9 A SPECIFICATION APPROACH

Thus the theory of description matters most.
Wallace Stevens,
Description Without Place,
1947.

It is written in the language of mathematics,
and its characters are triangles, circles, and other geometrical figures,
without which it is humanly impossible to understand it;
without these, one is wandering about in a dark labyrinth.
Galileo Galilei,
Il Saggiatore,
1623.

9.1. INTRODUCTION

In order to speak precisely of the security attributes of a system, we must develop precise notions of what ''security'' and a ''system'' really are. This section presents a system model that we believe to be appropriate for examining questions of security, and for dealing with issues of composition. This model may be applied to systems at any level of abstraction, which means that we can examine security issues of widely varying form in the context of systems defined at very abstract or very concrete levels.

Having a model and security criteria that are ''portable'' across a wide range of abstraction ensures that our results can be applied to high-level security objectives, circuit-level building blocks, and all descriptions in between. Therefore we make no real distinction here between the notions of *specification* and *implementation* , except that we recognize that the former are generally less detailed than the latter. For example, a subroutine coded in a programming language, usually considered an implementation to fit some more abstract purpose, can be seen as a specification that a compiler must adhere to when generating code in a lower-level language.

9.2. SYSTEM MODEL

We begin by describing the formalisms used to model systems and their characteristics. The following definitions are largely from [Abadi88] and

[Abadi90b]; however, some of our definitions and notations differ, since neither of these reports addresses security concerns.

The systems we are modeling here are nondeterministic state machines. They are described by means of *behaviours*, which are individual system histories, each representing one possible execution of a system. These behaviours consist of an *initial state*, representing the state of the system at its creation, and an infinite number of *behaviour steps*, which describe the transitions of the system from state to state. Each step not only describes the state reached, but also the *agent* responsible for causing the step. Identifying the agent at each step allows the development of models representing the concurrent operation of multiple systems, each agent representing the actions of an individual system. A behaviour thus records the sequence of states traversed by a system execution, as well as the agent responsible for each transition.

Inputs and outputs are encoded in the state at each step; if a particular state variable changes value over a step, this can be regarded as an output by that step's agent, while other agents may consider the state variable change to be an input. A step need not result in the change of system state, however. Such a step is referred to as a *stuttering step*, in which case we may leave the agent responsible for this step undefined. We say that two behaviours are *stuttering-equivalent* if removing all stuttering steps from each leaves identical behaviours.

A system is modeled by the set of behaviours that describe each possible complete execution of the system. We make the additional requirement that this set is closed under stuttering, which means that, for each behaviour in the set, all behaviours stuttering-equivalent to that behaviour are also in the set. A stuttering-closed set is referred to as a *property*. Since an infinite number of behaviours are stuttering-equivalent to any behaviour, properties contain an infinite number of behaviours.

We are interested here in the design of components of concurrent systems. This rules out the establishment of any finite state set in which our machines are to operate, since the addition of a new system to the composite can introduce arbitrary new state variables. Therefore we assume an infinite state space, though each property is defined in terms of some subset of this space. Similarly, we assume an infinite agent space, since we cannot put an upper bound on the number of components in a concurrent system. Each property then describes the actions of a subset of the infinite agent space.

This formalism models systems by clocked state machines, and simulates concurrency by the interleaving of steps from different systems. To allow flexibility in implementing specifications in this formalism, we generally allow at most one state variable to change value per behaviour step. We assume this in the discussion below, and justify it by arguing that changing multiple state variables per step is a form of true concurrency, which we are replacing by interleaving.

9.2.1. Notation

Behaviours will be represented by lower-case Greek letters. Given some behaviour σ, we denote by σ_i the i^{th} behaviour step. The notation $\sigma_{i \to j}$ denotes a *behaviour segment*, the subsequence of σ from the i^{th} through j^{th} steps, inclusive. Note that $\sigma_{i \to j}$ is not a behaviour unless j is infinite; removing the first i steps from a behaviour σ leaves the behaviour $\sigma_{i \to \infty}$. The notation σ can then be seen as an abbreviation for $\sigma_{0 \to \infty}$.

The agent portion of a behaviour step may be extracted by $\mathbf{a}(\sigma_i)$. This operator can also be applied to behaviours and behaviour segments to yield a sequence of agents: for example, the sequence of agents responsible for behaviour σ is represented by $\mathbf{a}(\sigma)$. If σ_i is a stuttering step, we may write $\mathbf{a}(\sigma_i) = \perp$. We refer to the agent set which behaves according to property P as μ^P; the remainder of the infinite agent set, whose actions are not restricted by P, is denoted $\neg \mu^P$.

Similarly, the state portion of a behaviour step may be extracted by $\mathbf{s}(\sigma_i)$. The notation $\mathbf{s}(\sigma_{i \to j})$ represents the sequence of states in the behaviour segment $\sigma_{i \to j}$, and the initial system state is $\mathbf{s}(\sigma_0)$.

The notation $\underline{\sigma}$ represents the behaviour σ with all stuttering steps removed. The notation $\sigma^{\wedge}_{i \to j}$ represents the behaviour formed by appending an infinite number of stuttering steps to the tail of behaviour segment $\sigma_{i \to j}$. Thus $\underline{\sigma}$ may or may not be infinite in length, depending on whether or not it contains an infinite number of non-stuttering steps. On the other hand, $\sigma^{\wedge}_{i \to j}$ is always infinite in length.

The behaviour $\sigma^{\wedge}_{0 \to j}$ will be referred to as a *stutter-prefix*; a behaviour prefix converted into a behaviour by appending an infinite stuttering suffix. Intuitively, a behaviour with an infinite number of stuttering steps at its tail represents a machine that has halted.

9.2.2. The Logical Form of Properties

We have attached a precise meaning to the words *behaviour* and *property* for the purposes of this investigation. This is the same terminology as in [Abadi90b]. It is important to remember that properties are only sets of behaviours of nondeterministic state machines. These sets are defined by predicates on the universe set of behaviours, U.

In practice, temporal logic is used in the definition of properties. This allows behaviours to be included or excluded from properties based on relations among values of state components, and the identities of agents, at different points in the system lifetime. Writing concise, correct formulas for even simple properties is surprisingly subtle and tricky. We will describe here a notation that is derived from Lamport's ''Temporal Logic of Actions''

[Lamport83] [Lamport89] [Lamport90], but modified for reasoning about composition, and to suit our definitions above. Much of the formalism to follow has appeared in [Abadi90a]; we adopt a less rigourous approach and make a few alterations.

We have described properties as sets of behaviours. When a property is expressed as a logical formula, this set-membership relation corresponds to the *satisfaction* or *modeling* of the formula by a behaviour. We can define the *satisfaction relation* between behaviours and arbitrary properties inductively via a set of rules describing the meaning of formulas in their various forms. More detail on this style of semantic description can be found in [Abadi90a].

The simplest formula is the *state predicate* , which consists of relationships among state variables that must hold at some step in the behaviour. A behaviour satisfies (models) a state predicate property if the state predicate holds in the initial state of the behaviour. This is written:

$$\sigma \models P \triangleq \mathbf{s}(\sigma_0) \models P,$$

where P is a state predicate, and the symbol $\models$ can be read as ''models'', or ''satisfies''. Thus a property whose formula consists only of a state predicate contains all behaviours whose initial states satisfy that predicate. For example, the property defined by:

$$x=0 \wedge y=1$$

consists of all behaviours with state variable x equal to 0 and state variable y equal to 1 initially.

Another basic formula makes use of a modified type of state predicate known as an *action* . An action consists of relationships among *primed* and *unprimed* state variables, where unprimed variables designate variables in one state, and primed variables designate variables in the next. Thus, actions express relationships that must hold among state variables across some step. A behaviour satisfies an action if its first step satisfies the action:

$$\sigma \models A \triangleq (\mathbf{s}(\sigma_0), \mathbf{s}(\sigma_1)) \models A,$$

where A is an action, the values for unprimed variables in A come from $\mathbf{s}(\sigma_0)$, and the values for primed variables in A come from $\mathbf{s}(\sigma_1)$.

Since an action is the work of some agent, we include the agent's identity by using the [ ] operator:

$$\sigma \models [A]_a \triangleq \mathbf{a}(\sigma_1) = a \Rightarrow \sigma \models A.$$

For example, the property defined by

$$[x' = x + 1]_g$$

consists of all behaviours in which state variable x is incremented in the first step by agent g.

We say that a behaviour satisfies the state-predicate *Enabled* (A), for any action A, if the values of the unprimed variables in A hold in the initial state.

For example, *Enabled* $(x = 0 \wedge y' = y + 1)$ holds for all behaviours with $x = 0$ in the initial state.

More general temporal formulas are constructed using the ''henceforth'' operator, defined by:

$$\sigma \models \Box F \triangleq \forall i \geq 0 : \sigma_{i \to \infty} \models F,$$

where F is an arbitrary formula, of the kind we are now describing. Intuitively, $\Box F$ means that the formula F must hold for all behaviour suffixes. For example, the property defined by

$$\Box (x = y)$$

consists of all behaviours in which state variables x and y have the same value in every state, and

$$\Box [z' = x - y]_r$$

consists of all behaviours in which every step by agent r sets state