# Chapter 1

# Models for Discrete Systems

## 1.1 Interface Modules

We define *interface modules,* a model for discrete system components that can communicate with their environment. Interface modules model both the behavior of a system component, and the interface between the component and its environment. The goal of interface modules is to support the following activities.

- **Top-down design decomposition.** A design is decomposed into components that can be designed and implemented separately. The component model is used to specify the task of each component; the interface specification ensures that the components, once implemented, can work together correctly.

- **Compositional verification.** In order to verify a complete design, each component is studied with the help of assumptions about its environment. The results for the single components are then combined into an analysis for the complete system. The interface specifications constitute the assumptions about the environment, while the component models describe each component.

- **Component-based design.** In component-based design, designs are created by combining pre-existing and application-specific components. The models of the components and their interfaces help in selecting and combining the components, and in checking that the component interfaces are compatible one with the other.

In the remainder of this course, interface modules will be simply called *modules.*

### 1.1.1 Modules, informally

The state of a module is described by a set of *state variables,* partitioned into sets of *input* and *output* variables. The input variables represent inputs to the module, and their value can be read, but not changed, by the module; the output variables represent outputs of the module, and their value can be changed (and read) by the module. The possible changes of output variables are described by an *output transition relation,* while the changes of input variables that are legal for the module are described by an *input transition relation.* Hence, the output transition relation models the module's behavior, and the input transition relation models design assumptions about the inputs provided by the environment. Finally, a set of *initial outputs* specifies the initial condition of the module, and a set of *initial inputs* specifies the desired initial condition of the environment. Hence, a *module* differs radically in spirit from the model of a physical system as found in usual physics or engineering courses. A model for a physical system defines the behavior of the system under *all* external behaviors: for example, the model of a linear system such as an (ideal) amplifier defines the output signal in response to any input signal. In other words, usual models of physical systems do not constrain their set of possible inputs. In contrast, a module encodes both the possible behaviors of the component (via its initial outputs and output transition relation), and the *legal* input behaviors (via its initial inputs, and the input transition relation).

The behavior of a module consists of an infinite sequence of states starting from an initial state. Each successive state is generated by the module and by its environment: the modules chooses the new values of the output variables according to the output transition relation, and the environment must choose the new values of the input variables according to the input transition relation.

### 1.1.2 Modules, formally

**Variables.** In order to define formally modules, we consider a global (infinite) set $\mathcal{W}$ of *typed variables,* from which the variables of the modules will be drawn. Each variable $x \in \mathcal{W}$ has an associated *domain,* or set of possible values, which we denote $\mathcal{D}(x)$. In this course, we will assume that $\mathcal{D}(x)$ is a finite set, restricting our attention to finite-state systems. An important special case is the one in which all variables are *boolean,* that is, their domain is the two-element set $\{\mathrm{F}, \mathrm{T}\}$.

**States.** Given a finite set $\mathcal{V} \subseteq \mathcal{W}$ of variables, a *state* $s$ over $\mathcal{V}$ is a function that associates with each variable $x \in \mathcal{V}$ a value $s(x) \in \mathcal{D}(x)$; we denote by $\mathcal{S}[\mathcal{V}]$ the set of all possible states over the variables $\mathcal{V}$. Note that, formally, the type of $s \in \mathcal{S}[\mathcal{V}]$ is $\prod_{x \in \mathcal{V}}(x \mapsto \mathcal{D}(x))$. Given a state $s \in \mathcal{S}[\mathcal{V}]$ and a subset $\mathcal{U} \subseteq \mathcal{V}$ of variables, we denote by $s[\mathcal{U}] \in \mathcal{S}[\mathcal{U}]$ the restriction of $s$ to the variables in $\mathcal{U}$: precisely, $s[\mathcal{U}]$ is defined by $s[\mathcal{U}](x) = s(x)$ for all $x \in \mathcal{U}$. For any two set $\mathcal{V}, \mathcal{U}$ of variables, and states $s \in \mathcal{S}[\mathcal{V}]$ and $t \in \mathcal{S}[\mathcal{U}]$, we write $s \simeq t$ if $s(x) = t(x)$ for all shared variables $x \in \mathcal{V} \cap \mathcal{U}$.

**State predicates.** We assume a logical language *Lang* in which assertions about the values of the variables in $\mathcal{W}$ can be written. For example, if all variables are boolean, then *Lang* can be taken to be predicate logic with the addition of the quantifiers $\forall$ and $\exists$ over the booleans. We say that a formula $\phi \in \mathit{Lang}$ is *over* a set $\mathcal{V}$ of variables if it only involves variables of $\mathcal{V}$; such a formula is also called a *predicate* over $\mathcal{V}$. We denote by $\mathit{Preds}[\mathcal{V}]$ the set of all formulas over the set of variable $\mathcal{V}$. Given a formula $\phi$ over $\mathcal{V}$ and a state $s \in \mathcal{S}[\mathcal{V}]$, we write $s \models \phi$ to denote the fact that $\phi$ is true under the interpretation that assigns to every variable $x \in \mathcal{V}$ the value $s(x)$. In particular, a formula $\phi$ over $\mathcal{V}$ defines the set of states $[\![\phi]\!]_{\mathcal{V}} = \{s \in \mathcal{S}[\mathcal{V}] \mid s \models \phi\}$.

**Example 1** Consider the set of boolean variables $\mathcal{V} = \{x, y, z\}$. The set $\mathcal{S}[\mathcal{V}]$ consists of $2^3 = 8$ elements. If we take *Lang* to be propositional logic, then the formula $x \wedge \neg y$ is satisfied by the two states $(x = \text{T}, y = \text{F}, z = \text{F}), (x = \text{T}, y = \text{F}, z = \text{T}) \in \mathcal{S}[\mathcal{V}]$. If we take *Lang* to be quantified boolean formulas, then the formula $\exists w\,.\,(w \equiv x \wedge w \equiv \neg y \wedge w \equiv z)$ is satisfied by the two states $(x = \text{T}, y = \text{F}, z = \text{T}), (x = \text{F}, y = \text{T}, z = \text{F}) \in \mathcal{S}[\mathcal{V}]$. ∎

**Transition predicates.** In order to be able to define *relations,* in addition to sets of states, we introduce the following notation. For each state variable $x$, we introduce a new variable $\bigcirc x$ (read: "next $x$"), with $\mathcal{D}(x) = \mathcal{D}(\bigcirc x)$, that denotes the value of the state variable $x$ in the successor state. Given a set $\mathcal{V} \subseteq \mathcal{W}$ of variables, we let $\bigcirc\mathcal{V} = \{\bigcirc x \mid x \in \mathcal{V}\}$ be the corresponding set of next variables. We denote the converse of $\bigcirc$ by $\ominus$ (read: "previous"): precisely, we let $\ominus\bigcirc x = x$ for all variables $x$. Given a predicate $\phi$, we denote by $\bigcirc\phi$ the result of replacing every variable $x$ in $\phi$ with $\bigcirc x$, and by $\ominus\phi$ the result of replacing every $\bigcirc x$ in $\phi$ with $x$; obviously, $\ominus\bigcirc\phi = \phi$.

Intuitively, in a transition predicate the standard variables refer to the current state, and the next variables refer to the successor state. Given a predicate $\rho$ over $\mathcal{V} \cup \bigcirc\mathcal{U}$, and states $s \in \mathcal{S}[\mathcal{V}]$ and $t \in \mathcal{S}[\mathcal{U}]$, we write $(s, t) \models \rho$

to denote the fact that $\rho$ is true when every $x \in \mathcal{V}$ has value $s(x)$, and every $\bigcirc y \in \bigcirc\mathcal{U}$ has value $t(y)$. A transition predicate $\rho \in \mathit{Preds}[\mathcal{V}, \neq \mathcal{U}]$ defines a relation

$$[\![\rho]\!]_{\mathcal{V},\bigcirc\mathcal{U}} = \{(s,t) \in \mathcal{S}[\mathcal{V}] \times \mathcal{S}[\mathcal{U}] \mid (s,t) \models \rho\}.$$

**Example 2** Consider the set of boolean variables $\mathcal{V} = \{x, y\}$. The transition predicate $(\bigcirc x \equiv y) \wedge \neg \bigcirc y$ defines the transition that copies the value of $y$ into $x$, and sets $y$ to F. ∎

**Modules.** Modules are defined as follows.

**Definition 1 (module)** A *module* (TM) $M = \langle \mathcal{V}^i_M, \mathcal{V}^o_M, {}_M, \theta^i_M, \theta^o_M, \tau^i_M, \tau^o_M \rangle$ consists of the following elements:

- A set $\mathcal{V}^i_M$ of *input variables,* and a set $\mathcal{V}^o_M$ of *output variables.* The two sets must be disjoint: $\mathcal{V}^i_M \cap \mathcal{V}^o_M = \emptyset$. We indicate by $\mathcal{V}_M = \mathcal{V}^i_M \cup \mathcal{V}^o_M$ the set of all state variables of $M$.

- A set ${}_M$ of *reserved variables,* such that $\mathcal{V}^i_M \cup \mathcal{V}^o_M \subseteq_M$. The set ${}_M$ contains variables that are reserved for use by the module, and constitute the module *name space.*

- A predicate $\theta^i_M \in \mathit{Preds}[\mathcal{V}^i_M]$ defining the legal initial values for the input variables.

- A predicate $\theta^o_M \in \mathit{Preds}[\mathcal{V}^o_M]$ defining the initial values of the output variables.

- An *input transition predicate* $\tau^i_M \in \mathit{Preds}[\mathcal{V}_M \cup \bigcirc\mathcal{V}^i_M]$, such that for all $s \in \mathcal{S}[\mathcal{V}_M]$, there is some $t \in \mathcal{S}[\mathcal{V}^i_M]$ such that $(s,t) \models \tau^i_M$. The predicate $\tau^i_M$ specifies what are the legal value changes for the input variables.

- An *output transition predicate* $\tau^o_M \in \mathit{Preds}[\mathcal{V}_M \cup \bigcirc\mathcal{V}^o_M]$, such that for all $s \in \mathcal{S}[\mathcal{V}_M]$, there is some $t \in \mathcal{S}[\mathcal{V}^o_M]$ such that $(s,t) \models \tau^o_M$. The predicate $\tau^o_M$ specifies how the module can update the values of the output variables. ∎

Associated with a module is a set of initial states, a transition relation, and a language. The set of initial states consists of the states that correspond to both possible initial values for the output variables, and legal initial values for the input variables. The transition relation consists of the state transitions that are both possible for the output variables, and legal for the input variables. The language of a module consists of all the possible infinite sequences of states that satisfy the initial conditions and the transition relations.

**Definition 2 (set of [initial] states, transition relation, trace, and language)** Consider a module $M = \langle \mathcal{V}^i_M, \mathcal{V}^o_M, _M, \theta^i_M, \theta^o_M, \tau^i_M, \tau^o_M \rangle$.

- The set of *states* of $M$ is $S_M = \mathcal{S}[\mathcal{V}_M]$.
- The set of *initial states* of $M$ is $I_M = \{s \in S_M \mid s \models \theta^i_M \wedge \theta^o_M\}$.
- The *transition relation* of $M$ is $R_M = \{(s,t) \in S_M \times S_M \mid (s,t) \models \tau^i_M \wedge \tau^o_M\}$.
- A *path* of $M$ from $s \in S_M$ is an infinite sequence $s = s_0, s_1, s_2, \ldots$ of $S_M$ such that $(s_k, s_{k+1}) \in R_M$ for all $k > 0$.
- A *trace* of $M$ is a path $s_0, s_1, s_2, \ldots$ such that $s_0 \in I_M$.
- The *language* of $M$ is the set $\mathcal{L}(M)$ consisting of all traces of $M$. ∎

The requirement on $\tau^i_M$ and $\tau^o_M$ ensures that every state in $S_M$ has a successor that satisfies both the input and output transition relations, ensuring that from every state, there is a transition that is both possible for the module, and legal for the environment. Note that if $\theta^i_M \wedge \theta^o_M$ is unsatisfiable, then $\mathcal{L}(M) = \emptyset$.

Transition modules are an example of *Moore* modules, in which the next value of the output and internal variables can depend on the current state, but not on the next value of the input variables. For instance, if the state variables of a module $M$ are $x$ (input) and $y$ (output), then in a transition from $s \in S_M$ to $t \in S_M$ the next value $t(y)$ of $y$ can depend on the old values $s(x)$ and $s(y)$, but not on the new value $t(x)$ of the input variable.

## 1.2 Closed and Universal Modules

### 1.2.1 Closed modules

A *closed* transition module is a module that has no input variables: hence, a closed module cannot be influenced by its environment.

**Definition 3 (closed transition module)** A transition module $M = \langle \mathcal{V}^i_M, \mathcal{V}^o_M, _M, \theta^i_M, \theta^o_M, \tau^i_M, \tau^o_M \rangle$ is *closed* if $\mathcal{V}^i_M = \emptyset$. ∎

Note that, if a module $M$ is closed, then the predicates $\theta^i_M$ and $\tau^i_M$ must be equivalent to T. why? Hence, we can specify a closed module $M$ simply as $M = \langle \mathcal{V}^o_M,_M, \theta_M, \tau_M \rangle$, where $\theta_M = \theta^o_M$ and $\tau_M = \tau^o_M$.

**Closure of an open module.** The *universal closure* of an open module consists in the closed module obtained by considering an open module in its *most general environment,* that allows any update of input variables consistent with the environment transition of the module.

**Definition 4 (closure)** Given an open module $M = \langle \mathcal{V}^i_M, \mathcal{V}^o_M,_M, \theta^i_M, \theta^o_M, \tau^i_M, \tau^o_M \rangle$, its *closure* $\mathit{Close}(M)$ is the module defined by $\mathit{Close}(M) = \langle \mathcal{V}^i_M \cup \mathcal{V}^o_M,_M, \theta^i_M \wedge \theta^o_M, \tau^i_M \wedge \tau^o_M \rangle$. ∎

### 1.2.2 Universal modules

A module is universal if it does not make any assumption about the inptu behavior.

**Definition 5 (universal module)** A module $M = \langle \mathcal{V}^i_M, \mathcal{V}^o_M,_M, \theta^i_M, \theta^o_M, \tau^i_M, \tau^o_M \rangle$ is *universal* if $\theta^i_M = \mathrm{T}$ and $\tau^i_M = \mathrm{T}$. ∎

## 1.3 Bibliographic Notes

The description of both module behavior and environment assumptions Interface modules describe both the module behavior and the environment assumptions. In this respect, interface modules are related to *trace theory* [?], as well as to the models for interface specification and compatibility checking of [?, ?]. Indeed, interface modules are a variable-based version of the *stateful assume/guarantee interfaces* of [?]. Universal modules do not specify environment assumptions, and are similar to the classical models used in model-checking [?, ?, ?].

In the definition of modules, we have followed the classical approach of describing a system by specifying its set of state variables, and by defining its state space and state transitions explicitly [?]. Another style of modeling takes as primary element the transitions, often called *actions,* rather than the states. Among the modeling languages that are action-based, we recall *process algebras* [?, ?, ?, ?] and *I/O Automata* [?, ?]. The duality in the modeling of component behavior and input behavior

Interface modules are an instance of *Moore* modules. In many cases it is useful to allow more general dependencies between the updated values of the variables: for example, in the modeling of synchronous hardware, these dependencies can be used to encode *Mealy* machines. Models for synchronous systems that capture increasingly general communication patterns, including Mealy machines, have been presented in [**?**, **?**, **?**].