Deriving and Using Descriptions of Purpose

David W. Franke, University of Texas at Austin

TED EXPRESSES THE PURPOSE OF COMPONENTS OR ACTIVITIES IN TERMS OF BEHAVIORS PREVENTED, GUARANTEED, OR INTRODUCED. OUR DESIGN METHOD CAPTURES THESE DESCRIPTIONS OF PURPOSE AND FACILITATES THEIR REUSE.

THE REPRESENTATION OF REAL-world systems and mechanisms was concerned initially with structure. Subsequently, researchers moved on to issues of representing and deriving the behaviors of these systems. From this examination arose efforts to derive and understand causal relationships. Given this base of representation and derivation techniques for structure, behavior, and causality, the next step in understanding real-world systems involves representing and deriving descriptions of teleology, or purpose.

We can understand the utility of descriptions of structure, behavior, causality, and purpose, as well as the differences among these descriptions, by considering the questions we answer with this information. A structure description addresses questions of the form "How is this mechanism constructed?" and "What are the physical (static) characteristics of this mechanism?" A behavior description addresses questions of the form "What does the mechanism do?" or "What are the dynamic characteristics of the mechanism?" Representations of structure and behavior provide the framework for a class of problem-solving techniques called model-based reasoning. Causal-reasoning techniques build on this framework, providing analyses and descriptions that address questions of the form "How does the mechanism accomplish its behavior?" Finally, a teleological description addresses questions of the form "Why is this portion of the mechanism designed in this way?" or "What is the purpose of this piece of the mechanism?"

When we examine human-generated descriptions of systems or mechanisms, we find they are rich with descriptions of purpose as well as of structure, behavior, and causality. In fact, descriptions of purpose are valuable in communicating and understanding design descriptions, since they convey the designers' intent. Consider the following design description:[1]

> Figure 3 shows a schematic of a pressurizer subsystem, the main purpose of which is to control the pressure of the primary circuit by maintaining a steam-water interface within the vessel through the controlled addition of heat and water spray.

This description mentions the purpose of the pressurizer subsystem, "to control the pressure of the primary circuit," as well as the technique by which this control is achieved, "by maintaining ...."

We have devised a language called Ted for representing teleological descriptions, along with a design method in which these descriptions can be captured and subsequently used for design reuse. This language is independent of any particular structure or behavior description language, but it builds on generalizations of such languages. Ted expresses the purpose of a component or activity in terms of behaviors prevented, guaranteed, or introduced by the component or activity. Given representation and acquisition schemes, system builders can use descriptions of purpose in explanation, diagnostic, and design systems.

0885/9000/91/0400-0041 $1.00 © 1991 IEEE

Teleology: what's the purpose?

One important use for teleological descriptions is explanation. For instance, de Kleer's Equal system generates teleological descriptions (in a form different from that in Ted) to explain electrical circuits.[2] The Lox Expert System represents and expresses the purpose of commands and command sequences, for human consumption and for analysis by automated systems.[3] The task domains of diagnosis and design can use teleological descriptions to extend the applicability and performance of automated problem-solving systems. In describing the Redesign system, Steinberg and Mitchell point out that in the domain of VLSI circuits, information regarding the purpose of circuit elements plays a role much like the problem-solving tasks of design and debugging.[4]

Diagnosis. A current approach in explanation and understanding systems[5-7] and in diagnosis systems[4,8-10] involves deriving and using causal relationships in mechanisms. However, in mechanisms with highly interconnected substructures, causal relationships can exist between virtually every pair of components in the mechanism (and between variables of a mechanism model). As Steinberg and Mitchell have noted,[4]

> The resulting focus (of causal reasoning) is generally broader than that determined from [the representation of purpose] because out of the many places in the circuit that can impact any given output specification, only a small proportion of these involve circuitry whose main purpose is to implement that specification.

Domain-specific heuristics can be applied to select among potential causes, but these heuristics are not applicable outside their domains. If an observed symptom of a mechanism is considered either an unwanted behavior (or a missing behavior), a teleological description relating a component with the prevention (or the introduction or guarantee) of that behavior provides a heuristic for selecting among potential causes. Certainly, teleological descriptions cannot introduce any relationship not discovered via causal analysis, as purpose necessarily requires causality. Therefore, teleological descriptions provide a more productive focus for diagnosis.

Design. Capturing and representing a design rationale can improve the design process.[11] If we can identify no relevant requirement when we try to relate a design decision to the requirements for the designed artifact, then the design decision is either superfluous or addresses a hidden or unelaborated design requirement. This is often the case for design decisions that address requirements stemming from the general design or engineering principles of a particular domain. For example, all circuit designers understand a body of design principles that are not explicitly elaborated in circuit design requirements.

NEITHER BEHAVIORAL NOR CAUSAL DESCRIPTIONS EXPLAIN WHY A CERTAIN BEHAVIOR IS DESIRED.

Given the ability to capture and represent teleological descriptions (generated by either people or programs), we can use these descriptions to classify mechanism descriptions. Disciplines such as electrical engineering have developed specialized vocabularies for denoting the purposes of mechanisms and components.[2] Further, system builders use these descriptions to index other design information whose relevance is determined by the component's current purpose. The ability to realize these descriptions in a formal language will facilitate their use as design knowledge. If this language is domain independent, it will allow teleological descriptions for mechanisms whose components come from several domains (electrical, mechanical, and hydraulic domains, for instance).

Design reuse. Design reuse is a research area in computer-aided design and engineering environments (electronics, software, and mechanical, to name a few). To realize design reuse, we must be able to capture and represent the information needed to classify a design component for subsequent retrieval. Design reuse also requires a language for describing the characteristics of design components to be examined as candidates for reuse. As Mostow and Barley explain,[12]

> For design by analogy, finding a suitable design to retrieve from a repository of previous designs requires knowing where to look. How can designers avoid a time-consuming search through such a repository when they've never seen the relevant entry or can't remember where to find it? As we develop a larger database of design plans, we expect the process of finding relevant ones to become a bottleneck...

Teleological descriptions add another dimension in which designs can be classified and retrieved. For example, a designer might want to examine components that control some variable (say tank fluid level) of a system under design. To construct a query for the search without having a description of purpose, designers must rely on their mental inventory, the structural features, or the specific behaviors of likely components. Without effective indexing methods into a database of existing designs, designers will more likely miss innovative solutions that do not fit their current mental model of how to solve the problem — that is, what kind of component to use.

Finally, when an existing design is being reused but does not match current requirements, it requires some modification. Knowing the purpose of components benefits the designer making the modifications in much the same way a teleological description aids the diagnosis task. In this case, the task is driven by a change in requirements (required or prohibited behaviors) as opposed to modified behavior on the part of the mechanism. Designers have used functional represention of device functions and structure to address the redesign problem.[13] The Redesign system, for example, applies both causal and teleological reasoning to redesign a circuit based on specification changes.[4] As we claimed for diagnosis tasks, Redesign uses teleological descriptions to provide a tighter focus to select candidate components for modification.

Ted — a language for teleology

Human-generated descriptions of purpose identify relationships between specific design decisions and design goals. We describe these goals in terms of the

Qualitative modeling with Qsim

Research in reasoning with qualitative models has been motivated by such concerns as providing programs with commonsense reasoning or with the ability to reason about the physical world in the face of incomplete knowledge. For example, specific numerical values might be unknown for parameters that measure the physical world, and specific details of the mathematical relationships between these parameters might be unknown. We might know that the level of liquid in a tank increases as the amount of liquid is increased, but the shape of the tank can make the liquid level a nontrivial function of the amount of liquid. For many purposes, however, we can effectively reason about this tank system knowing only that liquid amount and liquid level are monotonically related.

Several approaches to qualitative reasoning have been investigated,[1,2] with the common theme of qualitative representation of parameter values and the functions that constrain these parameters. This article describes the modeling approach and language of Qsim.[3,4]

In qualitative modeling, a parameter can assume values from a quantity space (an ordered set of discrete landmarks). The simplest quantity space specifies negative, zero, and positive values (sometimes described as −/0/+). In Qsim, quantity spaces are expressed as ordered sets of landmarks. The simplest quantity space is expressed as $(-\infty\ 0\ \infty)$, where $-\infty$ represents minus infinity and ∞ represents infinity. The possible values for a parameter are the individual landmarks in the quantity space, and the open intervals implied by adjacent landmarks. For example, two values from this quantity space are $(-\infty\ 0)$ and 0. Additional landmarks can be added to a quantity space, such as the height of a tank for the parameter representing fluid level. The quantity space would then be $(0\ \text{full}\ \infty)$. The landmark $-\infty$ has been removed from this quantity space, indicating that a negative value for the height parameter is not meaningful.

Qsim represents the value of a parameter as a pair — a qualitative magnitude (taken from the parameter's quantity space) and a qualitative direction (taken from Dec, Std, Inc, and Ign, representing decreasing, steady, increasing, and ignore). Hence, the value representing a decreasing liquid height in the tank somewhere between empty and full would be ((0 full), Dec).

Some of Qsim's qualitative constraints among parameters are M+ (monotonically increasing), M− (monotonically decreasing), d/dt (derivative), and add (additive). For example, the amount of fluid in a tank (Amt) and the level of fluid in the tank (Level) are monotonically related, and expressed as (M+ Amt Level). For purposes of simulation in Qsim, these constraints are not causal, and information can be propagated through the constraint in either direction.

Given a qualitative model expressed as parameters, quantity spaces, and constraints, Qsim generates the model's possible behaviors. There can be (and usually are) more than one possible behavior for a model, since qualitative models are less constraining that quantitative ones. Hence, the behavior descriptions generated by Qsim are expressed as a tree (one initial state) or a forest (multiple initial states) in which a path from the tree root to a leaf represents one possible behavior. Each node in the tree represents a state, and branches from a state lead to possible successor states.

For example, if a constant rate of input is initiated into a empty tank with an open drain, three qualitatively different behaviors result. One possible behavior is that this inflow/outflow system reaches equilibrium (that is, inflow equals outflow, and the level is constant) for a level between empty and full. A second possible behavior is that equilibrium is reached when the level exactly equals full. Finally, the third (qualitatively distinct) behavior states that equilibrium is not reached when the level reaches full (that is, the tank overflows). This characteristic of Qsim is very important for acquiring descriptions of purpose, in that from the given initial states, Qsim generates all possible behaviors. In other words, any real behavior that the device will generate maps to one of the qualitative behaviors generated by Qsim.

References

1. *Qualitative Reasoning about Physical Systems*, D.G. Bobrow, ed., MIT Press, Cambridge, Mass., 1985. Reprinted from *Artificial Intelligence*, Vol. 24, 1984.
2. K.D. Forbus, "Qualitative Physics: Past, Present, and Future," in *Exploring Artificial Intelligence: Survey Talks from Nat'l Confs. Artificial Intelligence*, H.E. Shrobe, ed., Morgan Kaufmann, San Mateo, Calif., 1988, pp. 239-296.
3. B.J. Kuipers, "Commonsense Reasoning about Causality: Deriving Behavior from Structure," in *Qualitative Reasoning About Physical Systems*, D.G. Bobrow, ed., MIT Press, Cambridge, Mass., 1985, pp. 169-203. Reprinted from *Artificial Intelligence*, Vol. 24, 1984.
4. B.J. Kuipers, "Qualitative Simulation," *Artificial Intelligence*, Vol. 29, No. 3, Sept. 1986, pp. 289-338.

desired artifact's behaviors and physical characteristics (for example, its dimensions or weight). We express relationships using verbs such as "control," "regulate," and "prevent."

Our language for descriptions of purpose, Ted, identifies (1) design modifications made during the design process — for instance, the addition of a component or the modification of a specific parameter value; and (2) the effects these changes have on the designed artifact's behavior. These effects are expressed in terms of the primitives Guarantees and Prevents.

Consider the case of a steam boiler with a pressure release valve.[14] The valve's purpose is to prevent the internal pressure of the boiler from exceeding some critical value, at or beyond which the boiler vessel will explode. A behavior description of the pressure release valve states that the valve opens at some prescribed pressure and that steam escapes the vessel via the valve. A causal description can explain how this behavior is achieved, but neither the behavioral nor causal descriptions explain why this behavior was desired, and hence why the pressure release valve was added to the design.

Ted has the following goals:

- to be independent of any structure or behavior language,
- to be independent of any domain of mechanisms, and
- to allow hierarchical descriptions referencing behaviors or other teleological descriptions.

Structure and behavior. To handle references to design modifications and behaviors, descriptions in Ted had to be based on languages that express structure and behavior. Instead of inventing new languages, we focused on using existing ones. The structure language was required to be able to express design modifications. (Most interactive design environments express design modifications in terms of editing primitives, and use these to undo editing operations at the designer's request.) The behavior language was required to express behaviors as ordered sequences of states of the designed artifact (model or device), and to describe device states in terms of a set of variables and their values. For the examples given here, we use the behavior language Qsim (see the sidebar).[14,15]

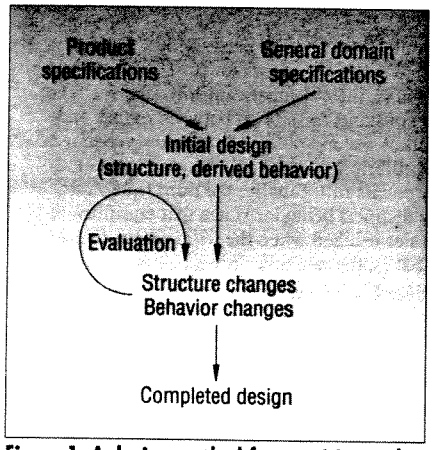


Figure 1. A design method for acquiring and using teleological descriptions.

Understanding these examples does not require a description of the structure language.

Before we provide terminology and definitions for Ted, we can demonstrate a teleological description for adding the pressure release valve. If δ represents the design modification of adding the release valve, we can say that δ prevents the steam boiler's internal pressure from exceeding some maximum value max. In Ted, we express this as

δ prevents <{(internal_pressure, ((max, ∞), ign))}>

To achieve this purpose, the release valve opens when Internal_pressure reaches the value Max.

Partial states and scenarios. To the behavior language's terminology (variables, states, and behaviors), Ted adds the terms "partial state" and "scenario." A partial state is an abstraction of a state, possibly equal to the state. A partial state describes the values of a subset of model variables, and can further abstract a state by specifying a range for a variable value, such as $x > 0$. The partial order $\sqsubset_v$ (read "is less general than") captures this notion of variable value abstraction. For example, for variable x with the domain of the union of real numbers and open intervals on real numbers, the values 5, (4, 6), (0, ∞), and R are related by $\sqsubset_v$ as follows: $5 \sqsubset_v (4,6) \sqsubset_v (0,\infty) \sqsubset_v R$.

The partial order $\sqsubset_s$ (read "is less general than") extends this notion to partial states and the states they abstract. Hence, for state s and partial state p of design d, with V denoting the variables of d, and $V_p \subseteq V$ denoting the variables referenced by p,

$$s \sqsubset_s p \Leftrightarrow \begin{cases} s = \{(v,x) \mid v \in V, x \text{ in the domain of } v\}, \\ p = \{(v,x) \mid v \in V_p, x \text{ in the domain of } v\}, \\ \quad \text{and} \\ \forall\, v \in V_p, (v,x_s) \in s \text{ and } (v,x_p) \in p \Rightarrow \\ \quad x_s \sqsubset_v x_p \end{cases}$$

A scenario is defined as a sequence of partial states in which the subset of variables is the same in all partial states in the sequence. Therefore, for design d, with V being the variables of d and $V_p \subseteq V$,

$$\sigma \text{ is a scenario of } d \Leftrightarrow \begin{cases} \sigma = \langle p_1,\ldots,p_n\rangle \text{ and} \\ \forall\, p_i, p_i = \{(v,x) \mid v \in V_p, x \text{ in the domain of } v\} \end{cases}$$

In addition to ignoring the model's variables, a scenario can also generalize a sequence of states by eliminating certain states in the sequence. Hence, partial states of a scenario need not be adjacent states. To demonstrate this point, consider the Qsim scenario

<{(x, (0, dec))}, {(x, (0, inc))}>

where Dec denotes decreasing and Inc denotes increasing. The states that are abstracted in this scenario cannot be adjacent in a behavior of the model, but are allowed as an abstraction of the behavior in the form of a scenario. Specifically, the scenario says that x at some time had the qualitative value (0, Dec), and at some later time (with an unspecified number of intervening values) had the value (0, Inc). In the semantics of Qsim, the variable x must take on a qualitative value whose direction of change is Std (steady) between Dec and Inc.

Mapping scenarios to behaviors. A scenario is said to Occur-in a behavior if a mapping exists from the scenario into the behavior such that (1) a partial state is mapped to a state that it abstracts, and (2) the order of states implied by the scenario is preserved in the behavior. Given two scenarios,

$\sigma = \langle p_1,\ldots,p_n\rangle$
$\sigma' = \langle p_1',\ldots,p_m'\rangle$

we can describe their temporal relationships as Occur-serially and Occur-synchronously. For instance, σ and σ' Occur-serially in behavior b if σ and σ' each Occur-in b in such a way that p_n is mapped to state s_i of b, p_1' is mapped to state s_j of b, and $i < j$. σ and σ' Occur-synchronously in behavior b if $n = m$, and σ and σ' each Occur-in b in such a way that corresponding partial states of σ and σ' are mapped to the same state of b. A notation for σ and σ' Occurring-serially is $[\sigma ; \sigma']$, and for Occurring-synchronously is $[\sigma \parallel \sigma']$.

Primitive teleological operators. Teleological operators are the language primitives for teleological descriptions. In the context of design modification, a single teleological operator relates the behaviors of the unmodified design to the behaviors of the modified design in terms of scenarios. In the following definitions, σ_i are scenarios, d and d' are designs, δ is a modification applied to d that yields d', E is the set of behaviors of design d, and E' is the set of behaviors of d'. The term Scenarios(b) denotes the set of scenarios that Occur-in behavior b. Each teleological operator makes a statement about both modified and unmodified designs:

(1) δ guarantees $\sigma \Leftrightarrow$
$$\begin{cases} \forall\, b' \in E' \;\; \sigma \in \text{scenarios}(b'), \text{ and} \\ \exists\, b \in E \;\; \sigma \notin \text{scenarios}(b) \end{cases}$$

(2) δ prevents $\sigma \Leftrightarrow$
$$\begin{cases} \forall\, b' \in E' \;\; \sigma \notin \text{scenarios}(b'), \text{ and} \\ \exists\, b \in E \;\; \sigma \in \text{scenarios}(b) \end{cases}$$

The third primitive operator provides a means for creating teleological descriptions that involve preconditions. This precondition references a set of scenarios on which a sentence (a single teleological operator or a sentence constructed from teleological operators and logical connectives) depends. In the following definition, T is a teleological sentence:

(3) δ conditionally (in $\{\sigma_1,\ldots,\sigma_n\}$) $T \Leftrightarrow$
$$\begin{cases} \forall\, b' \in E' \{\sigma_1,\ldots,\sigma_n\} \subseteq \text{scenarios}(b') \\ \quad \Rightarrow T \text{ holds in } b', \text{ and} \\ \exists\, b \in E \;\{\sigma_1,\ldots,\sigma_n\} \subseteq \text{scenarios}(b) \\ \quad \wedge\, T \text{ does not hold in } b \end{cases}$$

These three primitive teleological operators form the basis on which we build teleological descriptions.

Composed teleological operators. The Ted language can define semantically richer operators in terms of the three primitive operators Guarantees, Prevents, and

Conditionally. When we examine descriptions of purpose generated by designers, we find verbs such as "introduce," "control," "regulate," "maximize," "reduce," and "allow." The following definitions are not derived from any particular temporal logic, but are merely hypothesized as useful in constructing teleological descriptions. These definitions demonstrate how to decompose such verbs into teleological primitives directly or via previously defined verbs:

(1) δ orders $\sigma_1, \sigma_2 \Leftrightarrow$

$$\begin{cases} \forall\, b' \in E' \ \{\sigma_1, \sigma_2\} \subseteq \text{scenarios}(b') \Rightarrow \\ \quad [\sigma_1 ; \sigma_2] \in \text{scenarios}(b'), \text{ and} \\ \exists\, b \in E \ \{\sigma_1, \sigma_2\} \subseteq \text{scenarios}(b) \wedge \\ \quad [\sigma_1 ; \sigma_2] \notin \text{scenarios}(b) \end{cases}$$

We can rewrite this as

δ conditionally (in $\{\sigma_1,\sigma_2\}$) guarantees $[\sigma_1; \sigma_2]$

(2) δ synchronizes $\sigma_1,\sigma_2 \Leftrightarrow$

$$\begin{cases} \forall\, b' \in E' \ \{\sigma_1, \sigma_2\} \subseteq \text{scenarios}(b') \Rightarrow \\ \quad [\sigma_1 \parallel \sigma_2] \in \text{scenarios}(b'), \text{ and} \\ \exists\, b \in E \ \{\sigma_1, \sigma_2\} \subseteq \text{scenarios}(b) \wedge \\ \quad [\sigma_1 \parallel \sigma_2] \notin \text{scenarios}(b) \end{cases}$$

We can rewrite this as

δ conditionally (in $\{\sigma_1, \sigma_2\}$) guarantees $[\sigma_1 \parallel \sigma_2]$

(3) δ introduces $\sigma \Leftrightarrow$

$$\begin{cases} \exists\, b' \in E' \ \sigma \in \text{scenarios}(b') \\ \forall\, b \in E \ \sigma \notin \text{scenarios}(b) \end{cases}$$

We can rewrite this as

δ guarantees $\neg$(prevents σ)

An environment for design

Given a language for expressing descriptions of purpose, our design method can capture these teleological descriptions during the design process. It can also use these descriptions to reuse designs and to diagnose the designed artifact in the future.

We used a design process model from the propose-critique-modify family described by Chandrasekaran.[16] Our model starts with a set of design specifications, including physical characteristics and descriptions of required, prohibited, and other behaviors. In addition to the design specifications, there should be a set of specifications — design rules, if you will — that describe general engineering practice for the domain at hand. The design process then proceeds as a series of structure modifications, starting from some initial structure. Accompanying this series of structure descriptions is the corresponding series of behavior descriptions that can be generated from the structure descriptions. This process terminates when the structure and behavior descriptions meet both the design-specific and the general engineering specifications (see Figure 1). By examining the changes that structure modifications induce in behavior descriptions, we can infer the purpose of the design modifications.

A related design environment described by Abelson et al.[17] supports interaction between a designer and an intelligent computer assistant that helps analyze and evaluate proposed designs. This interaction identifies a design's undesirable behavior (specifically, an oscillation at a particular frequency for a floating ocean platform), at which point the designer initiates a design modification to correct this behavior ("an active stabilizer to damp the family B motions.") A description of purpose can be acquired at precisely this point: The purpose of the addition of the "active stabilizer" is "to damp the family B motions."

Design example. Consider the desired behavior of a simple CMOS circuit (shown in Figure 2) containing a pass transistor t_1 and an inverter. When the control signal's value is High, the data signal's value is transmitted to the in signal (that is, they are electrically connected). The inverter inverts this value (High to Low, or Low to High), which then becomes the value of the out signal. The logic values of High and Low are landmarks of the quantity space in which the data, control, in, and out parameters range. These landmarks are the desired values for signals in the circuit, which brings us to the first undesirable behavior.

Modification 1. The operating characteristics of t_1 (an n-channel MOS transistor) are such that when the data and control signals both have the value High, the value transmitted to the in signal is (High – *Th*), where *Th* is the threshhold value (> 0) of t_1. Let *HTh* equal High – *Th*. The value *HTh* is between the landmark values Low and High, and hence not a desired value for the in signal. In the domain of CMOS circuit design, it is a design goal to have a signal at values between High and Low only when the signal is transitioning between High and Low. Steady, intermediate values between Low and High should be eliminated.

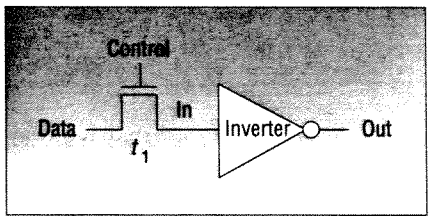


Figure 2. A CMOS circuit.

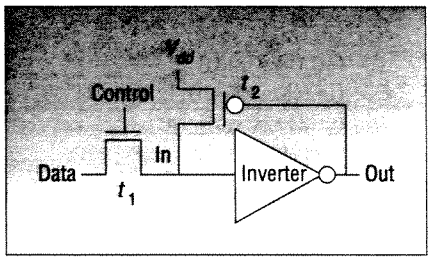


Figure 3. A circuit with a feedback transistor.

Adding the feedback transistor t_2 (a p-channel MOS transistor) modifies this behavior in terms of the in and the out signals, as shown in Figure 3. As the in signal transitions from Low to High, the out signal transitions from High to Low. As the out signal moves away from High and toward Low, transistor t_2 electrically connects the in signal with V_{dd}, enabling a current flow from V_{dd} to the in signal. This in turn increases the in signal's value to that of V_{dd} (High). Consequently, adding t_2 prevents the scenario in which the in signal reaches a value less than High and remains steady.

To express this purpose of t_2 in the Ted language, let δ_1 represent the design modification of adding t_2 to the design, and let σ_1 be the scenario

```
<{  (in, ((low, high), std)),
    (ctl, (high, std)),
    (data, (high, std)) }>
```

Then, as expressed in Ted, δ_1 prevents σ_1.

The behavior that electrically connects the in signal to V_{dd} also addresses another problem that occurs when the in signal has value High and the control signal transitions from High to Low. In this situation, the in signal is no longer electrically connected to the data signal; it becomes a memory element, which should preserve its value, High. However, in the absence of t_2, the charge at the in signal will dissipate and move the signal value away from the landmark value High, resulting in the value of

the out signal changing also (moving away from Low). By introducing t_2, we can maintain the charge at the in signal at High, preventing the in signal from decreasing in value.

To express this purpose of t_2 in the Ted language, let σ_2 be the scenario

```
<{   (in,(high, std)),
     (ctl,(low, std)) },
 {   (in,((low, high), ign)),
     (ctl,(low, std)) }>
```

Then, as expressed in Ted, δ_1 prevents σ_2.

The design modification of adding t_2 to the circuit can be assigned the purpose of preventing a steady or nontransitional signal value between Low and High when the data and control signals have the value High. Further, from starting conditions where the in signal has the value High and the control signal has the value Low, the modification prevents the in signal from changing its value.

Modification 2. While the first design modification addresses problems associated with the in signal achieving and maintaining value High, it also introduces a new problem. If the in signal has value High, the data signal has value Low, and the control signal transitions from Low to High, then the charge stored at the in signal (representing the value High) should be drawn off via the connection through t_1. However, if the channel resistance of t_2 (when open) is low, current will flow from V_{dd} to the in signal. If this happens at a sufficient rate, an intermediate value will be reached for the in signal such that the complementary value at the out signal is not high enough to "turn off" t_2 (a p-channel transistor is off when the gate voltage is High). Hence, the second design modification raises the channel resistance of t_2 to a higher value when it is "open," to prevent the in signal from reaching an equilibrium point between High and Low during its High-to-Low transition.

Again, a teleological description relating the design modification (changing the channel resistance of t_2) can be related to the desired change in behavior, namely that the circuit can successfully switch the in signal value from High to Low.

The purpose of t_2's channel resistance value can be expressed in Ted as follows. Let δ_2 represent the design modification of increasing t_2's channel resistance, and let σ_3 be the scenario

```
<{   (in, ((low, high), std)),
     (ctl, (high, std)),
     (data, (low, std)) }>
```

Then, as expressed in Ted, $\delta 2$ prevents $\sigma 3$.

To summarize this example, we added t_2 to prevent (in Ted terminology)

- the scenario in which the in signal reaches a steady value between Low and High when transitioning from Low to High, and
- the scenario in which the in signal's value decreases from High when the signal is acting as a memory element storing the value High.

THE TED LANGUAGE COMPLEMENTS FUNCTIONAL REPRESENTATION BY PROVIDING A DETAILED, FORMAL LANGUAGE FOR EXPRESSING PURPOSE.

The channel resistance of t_2 was set high to prevent (also Ted terminology) the scenario in which the in signal reaches an equilibrium value between Low and High during the transition from High to Low.

Reusing designs. Consider the CMOS circuit design example in the context of an accompanying database of design modifications and components, which have associated teleological descriptions. (A design modification or component can have several teleological descriptions associated with it, since the modification might have been applied in different contexts with different results.) If the design modification of adding the feedback transistor is recorded in the database with the associated teleological description of preventing the intermediate signal value, then this design modification is available to the designer via the query "Show me design modifications that prevent the behavior in which a signal maintains an intermediate value between Low and High." This query could be generated by the designer or by a design critic examining the behaviors of the design and comparing those behaviors with the specifications. This design critic can then present discrepancies between the design's behavior and its specifications, and suggest possible modifications to correct these discrepancies.

More generally, the goals of the designer at each design step can be used to index a database of existing design modifications (including complete components) to retrieve modifications or components for reuse. The designer can access existing design modifications and components in terms relevant to the task at hand, namely solving a specific design problem. Indexing existing design information in terms of structural or behavioral aspects alone cannot provide this relevance.

Interestingly, behaviors referenced in teleological descriptions can be abstracted beyond specific domains by ignoring variable types, and hence provide a mechanism for retrieving design solutions from other domains (for example, electrical versus mechanical). Retrieved design modifications or components offer potential solutions that designers can subsequently apply when suitable structural analogies are available.

Finally, the design method shown in Figure 1 is relevant when an existing design is being modified to meet a changing set of specifications. In this instance, teleological descriptions can assist the redesign task in two ways. First, as a specification is changed, all design modifications and components with teleological descriptions that reference the specification (for example, required or prohibited behaviors) are primary candidates for modification to meet new specifications. Similar approaches to redesign are described elsewhere, using design plans[4] and functional representations.[13] Second, as the designer explores possible modifications, teleological descriptions associated with the current design structure give the designer information concerning other design behaviors that might be affected if a particular component is modified.

Diagnosis

The role of teleological descriptions in diagnosis is essentially that described for redesign — namely, providing a focus for selecting structural components that are likely to contribute to observed or desired behaviors. In the task domain of model-based diagnosis,[10] this is called candidate

or hypothesis generation. This process generates a set of candidate structure compo-nents that could possibly account for missing or undesirable behaviors. The system evaluates each candidate and finally chooses a single candidate or set of candidates that best accounts for the aberrant behavior.

Techniques for generating the candidate set include dependency tracing and causal analysis. For devices with highly interconnected structures, this set can comprise a large percentage of the device's structural components — possibly all of them. Since all these candidates might require evaluation, it is important to focus the candidate generation process where possible. Teleological descriptions can provide this focus, allowing the system to generate an initial candidate set based on structural components known to have been placed in the design for the purpose of affecting the currently aberrant behavior. However, this candidate generation focus is not necessarily complete. The candidate set generated in this way might not contain the structural component causing the aberrant behavior.

WE HAVE IMPLEMENTED algorithms for deriving teleological descriptions by comparing design modification with design specifications, and we are modeling designs using Qsim. We are also developing a design environment (as described in Figure 1) that integrates derivation techniques for descriptions of purpose.

In comparison with the research on functional representation and functional modeling,[7,13,18] this work has focused on representing and acquiring statements of purpose, and does not address the representation of causal descriptions found in functional representation. Statements represented in the Ted language correspond to the Tomake clauses found in functional representation. The Ted language complements functional representation by providing a detailed, formal language for expressing purpose, which functional representation enriches with causal information. This causal information describes how a design accomplishes the behaviors referenced in the teleological description; in other words, how a design achieves the designer's purpose.

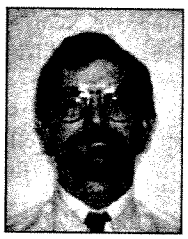
Acknowledgments

This work was done by the Qualitative Reasoning Group at the Artificial Intelligence Laboratory of the University of Texas at Austin. The group's research is supported in part by NSF grants IRI-8905494 and IRI-8904454, by NASA grant NAG 2-507, and by the Texas Advanced Research Program under grant 003658175.

Ben Kuipers has motivated and supported this work. Members of the Qualitative Reasoning Group, especially Dan Dvorak and Dan Berleant, contributed many useful comments, questions, and suggestions. Fruitful discussions have also occurred with members of the functional-representation community, especially Ashok Goel, Jon Sticklen, Tom Bylander, and B. Chandrasekaran.

References

1. M.R. Herbert and G.H. Williams, "An Initial Evaluation of the Detection and Diagnosis of Power Plant Faults Using a Deep Knowledge Representation of Physical Behaviour," *Expert Systems*, Vol. 4, No. 2, May 1987, p. 91.
2. Johan de Kleer, "How Circuits Work," in *Qualitative Reasoning about Physical Systems*, D.G. Bobrow, ed., MIT Press, Cambridge, Mass., 1985, pp. 205-280. Reprinted from *Artificial Intelligence*, Vol. 24, 1984.
3. E.A. Scarl, J.R. Jamieson, and C.I. Delaune, "Process Monitoring and Fault Location at the Kennedy Space Center," *SIGArt Newsletter*, special section on reasoning about structure, behavior, and function, No. 93, July 1985, pp. 38-44.
4. L.I. Steinberg and T.M. Mitchell, "A Knowledge-Based Approach to VLSI CAD: The Redesign System," *Proc. 21st Design Automation Conf.*, Computer Soc. Press, Los Alamitos, Calif., 1984, pp. 412-418.
5. R.J. Doyle, "Constructing and Refining Causal Explanations from an Inconsistent Domain Theory," *Proc. Fifth Nat'l Conf. Artificial Intelligence*, MIT Press, Cambridge, Mass., 1986, pp. 538-544.
6. B.J. Kuipers, "Qualitative Simulation as Causal Explanation," Tech. Report AI-TR86-24, AI Lab, Univ. of Texas at Austin, Austin, Texas, Apr. 1986.
7. J. Sticklen, B. Chandrasekaran, and W.E. Bond, "Applying a Functional Approach for Model-Based Reasoning," *Proc. 1989 Workshop Model-Based Reasoning*, Boeing Computer Services, Bellevue, Wash., 1989, pp. 165-176.
8. R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," in *Qualitative Reasoning about Physical Systems*, D.G. Bobrow, ed., MIT Press, Cambridge, Mass., 1985, pp. 347-410. Reprinted from *Artificial Intelligence*, Vol. 24, 1984.
9. *SIGArt Newsletter*, special section on reasoning about structure, behavior, and function, No. 93, July 1985.
10. R. Davis and W. Hamscher, "Model-Based Reasoning: Troubleshooting," in *Exploring Artificial Intelligence: Survey Talks from the Nat'l Confs. Artificial Intelligence*, H.E. Shrobe, ed., Morgan Kaufmann, San Mateo, Calif., 1988, pp. 297-346.
11. J. Mostow, "Towards a Better Model of the Design Process," *AI Magazine*, Vol. 6, No. 1, Spring 1985, pp. 44-57.
12. J. Mostow and M. Barley, "Automated Reuse of Design Plans," *Proc. 1987 Int'l Conf. Eng. Design (ICED87)*, Am. Soc. Mechanical Engineers, New York, 1987, pp. 632-647.
13. A. Goel and B. Chandrasekaran, "Functional Representation of Designs and Redesign Problem Solving," *Proc. 11th Int'l Joint Conf. Artificial Intelligence*, Morgan Kaufmann, San Mateo, Calif., 1989, pp. 1,388-1,394.
14. B.J. Kuipers, "Commonsense Reasoning about Causality: Deriving Behavior from Structure," in *Qualitative Reasoning About Physical Systems*, D.G. Bobrow, ed., MIT Press, Cambridge, Mass., 1985, pp. 169-203. Reprinted from *Artificial Intelligence*, Vol. 24, 1984.
15. B.J. Kuipers, "Qualitative Simulation," *Artificial Intelligence*, Vol. 29, No. 3, Sept. 1986, pp. 289-338.
16. B. Chandrasekaran, "Design Problem Solving: A Task Analysis," *AI Magazine*, Vol. 11, No. 4, Winter 1990, pp. 59-71.
17. H. Abelson et al., "Intelligence in Scientific Computing," *Comm. ACM*, Vol. 32, No. 5, May 1989, pp. 546-562.
18. V. Sembugamoorthy and B. Chandrasekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems," in *Experience, Memory, and Reasoning*, J.L. Kolodner and C.K. Riesbeck, eds., Lawrence Erlbaum, Hillsdale, N.J., 1986, pp. 47-73.



David W. Franke is a doctoral candidate in computer science at the University of Texas, and a senior member of technical staff for the CAD program of Microelectronics and Computer Technology Corp. He earned his BS in mathematics from the University of Oklahoma and his MS in computer science from Pennsylvania State University. His research interests include design knowledge representation, design reuse, and model-based reasoning. He is a member of the IEEE Computer Society, AAAI, ACM, and Upsilon Pi Epsilon.

Franke can be reached at MCC, 3500 W. Balcones Center Drive, Austin, TX 78759; e-mail, franke@mcc.com