for information from previous or subsequent invocations of the function along the time path.

In a conventional programming language, information can only be passed forward in time. All state information used by a function must have been calculated before the function which uses this state information is invoked. There is no alternative to this because the temporal dimension is not represented explicitly in the language. It is implicit in the calling sequence of the function and corresponds to the real time in which the function is being executed.

---

chain and has all the history data.

[7]The current implementation of TCS uses a single scheduling strategy for all modules. One avenue for extending the paradigm involves allowing the user to provide advice to enable more efficient scheduling.

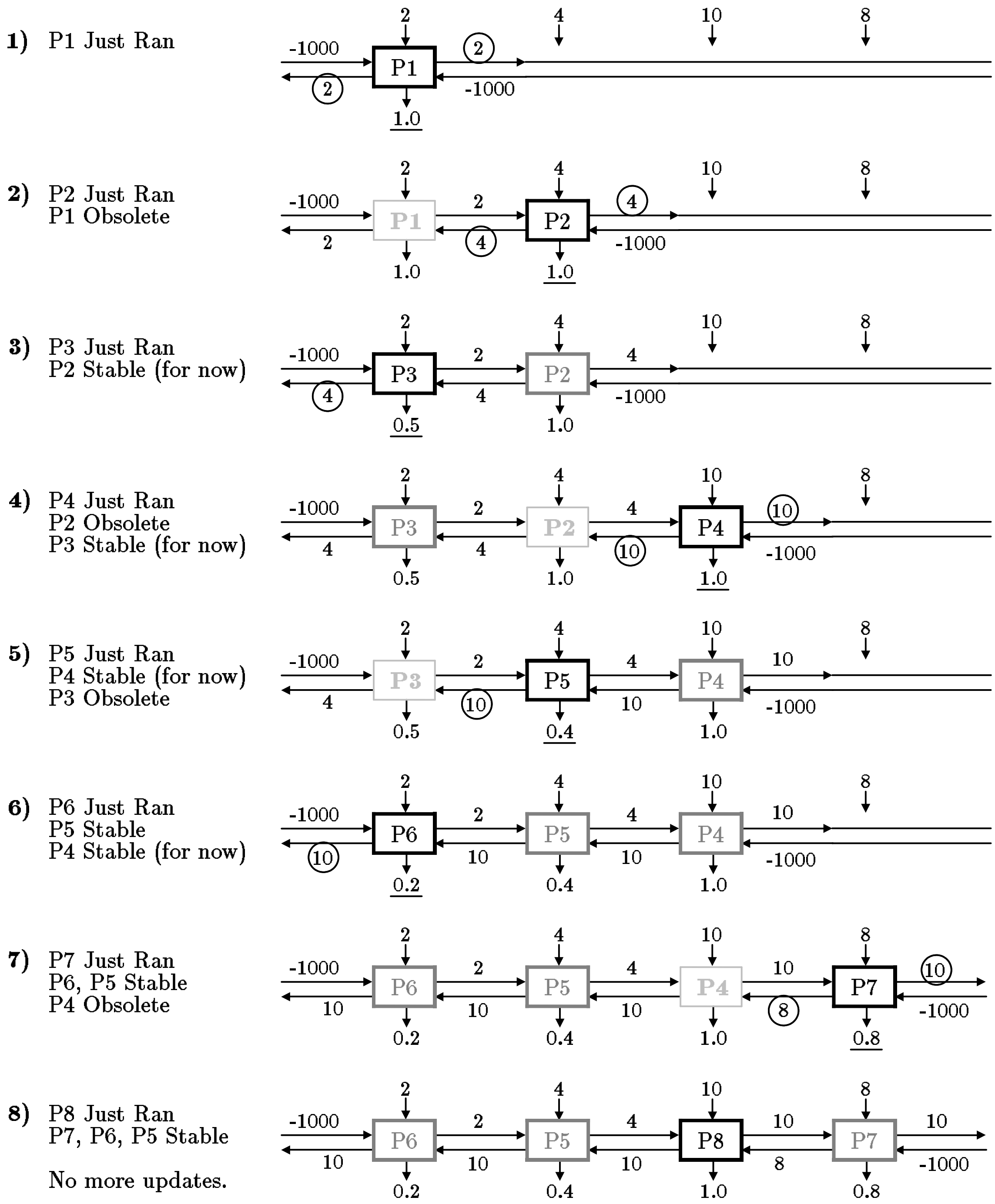


Figure 2.14: Histories and Oracles: Percent Process Trace

For symbol key, see figure 2.13. Underlined numbers are new output values. Circled numbers are newly propagated history or oracle values.

Although TCS has the same constraints at low level, the ability to reinvoke functions as needed allows one to hide this limitation from the user. Because the timeline and the resulting process state are represented explicitly, the order in which the functions are actually executed does not have to match the order of the timeline. If oracles are used, the order cannot be chronologic. One consequence of this architecture is that functions can potentially be invoked more than once in the same time interval. Since all functions must compute a deterministic result based only on their inputs, histories and oracles, multiple invocations of the function with the same data, must produce the same answer. This means that the answer is independent of the number of function invocations in any time interval. This allows mutually dependent invocations (i.e., ones using both histories and oracles) to be handled. One time interval is arbitrarily chosen for execution. The process state information is passed to the function for the adjacent time interval. If its process state variables change, then the function is invoked again over the first interval. The cycle continues until the mutually dependent functions reach a steady state wherein no more changes are being made to their outputs. This stable state must exist for any such functions in order for the TCS-based system to terminate its reasoning. This mechanism applies equally well to cycles that occur between modules linked by TCS variables and to those linked by the history/oracle mechanism.

The ability to write mutually dependent functions means that it is possible to construct non-terminating loops. It is assumed that in any well-ordered monitoring system this will not happen. The only safeguard, however, is to use care when setting up reasoning cycles.

2.5 Formal Definition of Module Function

A TCS application is created by defining the modules that do the reasoning. In the general case, the processing function in each module has three sources of input:

1. **Input Variables.** Input variables are used for inter-module communication, as well as for the interface between the TCS code and the outside world. Variables may take on arbitrary programmer-specified values. Each input variable holds the portion of the value of the variable valid over the execution interval of a process instance (PI). For an interval variable, this is a single value for the entire execution interval.

2. **Execution Interval.** Each process runs in an interval as scheduled by TCS. The endpoints of this interval are available for examination (and restricted modification) by process instances.

3. **Process State.** The process state provides a link along the time axis between temporally adjacent process instances. These are PIs whose execution intervals fulfill Allen's MEETS relation with the current PI. There are two subtypes:

(a) **History.** Transmits information forward in time. It provides the process state from the previous adjoining process instance along the timeline.

(b) **Oracle.** Transmits information backward in time. It provides the process state from the next adjoining process instance along the timeline.

The output of each module function can also be divided into three classes:

1. **Output Variables.** Output variables are used for inter-module communication, as well as for the interface between the TCS code and the outside world. They may be assigned arbitrary values. Each output variable can be assigned a value only for the execution interval of a process instance (PI). For interval variables, this must be a single value for the entire execution interval.

2. **Execution Interval.** The beginning and ending times of the execution interval can be modified within the following restrictions:

   (a) The begin time must be strictly less than the end time.

   (b) The new execution interval must not extend beyond the original execution interval. In Allen's calculus, this means the new execution interval must EQUAL, START, END or be DURING the original execution interval.

   By default, the endpoints remain unchanged.

3. **Process State.** The values of the process state variables are under the control of the program. By default, the input values of the process state are propagated unchanged. Using these variables is the only way for process instances to gain access to information outside their own execution intervals.

2.5.1 Terminology

In order to specify the functions formally, I introduce the following definitions:

Variable:	$\mathcal{V}$	Begin Time:	$\mathcal{T}_{begin}$
Point Variable:	$\mathcal{V}_{pt}$	End Time:	$\mathcal{T}_{end}$
Interval Variable:	$\mathcal{V}_{int}$	Execution Time:	$\mathcal{T}_{exec} \equiv \langle \mathcal{T}_{begin}, \mathcal{T}_{end} \rangle$
Process State:	$\mathcal{S}$	Zero or more X's:	X^*
History:	$\mathcal{S}_{hist}$	One or more X's:	X^+
Oracle:	$\mathcal{S}_{orac}$	Identity Function:	$\mathcal{I}d$
		Null Function:	$\emptyset$

2.5.2 Formal Definition

Each module contains a function described by the applications builder which implements the reasoning in such a module. The general form of this function is then

$$\text{Module: } f : \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^* \mapsto \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^*$$

For analysis, the module function f can be decomposed into three components, each of which handles a different part of the output. This decomposition yields functions that calculate the output variable values (f_v), the execution time (f_x) and the process state (f_s):

1. $f_v : \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^* \mapsto \mathcal{V}^+$
2. $f_x : \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^* \mapsto \mathcal{T}_{exec}$
3. $f_s : \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^* \mapsto \mathcal{S}^*$

I will use these formal definitions in a later chapter to describe certain restricted types of modules with special properties. For example, some modules do not need to modify their initial interval of execution, and so have the identify function for f_x. Since this is the default behavior of modules defined in TCS, no program code is required to specify this behavior. For such modules, the programming task is therefore simplified.

Chapter 3

Example: Pharmacokinetic Model

The top-level structure of a system implemented using the TCS consists of variables linked through modules. The decision is modeled as a process over time, and divided into individual process instances in a manner that is convenient to the calculation. The inputs that can be used are data associated either with a single point in time (points) or with an extension over a period of time (intervals). The TCS design philosophy assumes that intervals are relatively stable, so that it makes sense to use different values of intervals as the basis for scheduling process instances. This conveniently eliminates the need for most processes to consider the effects of time at all.

To illustrate the concepts, I describe the implementation of a pharmacokinetic model for lidocaine. This is a mathematical model, developed by statistical methods, that predicts the blood concentration of the drug lidocaine. It is a two-compartment model, with the structure shown in figure 3.1. This is the same model introduced in the previous chapter. Each compartment has a volume of distribution (V) and an amount of drug in the compartment (A). The blood concentration is the amount of drug in compartment 1 divided by the volume of compartment 1. Drugs enter the first compartment as a function of time (D(t)). Movement of the drugs in the model is controlled by rate constants linking the compartments (k_{12}, k_{21}) and providing a pathway for the elimination of the drug (k_{el}). The behavior of the model is governed by the following discrete time difference equations:

$$\mathrm{A}_1(t+1) = \mathrm{A}_1(t) + \mathrm{D}(t+1) + \mathrm{k}_{21}\mathrm{A}_2(t) - \mathrm{k}_{12}\mathrm{A}_1(t) - \mathrm{k}_{\mathrm{el}}\mathrm{A}_1(t) \tag{3.1}$$

$$\mathrm{A}_2(t+1) = \mathrm{A}_2(t) + \mathrm{k}_{12}\mathrm{A}_1(t) - \mathrm{k}_{21}\mathrm{A}_2(t) \tag{3.2}$$

Functionally, the model takes external information about the drug doses ($\mathrm{D}(t)$) and provides information about the drug concentration ($\mathrm{A}_1(t)/\mathrm{V}_1(t)$). Lidocaine can be administered as a continuous infusion over some time period or as a single shot (bolus) at a particular time. The drug inputs are divided into an interval input for the continuous infusion and a point variable for the bolus. The output for this example will consist of samples of the drug concentration, and will be represented by a point

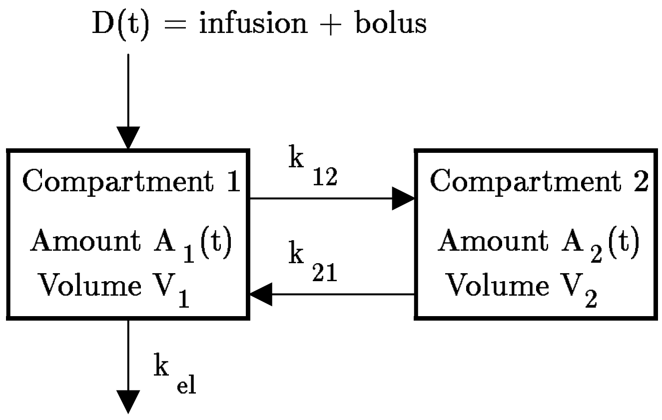


Figure 3.1: Structure of a Two-Compartment Model

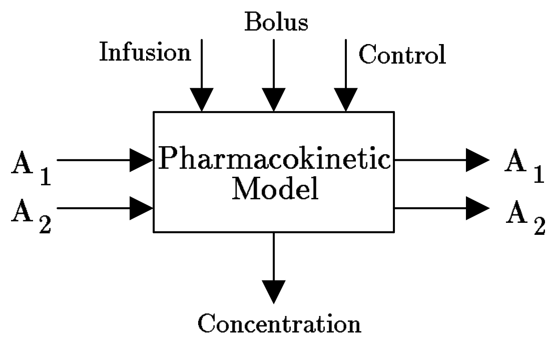


Figure 3.2: Schema for Pharmacokinetic Model Simulation

variable. Other potential outputs could be samples produced in response to an external trigger (also point variable output) or intervals in which the concentration is in important ranges.

As noted before, in order to run a simulation of the model, internal state information (the contents of $A_1(t)$ and $A_2(t)$ must also be maintained. This calculation is shown schematically in figure 3.2 and could be coded as shown in figure 3.3. The code separates the external information about doses and concentrations (the inputs and outputs) from the internal state variables containing the amounts (the history variables). It is necessary to keep the amount of drug in each compartment available so that the simulation will work with multiple process instances. If the infusion rate changes, then a new process instance will need to be created to maintain the static abstraction of interval values. Note that the code in figure 3.3 does not need to consider the temporal limits of the value of the variable `infusion`, since its validity is implicitly limited by the duration of the process instance (`begin_time` to `end_time`).

```
(defmodvar infusion :interval :initial-value 0)
(defmodvar bolus :point)
(defmodvar concentration :point)

(defconstant k12 ...)
(defconstant k21 ...)
(defconstant kel ...)
(defconstant v1  ...)

(defun get-bolus-value (bolus time)
  ;; Returns bolus amount for  time or zero if no bolus is given then.
  ...)

(defmodule pk-model (infusion bolus) (concentration)
        ((:history a1 :initial-value 0)
         (:history a2 :initial-value 0))

  (loop for time from begin_time to (- end_time 1)
        initially (setq concentration nil)
        do (psetq a1 (+ a1 infusion (get-bolus-value bolus time)
                        (* k21 a2) (- (* k12 a1)) (- (* kel a1)))
                  a2 (+ a2 (* k12 a1) (- (* k21 a2))))
           (pushpoint (/ a1 v1) time concentration)))
```

Figure 3.3: Code for Pharmacokinetic Model Simulation

The time units for this simulation are minutes. For convenience, the step size of the simulation is one time unit, although this is not required. The function `pushpoint` is a TCS construct which creates a point value data structure from a value and a point and pushes it onto a list of such points.

It is, however, necessary to provide special time-aware processing to extract the value of a bolus.

The output of the module, `concentration`, is a series of individual point variables, corresponding to the estimated blood concentration of the drug at different times. With one value produced per minute, this process will create many data points. The builder of a real system may want to perform an abstraction operation or use only every n-th value to reduce the number of values produced. Since there is the potential for producing many similar values, the process-scheduling algorithm allows for bulk processing of point values, since greater efficiency is possible if new process instances do not need to be created to handle each of a large number of individual points.

There is only one problem with the module as currently programmed: namely, the simulation will try to cover the entire timeline from negative to positive infinity. This would clearly take a long time to calculate, and would not be suitable for a real system. In the real world, there is only a certain amount of time over which one requires the information provided by the simulation. The time of simulation can be restricted by the addition of another external variable used for control purposes. The simulation would then only operate during the period(s) the control variable allowed. Since control can easily be achieved by using a TCS variable, adding the control information does not require any changes to the underlying structure of the TCS. The value of the control variable can either be set externally to this process or it can be produced internally. Since the control information is available as a regular variable, an expert system can set it as a result of the inferences that the program itself makes. A simple control scheme would involve only doing the calculation for a limited span around `now`, a capability that is easily implemented using persistence and anticipation modules.[1] The pharmacokinetic system, modified to use a control variable, appears in figure 3.4. The output from a system using standard lidocaine parameters from Thomson [84] is shown in figure 3.5.

The control variable used to limit the execution of the pharmacokinetic model can either be supplied as an input from outside the TCS, or it could be computed. One such computation, which makes use of the system time variables `now` and `past?`, is shown in figure 3.6. The variable values for such a control structure are shown in the graph in figure 3.7, for a starting time of 0 and a current time of 600. This sample control strategy limits the calculation in the past to "interesting" times as defined by the variable `start-time`. In the future, projections will be made for 500 time units. Landmark times combined with the current system time can be used to control the operation of other modules. The ability to program the control into the system allows a great deal of flexibility to handle domain- and application-specific problems. For example, one could imagine an "as needed" control system where the control variables were set in response to an outside request from the user.

[1] An only slightly more complicated control would use an offset from `now` to control projection in the future and the time since the beginning of the consultation in the past. This would require the TCS-based system to include the initial consultation time as one of its variables.

```
(defmodvar control :interval :initial-value nil)

        ⋮

(defmodule pk-model (infusion bolus control) (concentration)
        ((:history a1 :initial-value 0)
         (:history a2 :initial-value 0))

  (if control
      (loop for time from begin_time to (- end_time 1)
            initially (setq concentration nil)
            do (psetq a1 (+ a1 infusion (get-bolus-value bolus time)
                               (* k21 a2) (- (* k12 a1)) (- (* kel a1)))
                      a2 (+ a2 (* k12 a1) (- (* k21 a2))))
               (pushpoint (/ a1 v1) time concentration))
      (setq a1 0 a2 0 concentration nil)))
```

Figure 3.4: Control Added to Pharmacokinetic Program

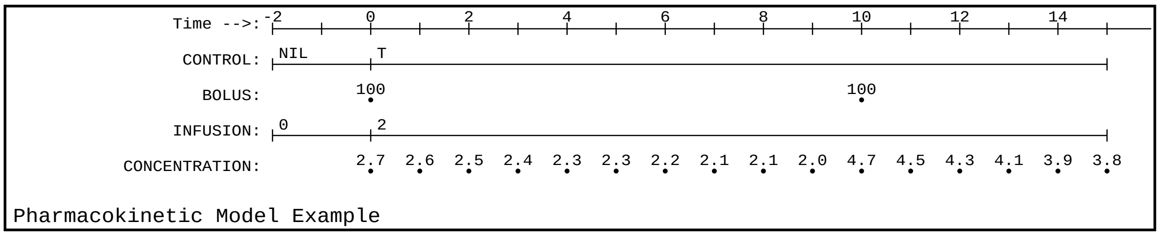


Figure 3.5: Pharmacokinetic Program Output

```
(defmodvar start-time :point)      ; Externally set to t at the starting time.

(defmodvar ctrl-1 :interval :initial-value nil)  ; Internal variable.
(defmodvar ctrl-2 :interval :initial-value nil)  ; Internal variable.


    ;; The arguments to a defpersistence form are the name of the module to
    ;; be created, a point variable input, an interval variable output, and options
    ;; that specify the length of persistence in the absence of new data, as well as
    ;; default values to cover the periods when no valid data is available.

    ;; ctrl-1 is  t from the starting time until infinity.
(defpersistence ctrl-start start-time ctrl-1 :persistence :infinity :default nil)

    ;; ctrl-2 is t for 500 time units from now into the future.
(defpersistence ctrl-future now ctrl-2 :persistence 500 :default nil)

    ;;  control is t from the starting time until now because the value of
    ;;    ctrl-1 in the past (which is t from start-time to infinity).
    ;;  control is also t for 500 units after now because of ctrl-2
    ;;    in the future.
(defmodule model-control (ctrl-1 ctrl-2 past?) (control)
     (setq control (if past? ctrl-1 ctrl-2)))
```

Figure 3.6: Simple Control Strategy

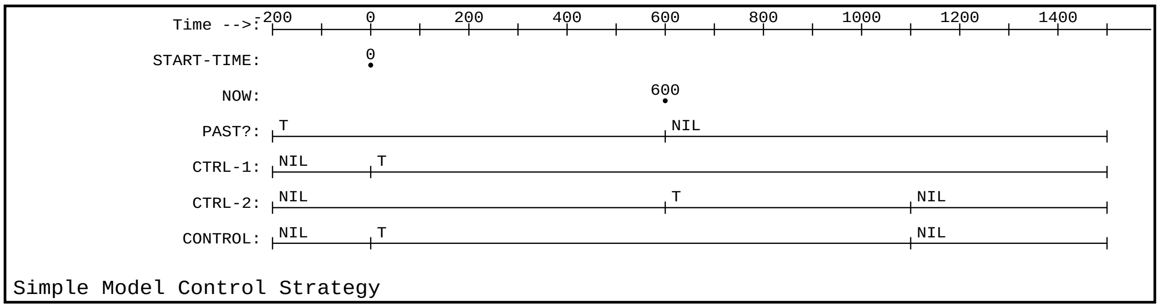


Figure 3.7: Control State at Time 600

The use of a control or data manipulation rule that uses one source of information in the past and a related, but often different, one in the future is a programming paradigm that recurs frequently in a TCS system. A common application is the use of the actual therapy in the past and the recommended therapy in the future when trying to project the effects of the advice. This arises in the use of a pharmacokinetic model. The model state must reflect the actual treatment in the past in order to form a valid basis for understanding the patient's current state. But the future expectations should be generated using the advice the program provides. In a situation where the output of the model is used in the process of designing the dosage regimen, as it was in the Arrhythmia Advisor [73] or in the extensions to the Oncocin project [39], this dichotomy around the current time (or next feasible decision point) is essential to the correct functioning of the system.

We can continue the expansion of this example by making the model parameters depend on the patient's state. For some drugs, estimates of the *a priori* effects of different medical conditions on the parameter values have been published [84]. In this study relevant parameters included the presence of congestive heart failure and liver disease. The use of contextual information changes the rate "constants" of the model into variables whose values depend on the state of the patient. Certain aspects of the state can change over time. For example, a digitalis (another cardiac drug) model [37] uses a measured parameter, creatinine clearance, to calculate the value of one model parameter. Since the measured patient parameter can change over time, so can the model parameters. Complicating matters is the problem that not all of the contextual information will be available initially. Liver disease may be detected only on the basis of a more complete diagnostic workup than would be possible before treatment begins for arrhythmias. The ability of the TCS to update past information as the underlying basis for those decisions changes is an important advantage in handling this situation. As more complete information becomes available, the parameters can be adjusted and the affected blood concentration estimates can be revised.

The back-and-forth nature of the decision-making becomes more pronounced if more sophisticated mathematical modeling techniques are applied. The model parameters, which are derived from population standard estimates, can be refined or adapted to an indivdual patient by comparing the estimates to measured samples. [36, 37, 39] It is important to update even the past drug concentration estimates because of the potential effects on the current and future decision-making process. For example, one of the rules from the arrhythmia advisor required a switch from lidocaine to procainamide if the patient appeared not to respond to therapeutic concentrations of lidocaine. The diagnosis of lidocaine resistance depends on the lack of response to an adequate amount of the drug. If revisions of the model parameters indicate that the model produced concentration estimates that were significantly higher than the actual values, a previous decision that the patient had lidocaine-resistant arrhythmias would need to be revoked. The prospective impact of revoking this past decision would be to return lidocaine to the armamentarium of therapeutic interventions. If this past decision is not reexamined and revised, then the old (and incorrect) observation of lidocaine resistance will prevent the use of a potentially useful drug. Note also that, because the model will have been modified, the future dosage recommendations should result in an effective drug concentration, because the dosage will be higher.

Chapter 4

Types of Reasoning Module

Using the formalism described in chapter 2, we can distinguish among the characteristics of different types of reasoning over time. This differentiation is determined by the types of variables that are used in the reasoning modules. In this section, I will describe several more abstract reasoning modules. In addition to the textual description, I will provide a schematic diagram of the reasoning flow over time, as well as a description in terms of the form of function embodied in the module functions.

A general overview of the interrelations of four classes of module can be seen in figure 4.1. These four basic types are characterized by the relation between the type of input and output variable. Two have the same type (both point or interval). The other two perform transformations between the two types (point to interval or vice versa). As one might expect, the modules which perform the type transformation have more complicated implementations. I detail each of these basic types, as well as other derivative types, in the following sections. To aid the comparison with fully general modules, I repeat the three functional parts of a standard module here:

1. $f_v : \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^* \mapsto \mathcal{V}^+$ Calculates output variable values.
2. $f_x : \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^* \mapsto \mathcal{T}_{exec}$ Calculates process instance execution interval.
3. $f_s : \mathcal{V}^+ \times \mathcal{T}_{exec} \times \mathcal{S}^* \mapsto \mathcal{S}^*$ Calculates process state (histories and oracles).

4.1 Rules

Rules are the most straightforward of the reasoning types to be explored, because they exhibit no overt time dependence. When viewed as a statement in a logical form, rules possess universal validity. The temporal aspect of a rule can be handled completely outside the rule itself. Figure 4.2 shows a typical rule schematic for a static rule. In the formal analysis, I discuss both static and temporal rules.

4.1.1 Static Rules

The functional definition of static rules is quite simple:

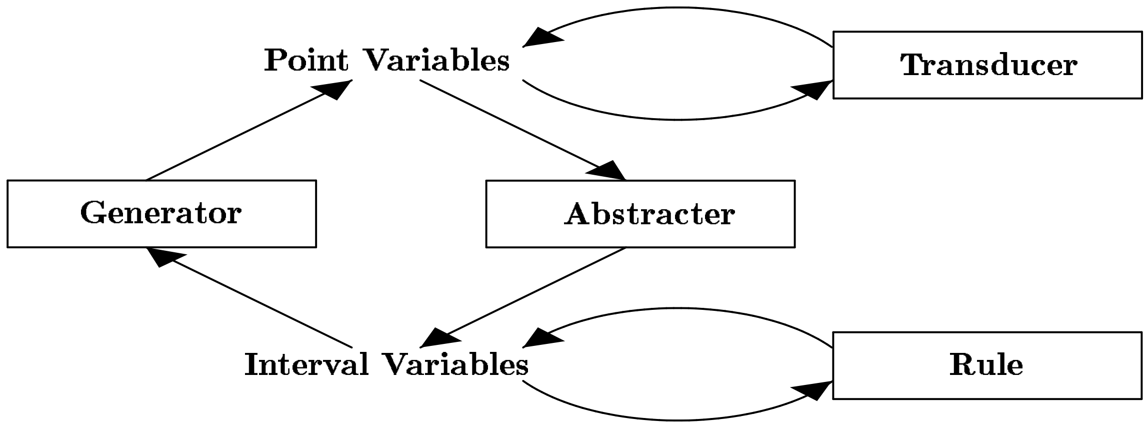


Figure 4.1: Basic Types of Reasoning Modules

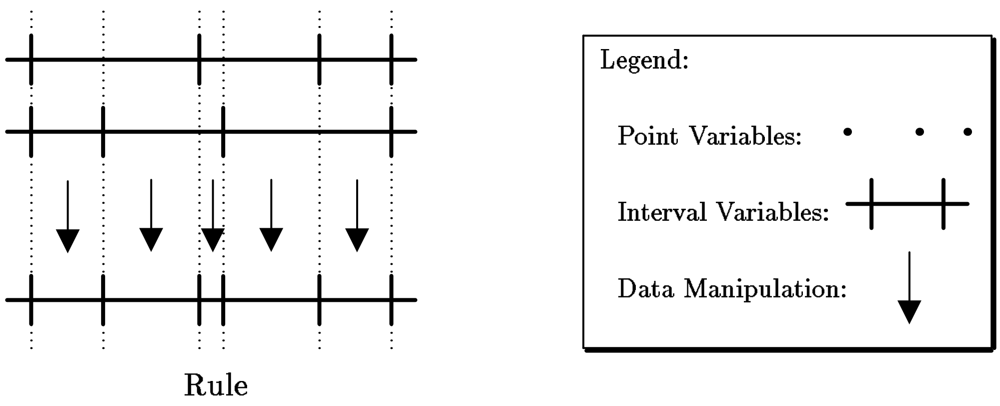


Figure 4.2: Schematic of a Rule Module

1. $f_v : \mathcal{V}^+_{int} \mapsto \mathcal{V}^+_{int}$

2. $f_x : \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec} \qquad (f_x \equiv \mathcal{I}d)$

3. $f_s \equiv \emptyset$

Since there is no change in the execution interval, and there are no functions that rely on internal state variables, this rule can only have effects inside the interval in which it is scheduled. The computational consequence is that the maximum number of process instances that can be spawned is limited to the length of the time interval over which an input value changes. The time interval affects the scheduling because of the other input variables.

4.1.2 Temporal Rules

The functional definition of temporal rules adds a function to manipulate the process state:

1. $f_v : \mathcal{V}^+_{int} \times \mathcal{S}^+ \mapsto \mathcal{V}^+_{int}$

2. $f_x : \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec} \qquad (f_x \equiv \mathcal{I}d)$

3. $f_s : \mathcal{V}^+_{int} \times \mathcal{S}^+ \mapsto \mathcal{S}^+$

Temporal rules use state information from previous or future process instances. This means that any change in an input variable can potentially affect all other intervals, as was the case in the percent calculation used in chapter 2. Realistic applications will most likely limit the extent of influence of any particular input interval. Since the applications programmer supplies f_s, the computational complexity of a temporal rule is under the control of the developer. The difference from a static rule is that the function f_s is defined and f_v has added a reference to the process state ($\mathcal{S}$).

4.2 Transducers

Transducers are atemporal reasoning units whose primary characteristic is that they convert the value part of their point variable input to a different value at the same time. There is thus no temporal processing involved, since the time part of the point value is not even examined by the function that does the reasoning. The function defining the value conversion is just applied to each point variable in turn. Schemata are shown in figure 4.3.

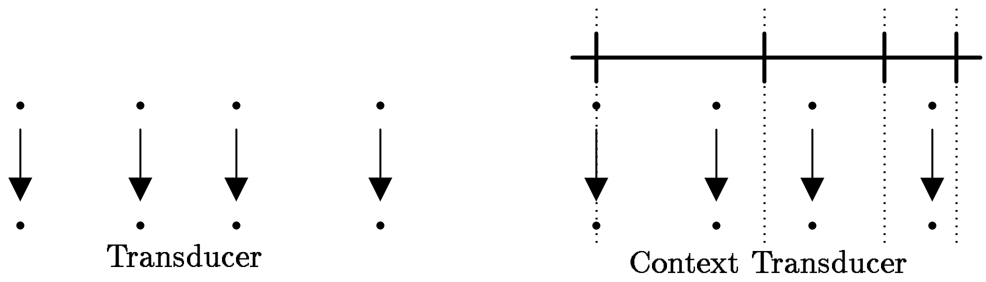


Figure 4.3: Schemata of Transducer Modules

4.2.1 Simple Transducers

The functional definition of a simple transducer is similar to that of a static rule. The only difference is the substitution of a point for an interval variable:

1. $f_v : \mathcal{V}_{pt} \mapsto \mathcal{V}_{pt}$
2. $f_x : \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec} \qquad (f_x \equiv \mathcal{I}d)$
3. $f_s \equiv \emptyset$

This module maps point values by considering only the value of the individual point variable and leaving the time part unchanged. The difference in which part of a point variable is processed defines the difference between a context-sensitive transducer and a generator.

4.2.2 Context-Sensitive Transducers

Context-sensitive transducers add interval variables which provide context:

1. $f_v : \mathcal{V}_{pt} \times \mathcal{V}_{int}^{+} \mapsto \mathcal{V}_{pt}$
2. $f_x : \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec} \qquad (f_x \equiv \mathcal{I}d)$
3. $f_s \equiv \emptyset$

This module is a transducer whose processing function determines the output value based on the value of a single point variable and the values of additional interval variable inputs. The presence of the context variables provides an easy method for modifying the point evaluation function over time without any direct reference to time in the calculation. The tables used in Fagan's VM for generating acceptable boundary ranges for parameter values are transducers of this type. Depending on the particular context in which the mapping from raw input data to a qualitative evaluation is done, the thresholds are different. Although the context can change over time, the extent

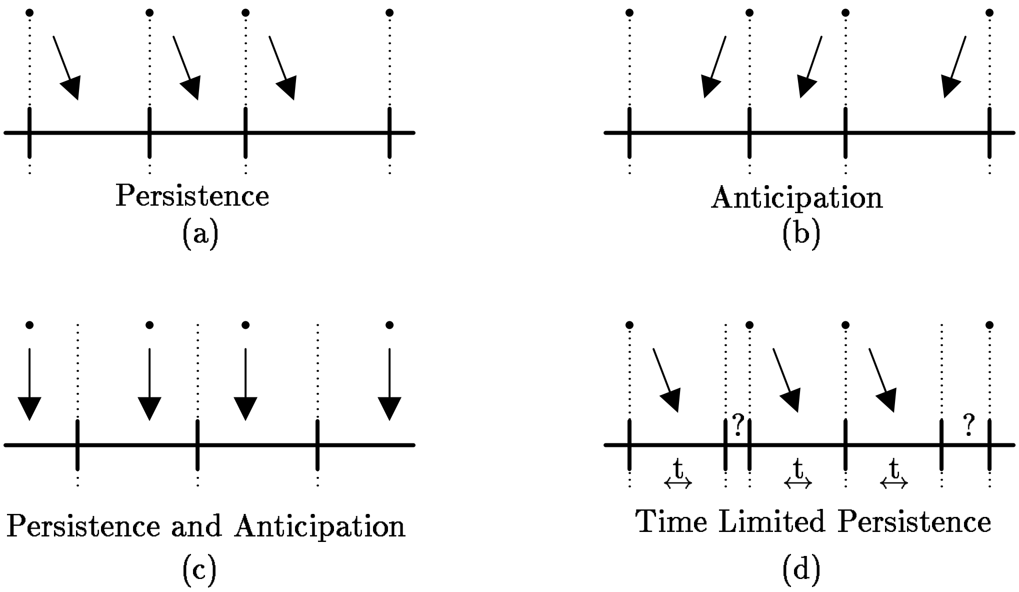


Figure 4.4: Schemata of Abstractor Modules

of the context's validity and the particular time at which a point datum is evaluated are not considered by the transducer function. It simply receives the value part of the point variable along with the values of the relevant contexts and produces the value part of the output point variable.

The common feature of all transducers is that they operate only on the value part of point variables.

4.3 Abstractors

Abstraction is the process of changing the form of the data from a collection of point variables to a set of intervals. This reflects an expansion of the duration of validity of point data samples. This is an inductive process and is not so straightforward as the deductive processes used in executing rules. As the time extent of the data expands, it becomes a more abstract entity. By setting up the boundaries and attaching an interpretation to the individual sample points, the abstraction process takes a loose collection of data and produces a more ordered description.

As an example of abstraction, consider the examination of a series of daily reports of rainfall and the resulting conclusion that a drought was in progress. A drought covers a longer time interval and is a more abstract description than a single dry day or even a sequence of two or three dry days. Furthermore, by attaching the label "drought" to the condition, the abstraction allows the application of more general knowledge associated with droughts. This knowledge could also be applied on the basis of an individual analysis of the run of dry days, but the application is much more tedious. Moving up the abstraction hierarchy in the reasoning also increases the size of time interval covered. More abstract descriptions tend to cover a larger time span than the more detailed descriptions.

TCS provides only the simplest form of persistence, namely the extension of data values for a fixed period of validity. Other researchers have investigated the deeper relation between causation and the notion of persistence that this simple model captures [18], as well as having proposed more rigorous models of the probalistic nature of the resulting interval value [34]. In the experiments I have conducted so far, the simple approach used by TCS has been adequate.

The expansion of the time covered can occur in both directions. I term the extension into the future *persistence* and into the past *anticipation.* Schemata are illustrated in figure 4.4. This specification of a simple persistence as shown in figure 4.4a requires only the specification of a maximum duration interval. In the case of anticipation, the resulting abstracted interval is projected into the *past*, rather than the future (see figure 4.4b). These can also be made into a combination which assumes that the abstract intervals extend both into the past and into the future, in effect splitting the timeline between adjacent sample points (figure 4.4c). The first three types of abstractor are distinguished only by the relationship between the time of the point variable's values and the endpoints of the resulting intervals, as illustrated in the diagram.

One can also limit the persistence of information from a sample point to a maximum interval. This time-limited abstraction can be used to implement reasoning in which the data become too old to be useful or safe to use in calculations (figure 4.4d).[1]

4.3.1 Time-Limited Persistence

In the formal analysis I will only address the last type of single point abstractor:

1. $f_v : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \mapsto \mathcal{V}_{int}$
2. $f_x : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \mapsto \mathcal{T}_{exec}$
3. $f_s : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \mapsto \mathcal{S}_{hist}$

Persistences provide a simplification for the user because he need only specify one parameter, the maximum duration of a point value, in order to define a time-limited abstractor. This parameter is sufficient then to generate code which implements all three functions shown above! The detailed implementation is complicated because the duration of the output $\mathcal{V}_{int}$ is influenced by any points during the system-chosen execution interval $\mathcal{T}_{exec}$ as well as by previously seen input values (contained in $\mathcal{S}_{hist}$).

[1] This type of limitation is not supported in all temporal reasoning systems. The tractibility of the Time Map Manager's contradiction resolution algorithm depends on the absence of time-limiting constraints on the extent of persistences [19, Rule 3, p. 45].

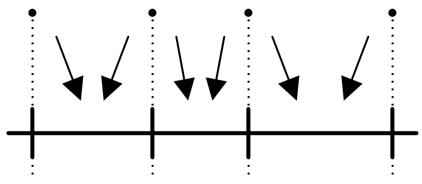


Figure 4.5: Schemata of Two Point Abstractor

The internal execution time function f_x must partition the initial execution interval into smaller pieces, each of which can be handled by f_v and f_s. This division of a large execution interval into smaller pieces is an example of the divide-and-conquer approach to simplifying the reasoning. The method of setting the duration of interval variables requires this type of programming style inside TCS modules. There is a dual to this function which operates in the opposite direction on the timeline.

4.3.2 Two-Point Abstractor

The difference between a two-point abstractor and the persistence module above is that information from two adjacent points is processed. Functionally, this is shown by the presence of both history and oracle variables in the process state:

1. $f_v : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \times \mathcal{S}_{orac} \mapsto \mathcal{V}_{int}$
2. $f_x : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \times \mathcal{S}_{orac} \mapsto \mathcal{T}_{exec}$
3. $f_s : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \times \mathcal{S}_{orac} \mapsto \mathcal{S}_{hist} \times \mathcal{S}_{orac}$

Although the internal details are complex, the user need provide only a single function $f'_v : x \times x \mapsto y$. This function takes two input values (one from each of two adjacent point variable values) and produces one output value (the value for the interval between the points). The data evaluation is shown schematically in figure 4.5. The remainder of the internal code is common to all instances of this type of abstractor and can be generated automatically. The programmer benefits by needing to provide only the domain-dependent part of the reasoning and allowing the system to implement the generic temporal reasoning task.

4.3.3 Memory

The memory used in the system thus far is limited to that of the history inside a single process. It is, however, easy to transform the value of an internal memory variable into an output variable. This allows the system to "remember" past values when it makes any given decision. I have implemented four types of memory module, all dealing

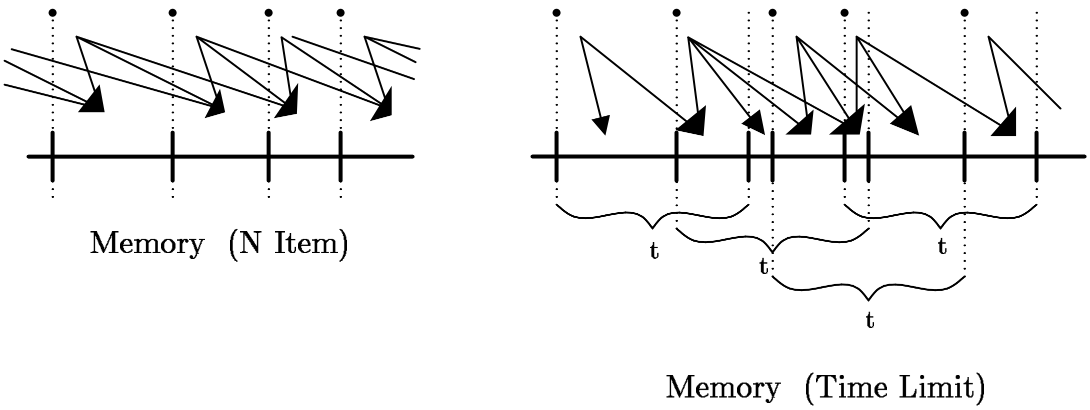


Figure 4.6: Schemata of Memory Modules

with point variable input. They represent the interaction of two design dimensions: the number of items to remember (either all events or just the most recent) and the length of time to remember them (either forever or limited to a specified time period). The first of these conditions can also be generalized to allow an arbitrary number of items to be remembered without requiring all items to be retained. The four memory types are:

1. Remember only the most recent event.

2. Remember the last n events.

3. Remember all events for the past t time units.

4. Remember all events for all time.

This list of memory modules covers two common domain-independent criteria for deciding how much information to keep for how long. It is not an exhaustive coverage of all memory modules. Making the length of time a datum is remembered depend on the value of point variable is an example of a domain-dependent strategy that I do not cover. Memory types 2 and 3 are shown in figure 4.6. Their formal specification is shown below.

Just as persistence has an analog that moves data backward in time, there is an analog to the memory module that also moves data backward in time. This is termed a “future” module. I will present an example of how this can be used in practice in chapter 6, in the section on measuring urine flow.

N Item Memory

Memory of n items, a variant of abstractor:

1. $f_v : \mathcal{V}_{pt} \times \mathcal{S}_{hist} \mapsto \mathcal{V}_{int} \qquad [f_v \equiv \text{cons}(\mathcal{V}_{pt}, \text{subseq}(\mathcal{S}, 1, n-1))]$

2. $f_x : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec}$

3. $f_s : \mathcal{V}_{pt} \times \mathcal{S}_{hist} \mapsto \mathcal{S}_{hist} \qquad [f_s \equiv \text{cons}(\mathcal{V}_{pt}, \text{subseq}(\mathcal{S}, 1, n-1)) \equiv f_v]$

This module has only one point input variable. The internal state contains the values of the previous n states of $\mathcal{V}_{pt}$. Any change in the value of an input will therefore cause at most $n+2$ processes to be executed. One is for the process that gets changed. Another is possible if the change causes an interval value to be shortened.[2] Finally, there are n executions as the information gets propagated to the next n process instances. The amount of recalculation in this module is therefore determined solely by the number of intervals remembered and is not affected by the size of the interval changed.

Time-Limited Memory

This module remembers all $\mathcal{V}_{pt}$ values for a fixed interval of time:

1. $f_v : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \mapsto \mathcal{V}_{int}$

2. $f_x : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec}$

3. $f_s : \mathcal{V}_{pt} \times \mathcal{T}_{exec} \times \mathcal{S}_{hist} \mapsto \mathcal{S}_{hist} \qquad (f_s \equiv f_v)$

The programmer need only specify the retention time. This differs from the persistence abstraction in two ways. First, both the value and the time of the point is retained in the interval output, and second, more than one value can be retained. In the persistence case, only the value of the point variable is used in determining the value of the output. The output of a memory module is therefore more general.

In a memory module $f_s \equiv f_v$, providing a additional simplification. This equivalence is not surprising since the module simply makes its internal history variable available to other modules as an output.

The abstractors show the benefit of introducing specialized modules for handling common reasoning tasks. Although the internal implementation of the modules requires the full generality of the TCS mechanism, much of the code can be automatically generated, allowing the programmer to control the process by specifying only those parts of the module that depend on the domain itself.

[2] This requires a new process instance because the endpoint changes; the value of the state variable at that time is indeterminate, since it has not yet been calculated, and is thus unavailable to TCS.

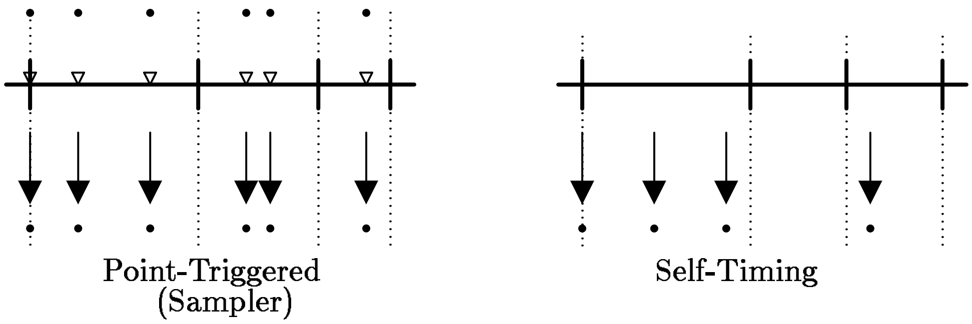


Figure 4.7: Schemata of Generator Modules

4.4 Generator

Data generation is the process by which some general description of a process (contained in an interval variable) is transformed into specific items of information or calls for atomic actions (contained in a point variable). This is the opposite of abstraction. For example, the general treatment directive "Take two aspirin every four hours" can be transformed into a series of individual actions "Take two aspirin" which are scheduled to be done as atomic actions at times four hours apart.

Similarly, a model can be used to generate information which can be viewed as point samples of a continuous process. For example, if it is known that over the period of four hours, the concentration of a certain drug in the blood will rise linearly from its present value of 1.0 to a final value of 2.0, intermediate predictions of 1.25 at one hour, 1.5 at two hours, etc. can be made. Figure 4.7 contains schemata of generators.

4.4.1 Point-Triggered Generator

A point-triggered generator ignores the value part of the input point variable and uses only the time part to determine when a value should be output:

1. $f_v : \mathcal{V}_{pt} \times \mathcal{V}^*_{int} \mapsto \mathcal{V}_{pt}$

2. $f_x : \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec} \qquad (f_x \equiv \mathcal{I}d)$

3. $f_s \equiv \emptyset$

The point variable is used solely for timing; data values come from the interval variables. This is in contrast to the context-sensitive transducers described above, which processed only the value part of the point variable. This is another example of the use of TCS variables for the control of reasoning.

Self-Timing Generator

Self-timing generators produce point output without an external trigger:

1. $f_v : \mathcal{T}_{exec} \times \mathcal{V}_{int}^{*} \mapsto \mathcal{V}_{pt}$
2. $f_x : \mathcal{T}_{exec} \mapsto \mathcal{T}_{exec} \qquad (f_x \equiv \mathcal{I}d)$
3. $f_s \equiv \emptyset$

In place of the external trigger, information from the execution interval is used. The functional definition shown above is a pure form of this type of module. The pharmacokinetic model simulation was a hybrid form which used a combination of interval and point input variables in its the basic calculation. The identification of output points was not determined by an external trigger, but came from the internal timing. In the model's case, every simulation step was used as a trigger, but it would only be a trivial change to make a simulation provide an output value for every fifth simulation step.

This completes the list of basic types of reasoning modules that TCS provides for a programmer. In the next chapter, I discuss higher-level reasoning concepts that can be implemented on top of the TCS substrate.

Chapter 5

Higher-Level Reasoning Abstractions

In this chapter I describe a level of reasoning further from the underlying module structure of the TCS . I will present a simple model for temporal pattern-matching. The purpose of the presentation is to demonstrate that such a matching function can be easily implemented in TCS. I make no claim that the pattern-matcher is complete. In fact, the system provided by TCS is currently very limited in scope. I discuss one method of expanding the functionality of the pattern-matcher.

In the second section I describe reasoning by hindsight. The revision of past inferences as new information becomes available plays to the strength of the TCS design. As in the previous chapter, the TCS substrate is used to provide the administrative functionality needed for updating in the face of changing information. Reasoning by hindsight uses this foundation for the implementation of a solution to a difficult medical reasoning problem.

5.1 Temporal Pattern-Matching

The modules I have introduced thus far have concentrated on the processing of data in a particular execution interval. Through the use of history and oracle variables, one can make decisions in any particular execution interval depend on the values of data from outside that interval. This capability allows the matching of temporal patterns to sequences of interval values. History variables in a module are used to remember past interval values, including information about the length of the intervals over which values were valid. Although the time span of interval variables is not provided to modules directly, it can be easily calculated by reference to the endpoints of the execution intervals of the process instances[1].

[1] Actually, some care must be taken to ensure that the current input value is different from the previous one, but since that information is being accumulated in a history variable anyway, this is easily accomplished.

```
((:value b :min 10) (:member (a e i o u) :min 20 :max 50))
```

will match any part of the timeline where there is a `b` of at least 10 units duration followed by a single instance of *a, e, i, o, u* of 20 to 50 units duration.

```
((:value a :min 10) (:not a :max 50) (:value a :min 10))
```

will match a pattern of two `a`'s of duration at least 10 separated by a single interval value other than an `a` of duration not greater than 50.

Figure 5.1: Examples of the Pattern-Matching Language

With such a history, one can search for patterns of values that have special significance. An example from the medical domain is a characteristic rise and fall in cardiac enzymes following a heart attack. If these enzyme levels were being monitored by a TCS system, this pattern could be detected. Looking for characteristic seismic waves in order to monitor nuclear testing treaties is another example.

5.1.1 The Current Implementation

At present TCS provides a limited pattern-matching language capable of matching fixed-length patterns. The fixed-length restriction is imposed in order to limit the amount of information that must be accumulated in history varibles of the pattern-matching module, thus improving the efficiency of the calculation in the face of changing data.

The primitives of the matching language are the following:

`(:value` *value*`)`	Matches if the tested value is equal to *value*.
`(:not` *value*`)`	Matches if the tested value is not equal to *value*.
`(:member` *value-list*`)`	Matches if the tested value is in *value-list*.
`(:none-of` *value-list*`)`	Matches if the tested value is not in it value-list.
`(:any t)`	Always matches.
`(:predicate` *function*`)`	Uses *function* to test the value (only) of the interval to be matched.
`(:full-predicate` *function*`)`	Uses *function* to test the value and duration of the interval to be matched.

A pattern consists of a list of these primitives, specifying the order of the matching. In addition to the basic forms listed above, all keywords except `:full-predicate` can also have `:max` and `:min` keywords followed by non-negative values. These values are used to put lower and upper inclusive bounds on the length of time that an otherwise suitable match must last. Even with a 0 lower bound, the pattern element *must* be present. A pattern of fixed time lengths can be enforced by making the `:min` and `:max` values the same. There is no disjunction. The example in figure 5.1 shows the use of this language.

Although the basic temporal model used in TCS requires that all times be exactly specified, the patterns used for matching against this database do *not* have to have exact endpoints.

5.1.2 One Potential Extension

The current implementation of pattern-matching uses a very rudimentary language and limits the matching to fixed-length patterns. A more flexible approach would be to build a regular language parser based on a finite-state machine. Since the amount of data needed to perform a match could not necessarily be calculated in advance, a different strategy would be needed. Assembling all of the values in each execution interval would be computationally prohibitive.

Instead of a complete evaluation of the pattern match in each process interval being done, the task could be broken down into segments. The state information for the finite-state machine could be transmitted via the history and oracle mechanism. Using that state information, each process instance could perform an incremental check for a match based on the current input value during that execution interval.

A number of details remain to be worked out. A way to transmit the news of a successful match to all of the intervals that contributed to that match is needed, as well as some administrative functions for data fusion between adjacent process instances with the same value. Neither of these problems should be insurmountable, however.

Dvorak and Kuipers [25] and Coiera [13, 14] are investigating the use of qualitative models to match behavior. The combination of their analysis techniques with a monitor built using the TCS could combine the flexibility of the model-based approach to matching with the convenience of being able to modify and retract the information that feeds into such a model.

In a development outside the scope of TCS-amenable applications, Kahn [38] has developed a technique for analyzing a collection of variable daily records in order to detect changes. Taking advantage of the cyclical nature of the records, his system identifies clinically meaningful changes in chronic medication doses. This system has been successfully applied to analyze home care journals of diabetic outpatients.

5.2 Reasoning by Hindsight[2]

In this section I describe the phenomenon of reasoning by hindsight, using an example drawn from cardiac patient care. As the clinical picture of a patient evolves over time, more information becomes available. The availability of more data allows a more accurate assessment of the patient. With more information one can revise observations, reinterpret previous data and confirm or retract assumptions. Uncertainties, guesses or errors that were made early in the clinical course of patient care can also be identified and resolved.

Hindsight also allows one to use response to therapy as diagnostic information. Since the response of a patient to a particular treatment is modulated by the underlying disease process, an analysis of this response can shed light on that process. The identification of errors, the discovery of violated assumptions, or simply the resolution of ambiguous findings becomes possible. But the use of hindsight in expert systems also requires that appropriate attention be paid to the temporal relations of the data and that care be exercised in revising decisions.

5.2.1 Clinical Example

The clinical example uses an abstraction of an actual case from cardiology. It shows the revision of diagnosis and the modification of therapy in response to evolving information about the patient's condition. The crucial feature of the example is that an analysis of the response to therapy is necessary in order to come to the correct conclusion.

Consider a woman presenting with a heart attack and ventricular premature beats (VPBs):

> The patient was a 56 year old female with acute chest pain, ice cold hands, clammy skin, bibasilar rales, left S3 gallop, no murmurs, blood pressure of 80/50 by cuff, pO_2 of 64 (slightly low), pCO_2 of 36 (a bit low, reflecting hyperventilation), pH of 7.36, BUN of 19, serum creatinine 1.1 and K 4.9. The ECG showed multifocal VPBs, short runs of ventricular tachycardia of 3–8 beats at a rate of 130–160, ST elevated in V1–5 (suggesting a fairly large anterior wall infarct), and no Q waves. She was treated with dopamine and lidocaine. She was excreting some urine but was oliguric ($<$ 500cc/day). After some hours a Swan-line was inserted, showing a PA pressure of 50/30 and a wedge of 29, confirming the left ventricular failure.
>
> There was limited response to the lidocaine or dopamine after a day. The blood pressure only went up to 90/50 and her hands remained ice cold and the S3 gallop and bibasilar rales persisted. The arrhythmias improved, but

[2]The material in this section has been separately published [71].

> multifocal VPBs and short runs of ventricular tachycardia still persisted. An arterial line was put in and the blood pressure was 200/120. [43, p. 34]

The initial presentation has the classic signs of an acute myocardial infarction. Based on the information available at presentation, the initial therapy decision is correct. Since the data indicate a patient in cardiogenic shock (a low blood pressure state), action should be taken to boost the performance of the heart in order to raise blood pressure and provide more oxygen to vital tissues (including the heart!).

Unfortunately, this case has a twist. One of the body's normal reactions to a fall in blood pressure is to reduce peripheral circulation in order to maintain adequate blood pressure in the central, vital part of the body. In this patient, the reduction in peripheral circulation was so extreme that the blood pressure reading obtained by using a cuff on the arm was no longer representative of the true blood pressure in the core of the body. This violates the basic (unstated) assumption of blood pressure measurement: the pressure in the upper arm is an accurate indicator of the blood pressure in the aorta.

Because the low blood-pressure measurement was expected in a heart attack victim, there was no reason to doubt the accuracy of the measurement during the initial assessment. The important ramification is that *data consistency checking cannot detect this mistake!* It is only clear that something is amiss *over the course of the next day*. The problem first becomes apparent when the expectations of therapeutic response were violated. The expected reaction to the dopamine would be a rise in blood pressure, an increase in urine flow and an improvement in the heart failure. These effects do not occur. At this point it is necessary to reassess the available information and the decisions based upon that information.

Since the therapy decision was correct, based on information available at the time, the focus must be on the data evaluation. A reconsideration of the data evaluation leads one to suspect that the equivalence between the measurement of the blood pressure at the arm and the underlying datum of interest, central arterial pressure, is not present. By using an invasive, but more accurate method for measuring blood pressure, it is possible to confirm this suspicion.

The new assessment removes the justification for giving a positive inotropic agent and also calls the initial dopamine therapy into question. The revised opinion, which benefits from hindsight, is that a drug to dilate the patient's arteries should have been used instead of one to make the heart beat more strongly.

5.2.2 Program Results

To demonstrate how the TCS supports this type of reasoning, I programmed a simplified version of the cardiac management decision above. The program used to generate the output shown here consisted of 26 variables (of which only seven are shown in the figures) and 14 reasoning modules. Selected portions of the program output after the initial consultation (figure 5.2) and after the reconsideration of the information

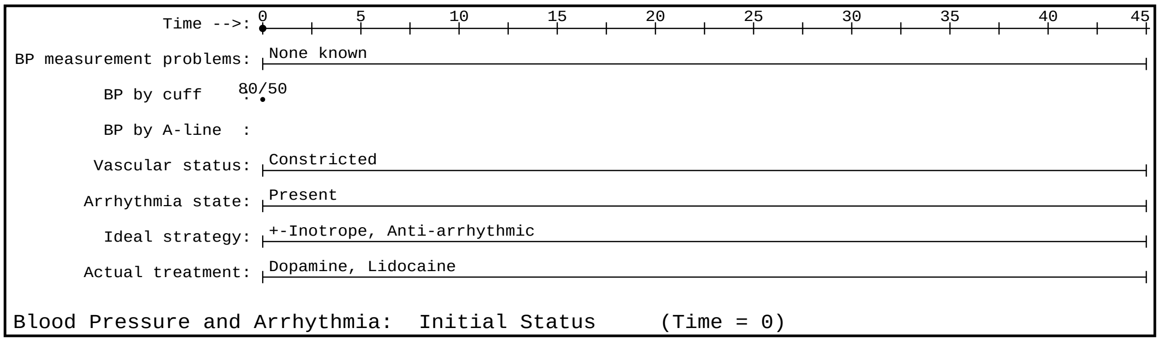
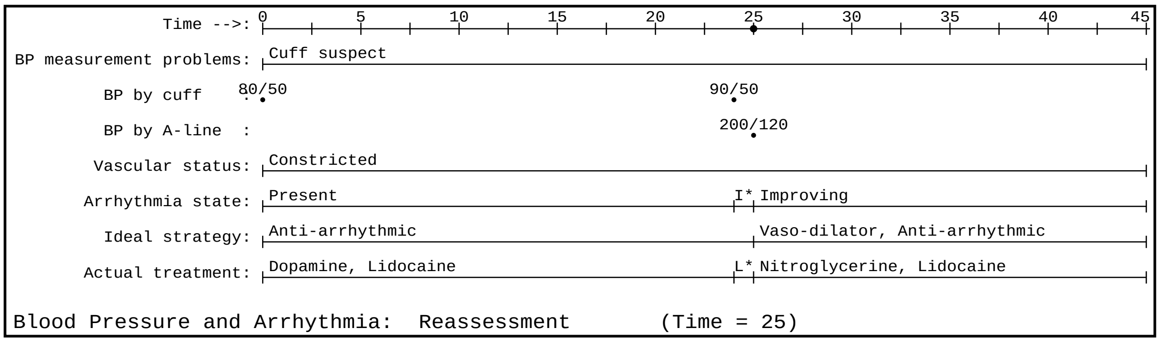


Figure 5.2: Initial Patient Advice



(The "I∗" and "L∗" stand for "Improving" and "Lidocaine." There was not enough room on the graph for the full labels.)

Figure 5.3: Revised Patient Advice

(figure 5.3) are shown. The initial evaluation executed 38 process instances for the 14 modules. The two stages of the revision at times 24 and 25 combined executed 85 process instances. For this example, the time scale uses units of hours.

The program takes the clinical observations and test data as its input and abstracts this into a description of the state of the patient. The initial decision uses the low blood-pressure measurement (80/50), the constricted vascular status (from cold, clammy skin), and the presence of arrhythmias to suggest the use of a positive inotrope and an anti-arrhythmic agent. This abstract strategy is refined into the concrete recommendation of dopamine and lidocaine (figure 5.2). I have already discussed the transformation of point data to intervals in the TCS .

Great care is taken to separate the ideal strategy from the concrete actions. This is important for allowing the process of hindsight to operate. It is a demonstration of the need to use different forms of reasoning in the future and in the past.

As more information becomes available (at times 24 and 25), the assessment is reconsidered. The arrhythmia remains a problem, but since it is improving, the program concludes that the choice of lidocaine is correct and should be continued. This is reflected in the retention of the anti-arrhythmic part of the ideal therapy strategy.

Since the lack of blood-pressure response is not consistent with the expected effects of dopamine, this part of the case analysis needs to be reexamined. The lack of response, combined with the vasoconstriction, makes the cuff method of blood-pressure measurement suspect; this is detected by a module monitoring the progress of therapy. This rule compares the actual response with the expected response and detects discrepancies. The discrepancies in turn lead to an examination of the assumptions underlying the original treatment decision.

Without a reliable blood pressure, the justification for the positive inotrope is missing, so it is removed from the ideal strategy.[3] The ideal treatment is modified so that it no longer includes dopamine and so that it uses a different method to measure the blood pressure. The concrete treatment, however, can only be changed in the future, so dopamine remains on the treatment list for the first 24 hours (figure 5.3). Once the arterial line is inserted and a reading obtained, a vasodilator is indicated to reduce the central blood pressure from its very high level of 200/120. This is reflected in the concrete suggestion that nitroglycerine be added.

In the implementation of this decision, the module used to evaluate the data in order to arrive at a treatment strategy considers the current values of the blood pressure, the arrhythmia state and the vascular status of the patient, as well as any known problems with drugs or blood pressure. The module that determines whether there are any drug or blood-pressure problems considers the treatment and current (input) and past (history) values of the clinical parameters. It also uses the history and oracle facility to make the conclusions about the blood-pressure difficulties available to earlier and later time periods. The detection of the problem at time 24 is therefore available for use in reconsidering the treatment strategy at time 0.

In a complete system, there will need to be a large number of "default assumptions" along with rules to monitor their validity. Although TCS does not relieve the programmer of the burden of identifying and implementing these assumption monitors, the TCS provides a mechanism for revising the affected decisions when assumptions are later determined to be violated.

5.2.3 Discussion

Hindsight is inherently a temporal process. It involves using data available at one time to evaluate decisions made earlier. The temporal aspect of reasoning by hindsight is illustrated in figure 5.4. The initial advice for Therapy_1 is based on the evaluation of the first blood pressure reading BP_1. After some time has passed, another reading is considered (BP_2). This second reading is used in two ways:

1. To evaluate the efficacy of the initial intervention Therapy_1 as well as the process (Patient Evaluation) that led to the choice of that therapy. This review can

[3] It would also be possible to implement a less radical strategy by suggesting the use of an arterial line before the therapy itself was changed. This is a change in the function used by the decision module and does not affect the demonstration of the action of TCS.

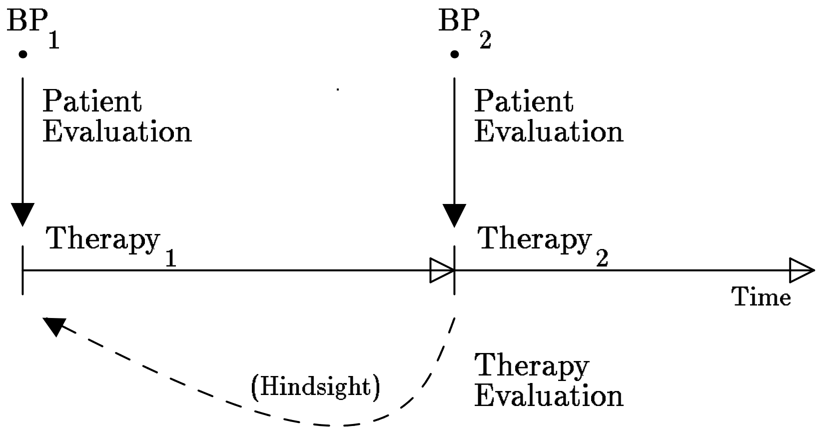


Figure 5.4: Temporal Aspect of Hindsight

confirm the correctness of the initial decision, suggest a modification of the therapy, change it completely or be neutral (i.e., not express an opinion).

2. To plan future therapeutic interventions (Therapy_2). The future plans will implement either the results of the review of therapy from 1 above or else involve some other changes, perhaps the progression to a new state of the therapy.

In the following sections I examine conceptual issues raised by the use of feedback, as well as technical considerations needed for the proper implementation of this reasoning.

Evaluation Feedback

Two different types of failure can be detected by the use of new information for the evaluation of a treatment. One is the failure in the choice of therapy. This could be due to an error in the reasoning which led to the choice of therapy, or it could be due to inherent uncertainty. An example of the latter would be the presence of arrhythmias that are resistant to lidocaine. In the above example, if the lidocaine had proved incapable of improving the arrhythmia, the program would conclude that the arrhythmia is resistant to lidocaine. However, since the anti-arrhythmic strategy would still be correct, only the implementation of the strategy need be changed. An appropriate alternate drug such as procainamide would be suggested.

A second form of failure that could be detected by hindsight could be an error in the data collection process or in the interpretation of the data in a particular patient's context. In the example there is an error in the data evaluation due to a violated assumption. For example, inappropriate data interpretation could occur if a hypertensive patient presented with a blood pressure of 115/75. In most patients this would be considered in the normal range. For an individual with high blood

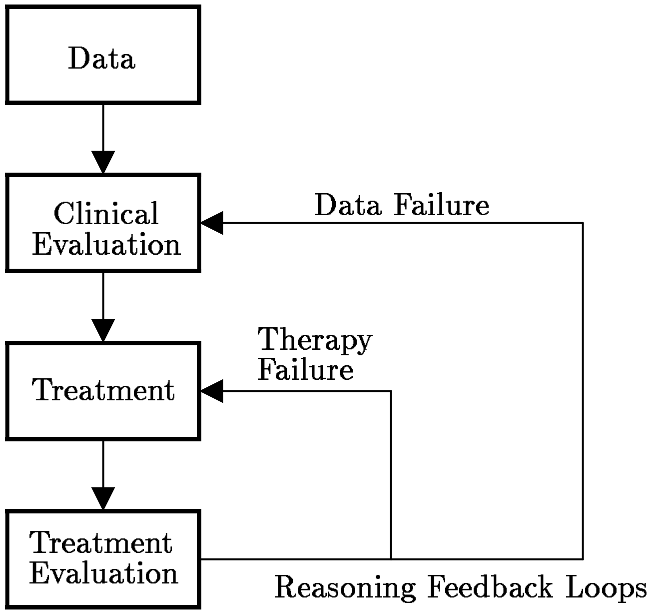


Figure 5.5: Evaluation Feedback Loops

pressure, however, this should be considered abnormal and a cause for the lower-than-usual blood pressure sought. If knowledge of a patient's high blood pressure only became available after the start of treatment, the reinterpretation of the blood pressure readings would be an example of hindsight. Both types of feedback in the reasoning are shown in figure 5.5.

Past–Future Distinction

When considering the way reasoning and actions interact in an advice-giving system, one must maintain a separation between reasoning about future events and reasoning about past events. In the future, one can freely change both the advice and the actions that follow from the advice. In the past one can, through hindsight, change the advice—deciding "what *should* have been done"—but actions must remain unchanged, reflecting what was actually done.

In a program, this can be accomplished by maintaining separate variables for the advice (the ideal treatment and strategy) and the actual interventions (the concrete treatment and strategy). This is combined with a system-maintained variable indicating whether the reasoning is in the past or the future. If reasoning is in the past, no changes are allowed to the concrete choices. One could accomplish the same end by having the concrete actions be entered from outside the system. This would also be a confirmation of what was actually done, since the clinical staff is not forced to follow the advice of the computer.

In addition to being a logical nicety, the maintenance of this distinction is crucial to the performance of reasoning by hindsight. The importance lies in the interaction between changing items in the past (through hindsight) and the dependency-directed updating system. If this distinction is not carefully made, one can be led into a circular argument of the following form:

1. We have a default assumption $\mathcal{A}$.
2. We make evaluation $\mathcal{E}$, based on assumption $\mathcal{A}$.
3. $\mathcal{E}$ indicates that the proper therapy is $\mathcal{T}$.
4. Treatment $\mathcal{T}$ leads to response $\mathcal{R}$, when $\mathcal{E}$ (using assumption $\mathcal{A}$) is believed.
5. When we later discover that $\mathcal{R}$ did not occur following $\mathcal{T}$, we can conclude that assumption $\mathcal{A}$ is invalid and should be retracted.
6. Since $\mathcal{E}$ depends on $\mathcal{A}$, $\mathcal{E}$ is retracted.
7. Since $\mathcal{E}$ is retracted, we have no longer have a reason for doing $\mathcal{T}$, so it is removed.
8. Without $\mathcal{T}$, the absence of the response $\mathcal{R}$ is not grounds for disbelieving $\mathcal{A}$.
9. Therefore, we make default assumption $\mathcal{A}$ and the cycle begins again at Step 1.

Trouble occurs at step 7, where an attempt is made to undo a past action. If this step is disallowed, then the circularity is broken and the reasoning chain remains valid, without the infinite loop. $\mathcal{T}$ must be removed from the list of actions that we wanted to perform (in the ideal world), while remaining on the list of actions that were performed (in the real world).

This potential circularity requires that the user keep those items of the history invariant which cannot be changed retroactively. Since real oracles do not exist, data is only available at the current time or from past times. Hindsight cannot undo physical actions. They must remain, not only for philosophical reasons, but also because of their logical necessity in support of the hindsight argument.

5.2.4 Hindsight Summary

The TCSis designed to support management of evolving situations that do not require novel strategies. In medicine this is known as following standard protocols. The steps of protocol-based care involve the identification of the appropriate protocol (diagnosis), the evaluation of the patient's state (assessment), and the decision about the therapy to be tried (plan instantiation). Following plan instantiation, diagnosis and assessment continue to determine whether the plan is being successful or not (monitoring). Monitoring allows the plan to be adjusted for differences in individual responses to a standard therapy. It uses feedback to adapt the plan to individual patient characteristics. It is also needed to detect changes that can indicate a new disease process, a fundamental change in an existing process, a flaw in the therapy chosen, or the existence of errors in the initial therapy choice.

As I demonstrate in the hindsight example above, some of the errors in patient evaluation and plan choice only become apparent over the course of time. It is therefore important to monitor the changes in a patient's condition, consider the effects of previous treatment in evaluating current patient data, and be able to change flawed assumptions at the point that they influence decisions and have the effects propagated forward in time.

Chapter 6

Ketoacidosis Advisor

Experience in the cardiology domain influenced the initial development of TCS. To further test the utility and demonstrate the generality of TCS, I used it as the basis for a new, medically interesting therapy advisor for diabetic ketoacidosis (DKA). A successful application in this domain must consider serial laboratory test data and additional clinical information in order to arrive at advice for patient treatment and management. The state of a patient with DKA must be tracked over time and the therapy adjusted as the problem resolves.

Diabetic ketoacidosis is a condition which occurs when insufficient insulin is present for the metabolic needs of the body. Since insulin enables the body to use sugar (glucose) for its energy needs, a lack of insulin forces the body to fall back on an alternate source of energy. This alternate metabolic pathway results in the production of ketoacids, hence the name of the condition. The presence of ketones and ketoacids upsets the normal acid-base balance and makes patients very sick.

Diagnosis is straightforward. History of diabetes and changes in either insulin-taking, diet or physical activity (such as illness or accident) strongly suggest the condition. Evidence of ketones in the blood or urine along with glucose provide a positive diagnosis. The medical challenge is not in the solution of a diagnostic problem, but in the careful adjustment of treatment over the 24 to 72 hours it takes to restore patients to normal.

Treatment is a combiniation of direct and supportive measures. The direct measure is the administration of insulin to allow the body to use glucose as a fuel and thus obviate the production of ketones. This has the beneficial side effect of also reducing the concentration of glucose in the blood. Supportive measures include fluid and electrolyte replacement. When serum glucose concentration is high, the kidneys begin eliminating the excess glucose from the blood. Unfortunately, glucose is a large molecule and when excreted in the urine, it draws water with it (osmotic diuresis), leaving patients volume-depleted (dehydrated). Patients in DKA can have volume deficits of three or more liters. In addition to the water, potassium is also lost through the kidneys. Both items must be replaced as part of the supportive therapy.

6.1 Development of the Ketoacidosis Advisor

The goal of the Ketoacidosis Advisor project was to produce a computer system that is able to generate advice that is similar in quality to actual human performance. The Advisor project was a collaborative effort between myself and physicians at the Tufts—New England Medical Center. I provided the computer-science expertise and the physicians, Michael Hagen and Klemens Meyer, supplied the medical expertise. The Advisor was able to achieve these goals, as demonstrated by a formal evaluation reported in the next chapter.

Realism was guaranteed by the use of actual patient data. We randomly selected sixteen cases from a pool of approximately 400 cases with a primary or secondary diagnosis of DKA treated at the New England Medical Center between 1986 and 1989. Joni Beshansky, a nurse in the Medical Center's Clinical Decision Making Unit, performed the database search and located the cases. We used ten of the sixteen cases for expert system development and reserved six for use in testing. The test phase used four of the six reserved cases. Before extracting data, we removed all patient indentifying information from the cases. The New England Medical Center Institutional Review Board granted approval for the use of anonymous information from the clinical records. We assigned the cases sequential numbers for identification within the Ketoacidosis Advisor project. We did all development and testing of the system using data collected retrospectively from the patient records. The Ketoacidosis Advisor was not used for patient care.

We abstracted the data in the medical records into a machine-readable form and used them for development. I did all of the record coding. The quality of the record-keeping varied from case to case, as did the amount of information that was retained in the permanent record. The information available generally included the emergency room record, complete laboratory test reports, flowsheets used in treatment, input and output records, medication record sheets, physician orders and the progress notes. Serial blood-pressure measurements were also generally present. The fluid balance sheets had the greatest variability and were in some cases not completely filled out. The largest problem for evaluation was the poor identification of the times at which fluids were added or urine output was recorded. Medication times and laboratory tests typically had good time stamps.

I encoded full chemical blood labs, blood gas and relevant parts of the urinalysis, but not laboratory studies that did not affect the insulin or fluid decisions. I encoded information from the narrative portion (House Officer and Nursing progress notes, as well as physical exam results) as it seemed relevant.

A major difficulty in data acquisition was determining when the patient was able to tolerate oral fluids. Although obvious to the clinical staff treating a patient, this fact is often not formally recorded. I have attempted to identify the point at which oral intake is sufficient by examination of the nursing progress notes, the physician orders for diet, and the appearance of other fluid input sources on the fluid balance flowsheets. Nevertheless, I consider this one of the weakest parts of the data acquisition. A second difficulty is the measurement of urine output. This is sometimes not recorded on the fluid balance sheet (urinalysis results occasionally appear from the laboratory without corresponding urine output!). Furthermore, if the patient is ambulatory, urine output is not entered into the record because the patient will just walk to the toilet. (Although it complicates the task of development and evaluation of an expert system, this gap in the data collection has no impact on actual patient care. Any patient that is sufficiently well to be up and out of bed is not severely dehydrated.)

Since the goal of the project was to do an in-depth analysis of each particular case, a large set of cases was not required for development. The data abstraction involved the reconstruction of the measurements, observations and events that occurred during the hospital stay. Because the major emphasis of this work is on the temporal nature of the decision-making, I made every effort to determine the times for all items in the record. A typical case would span 48 to 72 hours. The development cases had an average of 75 specific times when information became available or treatment was changed, and slightly more than 300 individual observations spread over those 75 sessions. This formed the input to the Ketoacidosis Advisor.

The physicians and I developed the knowledge base over a period of three years through a study of the literature, conferences with domain experts and the analysis of the case data. Once the initial framework of the Advisor was complete, we could use the cases to exercise the decision-making and to identify deficiencies in the reasoning. We performed a formal evaluation once the Ketoacidosis Advisor was performing satisfactorily on the set of development cases.

6.2 The Assessment Process

Monitoring and management problems characteristically have a recurring pattern of assessment and action. Information is obtained in order to arrive at an initial view of the problem encountered. Therapy (or more generally any action) is based on an evaluation of the initial state. Time then elapses until more data become available (i.e., more observations are made) and modifications of the actions can be made.

The amount of time that passes between the initiation of therapy and its evaluation must be long enough so that the therapy has time to produce an effect. In the case of fast-acting drugs like nitroprusside (which decreases blood pressure), this could be a matter of minutes. For other drugs, such as diuretics, more time must pass before an effect is discernible. Finally, a drug's effects wear off over time as it is eliminated from the body. This underscores the importance of the temporal component in the reasoning.

In DKA , the assessment of the patient is based heavily on blood tests. Blood glucose concentration is the main determinant of insulin therapy. Physical examination and other laboratory data contribute to the assessment of fluid status. Potassium status is assessed based on laboratory test results.

The laboratory tests are point samples that measure an underlying process. The therapies themselves are carried out over time. There is a fundamental difference in the temporal extent of these two quantities. To reflect the persistence of states even though they are observed only at single moments in time, the description of the patient's laboratory values is extended in time. This is appropriate because values reported by the clinical laboratory cannot change instantly. Since the values do respond over a longer time period, both to external therapy and internal evolution, it is important to limit the extent of the persistence.

This extension over time can be handled by using the persistence operators of the TCS. In the Ketoacidosis Advisor, this persistence is a fixed length of time. It is chosen so that the information will be remembered long enough to span the time until the next likely point when information will be gathered. It would perhaps be better to vary the time and have it depend not only on the value reported in the laboratory data, but also on what was done in the meantime.

For example, the serum potassium concentration in the blood will not stay constant if potassium supplements are given to the patient. When considering therapy decisions, it is important to consider the impact of treatment on the measured quantity when deciding how to use older laboratory data. Some of the therapy decisions can be based on abstracted intervals of time; others must be based on the sample itself.

For some parts of the therapy, trend information is also important. First, one wishes to have the serum glucose levels fall rather than rise further. But the rate of fall should be such that it produces no undue dislocations in the body. By using sequential measurements, a rate of fall can be calculated and used as the control variable for insulin therapy. This trend could be extrapolated to indicate when additional test results are warranted. This is important because it is necessary to adjust the dose as the glucose levels approach normal. Overshooting the goal causes hypoglycemia (low blood sugar) and endangers the patient.

The assessment also includes monitoring what has been done to the patient in the past. This is important for three reasons:

1. The actual clinical actions may not be what the Advisor has recommended. In making further recommendations, it is important that the Advisor know what the starting point of the actual therapy is. This is vital to the correct evaluation of the effectiveness of the therapy.

2. The treatment can affect the assessment because the response to therapy can itself yield diagnostic information. In particular, insulin resistence is identified by observing the effects of a particular insulin dose on a patient.

3. Some of the assessment involves the cumulative impact of treatment over time. The restoration of a volume deficit requires a certain amount of fluid excess to

be infused. Judgement of progress toward this goal is determined by summing the volume of fluids added and subtracting the measured and estimated losses. This process is aided by the use of clinically observable features such as urine flow, blood pressure, heart rate and weight changes.

6.3 The Therapy Decision

As I noted above, the three major components of DKA treatment are control of the amount of insulin, restoration of a proper fluid balance, and replacement of missing electrolytes.

6.3.1 Insulin

Insulin is used to arrest the further production of ketones and reduce the serum glucose concentration. The amount of insulin given is determined primarily by measurement of the glucose concentration. The zones for insulin therapy are high, moderate, normal, low and very low. If glucose is low, then no insulin is given. At very low levels supplemental glucose is also given, since eliminating all glucose will result in a coma . At normal glucose levels, insulin therapy is aimed at maintaining a steady state. At moderate and high levels, the goal is to reduce the insulin level. The rate at which this is done varies, because an overshoot can have swift negative consequences. As the glucose levels approach normal the rate of decline should level off to provide a "smooth landing."

Depending on the degree of sickness, insulin can be adminstered either through an intravenous (IV) infusion or via subcutaneous (SQ) injections. The therapy decision regarding insulin involves choosing a route of administration as well as the amount of insulin to give. These decisions are influenced by the patient's normal schedule of insulin needs as well as by his clinical state. A slightly simplified diagram of the rule for initial insulin administration is shown in figure 6.1a. The parameters influencing the control of an IV insulin infusion are shown in figure 6.1b. The infusion rate is titrated to keep the patient inside the glucose decline envelope. Time influences this decision through the calculation of a rate (change per unit time) and in the nature of a feedback control system (treatment evolving over time).

Because insulin must be given until the ketoacids have been eliminated, it is still necessary to continue therapy after normal glucose levels have returned. To maintain a steady glucose level, the continued administration of insulin is combined with the administration of additional glucose. Until the patient is well enough to eat, this additional glucose is provided by changing the composition of the fluids.

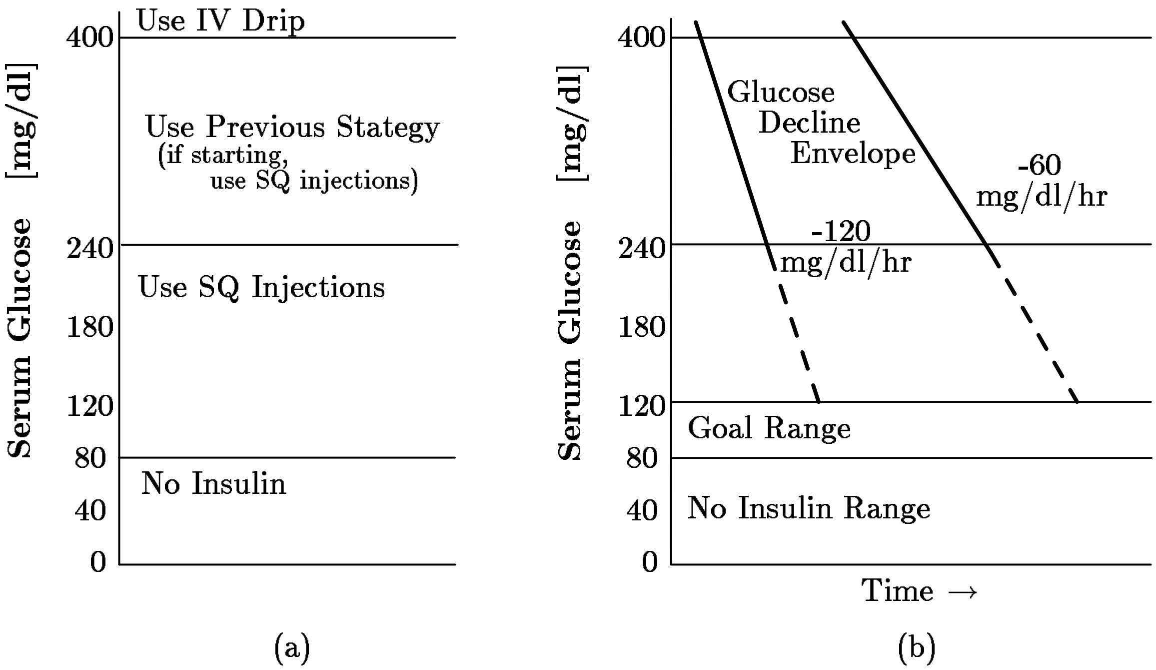


Figure 6.1: Insulin Treatment Strategy

6.3.2 Fluids

There are two components of fluid therapy: the type of fluid and the rate at which it is infused. The patient's initial condition typically includes high serum glucose levels and a volume deficit. Normal saline (without additional glucose) at high flow rates is used to remedy this condition.

As the treatment progresses the two problems resolve. The degree of fluid deficit and the need to offset continuing urinary losses determine the rate of administration. Progress in controlling the hyperglycemia (high blood sugar) and the success in overcoming the initial deficit affect the type of fluid used. These two controls are independent. Normal glucose levels are often achieved before all of the ketoacids have been eliminated. Insulin therapy must therefore continue. Since the serum glucose levels have dropped, it is necessary to supply additional glucose through either the IV fluids or oral (dietary) supplements to prevent dangerous hypoglycemia. Similarly, restoration of normal cardiovascular function as the volume deficit is replaced shifts the emphasis from pure volume replenishment toward maintenance of a steady state. Once a patient is able to tolerate food and drink, the need for an infusion will decline.

The initial rate decision is based on a desire to restore a normal volume in a controlled manner. The goal is to restore half of the deficit over the first eight hours of treatment and the other half over the following sixteen hours. Typically this means an infusion rate of 500 to 1000 ml/hour. After the initial volume deficit is made up,

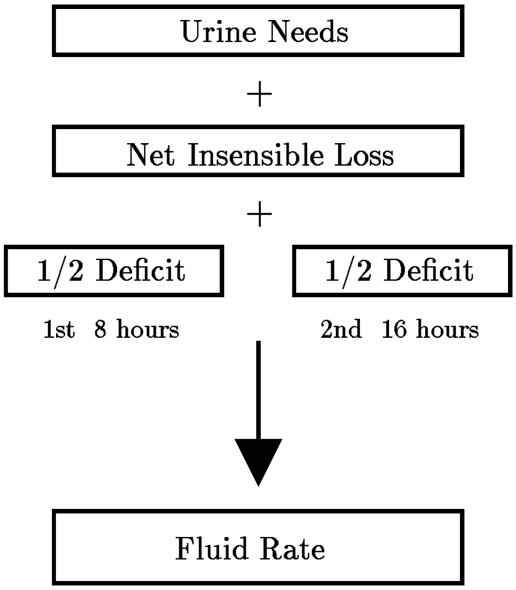


Figure 6.2: Fluid Rate Calculation

the replacement rate can be reduced to around 200 to 250 ml/hour. Once the patient stabilizes and the glucose and ketone levels are normal, a maintenance rate can be used. The factors which affect the fluid rate calculation are shown in figure 6.2. Some of these, such as the ongoing urinary fluid loss, can only be measured retrospectively. This means that therapy based on expected urine flow may need to be revised—a situation the TCS makes much easier to handle.

In a normal patient, there is little possibility of giving too much fluid, since the kidneys are able to eliminate the excess. In the case of kidney or heart failure, the body is not able to tolerate excess fluids. The Ketoacidosis Advisor's knowledge base does not contain rules for handling these special cases. This limits the clinical usefulness of the current prototype, and also imposed restrictions on the set of usable cases for the evaluation. I included no cases with renal or heart failure in the development or test set.

6.3.3 Electrolytes

The primary electrolytes that DKA treatment must handle are potassium and phosphate. Potassium management is complicated because the high urine flows from the osmotic diuresis cause potassium loss. Furthermore, the management is complicated by the lack of a clear assessment. The laboratory test results are skewed by a shift in the location of potassium from the cells to the bloodstream that is caused by the acidosis. This means that patients in the early stages of DKA treatment are low in potassium, but have laboratory test results that are in the normal range! Since the total mass of potassium in the cells is much higher than in the blood stream, small shifts in the amount in the cells can lead to large changes in blood concentration. This is a problem because excessively high or low serum potassium levels can disturb heart rhythms and lead to death.

Further complicating this is the fact that in advanced dehydration the body's mechanism for regulating potassium balance, the kidneys, may not be functioning. It is therefore necessary to ensure kidney function before beginning aggressive potassium therapy. If kidney function is ensured, then it is relatively difficult to overmedicate with potassium supplements. The need for potassium therapy is related to the success of the insulin therapy at correcting the acidosis. As the acidosis resolves, potassium shifts back into the cells, necessitating replacement with external sources. The effectiveness of one therapy therefore influences the effects of the other therapy.

6.4 The Bookkeeping Functions

Certain auxilliary functions proved useful for coordinating data arrival with the implementation of therapy. Other functions calculate derived information about the state of the patient to aid in the decision-making procedure.

6.5 Agendas, Urgent and Non-Urgent Changes

When continuous IV fluid therapy is in place, it is used as a means of administering other forms of therapy continuously over a period of time. Potassium and insulin are often given this way. Because the rate of insulin administration is closely regulated to keep patients in the proper glucose decline envelope, it is generally administered through a dedicated IV line.

Since potassium and phosphate administration rates are less critical, they are piggy-backed on the fluid replacement. Since supplement administration is not urgent, changes in treatment such as adding or deleting supplements wait until a new IV bottle is started. This happens when the present bottle is exhausted or a change to another type of fluid is indicated. The bookkeeping challenge is to coordinate the desire to change the supplement, made in response to a laboratory test result, with the independent event of an intravenous bottle being changed. The solution I used in the Ketoacidosis Advisor was to implement a higher-level data structure called an *agenda*, and use TCS modules to manage the addition and removal of items. Non-urgent treatments are posted to an agenda whose items are taken care of as the opportunities present themselves. For example, potassium supplements are added to the agenda when indicated by laboratory tests, and they take effect at the next fluid change.

A similar mechanism can be used for urgent changes. In this case, the agenda is used to retain actions triggered by a point variable value until they are either superceded by new advice, or until the clinical staff follows the advice. This provides a coordination between recommendations of the Ketoacidosis Advisor in response to newly received data action and the ability of the clinical staff to carry out the advice.

Agendas are implemented by using the history mechanism. The TCS module that maintains the state of the agenda uses lists of items to be added or deleted. The add list comes from advice-generating modules. The delete list comes from monitoring which actions have been carried out by the clinical staff. This data flow structure is like producer-consumer co-routines, but with the additional feature that the producers can retract items from the agenda before they are consumed. In other words, if an action which has not already been carried out becomes unnecessary, it is retracted from the agenda.

6.5.1 Assessing the Urine Flow

Replacing the fluid deficit requires a positive balance between the fluids that a patient receives and the fluids that he loses. This balance is calculated by subtracting the losses from the inputs. The sources of fluids are intravenous infusions, oral fluid intake, and water produced by metabolism. Losses are via urine and stool output (which can be easily measured) and so-called "insensible losses" that occur through breathing and sweating. There is a minimal urine flow that is required to clear metabolic waste products from the body. There is also a minimal amount that will be lost via the insensible route. In order to have a positive balance, the intake must exceed this amount.

IV and oral intake can be accurately monitored, and the metabolic sources estimated. Similarly, the urine output can be measured, while insensible losses must be estimated. Recall that the hydration control strategy that I used in the DKA advisor called for the replacement of one half of the calculated fluid deficit in the first eight hours and the replacement of the remaining half of the deficit over the next sixteen hours. The adjustment of the IV infusion rate depends on a real time assessment of the fluid balance. This requires that the losses be subtracted from the intake. Unfortunately, urine output is buffered in the body via the bladder. This introduces some problems into the continuous assessment of the fluid status.

Over the course of the project, I tried numerous techniques to handle this difficulty. I describe them below:

Calculating Flow Rate

The simplest method is to calculate the flow rate based on the amount of urine produced and the time since that last voiding. This is computationally simple and is based on the assumption that urine production was constant during the interval between voidings. The problem is that each trip to the bathroom does not necessarily completely empty the bladder. Particularly when there was not much time between voidings, it was possible to get short stretches with greatly different flow rates. This can be seen on the “Urine Flow, straight” line of the graph in figure 6.3.

I initially addressed this particular problem by consolidating all urine output that occured within a period of 60 minutes into a single point and using that as the urine output point. The effect was never to calculate a rate over a period of time that was less than 60 minutes. This averages out the small variations and reduces the size of the flow rate. This can be seen on the “Urine Flow, lagged” line of the graph in figure 6.3. The lag solution combines the results from 12:00 and 12:45 into a single value at time 12:45.

Previous Eight Hours—Shift Aligned

Unfortunately, the solution outlined above was not sufficiently robust for my needs. The second solution that I attempted was to observe the previous eight-hour shift and use that data. This method had the appeal that it closely followed the availability of data to the clinician, since fluid balance was generally tabulated only at the end of an eight-hour nursing shift.

Since shift balance data were already being calculated, it was easy to transform the balance from the previous shift into an interval for use in decision making in the following interval. The disadvantage of this method was that it did not provide current information. All of the balance calculations were being made eight hours after the fact. In other words, the effect of urine flow was not taken into account until one shift later. This can be seen on the line “Urine Flow, 8hr Avg” in figure 6.3.

Previous Eight Hours—Dynamic

To maintain the advantage of a larger averaging period while still remaining responsive to acute changes in the urine flow rate, I finally shifted to an eight-hour moving average. Aside from special provisions for the beginning of the hospitalization period when a full eight hours of data are not available, this was quite easy to implement.

All that was required was that the urine output be available (as part of an oracle variable) for the eight hours before the output was measured. At any point in time, then, the values of all of the urine output in the time window were available for averaging. This is illustrated by the “Urine Flow, 8hr Back Avg” line of the graph in figure 6.3. Note that the lag solution has fewer distinct values, because the lag solutions’ intervals are bounded by the individual data values. The averaging method also adds an eight-hour event horizon which can fall between existing data points and introduce more time periods. The magnitude of the averaged values shows more consistency, as one would expect with averaging. The averaging solution had more intervals (as expected), but the magnitude of the change between adjacent values was much smaller, producing smoother fluid rate recommendations. The size of the urine flow estimates are also higher in the crucial early phase of the treatment, when most

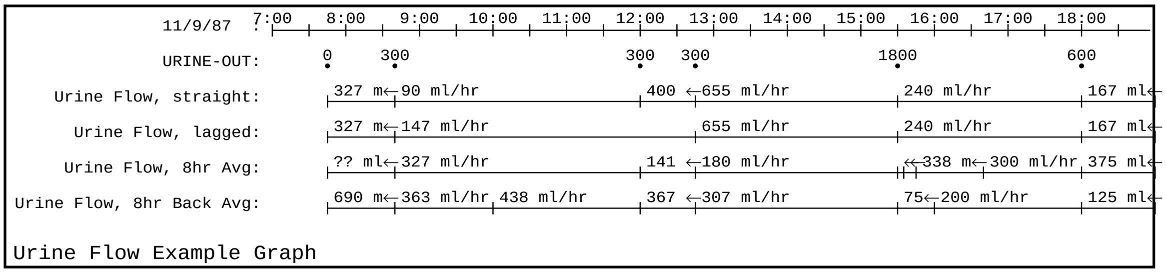


Figure 6.3: Various Urine Flow Measures

of the osmotic diuresis occurs. This leads to higher initial fluid rates, which is the behavior we desire.

The code for this module is quite easy to program, as shown in figure 6.4.

6.6 Special Module Schemata

The implementation of the Ketoacidosis Advisor showed the need for a number of other general-purpose modules. Although developed in response to specific reasoning goals in the DKA domain, they are sufficiently generic to be useful in a wider range of applications.

I implemented special modules to handle the following tasks:

1. Establishment of cyclic time patterns. The standard insulin dosage patterns call for injections at various times of the day, typically morning, evening and bedtime. One specific time would be too inflexible for a real world setting, so a period of time each day needs to be designated for each of these dosage times. The dosage rules used by TCS will recommend the baseline insulin dosage for patients who are well enough to be handled by SQ injections. If the dose is not given within its "dosage window," then it is deferred until the next day, and an interim maintenance strategy takes over.

   In the Ketoacidosis Advisor, I divided the timeline into a daily cycle with segments which include morning, evening and bedtime. I generalized the module implementing this division to produce a macro which takes a description of the cycle and a control variable that determines how far into the future to project the cycles. The time line is then automatically segmented, with the projection into the future controlled by the separate TCS variable. This is the same technique I used to run the pharmacokinetic model in chapter 3.

2. Periodic output summaries. At the end of each eight-hour shift, the nurses prepare a summary of the fluid input and output for each patient. This is a generator-type function which requires a different type of period timer. Again,

```
(defmodvar urine-out :point)
(defmodvar 8hr-out-fut :interval)
(defmodvar 8hr-out-back-av :interval)

  ;; Make urine output available for reasoning for the eight hours
  ;; before the value was measured.
(deffuture 8hr-out-fut urine-out 8hr-out-fut :full-future t
           :decay-time #.(* 8 60))

  ;; Calculates the rate component of each value by dividing the value by the amount
  ;; of time the value was observable.  This means that urine output within eight hours of
  ;; the start of treatment contributes more to the rate.  This is needed so that the
  ;; integral of the rate from the start of treatment until any given input point equals
  ;; the sum of the urine output recorded thus far.
(defun adjust-for-time (pv time)
  (let ((diff (tcs:time-sub (tcs:time-part pv) time)))
    (cond ((<= diff 0) 0)
          ((< diff #.(* 8 60)) (/ (tcs:value-part pv) diff))
          (t (/ (tcs:value-part pv) #.(* 8 60))))))

(defmodule 8hr-out-back-av (8hr-out-fut starting-time)
                           (8hr-out-back-av)
               ()                      ; No internal process state.  The oracle variable is
                            ; handled separately by the module  8hr-out-fut.
                            ; This separates the temporal component in one module
                            ; and allows a simpler implementation in this module.

  (cond ((unknown starting-time) (setq 8hr-out-back-av :unknown))   ; No session yet.
        ((tcs:time< begin_time starting-time)      ; No calculation before session starts.
         (setq 8hr-out-back-av :unknown
               end_time (tcs:time-min starting-time end_time)))
        ((tcs:time>= end_time (+ #.(* 8 60) starting-time))
         (setq 8hr-out-back-av
               (/ (loop for pv in 8hr-out-fut sum (tcs:value-part pv))
                  #.(* 8 60))))
        ;; The following code adjusts the values so that the amounts are scaled for
        ;;   the time to start.  It should integrate to the same value as the sum of
        ;;   the urine output data.
        (t (setq 8hr-out-back-av
                 (loop for pv in 8hr-out-fut
                       sum (adjust-for-time pv starting-time))))))
```

Figure 6.4: Eight-Hour Average Urine Flow Code

I developed a general module which can generate period timing information to trigger the collection and aggregation of data.

3. Reminders. Since some actions, such as changing the intravenous fluid bottles, do not happen at set times, but rather in response to need (i.e., when the bottle is empty), reminders of imminent chores can be helpful. Determination of the emptying time of bottles is an example of the more general phenomenon of the projection of the time a process will take to complete. This generalization can be used in any such circumstances. A related task is calculating when a patient's fluid deficit will be eliminated. This triggers a change in the fluid management strategy.

4. Coordination between advice and actual treatment. The ability to separate the generation of advice from the monitoring of its execution introduces a problem decomposition that simplifies the design of a reasoning program. A general mechanism for accomplishing this is to use the agenda to keep track of pending advice. The agenda can hold actions that should be performed immediately (like giving glucose supplements for hypoglycemia) or they can be linked to external triggering events (such as intravenous bottle emptying). A short discussion of this is published in [72].

The routines I have developed to support the programming of the Ketoacidosis Advisor provide tools that can be reused in other projects. Some of the tools are sufficiently general that I will include them in future releases of the TCS program.

6.7 Summary

In the implementation of the Ketoacidosis Advisor, the TCS methodology and tools proved adequate to handle all of the domain-specific reasoning. The existence of the automatic updating made the solution of many of the problems simpler, since significant measurements (e.g., urine flow) were often available only after a substantial delay. This required a recalculation and reassessment of the situation. The variables that were affected were automatically determined by the TCS system, which also scheduled the execution intervals for the process instances needed to do the recalculation. That alone provided a significant savings in development time.

Chapter 7

Formal Evaluation of the Ketoacidosis Advisor

In a formal evaluation, the decisions made by the Ketoacidosis Advisor were indistiguishable in quality from the clinical decisions made by the residents and interns treating the patients. There is evidence that the computer-generated advice was better, although the advantage was not sufficiently high to reach the .05 level in all of the statistical tests I applied to the data.

The Advisor's performance was evaluated by experienced physicians specializing in nephrology. The actual clinical treatment was given by the house staff at the Tufts—New England Medical Center. The house staff is made up of residents (physicians in the first three years after medical school).

7.1 Evaluation Design

An evaluation of an expert advice-giving system could take one of several forms. The form is influenced by the questions one wishes to have answered, which are in turn dictated by the purpose of conducting the evaluation. An evaluation designed to measure the overall performance of a system is not necessarily suited to identifying specific problems. An evaluation that concentrates on the rules which result in specific recommendations can lead to difficulty in forming a global view of the performance for comparison to actual clinical practice. Finally, one may wish to elucidate a "gold standard" for comparison with both advice and actual treatment, as well as for a guide for implementation of the reasoning.

The evaluation I report here is of the first type mentioned above. I present a composite assessment of the performance of the Ketoacidosis Advisor as an integrated unit. The goal of the experiment is to demonstrate that human-level performance can be achieved by an expert system in a domain in which the data are changing rapidly. Unlike a detailed evaluation of parts of the advice, such a global assessment is not directly useful in refining the medical content of the rules. (It is indirectly useful, since attention can be focused on areas where the composite advice was judged to be objectively poor.)

I did not evaluate the timing of the advice because of the fear of introducing bias in favor of a dedicated system as well as because of the limited resources that were available for evaluating the system's performance. I address this concern below in the discussion of the choice of decision points.

The formal evaluation was constrained by the amount of time that the expert panel was able to dedicate to the evaluation process. The evaluation needed to be completed in approximately one hour by each panel member.

A combination of time and personnel constraints made the development of a gold standard impractical. First, time would need to be found to reach a consensus. As I discuss at greater length below, there is no natural consensus on the evaluation of particular decisions. This lack of consensus is an inherent property of the domain of DKA treatment. Furthermore, to avoid biasing the evaluation process, separate groups would be needed to create the gold standard and to assess the performance of the Advisor on the case. This would have required more physicians than were available to serve on the expert panel.

7.2 Methodology

As I noted in the previous chapter, I selected the four cases for the evaluation phase of the project at random. To avoid any influence on the design and implementation of the Ketoacidosis Advisor, I first froze the program design, then examined and abstracted the cases in the evaluation set and presented them to a panel of experts.

7.2.1 Panel of Experts

I presented the set of cases chosen for the evaluation to a panel of five attending physicians and five fellows in the Division of Nephrology. Although two medical students also filled out evaluation forms, the statistical tests and the analysis below use only evaluation forms provided by the attending physicians and the fellows. This limits the panel to physicians recognized as experienced in the field of nephrology.[1]

None of the physicians on the expert panel was involved in the design or training of the Ketoacidosis Advisor. They had not previously seen the cases used for the evaluation.

I chose a panel of nephrologists for two reasons. First, since DKA is a disturbance of acid-base physiology, nephrology is one of the two relevant specialties. Endocrinologists, experts in the other relevant specialty, served as a backup discussion group

[1]Fellows are physicians who have completed their initial training (three years beyond the M.D. degree) and are beginning their specialty training, which typically lasts two years. Attending physicians are certified specialists. In a teaching hospital, the attendings are responsible for training the fellows.

to perform a less formal analysis of the program's output. The other reason for evaluation by nephrologists is that the domain expert who aided me in the design of the program was a nephrologist by training. This should be a more realistic evaluation of the effectiveness of the reasoning captured in the expert system, since the domain expert and the expert panel should have roughly similar approaches to the problem. Using a panel from another specialty, although interesting as an evaluation from the medical perspective, would have been less useful from the point of view of assessing the ability of this technique to capture a particular type of expertise. Thus, the specialty of the source of the program's expertise is the same as the specialty of the evaluation panel.

7.2.2 Format of the Questionaire

I designed a questionaire to present the temporal course of a patient who was admitted to the hospital for treatment of diabetic ketoacidosis. I coded each case by hand into a machine-readable form. The evaluation sheets were generated automatically from information in machine readable form. This included a textual overview which provides background for the evaluation panel, but which was not considered by the Ketoacidosis Advisor.

7.2.3 Choice of Decision Points

I used a mechanical procedure to choose the decision points to avoid bias introduced by the researchers. A decision point was deemed to occur whenever there was a significant change in the actual treatment. A significant change was:

- any change in the composition of the intravenous fluid (including changes to supplements).
- a change in the fluid infusion rate greater than 10%
- a change in the form of insulin treatment (infusion or subcutaneous injection)
- a change in the rate of insulin infusion greater than 10%
- the administration of insulin by subcutaneous injection
- the administration of an intravenous glucose bolus.

Each of the four evaluation cases was searched from the beginning of the record until either twelve decision points were found, the case was three days old, or the patient was discharged from the hospital.[2] The cases used in the evaluation varied in length

[2] One patient was sent home overnight and returned the next morning. This was treated as a single admission.

from 22 to 72 hours. The number of decision points varied from 9 to 12 per case. A total of 42 points were available for evaluation by each panel member.

I chose to limit the decision points to those times when a change in the actual therapy took place because that gave reasonable assurance that the clinical staff had examined the patient and any relevant data that had arrived up to that time. In particular, I did not feel that a fair evaluation of the content of the decision rules would have been possible if treatment changes that would have been initiated by the Ketoacidosis Advisor itself were chosen as decision points. Since the reaction time of the Advisor would never be slower than the clinical staff, and would most likely be faster, this would involve comparing the Advisor's recommendation using the latest available information with the old decision made by the clinical staff, based on information that was no longer current. I feel that this would have biased the evaluation unfairly in favor of the performance of the expert system.

However, this consideration made it impossible to assess the timeliness of the program's advice or to assess its potential to respond to changing conditions more rapidly than the human decision-makers.

7.2.4 Method of Evaluation

Each case was presented to the physicians in a summary form. Each decision point was on a separate page of the evaluation form. For each decision point, the case up to the time of that decision was presented to the panel of experts. Each case had between nine and twelve evaluation points. The cases were printed on forms which provided an initial narrative derived from the emergency room admission notes and the initial physical examination. Other information relating to the results of laboratory tests, vital signs and information about the treatment given up to the time of the decision was provided. At each decision point, two treatment plans for the next action were presented. One plan was generated by the Ketoacidosis Advisor and the other was the treatment actually given to the patient. Treatment plans were listed in random order and their source was not identified. As a safeguard against the order of presentation affecting the evalation, I prepared two sets of forms, identical except for the order of the treatments. Roughly equal numbers of each set of forms were used. Figure 7.1 shows the layout of an evaluation form. All of the information from the beginning of the case was reprinted in chronological order on each page of the form with the newly available information highlighted.

The panel assembled in a single room and worked through the evaluation forms without consulting one another. Dr. Meyer and I were present to guard against the evaluators' looking ahead in the record to see either what was actually done in the case (breaking the randomization), or seeing data that were only available in the future (acquiring information that would not be physically possible in a real setting). The panel used two methods to evaluate the therapy suggestions. First, each of the two treatments was rated on a five-category scale: dangerous, poor, acceptable, good

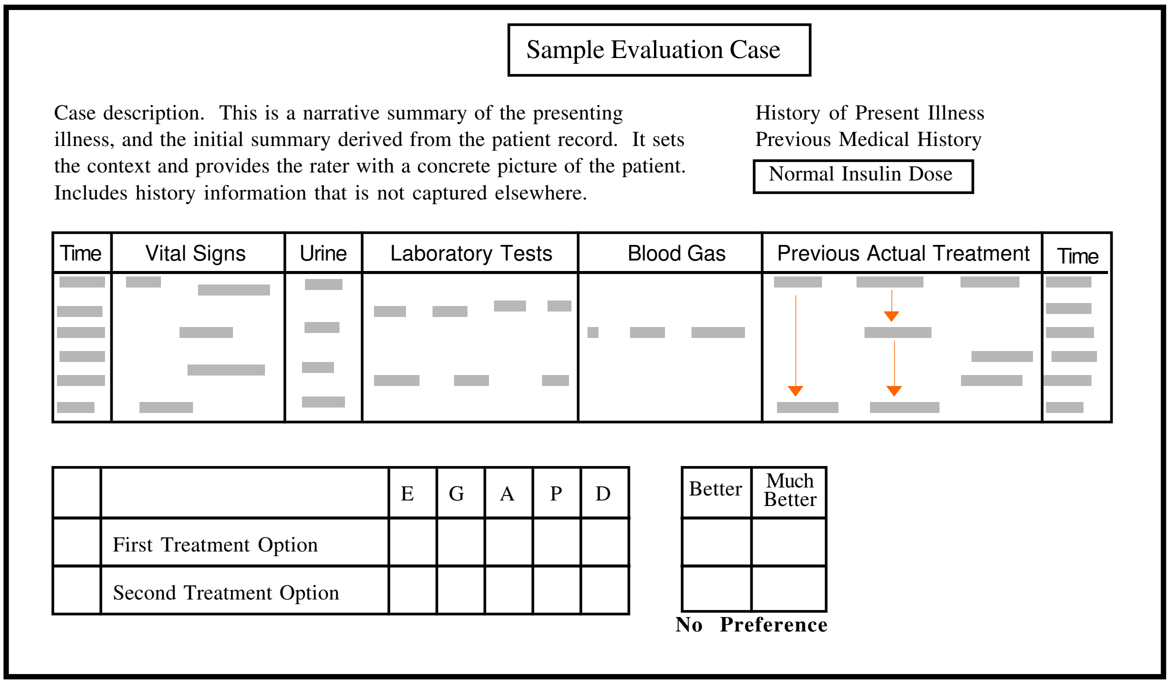


Figure 7.1: Sample of Evaluation Questionaire

or excellent. This provided an absolute measure of the quality of the advice. I refer to this below as the five-category test. The second measure recorded the relative quality of the two suggestions. The evaluator could express a preference for one treatment over another even if both fell into the same qualitative category. The scale allowed the rater to choose between no preference, one treatment was better or one treatment was much better. I call this the preference test.

For subsequent data analysis I introduced a two-category scale, derived from the five categories of the evaluation instrument. Dangerous and poor were combined into a new category called "Bad," and acceptable, good and excellent were combined into a new category called "OK." I call this the two-category test in the tables below.

My hypothesis before the evaluation was carried out was that the preference scale would be more sensitive to subtle differences in the treatment, since two treatments could fall within one of the absolute categories, but still not be considered equally good. The ability to express a relative preference allows a finer comparison than the rankings from the five-category scale. A sample evaluation question is shown in figure 7.2.

After collecting the data I discovered that not enough information was provided at two of the forty-two evaluation points. Laboratory test results which were available to the clinic staff when the decisions were made were not presented to the panel or made available to the program. Since the decisions involved the administration of insulin

[10] At **6/25 2:30a**, one of the following options was taken. Please rate them:

Choice	**Treatment**	Excellent	Good	Acceptable	Poor	Dangerous		Better	Much Better
1	NS with 30mEq Kcl/l at 200 ml/hr No IV insulin drip 5ml Kphos								
2	NS at 300ml/hr No IV insulin drip								

No Preference

Figure 7.2: Typical Treatment Evaluation Question

and the laboratory tests were the first measures of serum glucose concentration, the omission of the test rendered those evaluation points invalid. I excluded the affected points from the data analysis. This reduced the number of available decision points by 4.5%.

7.3 Unanswered Questions

Because the amount of expert panel time was strictly limited, the choice of questions to evaluate was circumscribed. This evaluation assesses both the absolute quality of the advice generated by the Ketoacidosis Advisor and its relative merit compared with actual hospital treatment. I discuss other questions that were not addressed by the formal evaluation below.

7.3.1 Detailed Evaluation of Advice

The evaluation method indicates the performance of the Advisor program in the aggregate. Comments from the evaluators indicated that at times they were in general agreement with some (real or advisor) treatment plan but had reservations about one particular part of the recommendation. The study design I used did not allow a detailed breakdown of the areas of agreement or disagreement with particular treatment options. An alternative would have been to separate the evaluation into distinct components such as fluid therapy, insulin therapy and potassium therapy. While providing additional information about the detailed performance, such a study design would create problems in assessing the overall performance of the system. How would one combine an excellent fluid recommendation with a dangerous potassium plan? I chose to have the evaluators integrate the different treatment components. One can measure overall system performance without having to create an ad hoc rating scheme to combine the individual parts.

As a prelude to further development of the expert system, though, such a detailed evaluation would be useful. System devlopment requires that the sources of controversy be identified and deficiencies in parts of the decision-making process be highlighted. This type of evaluation should be carried out before further development of the system is done.

7.3.2 What Would be Optimal

A fundamental weakness of this approach to the evaluation is that there is no indication of what an optimal decision should be in any particular case. In effect, the evaluators were constrained in the treatments they could vote for. Again, I chose this format because I was interested in comparing the performance of the Ketoacidosis Advisor to human clinical practice.

An alternate approach might have been to present the cases to a different panel of experts, who would reach a consensus decision on the appropriate treatment. This would provide three treatment options in place of the two that were offered to the evaluators in this study. I was unable to do this because of a shortage of experts to create all of the panels that would have been necessary.

Another method of getting information about a better treatment plan would be to have the evaluators also indicate what they would do in the situation being evaluated. This would then create the difficulty of merging conflicting treatment plans from each of the evaluators in order to determine the optimal treatment. Below I discuss the lack of agreement in individual recommendations. Since evaluations of particular treatment options could cover the full scale of the evaluation range, it would not be reasonable to assume that a consensus could be constructed from individual notes written by the evaluators. A conference would be needed to resolve the differences in the approaches of the individual evaluators.

7.3.3 Advantage of Faster Intervention

In order to avoid a temporal bias in favor of the Ketoacidosis Advisor, I selected all of the decision points at times that the actual treatment changed. This limited the choice of decision points, excluding times when only the Advisor recommended a change in treatment. Including the times when the Ketoacidosis Advisor would have changed treatment but the real treatment was unchanged, would have roughly doubled the number of decision points over the time frame used for the evaluation of the cases used in this evaluation.

To a certain extent, the decision not to look at the times when the Ketoacidosis Advisor would have recommended change when the real treatment remained the same reflects an assumption about the outcome of that examination. I assume that the advice given at the selected evaluation points is an accurate sample of the quality of the advice generated by the system. Furthermore, I assume (with much greater confidence) that the Ketoacidosis Advisor would react more quickly to data, since it

Table 7.1: Frequency Test Results for Rater Agreement.

	Real			**Advice**		
	Attending	Fellow	Both	Attending	Fellow	Both
Dangerous	11%	2%	6%	4%	4%	4%
Poor	36%	37%	36%	29%	26%	27%
Acceptable	28%	26%	27%	38%	32%	35%
Good	21%	31%	26%	14%	34%	25%
Excellent	4%	4%	4%	14%	4%	9%
N	120	136	256	120	136	256

Frequency of score by rater group, for both real and advice. Percentages may not sum to 100 because of rounding. The Pearson χ^2 was used to test the distributions for agreement. For the real treatment p = 0.037 (χ^2 = 10.187, DF = 4) and for the advice p = 0.001 (χ^2 = 17.724, DF = 4). In both cases, the differences are significant.

will immediately process all data presented to it.

As a result of this inability to evaluate timing, the evaluation carried out was a comparison of the advice from the system serving in a consultant role. The usefulness of the expert system advice in a monitoring role remains somewhat speculative, although there is no reason to believe that the performance would be any worse. Given the greater demands on the expert panel, a full evaluation of the effects of timing should wait until a system is designed that can be shown to be superior in the other evaluation. The next implementation should be tested in this manner.

7.4 Results of the Evaluation

The overall evaluation is that the Ketoacidosis Advisor performs no worse than the clinical staff that treated the patients. There is evidence that the computer-generated advice was better, although the advantage was not sufficiently high to reach the .05 level in the statistical tests that I applied to the data.

The effects of the evaluators' levels of training can be clearly demonstrated. The grades given to the program by the attending physicians were significantly different than the grades given by the fellows. (See table 7.1). The computer advice was viewed more favorably by the attending physicians. Since all cases except number 16 had the same number of fellows and attendings evaluating them, the difference is not likely to be the result of a skewed data mix.

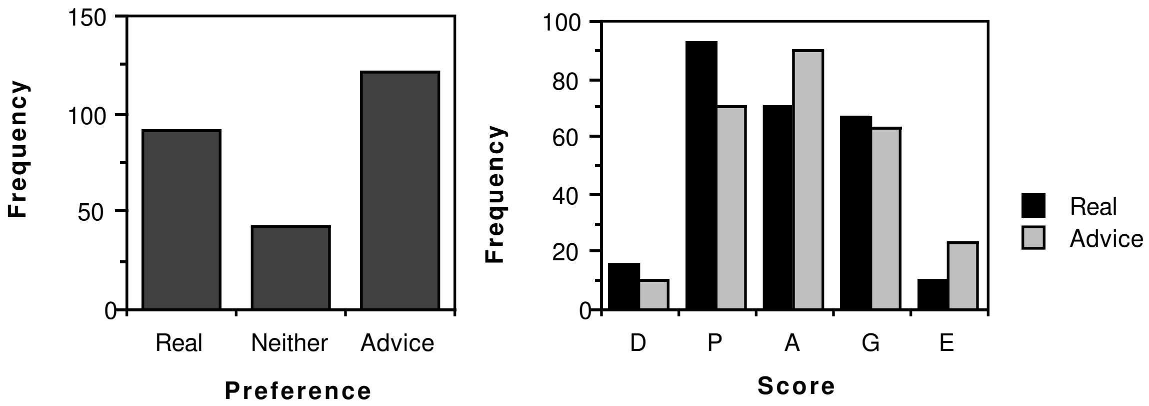


Figure 7.3: Preference and Evaluation Scores of All Cases

The preference graph shows the number of times real or advice was preferred. The score graph shows the number of times each score was given to the real or advisor treatment plans. Score key: D = Dangerous, P = Poor, A = Acceptable, G = Good and E = Excellent. Data is aggregated from all raters (N = 256).

7.4.1 Statistical Tests

I performed statistical tests on the data gathered from the evaluation forms. Because all of the data were paired and the evaluation categories were totally ordered, I chose the sign test for the analysis. The sign test evaluates the sign of the difference of paired observations. If the populations from which the samples are drawn are the same, then one would expect the difference in the signs of the evaluation to be zero. This hypothesis is tested using a binomial approximation. I carried out calculations using the SYSTAT 5.0 program on an Apple Macintosh. The results are presented below:

Table 7.2 summarizes the results of the sign test applied to the aggregate data from the evaluation. The three tables report the results of the analysis using the full five-category range of the test instrument, using the two-category collapsed scale, and a direct comparison of the preference results. Of these choices, the most appropriate is the preference results, because this question is posed in the same terms that the sign test evaluates: Is option A better or worse than option B? The results suggest a difference between the real and the advice. Significance results are consistent with results of the McNemar Symmetry χ^2 test.[3]

This claim is, however, rendered suspect, because not all evaluators looked at every

[3] The five-category evaluation uses a five-by-five table which had too many sparse cells to yield accurate results. For the two-category evaluation p = 0.011 (χ^2 = 6.519, DF = 2) and for the preference evaluation p = 0.040 (χ^2 = 4.206, DF = 2).

Table 7.2: Sign Test Results

	5 Category	2 Category	Preferences
Real Better	84	50	92
Advice Better	104	79	122
Ties	68	127	42
	$p = 0.047$	$p = 0.014$	$p = 0.047$

Each row lists the number of times the particular treatment plan was judged better in a paired comparison. P values are from the two-sided probability of the sign test. Data aggregated from all raters ($N = 256$).

case. Because my hypothesis of great inter-rater variability was confirmed by the data, this makes the combination of different cases questionable. I used two separate methods to correct for this bias. One correction involves examining only the results of the raters who evaluated every case. The second method involves aggregating evaluations for each decision point so that each decision point is weighted equally.

If the analysis is restricted to those three raters who examined every case, no statistically significant difference is apparent. With 120 data points, a difference of opinion of 75–45 (5–3 or 62.5%–37.5%) would be significant at the $p = 0.05$ level. This level of difference was not achieved. Applying the same three tests, there was a difference of 5 to 6 choices, with the five-category and preference tests in favor of the real therapy and the two-category test in favor of the Advisor. The case with the fewest evaluators (case 15) was coincidentally the case in which the Advisor had the worst relative performance. This taints the aggregate results from table 7.2 because evaluations from the Advisor's worst case are fewer than evaluations from the cases in which it performed better. On the other hand, the absolute performance on case 15 was the best.

An aggregate evaluation measure at each decision point can be constructed by subtracting the proportion of evaluators who preferred the real therapy from the proportion who preferred the advice. This procedure will yield a value in the range -1 to 1 representing the net preference fraction. A score of 1 would indicate unanimous preference for the advice, -1 unanimous preference for the real treatment, and 0 no net preference (the same number of votes for each). The results of this measure are shown in figure 7.4. 19 decisions favored advice, 13 favored the real action and 8 were ties. No significance was demonstrated either by the Wilcoxon Signed Ranks test or the paired samples t test[4].

[4] The tests were applied to the underlying proportions before the subtraction used in the aggregate measure. Each value ranged from 0 to 1. The Wilcoxon Signed Ranks test showed $p = 0.441$ ($Z = -0.770$). The paired samples t test had $p = 0.423$ ($t = 0.810$, $DF = 39$, mean difference = 0.064, standard deviation of difference = 0.501)

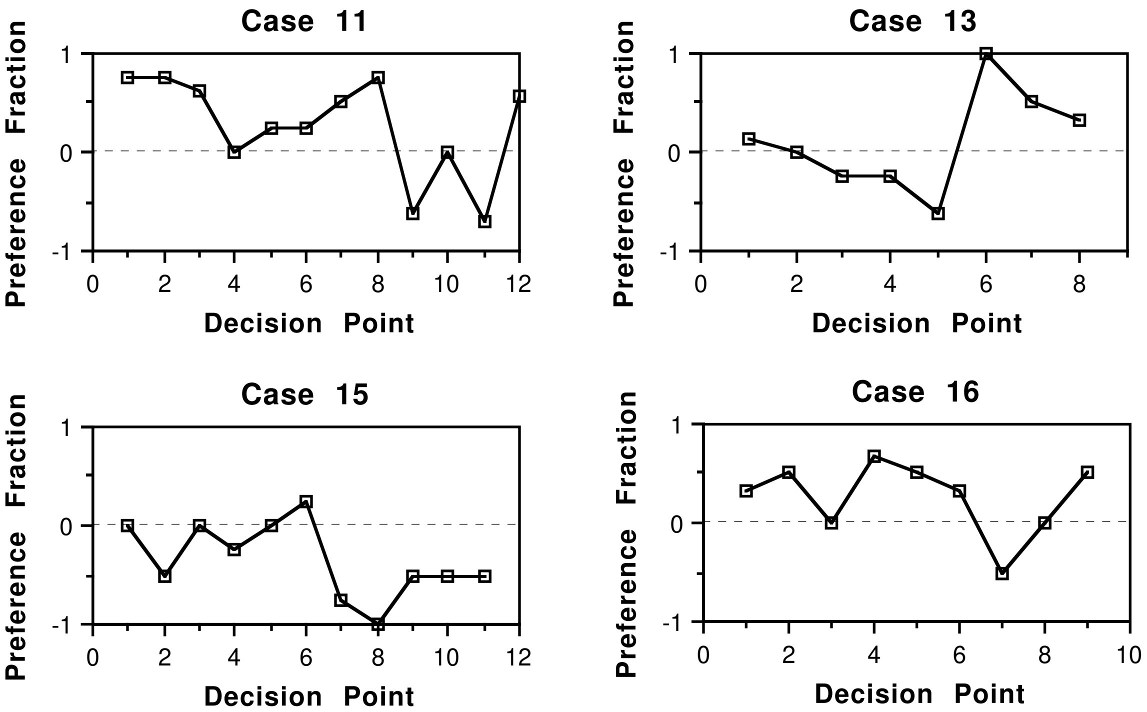


Figure 7.4: Net Preference Fraction by Case

Preference fraction is calculated by subtracting the proportion of evaluators who preferred the real therapy from the proportion who preferred the advice. Values range from -1 to 1: 1 indicates unanimous preference for the advice, -1 unanimous preference for the real treatment, and 0 no net preference. Each decision point is calculated and plotted separately.

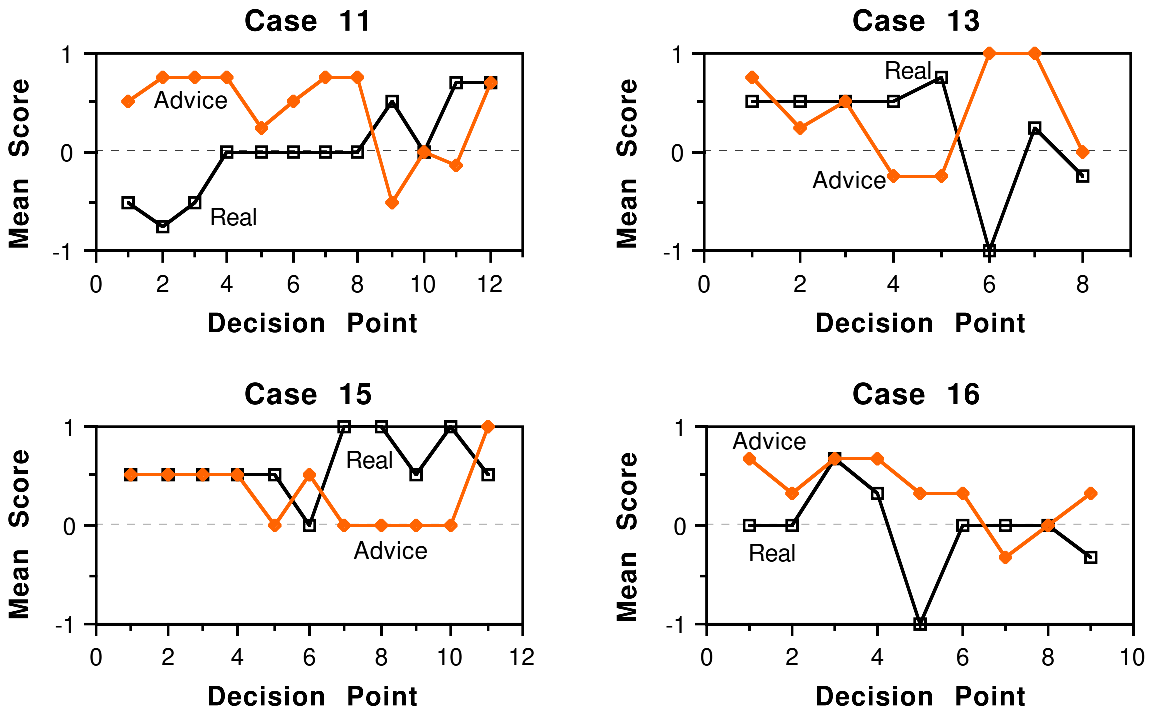


Figure 7.5: Two Category Evaluation by Case

Score calculated by assigning the value 1 to any evaluations in the top three ("OK") categories and -1 to any evaluations in the bottom two ("Bad") categories. The mean score is the sum of scores divided by the number of raters for each decision point. It indicates the relative proportion of OK and Bad evaluations: Unanimitiy results in a score of 1 (or -1) and equal numbers of OK and Bad evaluations gives a score of zero. Real score is plotted in black; advice is shaded.

In light of these results it is clear that there was no discernable difference in the quality of the decisions evaluated by the panel. The raw data do indicate that particular decisions varied widely in their acceptance, both in absolute and relative terms. In the next section I examine the differences in depth.

7.4.2 Breakdown by Cases

The cases showed differing success. The simplest summary involves aggregating the two-category evaluations. I assigned the value 1 to any scores in the top three ("OK") categories and -1 to any scores in the bottom two ("Bad") categories. The mean score for each decision point is shown in figure 7.5. The mean score indicates the relative proportion of OK and Bad evaluations. A score of zero indicates equal numbers of

OK and Bad evaluations. Case 11 had the best Advisor performance and case 15 had the worst. The other two cases were roughly similar in the aggregate, although a difference in opinion between the attendings and the fellows was evident. The fellows scored case 13 higher than attendings, while attendings scored both case 11 and case 16 higher. Scores on case 15 showed no major difference.

Closer examination of the results in figure 7.5 reveals that the advice generally remains in the upper half of the rating space. This means that the advice was rarely judged to be poor or dangerous by a majority of the evaluators. In contrast to the real treatment, there was never a unanimous evaluation of the advice as being bad. (Compare with the real results in case 13, decision 6 and case 16, decision 5.) In the case with the worst relative performance (case 15), the lowest scores were 0, meaning an even split between the OK and bad evaluations.

In the following sections, I examine cases where the recommendations were markedly inferior to the actual treatment. I will give a short summary of the issue in the decision. If the reader wishes more detail, the case data are included in appendix A, and the evaluation details in appendix B.

For each case, the domain expert and I examined those decisions where the Advisor's performance was judged significantly worse than the actual treatment. In this analysis we endeavored to identify likely elements of the decision that were responsible for the poor performance. The explanation of the poor scores comes from our *post hoc* analysis and does not have input from the evaluation panel. Because the evaluators were identified only by level of training, the study design did not allow for followup discussions about individual evaluations. We also did not have the evaluator time for a general discussion of either the cases or particular decisions with the nephrology group.

Case 11

The results for case 11 are shown in figure 7.6. The graph shows the average weight of the absolute scale. I calculated this by assigning dangerous a value of 1, poor a value of 2, ..., and excellent a value of 5. I then added the values together and divided by the number of evaluations to provide a mean. The bars show the value of plus or minus one standard deviation. From the graph it is apparent that a large amount of variability in the evaluation was present. The transformation of an ordinal into a cardinal scale is not without its pitfalls. The difference between excellent and good is not necessarily the same as the difference between poor and dangerous. Nevertheless, the use of a scale similar to scholastic grade point averages has some benefits. The major advantage is that it allows the standard deviation to be used to quantify the disagreement about the objective rankings. As figure 7.6 shows, there is a great deal of variability in opinion.

The Ketoacidosis Advisor had the best performance on case 11, with a significantly better performance as evaluated by eight of the raters—four attendings and four fellows. (See table 7.3).

Table 7.3: Case 11 Sign Test Results

	5 Category	2 Category	Preferences
Real Better	26	16	29
Advice Better	47	36	54
Ties	21	42	11
	p = 0.008	p = 0.008	p = 0.008

Each row lists the number of times the particular treatment plan was judged better in a paired comparison. P values are from the two-sided probability of the sign test. Data aggregated from all raters (N = 94).

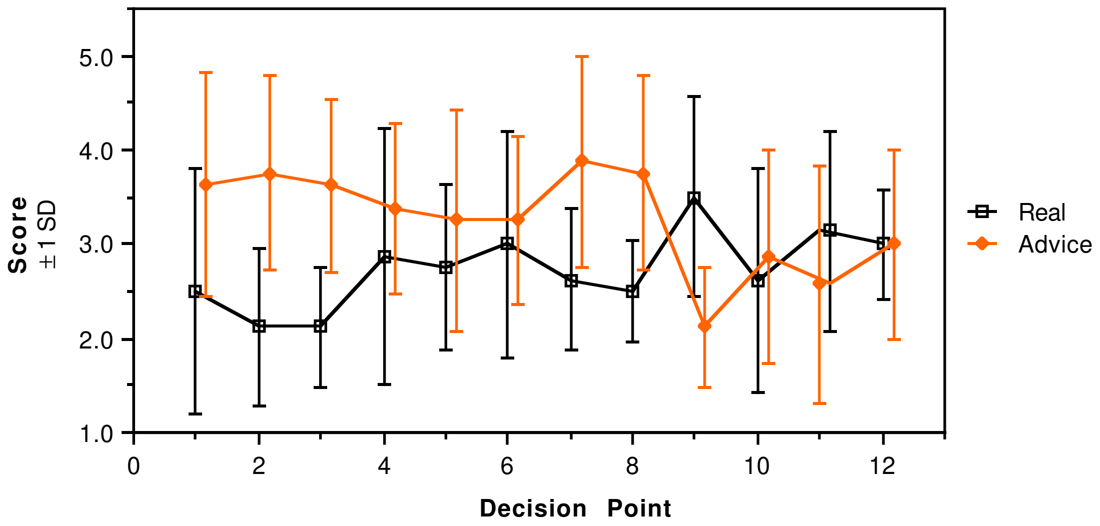


Figure 7.6: Case 11 Average Scores

Scores are calculated by assigning numerical values to the categories analagous to grade point averages. The following assignments were used: Dangerous = 1, Poor = 2, Acceptable = 3, Good = 4, and Excellent = 5. The bars show ± 1 standard deviation. The advice graph has been offset horizontally to increase legibility.

Decision 9. The reason for this low evaluation was not completely clear. The chief suspects are the decision to recommend stopping the IV fluid infusion, and the decision not to recommend a subcutaneous injection. The former relies on an assessment in the absence of strong evidence one way or the other about the patient's fluid status. Whether the patient is capable of eating or not is difficult to assess from the records. The program was working on the assumption that the patient was able to eat, and that fluid balance was therefore not a major problem.

The issue of insulin injections is the other likely reason for the relatively poor Advisor performance on this question. The actual treatment was to give the patient his normal morning dose of 50U Lente insulin at 11:30am. This is outside the time window that the Advisor uses for giving a morning dose of insulin. Since it is too late to give this morning's insulin, the Advisor would want to continue a low level of insulin infusion and wait until the next morning to start the patient on his regular course. In fact, the Advisor had recommended giving the 50U Lente insulin from 7am until 9:45am, showing a more timely recommendation.

Decision 11. The Advisor recommends potassium supplements; the real treatment did not. This decisions runs into the domain controversy surrounding the administration of potassium. The Advisor follows a more aggressive rule than the panel seemed comfortable with. The problem could also be one of degree, influenced by the exact point where the threshold is set. The Advisor is following its late-stage, less aggressive rule in making this particular decision, but it is still perhaps more aggressive than the panel would like.

Case 13

Case 13 received the best evaluation from the fellows and the worst absolute rating from the attendings. In terms of preferences, this case had the best results among the fellows and was in third place among the attendings. The case was evaluated by eight raters—four attendings and four fellows. (See table 7.4).

Decision 2. The most likely point of dispute was the inclusion of bicarbonate supplements in the advice and not in the real therapy. The use of bicarbonate in treating DKA is also controversial. The rationale in its favor is that patients with very low serum bicarbonate are maintaining their pH by hyperventilation, thus reducing the amount of carbon dioxide in the blood. If they tire, then they may not be able to compensate anymore, with a resultant quick fall in the blood pH. The argument against giving the bicarbonate is that it is unnecessary and that too much could could cause harm by changing the acid-base balance too quickly.[5] This decision was particularly difficult to analyze because the expert

[5]Bicarbonate needs time to diffuse across the blood-brain barrier.

Table 7.4: Case 13 Sign Test Results

	5 Category	2 Category	Preferences
Real Better	21	14	23
Advice Better	25	19	30
Ties	18	31	11
	p = 0.410	p = 0.486	p = 0.410

Each row lists the number of times the particular treatment plan was judged better in a paired comparison. P values are from the two-sided probability of the sign test. Data aggregated from all raters (N = 64).

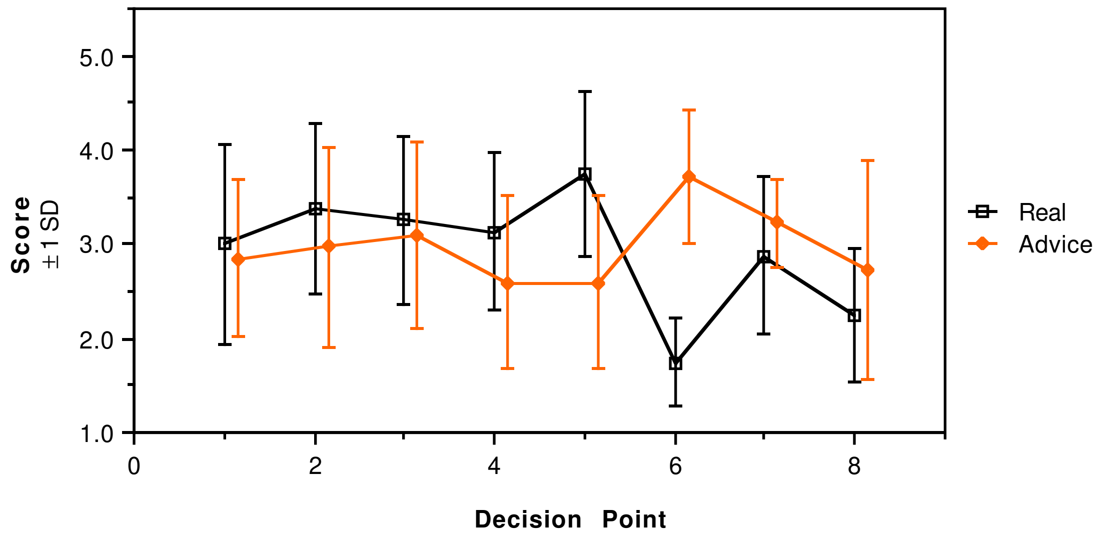


Figure 7.7: Case 13 Average Scores

Scores are calculated by assigning numerical values to the categories analagous to grade point averages. The following assignments were used: Dangerous = 1, Poor = 2, Acceptable = 3, Good = 4, and Excellent = 5. The bars show ± 1 standard deviation. The advice graph has been offset horizontally to increase legibility.

Table 7.5: Case 15 Sign Test Results

	5 Category	2 Category	Preferences
Real Better	25	12	25
Advice Better	8	6	9
Ties	11	26	10
	p = 0.010	p = 0.238	p = 0.010

Each row lists the number of times the particular treatment plan was judged better in a paired comparison. P values are from the two-sided probability of the sign test. Data aggregated from all raters (N = 44).

panel did not agree on the need to use bicarbonate. Some members would have used aggressive bicarbonate therapy, while others thought it unnecessary.

Decision 4. This Advisor's decision was probably faulted for a combination of a potassium recommendation and its recommendation of a higher infusion rate for intravenous fluids. The potassium recommendation follows the aggressive rule derived from Alberti and Hockaday [1]. The fluid rule can be attributed to the deficit-estimation problem. The Advisor does not modify the deficit estimate based on other clinical signs. Since three liters of fluid were infused, blood pressure had risen, and pulse had dropped, the patient was most likely no longer dehydrated. The inability to recognize this change of state is an error in the knowledge base.

Decision 5. Same as above. There is also the additional difference that the real action was to reduce the insulin infusion. The Advisor would wait until the serum glucose concentration dropped below 240mg/dl. The most recent laboratory measurement before this decision returned a value of 247. It is likely that the threshold chosen in the Advisor's rule was too low to satisfy the evaluation panel.

Case 15

In case 15, the Advisor had the worst performance relative to the actual clinical treatment (see table 7.5). It received the second highest absolute performance rating from the fellows, and the lowest absolute rating from the attendings. Nevertheless, examination of figure 7.5 reveals that the Advisor's performance never drops below the zero line—the line at which there are equal numbers of OK and Bad ratings—so there was no consensus that the Advisor's recommendations were unacceptable, even on its worst case. The case was evaluated by two attendings and two fellows.

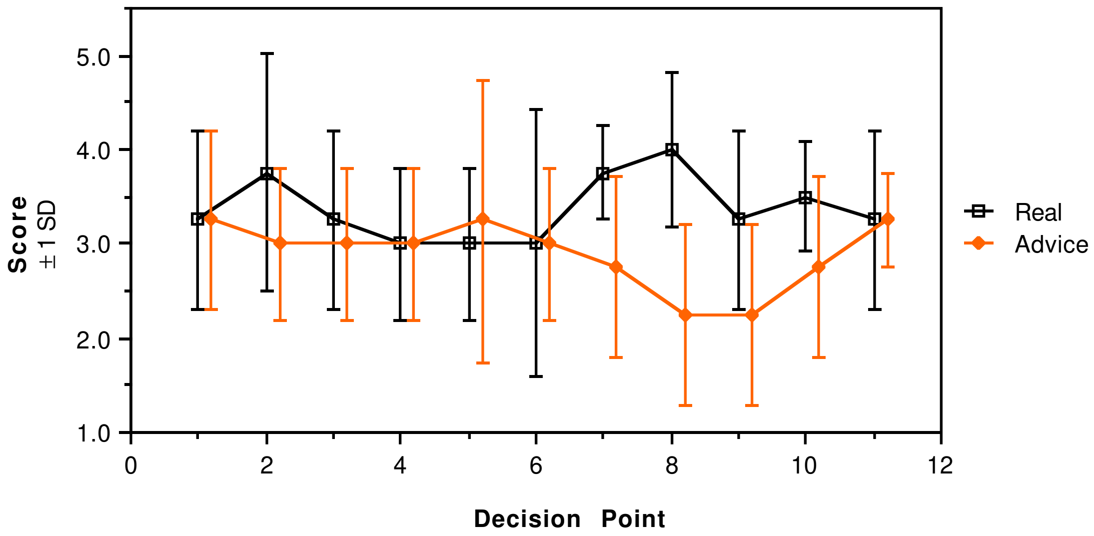


Figure 7.8: Case 15 Average Scores

Scores are calculated by assigning numerical values to the categories analagous to grade point averages. The following assignments were used: Dangerous = 1, Poor = 2, Acceptable = 3, Good = 4, and Excellent = 5. The bars show ± 1 standard deviation. The advice graph has been offset horizontally to increase legibility.

Decision 5. The difference between the real treatment and the recommendation concerns when to start adding glucose to the fluids. The Advisor waits for a serum glucose below 240mg/dl before adding supplemental glucose to the fluids. The most recent previous measurement was 261mg/dl. This is similar to decision 5 in case 13.

Decision 7. The poor performance was due to a programming bug in a Ketoacidosis Advisor rule. The Advisor was trying to switch from intravenous insulin to a subcutaneous regimen. This involved continuing a low flow (1U/hr) of insulin infusion while awaiting an appropriate opportunity to give an injection. Unfortunately, the wean strategy was not recognized as an insulin infusion by the section of the Advisor which recommended fluid type. The consequence was that the Advisor did not recommend adding glucose (D5) to the fluid infusion, even though the patient's serum glucose level was fairly low (129–195). This was the result of a programming error in the rule for deciding fluid type.

Decision 8. Same as above, but with even lower serum glucose. A second factor was the inability to identify the actual strategy as weaning the patient from intravenous insulin. The infusion rate used for weaning in this case (2U/hr) is higher than the 1U/hr weaning rate the program recognizes. Also, the weaning continued after the subcutaneous insulin was given. Since this is a common strategy, it should have been considered by the Advisor, but it was not.

Decision 9. Similar to decision 8. The Advisor did not suggest adding glucose to the infusion fluids because it wanted to stop giving intravenous insulin and the patient was able to eat. The panel may not have attached the same importance to the patient being able to eat as the program did.

Decision 10. The real action was to end the 2U/hr insulin drip, whereas the Advisor wanted to continue the infusion. This decision was the result of a different knowledge-base error. In this case a mistake was made in the insulin strategy determination rule. Because of this mistake, the program incorrectly thought the patient was becoming much more acidotic, and therefore not stable enough to be weaned from the insulin infusion. After discussion with Dr. Meyer, I concluded that the rule responsible for this decision was incorrect.

Case 16

Case 16 was handled well by the Advisor, but there were insufficient evaluations to show a statistically significant difference. This case was rated by two attendings and four fellows. The attendings had a much stronger preference for the computer-generated advice than the fellows in this case.

Decision 7. The only significant difference in the treatment suggestions was in intravenous fluid therapy. The Advisor recommended stopping fluids, whereas

Table 7.6: Case 16 Sign Test Results

	5 Category	2 Category	Preferences
Real Better	12	8	15
Advice Better	24	18	29
Ties	18	28	10
	p = 0.050	p = 0.078	p = 0.050

Each row lists the number of times the particular treatment plan was judged better in a paired comparison. P values are from the two-sided probability of the sign test. Data aggregated from all raters (N = 54).

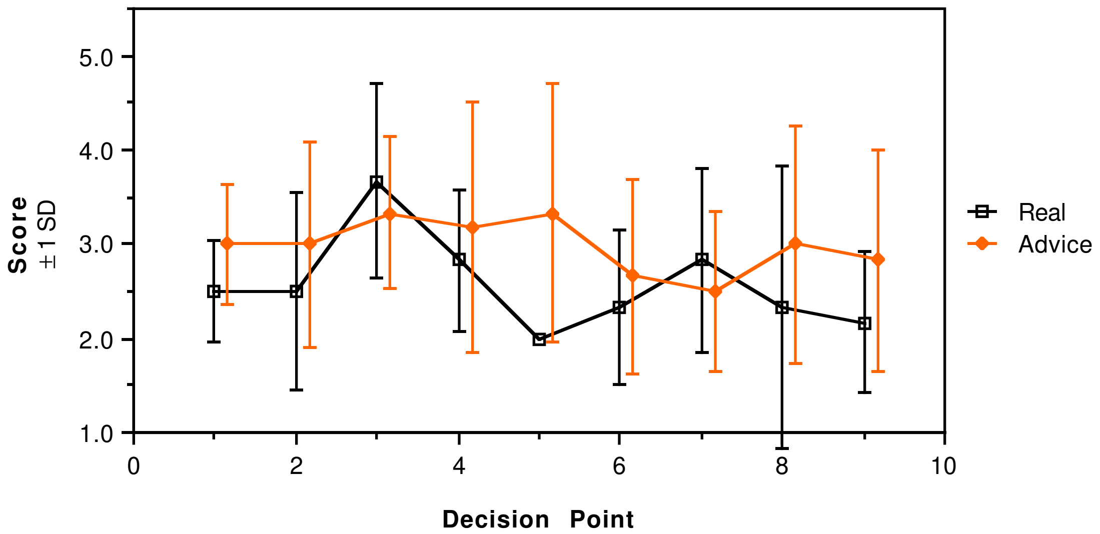


Figure 7.9: Case 16 Average Scores

Scores are calculated by assigning numerical values to the categories analagous to grade point averages. The following assignments were used: Dangerous = 1, Poor = 2, Acceptable = 3, Good = 4, and Excellent = 5. The bars show ± 1 standard deviation. The advice graph has been offset horizontally to increase legibility.

the clinical staff elected to continue IV fluids at 175ml/hr. Because the patient had a blood pressure of 142/92, had a pulse of 84, had already received almost three liters of fluid, and was able to take fluids orally, the domain expert did not feel the data supported a need for continued IV fluid therapy.

Summary

I have discussed the decisions for which the Ketoacidosis Advisor was judged to be inferior to the actual treatment above. The problems in the advice can be summarized in the following categories:

Data Problems There are fundamental problems in assessing hydration status when one is unable to examine the patient physically. The medical record typically does not contain enough information to make that determination on the basis of the entries. The mental state of a patient is generally not recorded, but enters into an assessment of whether they are dehydrated or not. Similarly, it is not always possible to determine from the clinical records when a patient was eating (although this is somewhat easier). In the absence of definitive data about these conditions, the program can run into difficulty in making these judgements.

If the information were available, the knowledge base could be reprogrammed to take advantage of the input. Some of this problem is the result of testing the system in an off-line mode. Because there is no interaction between the clinical staff and the Ketoacidosis Advisor, it is not possible for the computer to request the types of information that are needed for its assessment. For example, finding out if the patient were eating could be trivially accomplished by simply asking the question. This is not invasive and would not require much thought on the part of the person interacting with the system.

Other data problems, such as the problem of unrecorded urine output from patients who are ambulatory, cannot be solved so easily. It is possible to make the fluid decisions consider the context of the patient care, since any patient who is well enough to be up and out of bed is not severely dehydrated. This could be used as a surrogate measure which is sufficiently precise to allow one to conclude that intravenous fluids are not needed (or are needed only under much different circumstances) compared to a patient who is confined to bed.

Domain Controversy As noted in the chapter describing the design of the Ketoacidosis Advisor, several aspects of the treatment of DKA remain controversial. In particular, the use of bicarbonate to treat acidemia and the aggressive use of potassium supplements early in DKA provoke differences of opinion. The choice of any strategy regarding these aspects of the treatment will result in some evaluators disagreeing not only with the specifics of the advice, but with the entire premise underlying the advice itself. In areas of medicine where there is no consensus as to the "best" treatment, this problem is inevitable. The controversy is addressed by Kassirer, et al. [41, p. 137]:

> The use of exogenous alkali in treating diabetic ketoacidosis is controversial. As with most medical controversies, the issue revolves around an assessment of the risks and benefits. . . . When the acidosis is severe (plasma bicarbonate concentration less than 8 to 10 mEq/L), however, the benefits of administering sufficient alkali to partially repair the deficit (i.e., raise plasma bicarbonate to approximately 12 to 14 mEq/L) far outweigh the risks.

Particularly in the case of potassium supplements, the nephrologists may be influenced by the large number of patients that they treat who have kidney disease. Since excessive potassium is also dangerous, there is a hesitation to prescribe potassium supplements when patients present with laboratory values in the upper part of the normal range. Normal healthy humans can easily handle excess potassium by excreting the surplus in the urine. Patients with kidney failure cannot. Giving too much potassium to a patient in kidney failure can cause major problems, while the same is not true of patients in DKA. The justification for using an aggressive treatment is that DKA patients have reduced total body stores of potassium. The effect of the acidosis is to cause potassium to shift out of the cells and into the blood stream, resulting in a normal laboratory measurement even though the total supply of potassium in the body is reduced. As the acidosis is corrected, the potassium will return to the cells, causing a drop in the serum concentration. This migration of potassium is also aided by the administration of insulin.

Advisor Deficiencies The one program bug identified as the likely cause of poor performance was one that could be easily fixed: the failure to consider the insulin weaning strategy (which includes intravenous insulin adminstration) as a type of intravenous insulin. This is a trivial change. The general problem of actual strategy identification is a more difficult, albeit artificially contrived, problem. In an actual clinical setting, the Advisor could ask for the strategy, a possibility not open to the program when working retrospectively with case records.

Adjustment of thresholds for recommending potassium supplements falls into the class of easily amended rules. There are currently two different strategies used for recommending potassium corresponding to the early and late phases of DKA. In the early phase, total body stores are depleted and imminent potassium shifts into the cells are anticipated. In the late phase, the patient is more stable, so a less aggressive approach is used. In each case, the program relies upon test values to determine how much potassium to recommend. The threshold values at which the supplements should be given can be easily modified. The experts at

New England Medical Center may be more amenable to a slightly less aggressive approach.

A more difficult problem is posed by the choice of thresholds for glucose tests. The choice of values was influenced too heavily by the endpoints of ranges on the less accurate finger-stick assessments. The discussion with the endocrinologists indicates that the finger stick is much less trusted as an assessment tool. Since I sought to have a less complex way of combining the two test types, I opted for compatible thresholds. Since one range of the finger stick covered 240–400mg/dl, I used 240 as the boundary below which glucose would be added to the fluid recommendation. This value is a little lower than would have been chosen in the absence of the desire to coordinate the two measuring systems, but it is consistent with published guidelines [15, 20]. Although it would involve more complicated programs, separate thresholds could be used, with appropriate safeguards to prevent "thrashing" between recommendations in the event that the two types of tests alternated. This type of safeguard could involve not having the range 240–400 change the previous fluid type therapy at all.

Although there were certain decisions for which the Advisor's decisions were considered to be inferior to the actual clinical practice, none of the problems can be traced to a systematic flaw in the TCS design. Aside from programming or rule-encoding problems, the difficulties that the Advisor encountered in the evaluation were related to properties of the domain which transcend the technology used to implement the particular expert system. In no case was the problem an inablility properly to integrate data that arrived over the course of time.

7.5 Discussion

The overall performance of the Ketoacidosis Advisor was sufficiently good to demonstrate that practical clinical applications can be implemented in TCS. The ability to give management advice comparable to actual treatment at a major medical center shows that the system can perform credibly in a dynamic environment.

7.5.1 Acceptable Performance

The statistical analysis does not prove that the advice given by the program is better than the actual clinical treatment. It does, however, give reason for confidence that the performance was similar to what was actually done in the clinic. (Unfortunately, the expert panel often did not really like what was actually done. Several of the evaluatiors expressed horror at the thought that DKA patients were being cared for in the manner that the medical records indicated.) In order to make the Ketoacidosis Advisor function at the level of the review panel, a more complete knowledge base would be needed. Alternate methods for determining hydration status, such as the analysis of blood pressure and pulse changes, would need to be added. Similarly, there would be a need to include such common-sense rules as concluding that a patient is in fairly good condition if he can leave his bed to void. Finally, special case features for handling more complicated cases involving heart and kidney failure would be necessary. Since the management of those cases relies on a careful monitoring of the fluid balance, the ability of a computer-based system to do precise calculations becomes more important.

The major question that the evalution as done here could not answer was whether there was a significant time advantage from the effectively instant response of the Ketoacidosis Advisor to incoming data. My own observations indicate that there is often a lag between data becoming available and the clinical response to those data. Since a dedicated monitoring system would not be distracted by other important chores, it would react more quickly. Since the data processing time is sufficiently rapid, the computer system would not react to new information any more slowly than the actual clinical staff. This part of the evaluation was not done for the reasons cited above, but one can safely assume that any effect from the increased speed of the decision-making would be an advantage for the Ketoacidosis Advisor.

Since the statistical tests were close to achieving statistical significance, it may be possible to show significance with a larger data set. One of the difficulties at the design phase was that it was not clear what the magnitude of the difference would be. This prevented the performance of an *a priori* power analysis before the evaluation was carried out.

7.5.2 Lack of Clear Consensus

On the other hand, the lack of a clear consensus as to the appropriate treatment is demonstrated by the number of cases with a very large spread of evaluations. For some questions[6] the same treatment was evaluated using the full scale of the measurement instrument. A single treatment suggestion in a concrete situation received ratings from excellent to dangerous! There was only one case of unanimous choice of evaluation.[7] The mean category spread for real treatment was 3.25 and for advice 3.5 categories. Such controversy makes it difficult to say anything with a high degree of confidence. There is an apparent lack of an unambiguous gold standard for measuring the treatment options. In spite of this lack of individual consensus, there is no apparent difference between the quality of the treatment recommended by the Advisor and that actually rendered in the hospital.

Matching human performance fulfills the performance goal of the Ketoacidosis Advisor experiment. The expert system was able to monitor patients successfully over a period of many hours and respond to changes in the patient state while coping with

[6]Advisor treatment case 11, question 11 and case 16, question 4; Real treatment in case 11, questions 1, 4 and 10.

[7]Real treatment in case 16, question 5, with six reviewers.

treatment decisions that did not match the advice generated by the program. The ability to function in such a demanding environment is a validation of the basic design premise of TCS. All of the necessary reasoning constructs could be implemented within the TCS framework. As indicated in the section on general-purpose modules designed for the Ketacidosis Advisor, some of the techniques used in the implementation have application beyond the specific area of expertise. These include the code used to keep track of the state of a changing therapy, as well as the management of the interaction between the time advice is generated as a result of the arrival of new data and the time that the advice is carried out.

Chapter 8

Related Work

The discussion of related work is divided into four parts. First, I dicuss work on temporal reasoning. This line of research focuses more on reasoning about the relationships among events happening at different times rather than on the process of updating conclusions in response to changing data. The TCS side-steps this issue by requiring exact time points, rendering the question of the relationship between two time points or two intervals a trivial computation. In return, TCS gives up the ability to deal with ambiguous time information. Fortunately, in the monitoring domain, this is not a major problem.

In the second section I discuss Truth Maintenance Systems, which share the TCS's emphasis on providing an efficient updating system based on the idea of data dependencies. In the third section I discuss medical systems, concentrating on the Ventilation Manager and TOPAZ, both patient management systems. Finally, I briefly discuss blackboard architectures and real-time expert systems.

8.1 Temporal Reasoning

Research in temporal reasoning has concentrated on the development of the representation of time and on reasoning about the temporal relationships among individual events or event clusters. Work on non-monotonic logics has focused on the problem of revising beliefs when new data becomes available. The problem that has been largely ignored in this line of research is the problem of characterizing data that is changing over time. The nonmonotonic logics, for instance, assume that there is only one true state of the world, but that this is revealed bit by bit, so that it becomes necessary to retract certain conclusions about the world. While this is often true, it does not provide support for histories.

8.1.1 Temporal Representation and Relations

Most of the AI work on temporal reasoning has fallen into two categories: representations for describing temporal events, and systems for reasoning about the temporal relations among events. Little work has been done on the interpretation of data that change with time, or on the reasoning processes associated with this type of interpretation.

The most extensive work on a temporal representation has been the development of the temporal interval algebra of Allen [3, 4, 5], along with attempts to extend it by Kandrashina [40] and Ladkin [50, 49]. These systems provide a strict definition of temporal intervals and an algebra for manipulating them. This allows reasoning about the relationships among distinct intervals, for example, the determination of whether they overlap or not. An application of this that is being pursued by Allen is the use of this algebra in planning [6]. It is interesting that there is a desire to avoid having both points and intervals together because that would destroy mathematical purity and introduce difficulties in interpretation. This sidesteps a major reasoning problem at the system end by requiring the user to perform the hardest part of reasoning, namely the structuring of raw sample data into abstract intervals.

Other researchers have been exploring the calculation of temporal relationships among data points. Kohane [44] has investigated a method of propagating information about the endpoints to restrict the relationships among events organized on a timeline. An application of linear programming to the problem of calculating the tightest bounds of a set of temporal constraints is reported by Malik and Binford [55]. By allowing uncertainty in the boundaries of intervals, these systems have a less restrictive temporal representation than the TCS. The restriction to specific intervals in TCS is dictated by computational convenience, since the specific intervals eliminate the potential for combinatorial explosion. Unfortunately, these techniques cannot guarantee unique endpoints, so they could not be used to add fuzzy endpoints to TCS.

Mittal [59] has examined the relationship of information in medical records in particular to certain key events, such as admission. He uses a disjoint decomposition of the time-line to produce efficient reasoning about the course of a patient's hospital stay. There is also a component of natural language processing involved in this effort (see Obermeier [65]).

8.1.2 Logic-Based Approaches

Attempts to specify temporal reasoning in classical logic run into several problems. On a philosophical level one is faced with the problem of using a formalism designed expressly for reasoning about the absolute truths of philosophy and mathematics. Attempts to modify logic for temporal reasoning must overcome the inherent design bias of logic as a language against representing changing concepts. The formalism was created for eternal rather than temporary facts. As the TCS experiments show, there is great power in attaching a special interpretation to time. Unfortunately, this can be difficult to do in the general case. The major practical difficulty is that useful temporal logics are undecidable [31].

It is also often difficult to formalize certain types of common-sense reasoning that people find quite easy to do. The circumscription approach [51, 58, 86] minimizes logical predicates according to syntactic criteria. Similar minimization techniques can be applied to nonmonotonic logics [79]. Hanks and McDermott [33] describe some of the difficulties of dealing with the frame problem using current nonmonotonic logics. The example which is used extensively in the literature is that of loading a gun, waiting and then shooting someone with the gun. The intuitive conclusion that the person who is shot will then be dead is not the only conclusion admissible under the logic's rules. As Hanks and McDermott show, it is possible to assume either that the gun stayed loaded during the waiting period or that it became unloaded. This was described by them as a very disturbing result, indicating that the approach being taken by the non-monotonic logics was in need of some help.

What makes the example so intriguing is that both outcomes make sense. The most likely intuitive result is that the person is dead because simply having a loaded gun sitting around will not cause it to become unloaded. On the other hand, if a sufficiently long period of time elapsed, say several years, then it would also be plausible to believe that the gun had become unloaded in the waiting period. Because there is no metric, it is difficult to express the dependence of the preferred conclusion on the length of the waiting period. The existence of a time metric in TCS makes it easy to encode this particular inference. Although this particular reasoning strategy for this particular domain can be programmed in TCS, a general-purpose reasoner without specific information about the domain and the particular inference to be made is still out of reach.

The following is a summary of the shortcomings of logic-based schemes:

1. **The Frame Problem.** The duration of propositions in a logical formalism must be explicitly stated. This means that every change of state must contain the information necessary to describe the new state completely. An active area of research focuses on the attempt to automate the process of limiting the effect of state changes without explicitly mentioning the entire world. Unfortunately, these schemes run into the next problem.

2. **The Syntactic Solution.** The circumscription approach creates an inference method that allows all normal properties to endure (unless explicitly declared to be abnormal). This provides a mechanism, but no guidance as to how it should be used. What is being proposed is a syntactic solution to what is essentially a semantic problem. Thus the circumscription approach is limited and has difficulty formalizing reasoning that most people find easy. The attempt to find a syntactic solution to a semantic problem is flawed. There is some recognition of this problem, since recent work in circumscription and nonmonotonic logic is

exploring ways of adding policies [52] or preferences [87] as a way of controlling the reasoning. These represent attempts to introduce semantic content to the default reasoning decisions.

3. **Undecidability.** In order to capture interesting temporal behavior, the logics used must be so complex as to become undecidable. If undecidability is present, then part of the appeal of using the language is lost. If decidability is already forfeit, then I would argue for adding Turing completeness to the system to allow the greatest flexibility in algorithms as well as to enable one to program in a convenient computer language.

I have concluded that a general solution to this problem will be impractical due to the complexity inherent in sufficiently powerful formal logical systems. It is also the case that in many domains, the types of decisions that will need to be made by an expert system are sufficiently constrained that they can be programmed without too much difficulty. These heuristic approaches have the advantage of reasoning efficiently in domains in which a strictly formal approach is too costly. By not restricting the form of the reasoning in the modules of the TCS, the greatest flexibility is preserved. The action that a programmer wishes a reasoning system to take is usually fairly straightforward.

8.2 Truth Maintenance

The idea of a Truth Maintenance System (TMS) can trace its roots to the dependency-directed backtracking and constraint-propagation work of Stallman and Sussman [83]. This work was extended by Doyle [23] and McAllester [57]. The central idea was that a more efficient recovery from errors or false assumptions could be made by keeping a record of the dependencies among conclusions. This introduced intelligence into what had been a blind backtracking approach. All the conclusions of a system were recorded in a dependency structure which provided the database for backtracking.

Although the details vary from system to system, the basic approach has data in the form of facts (or propositions) in the database. They are connected by inference rules or logical clauses. Since the system knows the meaning of the inference rule or the logical combination rules, it can detect an inconsistent state. This triggers a backtracking procedure focused on the data which contributed to the contradiction. A TMS provides one level of Boolean inference (see [56] for a more detailed discussion). A key feature of this strategy is the requirement that the system understand the inference procedure. This means that programming inside the TMS is limited to use of the system-provided inference mechanisms. Other types of reasoning must take place outside the system. A problem-solving system is then built on top of this to handle the decision-making and manipulate the TMS clauses. A TMS is a low-level substrate upon which the larger reasoning program is built. The function of the TMS is to guarantee a consistent state of the database. This type of architecture is shown

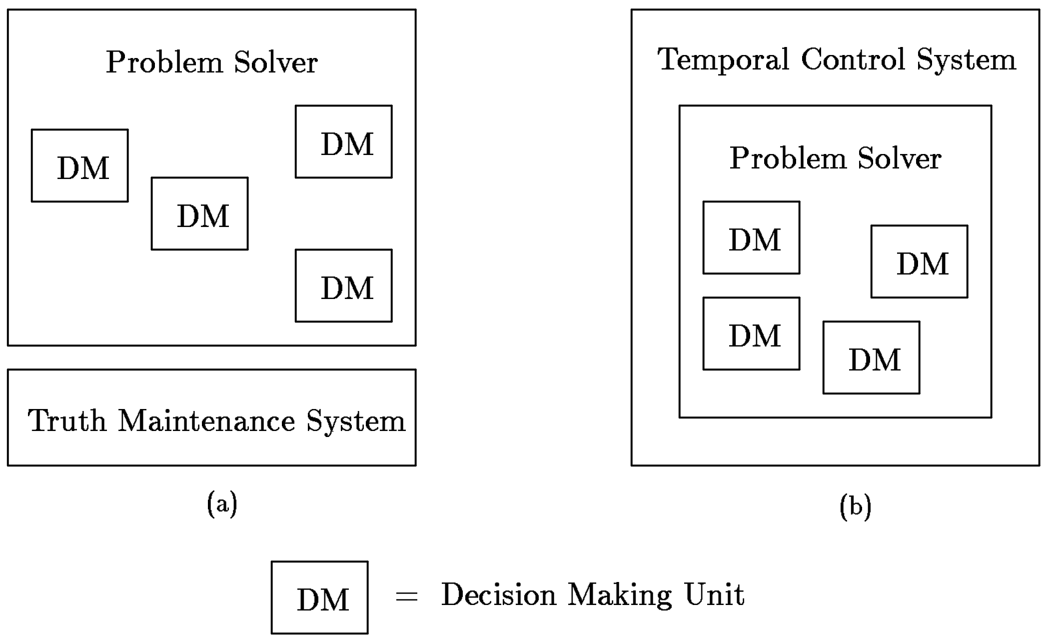


Figure 8.1: Architectural Differences Between TMS and TCS.

in figure 8.1a. The user program's decision units manipulate the TMS database. A system based on this paradigm is described by Dhar [22], in which he builds his domain-specific constraint satisfaction code on top of McAllester's TMS package. The TMS system is the substrate that is manipulated by a constraint-satisfaction problem-solver.

An alternate approach pursued by TCS is to allow arbitrary programmer-specified types of reasoning to connect the variable values in a system. This causes the system to lose the ability to analyze the reasoning units. To the system they appear as "black boxes" which implement some decision-making procedure. All of the inputs to the "black box" are known to the system. The only constraint is that the inference not depend on any variable value that is not explicitly identified as an input to the "box." Because the main reasoning units are contained inside the TCS system, this yields a qualitatively different architecture for a problem-solving system. (See figure 8.1b.) The decision making chores of the problem-solver are embedded in the TCS system. Rather than being a utility program that is manipulated by the problem solving application, the TCS provides an environment in which the application itself is run.

Because the TCS system is not able to analyze the reasoning functions, the system itself has no way to predict what the output of the inference mechanism will be. It can, however, execute the black-box procedure to calculate the new output values. Propagation of values continues until the system reaches a state of quiescence. A comparison using equality (the simplest data comparison test) is used to control propagation of information. The data are only propagated in a forward direction. By stopping the forward propagation when values don't change, a limit is imposed on the amount of processing needed to perform any update.

One major difference between the TCS and the TMS is in the degree of knowledge about the inference methods that are used. The TCS does not make any assumptions about the nature of the inference method that is used, and is therefore able to accommodate a wider range of methods. A TMS constrains the types of inference used to some specific method. Doyle's original TMS system [23] used the presence or absence of specific nodes to determine the validity or lack of validity of a given result. Other organizational principles also exist in Doyle's and de Kleer's [16] systems, but the basic framework and system architecture are the same.

The other major difference is that the TCS requires all of the variables and the entire dependency structure of the decision process to be known in advance, since the structure is compiled into the system via the declarations of dependencies. No new variables can be introduced into the system while it is running. A TMS has a more flexible structure than a TCS , because new nodes, new types of nodes, and new constraints can be added at any time. Since the type of inference is limited in a TMS, efficient algorithms for performing the updating can be implemented.

Although the standard TMS does not have special provisions for time-related reasoning, Dressler and Freitag [24] have produced a variant on an assumption-based TMS which propagates temporal labels as well as the traditional assumption sets. This provides a second indexing method for database query answering. The propagation method they use requires time to be a symbolic interval, so it is not able to handle metric information.

Time Map Manager Dean and McDermott [19] have constructed a temporal extension of a TMS system called a Time Map Manager. This augments a conventional TMS with special constructs for handling the temporal extent of propositions. Time maps are designed to assist a reasoning program doing planning with time constraints. It differs from the TCS in several ways:

1. Time maps use the predicate calculus as the basic knowledge representation. It is thus awkward to implement specific algorithms, since predicate calculus does not have the same rich supply of control constructs. Furthermore, the system relies upon the user to specify which propositions are contradictory.

2. Time maps allow inexact endpoints and multiple competing viewpoints. This makes the updating algorithms more complex and leads to the third difference:

3. States and persistences are handled asymmetrically. The beginning point may be fixed by the user, but the ending point must "float" in order to guarantee correctness of the algorithms used. This is not a fundamental requirement of the approach, but it is needed to allow the use of an efficient algorithm for

clipping the persistence of states. Time-limited persistence, such as that used with laboratory data in the Ketoacidosis Advisor, cannot be implemented.

4. Some of the updating functions are not automatic, and must be explicitly programmed by the user. Although this is cited as a benefit because it allows the user to control which inferences will be made as a result of changes to the database, it introduces the possibility of incompleteness in the reasoning, by not having all conclusions made, or inconsistency, by not having contradicted conclusions removed from the system.

8.3 Medical Systems

8.3.1 Non-AI Management Systems

Several computer systems have been designed for the outpatient management of diabetics. These systems use data from several days of observations to adjust the regular insulin dosage of the patients. The degree of modelling varies from the simple [21] to the more complicated [9]. There are other data analysis systems that seek to identify trends in the data and detect when changes in regimens have taken place [38].

The Ketoacidosis Advisor differs in having to assimilate more types of data in a more dynamic environment. In all the modeling systems, there is an underlying assumption that the basic lifestyle and meal pattern remained constant. (There are some models that allow one to vary the food intake while holding other parameters constant [74]) In DKA, however, the patients are not well-compensated insulin-taking diabetics, so the underlying premises of those systems are violated. Also, in most cases either the treatment is based on the analysis of more data than is available in the acute phase of DKA , or the solution covers only a part of the treatment regimen. Although I had to add other parts to the Ketoacidosis Advisor, the work on dosage adjustment proved to be a good starting point for my own subcutaneous insulin adjustment algorithms. Like [2, 77], I applied a computerized process to the implemetation of what were originally "paper" algorithms based on Skyler's work [81].

8.3.2 Ventilation Manager

The Ventilation Manager program VM [28] is the work most closely related to this thesis. VM is a rule-based expert system designed to monitor the progress of patients who are being weaned from respiratory support devices. This task requires the monitoring of the patient's physiological parameters, their evaluation, and a comparison with expected values generated by the use of a standard weaning protocol. The major effort in the design of the VM program went into the parameter monitoring and evaluation functions. It is in this part of the work that the temporal aspects of the domain exist.

The monitored parameters change over time, thus requiring the program to be able to interpret not just a single value for any result, but rather a sequence of values. In addition, there is the need to consider the history of the parameter values in making some decisions about whether a patient is doing well on a particular type of ventilation support, or whether they are ready for the next stage of weaning. These historical summaries are provided through the use of special purpose functions in the rule clauses. For example, it is possible to have a rule clause that matches on the basis of a stable parameter value in the last 20 minutes.

A further interesting feature of the VM data interpretation system is that the mapping from measured numeric values to symbolic interpretations is made context-dependent. This is done by setting up a mapping table that is initialized each time the context changes. Each context represents a different type of ventilation support. Thus, as a patient progresses from one type of support to another, the exact thresholds used to establish ideal and acceptable monitor results change. This capability is required by the temporal aspect of the domain, since the type of interpretation required in any one patient will change with time.

Relation of Ventilation Manager to TCS

The types of reasoning described above can be explained as a combination of certain types of more primitive reasoning activities using data that changes over time. In this domain, the interpretation of data (i.e., the mapping from continuous numeric values to discrete symbolic categories) is, in itself, an atemporal reasoning process. In the simplest case, there exists a function F which maps numbers to categories. This function considers only the value of the number in deciding to which category the parameter belongs.

Making this context-sensitive simply requires that rather than a single function, there exist a set of functions, indexed by contexts: $\{F_c : c \text{ is a context}\}$. The temporal aspect is then controlled entirely by controlling the temporal extent of the contexts. VM's establishment of context-sensitive evaluation functions for patient parameters was a major innovation. These were defined as tables of thresholds that defined symbolic ranges such as normal, low, very high, etc. These definitions were used whenever the type of ventilation method changed. Fagan refers to the tables as initialization rules which set up the context for data interpretation. This can be simply modeled in the TCS by having contexts be the values of interval variables (since each context has a duration). The function to be applied for data evaluation can then be selected based on the value of this interval variable's value. There is no need for the reasoning process itself to deal explicitly with the temporal aspects of the reasoning at all. This is an example of what TCS terms a context-sensitive transducer. The temporal dependency of the reasoning process is identical to that of a temporal variable. In this case the control system itself can handle the temporal dependency of the reasoning, simplifying the programming. The TCS approach to this type of definition involves the use of a context variable `ventilation-type` which

```
(defmodvar ventilation-change :point)          ; Reports change in ventilation
(defmodvar ventilation-type   :interval)       ; Holds the current ventilation type
(defmodvar vital-signs-raw    :point)          ; Measured vital signs
(defmodvar vital-signs-eval   :point)          ; Evaluated vital signs

(defpersistence ventilation-change ventilation-change ventilation-type)
   ; Just remember all changes in ventilation

(deftransducer evaluate-patient vital-signs-raw vital-signs-eval
               #'(lambda (raw type)
                   (evaluate-using-table raw (select-table-for type)))
               :context (ventilation-type))
```

Figure 8.2: TCS Implementation of VM Parameter Evaluation

is an abstraction of the point variable `ventilation-change`. The latter reports a change in the settings of the ventilator. Evaluation is done by using the appropriate table from a list. Sample code is shown in figure 8.2

The evaluation of historical data (e.g., for stability or trend detection, as is done in VM) can again be divided into two simpler processes, one atemporal and the other temporal. For example, consider the clause "heart beat stable for 20 minutes." This consists of the evaluation of some decision procedure ("heart beat stable") and a restriction on the temporal extent of this decision (20 minutes). The stability criterion can be modeled as a function which takes a list of input values and determines whether they fulfill the definition of stability[1], and another procedure that determines which values should be in the list of input values. This concept is implicitly temporal since it refers to the change (or rather the lack of change) in a dynamic variable.

The decomposition suggested above is used to isolate the temporal aspects of the reasoning. The explicit temporal reasoning consists of a memory function (see section 4.3.3) which constructs the list to be tested for the presence of the stability property. This memory has the fairly simple task of remembering all of the values of the heart beat parameter for the previous 20 minutes. This can again be modeled by an interval variable which, in any given interval, retains the values of the heart beat samples from the previous 20 minutes (it is assumed that the heart beat is provided as a series of discrete samples, rather than as a continuous function.) Given this memory, the stability function can be applied in each interval. The temporal dependence of the stability criterion is reflected only in its use of a list of values rather than a single value. The extent to which time enters into the picture is again handled implicitly by the control structure for the stability evaluation *per se*, and explicitly in the use of an auxiliary memory variable and associated reasoning machinery which maintain

[1]The exact stability definition is not important here, so long as there is an effective procedure available for determining whether or not the property holds. In this case, one could imagine stability meaning that the deviation from the mean heart beat over the given time period did not exceed some threshold value, say 5%.

the temporal aspects of the stability determination.

As these examples have shown, the TCS is capable of handling the temporal reasoning used in VM. It is further evident that there will be variety in the degree of time dependence of the reasoning. One benefit of the TCS approach is the identification of the degree of temporal dependence by allowing one to decompose the reasoning into individual units, some of which are, as we have seen, atemporal forms of reasoning. It is an advantage of the TCS that atemporal reasoning can be added without the need to deal with time. More important, atemporal reasoning can be embedded in a TCS in such a way that the atemporal reasoning description retains its simplicity, but becomes time-dependent because of the supporting framework. By providing some standard building blocks, the TCS facilitates the design of systems that interpret time-varying data.

The ability of the TCS to accommodate these types of reasoning in a natural manner will allow VM-style systems to gain the benefits of having a facility for updating data as well. The VM architecture is a strictly forward-chaining decision-making system without backtracking or belief revision. Data are assumed to arrive instantly and in chronological order. If these expectations are violated, one of two courses of action are open: either the data are treated as still reflecting the current state of the patient (i.e., they are treated as if they were current, new data), or they are rejected as being too old (i.e., no use is made of the information). In VM's domain this is not a problem because the program has the highly circumscribed task of monitoring the progress of a patient following a prescribed path through a series of mechanical ventilator settings. There is no need to explain what is happening with the patient in terms of disease processes. The only information relevant to the program's task is the current information and a restricted view of the history. The rule premises can refer to past data in a limited fashion. This access is implemented via certain special functions that provide summary information about the value of parameters over past time periods. What is missing is a mechanism that provides a framework for defining more functions that can assimilate a series of data points and arrive at some conclusion about their meaning (the data interpretation problem). Certain functions are provided by the system, but one is not able to add more without leaving the paradigm of the rule-based system. While this is certainly adequate in VM's domain, it is not general enough to serve as the basis for a more extensive temporal reasoning support system.

Because past information could have an impact on the interpretation of events and also on current therapy, any information that becomes available should be considered. The TCS can support this type of updating by propagating changed (or late-arriving) data along the dependency links in the reasoning structure so that the proper updates are made. In particular, this can be done for a VM-type system. This will allow proper utilization of all available data. It will also allow the correction of data later found to be in error.

8.3.3 Time Oriented Patient Analyzer

Michael Kahn implemented the Time Oriented Patient Analyzer (TOPAZ) [39], a system which uses multiple models to evaluate information and track a patient over time. The research domain was cancer therapy. Patients were seen at visits over a period of months. Information about the course of the patients was used to adjust the amount and type of drugs given to patients. This is an example of the patient-management problem, since it involves tracking the effectiveness of treatment over time.

Kahn uses three types of model in TOPAZ. The first is a multi-compartment physiological model similar to the pharmacokinetic model described earlier. It is extended by the ability to adapt its parameters to match the current patient. Kahn uses this model to interpret laboratory measurements and translate them into clinically useful concepts. He describes this as the data-interpretation part of the management problem.

Once the data has been interpreted, it is abstracted into clinically useful "states," using a second (non-mathematical) model. For this, an interval-based temporal model of time is used. The abstraction procedure is similar to the persistence abstractions described in this thesis.

Finally, there is a model of explanation that is used to generate patient summaries. It is used to structure the information in the temporal database and provide summaries to be read by people. It uses augmented transition networks and a straight chronological event order to translate the information about the patient's case.

The organization and strengths of TCS and TOPAZ reflect the differences in their domain features. TOPAZ was designed to operate in a world in which consultations were discrete events and there was no overlap between the data-gathering functions and the therapy decisions. Furthermore, the need to follow patients for a long period of time requires the use of more permanent data storage than TCS's variables. TCS, on the other hand, needs to be able to revise its decisions in response to data that arrives in the middle of a consultation. It is this need for efficient updating at unpredictable times that shaped the implementation of TCS. The major differences between TOPAZ's domain and the problems that are addressed in TCS are:

1. TOPAZ has a consultation structure in which all the information arrives in chronological order and is available at the next consultation. It is possible to correct past errors. Because there is no updating, however, only future invocations of the decision procedure will get the corrected information. TCS is designed to address the problems that occur when decisions are made in the middle of the data gathering process.

2. TOPAZ works with an external temporal database. The rules contain queries that retrieve information from the database. The temporal aspects of the data are handled by the rule predicates. Because the database is not active, changes in information going into it do not affect rules once they have been executed. All

future invocations will get the information. TCS is integrated with its database and takes an active role in scheduling the information that gets executed.

3. TOPAZ is consultation-driven. TCS is data-driven.

4. TOPAZ focused on the types of models that are needed to make medical decisions. TCS focuses on the temporal attributes of decision procedures. In this way the work is complementary. The model-based data interpretation could be embedded inside a TCS module, which would schedule the interpreter to run whenever new data became available. The difference in focus is that TCS highlights the dependence of information-processing on temporal data, whereas TOPAZ describes the conceptual tasks that must be accomplished in the decision-making.

Kahn's program uses three models of temporal reasoning. The first is a "process model" of the physiology that underlies observations. In his implementation this is a pharmacokinetic/pharmacodynamic model of bone-marrow effects of anti-cancer agents. The second is an "interval model" which abstracts the interpreted information from the process model. This provides a clinical context for the reasoning. The third model is an explanation model which can justify the results from the other two models.

Time is handled differently in each of the three models. In the process model, it is a continuous parameter. In the context model, intervals describing clinically relevant parts of the patient data are used. In the explanation model, the same database as in the context model is used. The conceptual view is that of a sequential model, which determines the order in which findings are discussed in the summary. The need to combine multiple models to handle a complex domain is a ratification of TCS's decision to allow maximum flexibility in specifying reasoning methods for heterogeneous systems.

8.4 Other Relevant Work

In this section I describe other work that is related to TCS. I discuss blackboard architectures for decision-making and special languages and hardware architectures for real-time expert systems.

8.4.1 Blackboard Systems

Blackboard systems [61, 62, 12] consist of a central database (the *blackboard*) that is shared by several independent reasoning units (*knowledge sources*). Each knowledge source (KS) is an independent unit which communicates with other units through messages placed in a commonly understood format on the blackboard. The independence of KS's allows the easy combination of heterogeneous reasoning methods. Since each KS is independent, there are fewer restrictions on the internal function calculated by an KS than is the case for TCS modules. On the other hand, the fact that each KS reads information of interest from the bulletin board and then posts its results means that there is no general method for retracting conclusions. Any KS that can retract conclusions must be specifically programmed to do so. Even then, there is no guarantee that another KS which used the retracted information will in turn revise its conclusions.

For example, HASP/SIAP uses a blackboard architecture to do sonar signal analysis. [63] The analysis takes place over time and the results of previous analyses are combined in calculating the current situation description. Since ships cannot arbitrarily appear and disappear, it is possible to use previous situation analyses as a source of guidance in identifying and classifying the current set of sonar signals. Two similar signals could be from the same source if they appear close enough together that the ship could have moved from one point to the other in the intervening time.

Unfortunately, since the system cannot backtrack, it cannot automatically retract earlier conclusions and have the effects propagated through the system. This is true of all blackboard systems. It may be easy to change the original data, but each knowledge source that relied on that information would have to be programmed to notice the change and deal with it appropriately.

It is also not clear to what extent the system can incorporate subsequent information. For example, it is possible to use *a priori* intelligence reports about the existence of enemy ships to identify signals, but apparently the reports must be available at the time the signal analysis is done. Such reports could therefore not be used to disambiguate an uncertain past identification.

In summary, the blackboard architecture allows more freedom in the implementation and integration of different types of reasoning than TCS, but it does not have the same level of support for the retraction and revision of data and conclusions.

8.4.2 Real-Time Systems

Real-time systems are characterized by the need to guarantee a response within a predefined period of time. If arbitrary calculations are permitted, this guarantee cannot be enforced. Real-time expert systems such as G2 [29, 60] can guarantee response time by limiting the language to rules which can be easily interpreted. By restricting information storage to statically allocated data areas, G2 can also avoid the need to garbage-collect. Given these constraints, it is not possible to have a general dependency-directed updating system, since the retraction of information can take arbitrarily long.

Researchers at Yale [26, 27] have explored the use of a parallel architecture called a *process trellis* for processing real-time monitor data. The process trellis uses a hierarchy of processors, each of which handles a small part of the interpretation. The connections between processors form a directed, acyclic graph. Processor outputs are coordinated by synchronization with a global clock. The synchronization simplifies the task of maintaining a coherent database, but it forces all processes to run at the speed of the slowest processor. For carefully selected, matched tasks in the signal-processing domain where the process trellis is used, these restrictions are acceptable.

The hindsight example presented earlier requires that previously-made decisions be retained in memory and then revised. The ability to go back in time and revise previous conclusions is incompatible with a real-time response deadline. The major difference between TCS and real-time expert systems is that TCS can reason at greater length about some of its problems. This ability makes it impossible for TCS to provide real-time response time guarantees. Since the updating model of TCS calls for completeness, real-time can be hard to achieve. TCS's ability to have reasoning loops, both through the use of circular data-dependency structures as well as the history-oracle mechanism, also prevents the guarantee of specific respnose times. In summary, real-time systems respond to current information, without spending a lot of time reasoning about the future, or analyzing past decisions. This allows them to fulfill their mission of providing a guaranteed time response, but does not allow the full sophistication of reasoning available in TCS.

There is potential for synergy between real-time systems and a TCS-based system. The process trellis, for example, could process the raw sensor data and provide inputs to a TCS system at an appropriate level of detail for further processing.

Chapter 9

Conclusion

Experience with the implementation of expert systems using TCS shows that it is flexible enough to support projects in disparate parts of the medical domain. The application of this methodology to the control of mathematical models, disease assessment, reasoning with expectation failures and the management of patient treatment over time shows breadth of coverage. On this basis, it is reasonable to assert that this approach can also be applied outside the area of medicine.

The contributions of this thesis consist of a programming system for constructing expert systems that use time dependent data as well as insight into the structure of reasoning over time. In the first section I discuss the practical applications that illustrate the usefulness of TCS as a programming tool. In the next section I discuss the conceptual contributions.

9.1 Practical Application

This thesis presents an engineering approach to dealing with time in AI reasoning. It is a language design that provides a basis on which to build temporally dependent applications programs. Some common types of temporal reasoning are supported via system utility routines, but there is no claim that the TCS will have a complete set of high-level reasoning routines. I do claim, however, that it will be possible to program such high-level routines inside the TCS framework.

Since this thesis project involves language design, it cannot be evaluated solely by a rigorous test. Certain aspects, such as the correctness of the change propagation algorithm, are amenable to such analysis. It cannot be rigorously proven that this system makes the development of temporal applications simpler by taking over the record-keeping functions that would otherwise need to be done by an applications programmer. The evaluation of the usefulness of the program is necessarily subjective. The argument in favor of using this language is that it provides the benefits of adding another layer to the programming system, making it a higher-level language for programming temporal processes. The projects that have been successfully implemented demonstrate that clinically useful behavior can be recreated using the system.

9.1.1 Projects Implemented

The practical utility of this approach has been demonstrated by the implementation of the following projects:

1. Demonstration Examples. These examples show system capabilities. This includes the percent program used to illustrate the data propagation method and the fever handler in the reference manual [70].

2. Reasoning by Hindsight. The use of hindsight shows a more sophisticated use of the history and oracle mechanism to solve a reasoning problem that relies on the observation of change over time and the inconsistencies between expected and observed behavior. This shows how TCS can be used to build a system that implements a difficult clinical reasoning task. I discussed this example at length in section 5.2.

3. Pharmacokinetic Modeling System. This example can model the changing pharmacokinetics of drugs and adjust model parameters in response to changes in relevant clinical data. An argument for the importance of using models in treatment programs can be found in the Oncocin project [80] and in TOPAZ [39]. The TCS provides a control environment in which the model can be executed. This demonstrates some of the control ideas and reasoning issues related to discrepancies between system recommendations and actual user actions.

4. Disease Assessment Program. Using an earlier version of TCS, Steve Novick developed an expert system for assessment of the underlying causes of ventricular arrhythmias as his Master's thesis project. [64] This demonstrates the use of TCS in a diagnostic tracking task.

5. Ketoacidosis. The major test project was the design of an advisor for acute acid-base and electrolyte balance disorders found in diabetic ketoacidosis. This collaborative work involved the Tufts-New England Medical Center, with two physicians, Michael Hagan and Klemens Meyer, serving as domain experts. In the formal evaluation reported in this thesis, the Ketoacidosis Advisor was shown to perform at a level indistiguishable from actual clinical care. This performance was in a domain in which the patient state changes over time and which requires an ability to track these changes.

9.2 Description of Temporal Reasoning

The division of temporal reasoning into a static and dynamic component creates a formalism that can be used to describe different aspects of reasoning over time.

9.2.1 Types of Temporal Reasoning

Several types of temporal reasoning have already been identified. Each of the classes below shares some characteristic that can be readily identified using the computational model of the TCS.

1. Atemporal rules and transducers. These are modules that use neither history nor oracle variables. They are static mappings from inputs to outputs, and are the simplest time-related reasoning methods. They can be handled automatically by the TCS.

2. Relationship between state change actions and states. Modules that embody this change make a transformation between the types of variables in the inputs and outputs. For example, going from points to intervals is a mapping from (potential) state change actions to states. Similarly, the process of producing points from intervals operates in the reverse direction. These modules generally require a limited use of history information. Oracles are typically not needed because of the unidirectional flow of time.

3. Forward temporal reasoning. Related to the previous point is the extension of data over time. This is a subcase of the above, but is sufficiently common to warrant separate discussion. In a changing domain such as medicine, data age and become less reliable. This limits the usefulness of the information, so that it is often convenient to limit the temporal extent of states induced from data samples. On the other hand, some planning problems do not have to worry about decay and can easily establish states that remain without change until acted upon by an outside action. By appropriately using the metric information from the timeline, either effect can be had. This includes the ability to make predictions of future events.

4. Backward temporal reasoning. This involves the use of future information to affect the past. Using hindsight is a complicated concept and is handled in its own section below.

5. History and forgetting. By establishing a model of how information propagates along the timeline (via the history variables) and providing a method for controlling the propagation (the function in a module), it is possible to develop applications that have differing degrees of dependence on the past or future. The facilities for using histories and oracles introduce a variable amount of time

dependence. It is possible to specify complete time dependence, in which case the output of a module depends on the entire time history of the entering data.

The full range of temporal dependencies can be supported. At one extreme are the rules and transducers that do not rely on information outside of their own execution interval. At the other extreme is a module with complete time dependence. A trivial example of such a module would be a module that calculated (internally) the average (over time) of all of the input values and had as its output each input value's percentage of that average. For each output datum, it is necessary to know something about each of the other values in order to compute the average. Furthermore, any change or addition to the data points would require the entire output over the complete timeline to be recomputed. This example is shown in the reference manual. [70, p. 30f.]

For efficiency reasons it is often desirable to put a temporal limit on the influence of any particular piece of data. Vere [85] demonstrates the practical utility of this concept in planning by using a temporal window to limit the time period that is examined by his reasoning system. Although the mechanisms differ, the underlying efficiency motive in both cases is supported by the fact that in most real systems there is a limited interaction between different world entities and processes. Furthermore, such interaction as occurs is often limited in scope, both physically and temporally. It is the temporal limitation that is exploited by establishing a cutoff on the length of time that a proposition can influence the decision making.

9.2.2 Explaining "Why I Changed My Mind"

One of the major goals of this system is to make it easy for new data to be incorporated into the reasoning framework. This means that the program should be able to "change its mind." By keeping records of the processes that were run, and the state of the data at that time, a history of the decision-making can be constructed. This history will provide the basis for an ability to explain why conclusions were changed. Unlike a standard updating system which can only give the current reasons for its beliefs, an historical record could even explain why an earlier (and since revised) opinion was held. For instance, the hindsight example from section 5.2 could then be explained by saying that on the first day, only the cuff pressure was available, and using the assumption that it was correct, a therapy to raise blood pressure was instituted. Later, when the assumptions was shown to be violated, a revision in the conclusion about the patient's fundamental problem was the impetus for changing the therapy to one which would reduce the blood pressure. It is important to remember that only conclusions can be revised, whereas past actions cannot be undone. Different database designs and the types of queries that they can answer are discussed by Snodgrass in [82].

The ability to maintain a history makes the system capable of providing explanations that involve changes in the belief structure about the world, by maintaining two separate temporal markers with each piece of reasoning: The world time to which the reasoning applies, and the real-time at which various bits of information that affect that reasoning process were available to the reasoner. The existence of such multiple views allows one to generate not only a picture of a program's belief about the changing values of variables in its world, but also allows one to capture a history of the changing beliefs of the program about the changing values of variables in its world. This is a latent capability in TCS that has not yet been exploited.

9.3 Future Work

The TCS functions as a programming system. There are a number of improvements that could be made to the implementation and some extensions to the way TCS is used.

9.3.1 Efficiency Improvements

The greatest efficiency loss in the present implementation involves the process queue. Process instances are spawned as soon as values change and then must wait in the queue until they can be executed. While waiting in the queue, the input values for the process instance could change. Since a process instance accesses its variable values at the start of execution, but has its duration determined when it is spawned, there could be a mismatch between interval values at execution time and the time bounds on the process instance. Since one of the guarantees that TCS makes is that interval variables have a single value, each time a module is queued, the entire queue must be checked to find processes for which this guarantee is violated. Such process instances must be rescheduled to take the new variable value times into account. In a large system with many items in the queue, the need to examine the entire queue every time a process is added contributes greatly to the running time of the program. The number of processes examined is proportional to the square of the length of the queue.

The most satisfactory solution would be to maintain a database of the intervals for which a module is scheduled but to delay the actual creation of process instances (and determination of their execution intervals) until the system is ready to run the process. This will require a complete overhaul of the process queueing code in the TCS. In the near term, processing time could be shortened by maintaining a separate queue for each module. Since the only process instances that can be affected by the new values are the ones for which a module is being queued, the queue scan need only examine the process instances from that module. The search space can be limited to the set of relevant process instances using the module (process type) as an index. Either of these solutions would also contribute to the distributed nature of the problem-solving model (see below). This redesign would be coupled with an efficient mechanism for determining which modules had pending process instances.

Other areas for performance improvement are the internal representation of the temporal database and the local execution strategy. The currently implemented representation uses a linked list of variable values ordered by time. They are stored in reverse chronologic order, based on the assumption that most of the changes occur at the "future" end of the timeline. Linked lists have the advantage that insertions and deletions can be easily performed. They have the disadvantage that locating a particular time involves a linear search. An alternate representation based on balanced trees should be investigated. The performance tradeoff between the searching time and the time to insert and delete items will need to be examined.

It is possible to use a more efficient execution order for modules like that used in the percent example in section 2.3.1. Since the TCS design treats reasoning modules as black boxes, it is not possible to determine the history and oracle behavior. One scheduling heuristic suggested by the percent module is to establish a direction of execution and move first from past to future and then back again. Since this can eliminate oscillations, the execution efficiency can be enhanced. Whether this speed improvement will be worth the increased overhead in queueing complexity requires further investigation.

9.3.2 Extension of Capabilities

The simplest extensions to TCS can be made within the framework that currently exists. By using the Common Lisp macro facility, certain general-purpose routines and abstractions can be provided. This is already the case for forms like the value persistence modules defined by `defpersistence`. This process can be extended to include those routines found useful in the Ketoacidosis Advisor, for example, the generation of regular time points for periodic actions (eight-hour fluid summaries) and the subdivision of the day into different dosage "windows" (the AM, PM and bedtime periods for insulin administration). This extensibility allows TCS to be customized for specific domain capabilities.

Other enhancements would require some changes to the implementation or the underlying computational model. A priority queueing mechanism (with fixed priorities) could be added with only minor changes. Executing process instances in priority order (depending on the module they implement) could enhance efficiency by allowing a programmer to specify the order of execution. This would allow tasks to be deferred until all of their predecessor processes had been executed. For modules that have multiple inputs, this could reduce the number of times they would need to be evaluated. For example, if a module with three inputs were executed after the first had changed, and then again as changes propagated to the second input, etc., it could be run more often than necessary. If it had a lower priority, then all of its inputs would have time to stabilize before it was invoked, yielding an increase in execution efficiency. A limited form of this prioritization could be performed following an analysis of the data dependency structure.[1]

A priority queue also has its uses in time critical applications. Because complete

[1] There would, of course, have to be provisions for loops.

updating is assured by the nature of the TCS design, it is not possible to guarantee real-time responses, because the system must run to completion. If priority queues were used, one could have all of the critical decisions run first, providing answers to the most important questions before other less critical tasks were done. In the ketoacidosis domain, this would mean handling low blood sugar (a very dangerous condition) before calculating the eight-hour fluid balance (a less important bookkeeping task). By displaying the results as they were produced, the important advice would be available first, with refinements to the advice and less important reasoning occuring later as time permitted.

A more ambitious extension would be to separate the determination of process instance scheduling intervals from the TCS internals. By making the scheduling program available as an option for individual modules, more flexibility in the determination of endpoints could be achieved. There would still be a need for TCS surveillance of the processes actually executed (to assure completeness of data propagation), but some of the scheduling decisions made centrally could be made specific to individual modules. For example, using the interval scheduling method, there are two options when an interval value changes. One is to schedule the minimal length interval that needs to be executed in order to handle the change. An alternate choice would be to schedule the largest interval with single values. Either choice will provide correct program results and complete updating. Depending on the domain, however, one choice may be more efficient than the other. By allowing this choice to be made on a module-by-module basis, the expert-system designer is given more control over the execution behavior of his program. This change would be consistent with the design philosophy of introducing the most flexible design space while preserving enough restrictions to maintain the fundamental guarantees of the TCS.

The final extension would be to embed a TCS-based advisor inside a program that could reason about the constraints on the temporal intervals. Since the TCS requires exact bounds, these could be provided by a constraint reasoner. As more constraints became available, it could change the exact bounds that were given to the embedded TCS, which would then propagate the effects of those changes on its internal reasoning. This division of labor would be a method of exploiting the strengths of both types of reasoning. It does run the risk of forcing the exploration of a combinatorial set of TCS models.

9.3.3 Distributed Model

The decomposition of the reasoning into modules with well-defined outputs provides a modular representation of parts of the program. The data dependencies describe the communications paths between individual parts of the entire system. Such a definition provides a specification for a distributed architecture for problem-solving. Since each module is complete and (except for its inputs) separate from all other elements of the system, one could create a distributed processing environment using the TCS process structure.

Outwardly, the dependency network of a TCS system resembles the process trellis architecture [26] developed at Yale. TCS provides a more general architecture because there is no need for synchronization between individual modules, and because loops are allowed in the data dependency structure. The existence of non-uniform communication paths and the lack of uniformity in the amount of time needed for individual modules to run are evidence for using a distributed architecture rather than a fine-grained parallel implementation.

9.3.4 Open Problems

There is a fundamental problem that any temporal tracking system must face. The amount of work that must be done in response to "clock ticks," (i.e., the advance of time without additional data becoming available) can be quite large. This is to a certain extent unavoidable. A module that projects the effects of following the advice in the future, given what has been done in the past, cannot avoid having to recompute its state as the clock advances. Since the premise of the projection is that the user will implement the advice right away, any time delay means that more of the other (actual) therapy was carried out. This changes the state from which the projection was made. There is no general answer to this problem, and the development of solutions to this difficulty remains an open problem.

There are also inherent limits to how cleverly a scheduling job can be done without knowledge of the workings of a module. Another open problem is whether general-purpose heuristics or methods can be developed that will enhance the optimality of a mechanical scheduler. An example of such a heuristic embedded in TCS is the desire to complete the execution of a given module's adjacent process instances before the process instances of modules further along the dependency path. One such heuristic would be to try to execute process instances in a single direction before moving information back in the opposite direction. Support for this can be found in consideration of the example of the percent program (section 2.3.1), where eight processes were used to calculate four values. The minimum needed to propagate the values would be seven—if all process instances were evaluated left to right and then right to left (the rightmost process instance would only need to be executed once).

9.4 Summary

In this work I have developed a computational model for describing reasoning over time. The key idea embodied in the model is that it is possible to decompose the task of reasoning over time into static and dynamic components. The use of static components simplifies the job of building expert systems by isolating the effects of temporal change, and in some cases hiding it entirely in the abstractions of the Temporal Control Structure.

Furthermore, the conceptual model provides a means for examining the temporal dependence of reasoning functions. This affects the efficiency of their execution and determines the extent to which they are exposed to information changing over time. Because both the temporal effects and the algorithms used in the implementation of the reasoning are explicit, knowledge engineers are free to choose appropriate methods for their domain. For design work, it is crucial that the person implementing a system have enough freedom to make whatever tradeoffs between accuracy and performance are suitable for the domain.

The computational model has a natural and efficient implementation that guarantees complete and carefully directed use of information. Except for conceptually needed distinctions between reasoning in the past or the future, a system programmed in the TCS is insensitive to the order of arrival of data. The bookkeeping chores needed to achieve this insensitivity are provided by TCS.

Finally, I have substantiated these claims through the implementation of several projects in real world domains. The Ketoacidosis Advisor clearly shows that human-level performance can be achieved in a medical domain. The characteristics of ketoacidosis required the system to track data which changed over time and which were not immediately available. The success of the program in the formal evaluation reported in this thesis validates the computational model used in its implementation.

Chapter 10

References

[1] K. G. M. M. Alberti and T. D. R. Hockaday, "Diabetic Coma: A Reappraisal after Five Years," *Clinics in Endocrinology and Metabolism*, 6(2):421–455, July 1977.

[2] A. M. Albisser, A. Schiffrin, et al., "Insulin Dosage Adjustment Using Manual Methods and Computer Algorithms: A Comparative Study," *Medical and Biological Engineering and Computing*, 24(6):577–584, 1986.

[3] James F. Allen, "Maintaining Knowledge About Temporal Intervals," *Communications of the ACM*, 26(11):832–843, November 1983.

[4] James F. Allen, "Towards a General Theory of Action and Time," *Artificial Intelligence*, 23:123–154, 1984.

[5] James F. Allen and Patrick J. Hayes, "A Common-Sense Theory of Time," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 528–531, 1985.

[6] James F. Allen and Johannes A. Koomen, "Planning Using a Temporal World Model," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 741–747, 1983.

[7] Ames Co., Davison Miles Laboratory, Inc., Elkhart, IA, *Modern Urine Chemistry: A Guid to the Diagnosis of Urinary Tract Diseases and Metabolic Disorders*, 1976.

[8] Eugene J. Barrett and Ralph A. DeFronzo, "Diabetic Ketoacidosis: Diagnosis and Treatment," *Hospital Practice*, pages 89–104, April 1984.

[9] M. P. Berger, R. A. Gelfand, and P. L. Miller, "Combining Statistical, Rule-based and Physiologic Model-based Methods to Assist in the Management of Diabetes Mellitus," *Computers in Biomedical Research*, 23(4):346–357, August 1990.

[10] Markus Berger and David Rodbard, "Computer Simulation of Plasma Insulin and Glucose Dynamics after Subcutaneous Insulin Injection," *Diabetes Care*, 12(10):725–736, November–December 1989.

[11] Christian Binder, Torsten Lauritzen, et al., "Insulin Pharmacokinetics," *Diabetes Care*, 7(2):188–199, March–April 1984.

[12] Blackboard Technology Group, Inc, "The Blackboard Problem-Solving Approach," *AI Review of Products, Services, and Research*, pages 27–32, July–August 1991, A publication of the American Association for Artificial Intelligence.

[13] Enrico W. Coiera, *Reasoning with Qualitative Disease Histories for Diagnostic Patient Monitoring,* PhD thesis, University of New South Wales, 1989.

[14] Enrico W. Coiera, "Monitoring Diseases with Empirical and Model-generated Histories," *Artificial Intelligence in Medicine*, 2:135–147, 1990.

[15] Mayer B. Davidson, "Diabetic Ketoacidosis and Hyperosmolar Nonketotic Coma," in *Diabetes Mellitus: Diagnosis and Treatment*, volume 2, chapter 9, pages 225–281, John Wiley & Sons, New York, second edition, 1986.

[16] Johan de Kleer, "An Assumption-based TMS," *Artificial Intelligence*, 28:127–162, 1986.

[17] Thomas Dean, *Time Map Maintenance,* Research Report 289, Yale University Department of Computer Science, 1983.

[18] Thomas L. Dean and Keiji Kanazawa, *A Model for Reasoning About Persistence and Causation,* Technical Report CS-89-04, Brown University, Department of Computer Science, February 1989.

[19] Thomas L. Dean and Drew V. McDermott, "Temporal Data Base Management," *Artificial Intelligence*, 32:1–55, 1987.

[20] Ralph A. DeFronzo, "Diabetic Ketoacidosis and Hyperosmolar Syndromes," notes on treating diabetic ketoacidosis, July 23, 1983.

[21] T. Deutsch, M. A. Boroujerdi, et al., "The Principles and Prototyping of a Knowledge-based Diabetes Management System," *Computer Methods and Programs in Biomedicine*, 29:75–88, 1989.

[22] Vasant Dhar and Casey Quayle, "An Approach to Dependency Directed Backtracking Using Domain Specific Knowledge," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 188–190, 1985.

[23] Jon Doyle, "A Truth Maintenance System," *Artificial Intelligence*, 12(2):231–272, 1979.

[24] Oskar Dressler and Hartmut Freitag, "Propagation of Temporally Indexed Values in Multiple Contexts," in *Proceedings of the Thirteenth German Workshop on Artificial Intelligence*, Informatik Fachberichte, pages 2–6, Berlin, 1989, Springer Verlag.

[25] Daniel Dvorak and Benjamin Kuipers, "Model-based Monitoring of Dynamic Systems," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1238–1243, 1989.

[26] Michael Factor and David H. Gelernter, "The Process Trellis: A Software Architecture for Intelligent Monitors," in *Tools for Artificial Intelligence: TAI 89"*, pages 174–181, IEEE Computer Society Press, October 1989.

[27] Michael Factor, Dean F. Sittig, et al., "A Parallel Software Architecture for Building Intelligent Medical Monitors," in *Symposium on Computer Applications in Medical Care*, pages 11–16, 1989.

[28] Lawrence Marvin Fagan, VM*: Representing Time-Dependent Relations in a Medical Setting,* PhD thesis, Stanford University, June 1980.

[29] Gensym Corporation, Cambridge, MA, USA, *G2 Reference Manual*, August 1990, For G2 Version 2.0.

[30] John W. Graef, editor, *Manual of Pediatric Therapeutics,* Little, Brown and Company, fourth edition, 1988.

[31] Joseph Y. Halpern and Yoav Shoham, "A Propositional Modal Logic of Time Intervals," in *Proceedings of the Symposium on Logic in Computer Science*, pages 279–292, IEEE, 1986.

[32] Walter Hamscher, "Temporally Coarse Representation of Behavior for Model-based Troubleshooting of Digital Circuits," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 887–893, 1989.

[33] Steve Hanks and Drew McDermott, "Default Reasoning, Nonmonotonic Logics, and the Frame Problem," in *Proceedings of the National Conference on Artificial Intelligence*, pages 328–333, American Association for Artificial Intelligence, 1986.

[34] Michael C. Higgins, Don Goodnature, et al., "An Architecture for Combining Alternative Reasoning Strategies in Real-time Patient Monitoring," in *AI in Medicine: Working Notes*, 1990 Spring Symposium Series, pages 74–76, American Association for Artificial Intelligence, March 27–29, 1990.

[35] T. D. R. Hockaday and K. G. M. M. Alberti, "Diabetic Coma," *Clinics in Endocrinology and Metabolism*, 1(3):751–788, November 1972.

[36] R. W. Jeliffe, D. Z. D'Argenio, et al., "A Time-Shared Computer Program for Adaptive Control of Lidocaine Therapy Using an Optimal Strategy for Obtaining Serum Concentrations," in *Symposium on Computer Applications in Medical Care*, pages 975–981, 1980.

[37] Roger W. Jelliffe, "Clinical Applications of Pharmacokinetics and Adaptive Control," *IEEE Transactions on Biomedical Engineering*, BME-34(8):624–632, August 1987.

[38] Michael G. Kahn, Charlene A. Abrams, et al., "Automated Interpretation of Diabetes Patient Data: Detecting Temporal Changes in Insulin Therapy," in *Symposium on Computer Applications in Medical Care*, pages 569–573, 1990.

[39] Michael Gary Kahn, *Model-Based Interpretation of Time-Ordered Medical Data,* PhD thesis, University of California, San Francisco, December 1988.

[40] E. Yu. Kandrashina, "Representation of Temporal Knowledge," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 346–348, 1983.

[41] J. P. Kassirer, D. E. Hricik, and J. J. Cohen, *Repairing Body Fluids: Principles and Practice,* W. B. Saunders, 1989.

[42] Saulo Klahr, editor, *The Kidney and Body Fluids in Health and Disease,* Plenum Medical Book Company, New York, 1984.

[43] Michael D. Klein, "Development of a Ventricular Arrhythmia Management Advisor," National Institutes of Health Research Grant Application, June 1983.

[44] Isaac S. Kohane, "Temporal Reasoning in Medical Expert Systems," in R. Salamon, B. Blum, and M. Jørgensen, editors, *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, pages 170–174, Washington, October 1986, North-Holland.

[45] Isaac S. Kohane, *Temporal Reasoning in Medical Expert Systems,* TR 389, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, April 1987.

[46] George P. Kozak and Arturo R. Rolla, "Diabetic Comas," in George P. Kozak, editor, *Clinical Diabetes Mellitus*, chapter 10, pages 109–145, W.B. Saunders Co., Philadelphia, 1982.

[47] E. W. Kraegen and D. J. Chisholm, "Insulin Responses to Varying Profiles of Subcutaneous Insulin Infusion: Kinetic Modelling Studies," *Diabetologia*, 26:208–213, 1984.

[48] Robert A. Kreisberg, "Diabetic Ketoacidosis, Alcoholic Ketosis, Lactic Acidosis, and Hyporeninemic Hypoaldosteronism," in Max Ellenberg and Harold Rifkin, editors, *Diabetes Mellitus: Theory and Practice*, chapter 30, pages 621–653, Medical Examination Publishing Co., Inc., third edition, 1983.

[49] Peter Ladkin, "Primitives and Units for Time Specification," in *Proceedings of the National Conference on Artificial Intelligence*, pages 354–359, American Association for Artificial Intelligence, 1986.

[50] Peter Ladkin, "Time Representation: A Taxonomy of Interval Relations," in *Proceedings of the National Conference on Artificial Intelligence*, pages 360–366, American Association for Artificial Intelligence, 1986.

[51] Vladimir Lifschitz, "Pointwise Circumscription: Preliminary Report," in *Proceedings of the National Conference on Artificial Intelligence*, volume 1, pages 406–410, 1986.

[52] Vladimir Lifschitz, "Circumscriptive Theories: A Logic-Based Framework for Knowledge Representation (Preliminary Report)," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 364–368, American Association for Artificial Intelligence, 1987.

[53] William J. Long and Thomas A. Russ, "A Control Structure for Time Dependent Reasoning," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 230–232, 1983.

[54] William J. Long, Thomas A. Russ, and W. Buck Locke, "Reasoning from Multiple Information Sources in Arrhythmia Management," in *Proceedings of the Conference on Frontiers of Engineering in Health Care*, pages 640–643, IEEE, 1983.

[55] J. Malik and T. Binford, "Reasoning in Time and Space," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 343–345, August 1983.

[56] David A. McAllester, "Truth Maintenance," in *Proceedings of the National Conference on Artificial Intelligence*, pages 1109–1116, American Association for Artificial Intelligence, 1990.

[57] David Allen McAllester, *A Three Valued Truth Maintenance System,* AIM 473, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA, 02139, 1978.

[58] John McCarthy, "Circumscription — A Form of Non-Monotonic Reasoning," *Artificial Intelligence*, 13(1):27–38, 1980.

[59] Sanjay Mittal, "Event-based Organization of Temporal Databases," in *Fourth National Conference of the Canadian Society for Computational Studies of Intelligence*, pages 1–8, 1982.

[60] Robert Moore, Lowell Hawkinson, et al., "The G2 Real-Time Expert System," in *Proceedings of the ISA/88 International Conference and Exhibit*, pages 1625–1633, Instrument Society of America, 1988.

[61] H. Penny Nii, "Blackboard Systems, Part 1: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *AI Magazine*, 7(2):38–53, Summer 1986.

[62] H. Penny Nii, "Blackboard Systems, Part 2: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective," *AI Magazine*, 7(3):82–106, August 1986.

[63] H. Penny Nii, Edward A. Feigenbaum, et al., "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *AI Magazine*, 3(2):23–35, 1982.

[64] Steven L. Novick, "Reasoning Over Time About the Causes of Arrhythmias," Master's thesis, Massachusetts Institute of Technology, 1987.

[65] Klaus K. Obermeier, "Temporal Inferences in Medical Texts," in *23rd Annual Meeting of the ACL*, pages 9–17, Association of Computational Linguistics, 1985.

[66] Mathew J. Orland and Robert J. Saltman, editors, *Manual of Medical Therapeutics,* Little, Brown and Company, 25th edition, 1986.

[67] Burton David Rose, *Clinical Physiology of Acid-Base and Electrolyte Disorders,* McGraw-Hill Book Company, New York, second edition, 1984.

[68] Thomas A. Russ, "A Knowlege-Based Approach to Ventricular Arrhythmia Management," in *Proceedings of the International Conference on Cybernetics and Society*, pages 10–14, IEEE, 1982.

[69] Thomas A. Russ, "A System for Using Time Dependent Data in Patient Management," in R. Salamon, B. Blum, and M. Jørgensen, editors, *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, pages 165–169, Washington, October 1986, North-Holland.

[70] Thomas A. Russ, *Temporal Control Structure Reference Manual,* TM 331, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, June 1987.

[71] Thomas A. Russ, "Using Hindsight in Medical Decision Making," *Computer Methods and Programs in Biomedicine*, 32(1):81–90, May 1990, Also published in *Proceedings of the Symposium on Computer Applications in Medical Care*, pp. 38–44, 1989.

[72] Thomas A. Russ, "Coordinating Advice and Actual Treatment," in Martin D. Fox, Mary Anne F. Epstein, et al., editors, *Proceedings of the 1991 IEEE Seventeenth Annual Northeast Bioengineering Conference*, pages 261–262, IEEE, April 1991.

[73] Thomas Anton Russ, "Ventricular Arrhythmia Management: A Knowledge-Based Approach," Master's thesis, Massachusetts Institute of Technology, 1983.

[74] A. Rutscher, E. Salzsieder, et al., "KADIS—A Computer-aided Decision Support System for Improving the Management of Type-I Diabetes," *Experimental and Clinical Endocrinology*, 95(1):137–147, February 1990.

[75] R. Salamon, B. Blum, and M. Jørgensen, editors, *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, Washington, October 1986, North-Holland.

[76] David S. Schade and R. Philip Eaton, "Diabetic Ketoacidosis — Pathogenesis, Prevention and Treatment," in *Clinics in Endocrinology and Metabolism*, volume 12, chapter 4, pages 321–338, W.B. Saunders Company Ltd, July 1983.

[77] Alicia Schiffrin, Marko Mihic, et al., "Computer-assisted Insulin Dosage Adjustment," *Diabetes Care*, 8(6):545–552, November–December 1985.

[78] Robert S. Sherwin, Karl J. Kramer, et al., "A Model of the Kinetics of Insulin in Man," *The Journal of Clinical Investigation*, 53:1481–1492, May 1974.

[79] Yoav Shoham, "Chronological Ignorance: Time, Nonmonotonicity, Necessity and Causal Theories," in *Proceedings of the National Conference on Artificial Intelligence*, volume 1, pages 389–393, 1986.

[80] Edward H. Shortliffe, A. Carlisle Scott, et al., "ONCOCIN: An Expert System for Oncology Protocol Management," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 876–881, 1981.

[81] Jay S. Skyler, Denise L. Skyler, et al., "Algorithms for Adjustment of Insulin Dosage by Patients Who Monitor Blood Glucose," *Diabetes Care*, 4(2):311–318, March–April 1981.

[82] Richard Snodgrass and Ilsoo Ahn, "Temporal Databases," *Computer*, 19(9):35–42, September 1986.

[83] Richard M. Stallman and Gerald J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, 9(2):135–196, 1977.

[84] Pate D. Thomson, Kenneth L. Melmon, et al., "Lidocaine Pharmacokinetics in Advanced Heart Failure, Liver Disease, and Renal Failure in Humans," *Annals of Internal Medicine*, 78:499–508, 1973.

[85] Steven Vere, "Temporal Scope of Assertions and Window Cutoff," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1055–1059, 1985.

[86] Brian C. Williams, *Circumscribing Circumscription: A Guide to Relevance and Incompleteness,* AIM 868, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA, 02139, October 1985.

[87] Włodek W. Zadrożny, "A Theory of Default Reasoning," in *Proceedings of the National Conference on Artificial Intelligence*, pages 385–390, American Association for Artificial Intelligence, 1987.

Appendix A

Cases Used for Evaluation

The cases used for the program evaluation are presented here. For each case the data is arranged chronologically in three tables: vital signs, oral fluids and urine output; blood chemistries and blood gas measurements; and the actual treatment given. This is followed by a table identifying the decision points and presenting the competing treatment plans. As in the evaluation results, an asterisk (*) identifies those decisions which were eliminated from the analysis because of data coding errors.

A.1 Case Keto 11

Keto 11, a 15yo male: This is one of several admissions for this patient with a 2 year history of IDDM. Since diagnosis, patient has been hospitalized 7–8 times for control of his diabetes. Patient was in his usual state of health until 1 week prior to admission when he developed upper respiratory tract infection syndrome with cough and runny nose. "He felt warm" all week but he never took his temperature. Aspirin offered no relief, and Contac simply made him drowsy. On the evening prior to admission he ate a late dinner, but had not eaten all day. The morning of admission he took his insulin, went to school without eating breakfast and started to feel extremely tired and nauseated with dyspnea on his way to school. He vomited once and returned home to fall asleep. Upon arrival at the hospital for his usual appointment an acetone-like odor was noted on his breath and he was referred to the Pediatric Walk In Clinic.
HPI: INFECTIONx3 days, NAUSEAx1 day, VOMITINGx1, INSULIN-TAKEN
PMH: IDDMx2 years, DKAx8

Baseline Insulin: 50U Lente SQ qam

	Vital Signs						Urine		
Time	Temp	HR	BP	RR	Weight	PO-In	Vol	Ket	Gluc
10/6 11:00a	34.8	96	110/60	24	49.0kg				
1:00p	35.8	88		18		240	1500	++++	++++
1:30p								+++	++++
3:00p	36.8	80	122/80	28	49.8kg				
10:30p							1100	++++	++++
10/7 12:00a						480			
1:00a	36.5	86	120/80	20					
8:00a	37.0	90		20					
11:00a	36.8	86	120/80	24	49.0kg				
11:30a							150		
3:00p							1950		
4:50p	36.9	84	122/84	24					
8:45p	37.1	84	124/90	20					
11:30p							950	++	++++

	Laboratory (Serum)									Blood Gas				
Time	Na	K	Cl	CO_2	Cr	BUN	Phos	Ket	Gluc	pH	pO_2	pCO_2	HCO_3	FIO_2
10/6 11:30a									240-400					
11:35a	140	4.0	104	12	0.7	12	3.9	++	437	V 6.96	26	36	8	RA
12:55p	136	4.4	106	9					468	V 6.94	39	32	7	RA
1:00p									240-400					
2:35p									240-400					
2:50p	138	4.4	112	8			2.7		366	A 7.05	148	16	4	
4:30p	136	4.3	111	11	0.5	9	2.7		347					
4:45p									180-240					
6:10p									120-180					
7:30p									120-180					
9:00p									120-180					
11:00p									120-180					
10/7 1:00a									120-180					
3:00a									120-180					
5:00a									80-120					
7:00a	137	3.4	112	18	0.7	10			194	V 7.26	55	38	17	
11:00a									180-240					
12:30p									180-240					
2:00p									240-400					
3:00p									180-240					
4:45p									180-240					
7:00p									120-180					
10:30p									240-400					
11:30p									240-400					

	IV Fluid Therapy							Insulin Therapy		
Time	Type	Rate	Addition	Type	Rate	Add.	Total	IV-Drip	IV-Bolus	SQ
10/6 11:00a										
11:30a	NS	857					0	0		
12:40p	↓	194					1000	0		
1:30p	↓	140	20KCl, 2.5KPhos				1162	0		
2:00p	↓	180	↓				1232	5U/hr		
3:00p	↓	400	40KCl, 5KPhos				1412	10U/hr		
4:00p	↓	118	↓				1812	5U/hr		
8:00p	↓	133	↓				2284	2U/hr		
9:00p	↓	123	↓				2417	↓		
10/7 11:30a	↓	110	↓				4201	↓		50U Lente
1:30p	↓	↓	↓				4421	0		
1:45p	—	0	—				4449	0		
11:30p	—	0	—				4449	0		

# Time	**Real**	**Advice**
1. 10/6 11:30a	NS at 1500ml/hr No IV insulin drip	NS at 80ml/hr No IV insulin drip 5U Reg insulin SQ
2. 10/6 12:40p	NS at 180ml/hr No IV insulin drip	$\frac{1}{2}$NS w/ 20mEq KCl/ℓ at 125ml/hr Insulin at 5U/hr 5U Reg insulin IV bolus
3. 10/6 1:30p	NS w/ 20mEq KCl, 2.5ml KPhos/ℓ at 180ml/hr No IV insulin drip	$\frac{1}{2}$NS w/ 20mEq KCl, 1amp Bicarb/ℓ at 600ml/hr Insulin at 5U/hr 5U Reg insulin IV bolus
4. 10/6 2:00p	NS w/ 20mEq KCl, 2.5ml KPhos/ℓ at 180ml/hr Insulin at 5U/hr	NS w/ 20mEq KCl, 1amp Bicarb/ℓ at 600ml/hr Insulin at 5U/hr 5U Reg insulin IV bolus
5. 10/6 3:00p	NS w/ 40mEq KCl, 5ml KPhos/ℓ at 180ml/hr Insulin at 5U/hr	NS w/ 20mEq KCl, 1amp Bicarb/ℓ at 600ml/hr Insulin at 2U/hr
6. 10/6 4:00p	NS w/ 40mEq KCl, 5ml KPhos/ℓ at 130ml/hr Insulin at 5U/hr	NS w/ 20mEq KCl, 1amp Bicarb/ℓ at 500ml/hr Insulin at 5U/hr
7. 10/6 8:00p	NS w/ 40mEq KCl, 5ml KPhos/ℓ at 130ml/hr No IV insulin drip	D5-NS w/ 20mEq KCl/ℓ at 250ml/hr Insulin at 5U/hr
8. 10/6 9:00p	NS w/ 40mEq KCl, 5ml KPhos/ℓ at 130ml/hr Insulin at 2U/hr	D5-NS w/ 20mEq KCl/ℓ at 125ml/hr Insulin at 2U/hr
9. 10/7 11:30a	NS w/ 40mEq KCl, 5ml KPhos/ℓ at 130ml/hr Insulin at 2U/hr 50U Lente insulin SQ	No IV fluids Insulin at 1U/hr 40mEq KCl
10. 10/7 1:30p	NS w/ 40mEq KCl, 5ml KPhos/ℓ at 130ml/hr No IV insulin drip	No IV fluids Insulin at 1U/hr 40mEq KCl
11. 10/7 1:45p	No IV fluids No IV insulin drip	No IV fluids No IV insulin drip 40mEq KCl
12. 10/7 11:30p	No IV fluids No IV insulin drip 6U Reg insulin SQ	No IV fluids No IV insulin drip 5U Reg insulin SQ 40mEq KCl

A.2 Case Keto 13

Keto 13, a 21yo male: Patient has had IDDMx8yrs. Approximately 24hrs before admission he was at a party and drank heavily such that he had a prolonged episode of nausea and vomiting. He reported to ER feeling dehydrated, with complaints of left flank pain. Patient presents afebrile and in moderate distress. Mucous membranes dry, lungs clear. Patient complains of polydipsia, denies polyuria. Patient denies infection of any sort, no diarrhea, cough, rhinitis, etc.
HPI: VOMITINGxMULTIPLE, NAUSEAx1 day, POLYDIPSIAx1 day, ETOH, INSULIN-TAKEN
PMH: IDDMx9 years, DKAx1

Baseline Insulin: 40U LENTE SQ qam

	Vital Signs						**Urine**		
Time	Temp	HR	BP	RR	Weight	PO-In	Vol	Ket	Gluc
6/23 11:35a	37.4	124	120/76	22					
11:45a	37.4	120	150/80	22					
2:55p		104	130/86	22			680		
3:00p								+++	++++
6:45p	37.4	92	140/90	22	66.2kg				
9:00p		96	150/90	20					
6/24 12:00a	36.5	89	110/66	16					

6/23 11:35a Pulse and BP: → 120 160/90, ↑ 124 120/76

	Laboratory (Serum)									**Blood Gas**				
Time	Na	K	Cl	CO_2	Cr	BUN	Phos	Ket	Gluc	pH	pO_2	pCO_2	HCO_3	FIO_2
6/23 11:35a	140	6.0	98	7	1.2	26	8.2	+++	572					
1:05p										A 7.12	138	19	6	RA
2:30p	142	5.0	108	12	1.1	22		++	342					
5:40p	138	4.7	108	17	1.0	20		++	247					
6:45p									240-400					
9:00p	138	4.9	108	21			1.9		165					
6/24 12:00a									40-80					

	IV Fluid Therapy							**Insulin Therapy**		
Time	Type	Rate	Addition	Type	Rate	Add.	Total	IV-Drip	IV-Bolus	SQ
6/23 11:45a	NS	2667					0	0		
12:30p	↓	500					2000	0	10U Reg	
1:00p	↓	↓					2250	3U/hr		
1:30p	↓	95					2500	6U/hr		
6:45p	↓	185					2999	↓		
7:30p	↓	186					3138	3U/hr		
10:20p	↓	200					3665	2U/hr		15U Reg
11:30p	↓	↓					3898	0		
6/24 12:00a	↓	143	30KCl				3998	0		

# Time	Real	Advice
1. 6/23 11:45a	NS at 1500ml/hr No IV insulin drip	NS at 1000ml/hr No IV insulin drip
* 6/23 12:30p	NS at 250ml/hr No IV insulin drip 10U Reg insulin IV bolus	NS at 350ml/hr No IV insulin drip
2. 6/23 1:00p	NS at 250ml/hr Insulin at 3U/hr	$\frac{1}{2}$NS w/ 1amp Bicarb/ℓ at 350ml/hr Insulin at 3U/hr
3. 6/23 1:30p	NS at 250ml/hr Insulin at 7U/hr	$\frac{1}{2}$NS w/ 1amp Bicarb/ℓ at 250ml/hr Insulin at 3U/hr
4. 6/23 6:45p	NS at 200ml/hr Insulin at 7U/hr	NS w/ 20mEq KCl/ℓ at 500ml/hr Insulin at 7U/hr
5. 6/23 7:30p	NS at 200ml/hr Insulin at 3U/hr	NS w/ 20mEq KCl/ℓ at 500ml/hr Insulin at 6U/hr
6. 6/23 10:20p	NS at 200ml/hr Insulin at 3U/hr 15U Reg insulin SQ	NS w/ 5ml KPhos/ℓ at 250ml/hr No IV insulin drip
7. 6/23 11:30p	NS at 200ml/hr No IV insulin drip	NS w/ 5ml KPhos/ℓ at 250ml/hr No IV insulin drip
8. 6/24 12:00a	NS w/ 30mEq KCl/ℓ at 200ml/hr No IV insulin drip	NS w/ 5ml KPhos/ℓ at 250ml/hr No IV insulin drip

A.3 Case Keto 15

Keto 15, a 25yo female: Patient with c/o nausea and vomiting since 5am (has vomited 4–5x). Also c/o pleuritic mid-sternal CP x 2 days. Pain not related to exertion and does not change with position. Patient was seen yesterday with negative CXR and EKG. No cough or SOB. No fever or chills. Recent UGI was negative. Has not been feeling well for about 4 weeks. Concerned about younger sister w/ cancer and an upcoming move to California. Has taken her insulin as usual (last dose last night). No dysuria.
HPI: NAUSEAx1 day, DIARRHEAx1 day, VOMITINGx6
PMH: IDDMx7 years, DKAxMULTIPLE

Baseline Insulin: 7U Reg, 35U Lente SQ qam

	Vital Signs						**Urine**		
Time	Temp	HR	BP	RR	Weight	PO-In	Vol	Ket	Gluc
6/3 7:23a		120	116/74						
7:30a		96	118/70	28					
8:30a		108	112/70	22					
10:30a		98	116/70	20					
11:15a	37.1	102		20					
1:00p	37.0	90	114/60	20	56.1kg	250			
2:00p						250			
4:00p	37.0	95	108/64	20			500	++++	+++
6:30p		90	110/60	18					
7:00p						180			
8:00p	37.4	101	108/64	18					
9:00p						120			
10:00p		89	104/68	16		120	400		
6/4 12:00a	36.7	72	104/66	16					
1:00a						180	600		
2:00a		76	114/76	16					
4:00a	36.8	74	110/70	16					
5:00a						180	575		
6:00a		79	120/88	18					
8:00a	36.2	83	102/64	18					
9:00a						180	800	0	
10:00a		88	102/66	16		180			
12:00p		98	106/64	16		180	700	0	

6/3 7:23a Pulse and BP: $\rightarrow$ 96 118/70, $\uparrow$ 120 116/74

	Laboratory (Serum)									**Blood Gas**				
Time	Na	K	Cl	CO_2	Cr	BUN	Phos	Ket	Gluc	pH	pO_2	pCO_2	HCO_3	FIO_2
6/3 7:35a	138	5.0	107	11	0.8	15	3.5		562					
7:57a										A 7.15	128	21	7	RA
10:13a	146	4.5	118	14	0.7	14			261					
2:45p	134	4.3	112	15					280					
3:00p									314					
4:00p									216					
5:00p									164					
7:00p	137	4.0	112	18					129					
8:00p									195					
9:00p									194					
10:00p									120-180					
11:00p									166					
11:40p	136	4.2	111	21					161					
6/4 1:35a									86					
4:00a	138	4.2	112	22	0.8	8	2.3	0	123					
8:00a									142					
8:40a	138	4.3	113	18	0.7	5	1.8	0	149					
10:00a									93					
12:00p									93					

	IV Fluid Therapy							Insulin Therapy		
Time	Type	Rate	Addition	Type	Rate	Add.	Total	IV-Drip	IV-Bolus	SQ
6/3 7:23a										
7:30a	NS	750					0	0		
8:30a	↓	↓					750	5U/hr	10U Reg	
8:50a	↓	414					1000	↓		
9:15a	↓	↓					1172	10U/hr		
9:20a	↓	↓					1206	4U/hr		
11:15a	D5-NS	133					2000	↓		
3:00p	D5-$\frac{1}{2}$NS	200	40KCl, 3KPhos				2499	↓		
8:00p	↓	169	10KPhos				3499	↓		
6/4 1:35a	↓	196	↓				4443	2U/hr		
8:30a	↓	200	↓				5799	↓		7U Reg, 35U NPH
10:30a	↓	↓	↓				6199	0		
12:00p	↓	↓	↓				6499	0		

# Time	Real	Advice
1. 6/3 7:30a	NS at 1000ml/hr No IV insulin drip	NS at 1000ml/hr No IV insulin drip
* 6/3 8:30a	NS at 1000ml/hr Insulin at 5U/hr 10U Reg insulin IV bolus	NS w/ 1amp Bicarb/ℓ at 250ml/hr No IV insulin drip
2. 6/3 8:50a	NS at 500ml/hr Insulin at 5U/hr	$\frac{1}{2}$NS w/ 1amp Bicarb/ℓ at 250ml/hr Insulin at 5U/hr
3. 6/3 9:15a	NS at 500ml/hr Insulin at 10U/hr	$\frac{1}{2}$NS w/ 1amp Bicarb/ℓ at 150ml/hr Insulin at 5U/hr
4. 6/3 9:20a	NS at 500ml/hr Insulin at 5U/hr	$\frac{1}{2}$NS w/ 1amp Bicarb/ℓ at 150ml/hr Insulin at 5U/hr
5. 6/3 11:15a	D5-NS at 200ml/hr Insulin at 5U/hr	$\frac{1}{2}$NS w/ 20mEq KCl/ℓ at 150ml/hr Insulin at 3U/hr
6. 6/3 3:00p	D5-$\frac{1}{2}$NS w/ 40mEq KCl, 3ml KPhos/ℓ at 200ml/hr Insulin at 4U/hr	$\frac{1}{2}$NS w/ 20mEq KCl/ℓ at 90ml/hr Insulin at 4U/hr
7. 6/3 8:00p	D5-$\frac{1}{2}$NS w/ 10ml KPhos/ℓ at 200ml/hr Insulin at 4U/hr	NS w/ 20mEq KCl/ℓ at 90ml/hr Insulin at 1U/hr
8. 6/4 1:35a	D5-$\frac{1}{2}$NS w/ 10ml KPhos/ℓ at 200ml/hr Insulin at 2U/hr	NS w/ 20mEq KCl/ℓ at 175ml/hr Insulin at 1U/hr
9. 6/4 8:30a	D5-$\frac{1}{2}$NS w/ 10ml KPhos/ℓ at 200ml/hr Insulin at 2U/hr 7U Reg, 35U NPH insulin SQ	No IV fluids No IV insulin drip 7U Reg, 35U Lente insulin SQ
10. 6/4 10:30a	D5-$\frac{1}{2}$NS w/ 10ml KPhos/ℓ at 200ml/hr No IV insulin drip	D5-NS w/ 5ml KPhos/ℓ at 200ml/hr Insulin at 2U/hr
11. 6/4 12:30p	D5-$\frac{1}{2}$NS w/ 10ml KPhos/ℓ at 200ml/hr D10 at 75ml/hr No IV insulin drip	D5-NS w/ 5ml KPhos/ℓ at 300ml/hr No IV insulin drip

A.4 Case Keto 16

Keto 16, a 25yo female: Patient has "kidney infection" and c/o nausea. She came in to have her BS checked. Is now being Rx for UTI from four days PTA. Three days ago BS = 325, declined with 5U regular insulin. BS 304 this am, 469 in pm. Called Endo-on-call who Rx'd 4U extra insulin. Rechecked in evening (> 400). Presented at ER.
HPI: INFECTIONx5 days, NAUSEAx1 day
PMH: IDDMx17 years

Baseline Insulin: 10U Reg, 30U NPH SQ qam; 8U Reg, 10U NPH SQ qpm

	Vital Signs						Urine		
Time	Temp	HR	BP	RR	Weight	PO-In	Vol	Ket	Gluc
8/14 7:02p	36.6	76	130/80	18					
10:30p								+	++++
8/15 12:30p	37.2	120	120/90						
6:15p	36.8	84	142/92	20	74.9kg				
8/16 7:00a						230		+	++++
8:00a	36.7	84	130/78	16					

8/15 12:20a Patient Discharged to home.
8/15 11:25a Patient returned to emergency room.
8/15 6:15p Patient enters ward from emergency room.

	Laboratory (Serum)									Blood Gas				
Time	Na	K	Cl	CO_2	Cr	BUN	Phos	Ket	Gluc	pH	pO_2	pCO_2	HCO_3	FIO_2
8/14 10:00a									469					
5:00p									400+					
7:25p	133	4.7	94	26	0.9	19			514					
10:45p									378					
8/15 7:00a									421					
12:30p	132	4.9	92	23	1.1	21			675					
2:29p								++						
10:00p									245					
8/16 2:00a									217					
7:00a									308					
8:30a	138	4.6	103	27					269					
12:00p									135					
4:30p									186					
8/17 7:00a									109					

	IV Fluid Therapy							Insulin Therapy		
Time	Type	Rate	Addition	Type	Rate	Add.	Total	IV-Drip	IV-Bolus	SQ
8/14 10:00a										
2:00p	—	0	—				0	0		4U Reg
5:00p	—	0	—				0	0		8U Reg, 10U NPH
8:45p	—	0	—				0	0		4U Reg
8/15 7:00a	—	0	—				0	0		10U Reg, 30U NPH
1:00p	NS	95					0	0		
2:10p	↓	↓					111	0		10U Reg
6:15p	↓	171					499	0		
9:45p	↓	186	10KCl				1097	0		
8/16 7:00a	↓	131	↓				2817	0		10U Reg, 30U NPH
4:00p	↓	78	↓				3996	0		8U Reg, 10U NPH
8/17 7:00a	↓	↓	↓				5166	0		

# Time	Real	Advice
1. 8/14 8:45p	No IV fluids No IV insulin drip 4U Reg insulin SQ	NS w/ 20mEq KCl/ℓ at 90ml/hr Insulin at 6U/hr 6U Reg insulin IV bolus
2. 8/15 7:00a	No IV fluids No IV insulin drip 10U Reg, 30U NPH insulin SQ	NS at 90ml/hr No IV insulin drip 10U Reg, 15U NPH insulin SQ
3. 8/15 1:00p	NS at 100ml/hr No IV insulin drip	NS at 90ml/hr No IV insulin drip
4. 8/15 2:10p	NS at 100ml/hr No IV insulin drip 10U Reg insulin SQ	NS w/ 20mEq KCl/ℓ at 90ml/hr Insulin at 6U/hr 6U Reg insulin IV bolus
5. 8/15 6:15p	NS at 175ml/hr No IV insulin drip	NS w/ 20mEq KCl/ℓ at 100ml/hr Insulin at 7U/hr 7U Reg insulin IV bolus
6. 8/15 9:45p	NS w/ 10mEq KCl/ℓ at 175ml/hr No IV insulin drip	NS at 100ml/hr No IV insulin drip
7. 8/16 7:00a	NS w/ 10mEq KCl/ℓ at 175ml/hr No IV insulin drip 10U Reg, 30U NPH insulin SQ	No IV fluids No IV insulin drip 15U Reg, 30U NPH insulin SQ
8. 8/16 4:00p	NS w/ 10mEq KCl/ℓ at 175ml/hr No IV insulin drip 8U Reg, 10U NPH insulin SQ	No IV fluids No IV insulin drip
9. 8/17 7:00a	NS w/ 10mEq KCl/ℓ at 175ml/hr No IV insulin drip 10U Reg, 30U NPH insulin SQ	No IV fluids No IV insulin drip 10U Reg, 30U NPH insulin SQ

Appendix B

Evalution Results by Case

Tables B.1–B.4 show the raw data for the Ketoacidosis Advisor evaluation. Raters are designated by a code which identifies their training: "S" for students, "F" for fellows, and "A" for attending physicians. Student ratings were not used in the performance evaluation. The scores for the "real" and the "advice" rows are encoded by the first letter of the scale: Dangerous, Poor, Acceptable, Good, Excellent. The "prefer" row uses "R" to designate the real treatment, "A" to identify the advice and a single plus sign (+) to indicate that the given treatment choice was deemed better. Two plus signs (++) denote much better. No preference is indicated by "None." An asterisk (*) in the "Decision Point" column indicates the value was not used in the data analysis because the data collection was corrupted—Information was available in the clinic that was not provided to the evaluators. The reader can confirm that eliminating these evaluations improves the relative performance of the actual treatment.

Table B.1: Evaluation of Case 11

Rater	Item Type	Decision Point											
		1	2	3	4	5	6	7	8	9	10	11	12
F1	Real	E	G	A	P	P	P	P	A	G	P	A	A
	Advice	P	A	G	G	G	G	G	G	P	G	A	A
	Prefer	+R	+R	+A	+A	+A	+A	+A	+A	+R	+A	+R	+A
A3	Real	P	P	P	G	P	P	P	P	P	P	A	A
	Advice	E	E	G	A	E	G	E	E	P	P	P	G
	Prefer	++A	++A	+A	+R	++A	++A	++A	++A	None	None	+R	+A
S5	Real	A	D	D	P	A	A	D	P	A	P	D	D
	Advice	P	E	G	A	D	P	P	P	D	D	D	D
	Prefer	+R	+A	++A	+A	+R	+R	+A	None	+R	None	None	None
F6	Real	P	P	P	P	P	P	A	P	E	E	G	A
	Advice	G	G	G	G	G	G	A	P	D	P	P	D
	Prefer	+A	+A	+A	+A	+A	+A	None	None	++R	++R	++R	+R
A7	Real	P	P	P	A	A	G	A	A	P	P	D	P
	Advice	G	G	A	P	P	A	P	A	P	P	D	A
	Prefer	+A	+A	+A	+R	+R	+R	+R	None	+R	None	+R	+A
A9	Real	D	D	A	E	G	E	G	A	G	D	A	G
	Advice	E	E	A	A	P	P	E	E	P	E	E	A
	Prefer	++A	++A	None	+R	+R	++R	None	+A	++R	++A	++A	None
F10	Real	G	P	P	G	G	A	P	P	G	A	G	A
	Advice	G	P	G	A	A	G	G	G	P	P	P	A
	Prefer	+A	+A	+A	+R	+R	+A	+A	+A	+R	+R	+R	+A
A11	Real	P	P	D	D	A	G	A	A	G	A	G	A
	Advice	A	G	E	E	G	A	E	G	A	A	A	G
	Prefer	+A	++A	++A	++A	+A	+R	++A	+A	+R	None	+R	+A
F13	Real	P	P	P	P	P	P	P	P	A	A	—	—
	Advice	P	A	P	A	P	P	A	A	A	A	—	—
	Prefer	+A	+A	+R	+A	+A	+A	+A	+A	+A	None	—	—

Table B.2: Evaluation of Case 13

	Item	**Decision Point**								
Rater	**Type**	1	*	2	3	4	5	6	7	8
	Real	G	P	G	A	P	G	P	A	A
F1	Advice	A	G	P	P	P	P	A	A	G
	Prefer	+R	+A	+R	+R	+A	+R	+A	+A	+A
	Real	A	P	P	A	A	G	A	A	G
S2	Advice	A	A	G	G	G	G	G	G	A
	Prefer	None	+A	+A	+A	+A	None	+A	+A	+R
	Real	G	P	G	G	A	G	P	G	P
F4	Advice	G	G	E	E	G	G	G	G	P
	Prefer	None	++A	++A	+A	+A	None	++A	None	None
	Real	P	P	A	A	A	P	P	P	D
S5	Advice	A	P	P	P	P	A	P	P	D
	Prefer	None	None	None	+R	+R	+A	None	None	None
	Real	G	P	P	P	G	G	P	P	P
F6	Advice	A	G	G	G	P	P	A	A	P
	Prefer	+R	+A	+A	+A	+R	+R	+A	+A	None
	Real	A	P	A	A	A	G	D	P	P
A7	Advice	A	A	A	A	A	A	G	A	P
	Prefer	None	+A	+A	+R	+R	+R	+A	+A	None
	Real	A	P	P	P	G	G	D	A	D
A8	Advice	A	A	A	A	P	P	G	A	D
	Prefer	+A	+A	++A	++A	+R	++R	++A	+R	None
	Real	D	P	G	G	G	E	P	G	P
A9	Advice	D	A	P	P	P	P	E	A	G
	Prefer	None	+A	+R	+R	+R	++R	++A	None	+A
	Real	A	D	G	G	P	P	P	A	A
F10	Advice	A	G	P	A	G	G	G	G	G
	Prefer	+A	+A	+R	+R	+A	+A	+A	+A	+A
	Real	P	A	G	G	A	A	P	P	A
A11	Advice	A	G	A	A	P	P	A	A	A
	Prefer	+A	+A	+R	+R	+R	+R	+A	+A	None

Table B.3: Evaluation of Case 15

Rater	Item Type	Decision Point											
		1	*	2	3	4	5	6	7	8	9	10	11
F6	Real	G	P	P	P	P	G	A	G	G	G	G	G
	Advice	G	G	G	G	G	P	A	P	P	P	A	A
	Prefer	None	+A	+A	+A	+A	+R	None	+R	++R	+R	+R	+R
A7	Real	A	P	G	A	A	A	P	A	A	P	A	A
	Advice	A	A	A	P	P	P	P	P	D	D	P	A
	Prefer	None	+A	+R	+R	+R	+R	+A	+R	+R	+R	+R	+R
A9	Real	P	D	E	G	A	P	P	G	E	G	G	G
	Advice	P	P	P	A	A	E	G	G	A	A	P	G
	Prefer	None	None	++R	None	None	++A	++A	None	+R	+R	+R	None
F10	Real	G	G	G	G	G	A	E	G	G	A	A	P
	Advice	G	A	A	A	A	G	A	A	A	A	G	A
	Prefer	None	+R	+R	+R	+R	+A	+R	+R	+R	+A	+A	None

Table B.4: Evaluation of Case 16

‘

Rater	Item Type	Decision Point 1	2	3	4	5	6	7	8	9
S2	Real	G	A	A	A	A	A	G	P	A
	Advice	P	G	A	G	G	G	P	A	A
	Prefer	+R	+A	None	+A	+A	+A	+R	+A	None
F4	Real	A	P	E	A	P	A	G	D	D
	Advice	G	E	G	G	E	P	P	G	D
	Prefer	+A	+A	None	+A	+A	None	+R	+A	None
F6	Real	A	G	G	G	P	A	G	G	P
	Advice	P	A	G	D	P	G	P	P	G
	Prefer	+R	+R	None	++R	None	+A	+R	++R	+A
A7	Real	P	P	P	P	P	P	P	D	A
	Advice	A	P	P	A	A	A	P	P	P
	Prefer	+A	+A	None	+A	+A	+A	+R	+A	+R
A8	Real	P	D	G	A	P	P	P	D	P
	Advice	A	P	G	A	A	A	G	E	A
	Prefer	+A	+A	None	+A	+A	+A	+A	++A	+A
F10	Real	A	A	G	P	P	D	P	G	A
	Advice	A	A	A	E	E	D	P	A	G
	Prefer	+R	+A	+R	+A	+A	None	None	+R	+A
F13	Real	P	A	A	A	P	A	A	A	P
	Advice	A	A	A	A	P	A	A	P	A
	Prefer	+A	None	+A	+A	+R	+R	+R	+R	+A