2007-05-02

Noninterference in Concurrent Game Structures

Piotr Mardziel
Worcester Polytechnic Institute

NONINTERFERENCE IN CONCURRENT GAME STRUCTURES

By

Piotr Mardziel

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in Computer Science

May 2007

APPROVED:

Professor Daniel Dougherty, Thesis Advisor

Professor Micha Hofri, Reader

Professor Michael Gennert, Head of Department

Abstract

Noninterference is a technique to formally capture the intuitive notion of information flow in the context of security. Information does not flow from one agent to another if the actions of the first have no impact on the future observations of the second. Various formulations of this notion have been proposed based on state machines and the removal of actions from action sequences. A new model known as the concurrent game structure [CGS] has recently been introduced for analysis multi-agent systems. We propose an alternate formulation of noninterference defined for systems modeled by CGS's and analyze the impact of the new approach on noninterference research based on existing definitions.

Acknowledgments

We would like to thank and recognize Professor Dan Dougherty, for his continued support, guidance, and dedication throughout the completion of this thesis. I would also like to thank Professor Micha Hofri for his suggestions and help during the polishing of this document.

Contents

List of Figures

List of Definitions

List of Examples

1 Introduction

Confidentiality

The study of security is composed of three main areas: *integrity* deals with establishing trust over data and resources, *availability* refers to the problem of maintaining the ability to use data and resources as desired, and *confidentiality* involves ensuring data and resources are secret or concealed [Bis03]. Confidentiality is becoming more and more important amidst such widespread concerns as identity theft and the increasing prevalence of information and the internet.

Initial work in confidentiality has been done by Bell and LaPadula [DL73, BL76] with the intent for securing military systems where access to information is of prime concern. These works focused on *access control* or techniques intended as means of *enforcing* confidentiality and has been applied to the design of a system by Millen [Mil76]. Access control pervades today's world from security in operating systems to content restrictions on media devices.

Central to the work on the military systems and confidentiality in general is information and access to this information by various entities. The military classification scheme studied by Bell and LaPadua consists of several tiers (see Figure 1) of information and clearances. Access control in such an example would need to ensure, for example, that an entity with secret clearance is allowed to access only the information with secret or lower classification (and thus prevent access to top-secret information). An alternative and perhaps more intuitive description of the confidentiality goals of a system is based on *information flow*, that is, information can flow only from lower classification levels to higher ones and never the other way around. It is noted by McLean [McL85] that access control techniques often fail to capture such intended (or intuitive) notions of security. The problems arise from *covert channels* [RMMG01] or not easily perceptible flows of information that seem to lie beyond the scope of an access control policy.

Noninterference

Noninterference is intended to formally model the intuitive notion of information flow. It has been first formulated by Goguen and Meseguer [GM82], building upon the work of Feiertag, Levitt, and Robinson [FLR77], Feiertag [FLR77], and Rushby [Rus82]. In most basic terms, noninterference states that agent a_1 does not interfere with agent a_2 if and only if no action performed by a_1 will have any effect on what a_2 observes. If agent a_1 does not interfere with a_2 then “no information has leaked” from a_1 to a_2.

Defining noninterference requires an explicit model of a system, agents, and their actions. Approaches derived from Goguen and Meseguer’s initial work define a system in terms of state machines and include provisions for agent actions and outputs to be “observed” after each action. Noninterference in such a model is formulated via runs of the state machine (or sequences of actions) and the *purging* of such runs. Having a run r_1 consisting of a sequence of actions by various agents being modeled, purging the actions of a certain agent results in a run with those certain actions removed. An agent a_1 is then said to not interfere with a_2 if purging a_1’s actions from any run does not change what a_2 observes (see Section 2 for more details).

An *information flow policy* is a specification of allowable information flow between agents as a binary relation between them. The policy based on the intended information flow in a military classification scheme (see Figure 1) as described earlier demonstrates that information from higher levels cannot leak down to lower ones. Such a policy is an example of a *multi-level* policy and has the property that the relation specifying allowable information flow is transitive.

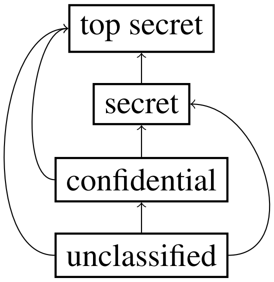


Figure 1: Information Flow in Multi-Level Security

There are cases in which multi-level policies are insufficient. Rushby notes that the treatment of noninterference as described by Goguen and Meseguer in [GM82] is incapable of dealing with polices that are not transitive such as *channel control* (see Figure 2) [Rus81]. The example illustrates a case in which information can flow from *in* to *out* but only if it flows through *encrypt*. Such non-transitivity is initially noted by Goguen and Meseguer in [GM84]. A treatment of the issue is presented by Haigh and Young [HY87] and later by Rushby [Rus92]. The work on channel control has since then been continued by Pinsky [Pin95], Roscoe and Goldsmith [RG99], and Backes and Pfitzmann [BP03].

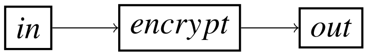


Figure 2: Information Flow in Channel Control

An important part of noninterference formulations is the establishment of an *unwinding theorem*. Such a theorem is intended to provide a means to show a system or model satisfies the restrictions of a noninterference policy purely based on local properties (i.e. transitions in the case of a finite state machine model). The utility of unwinding theorems lies in the fact that noninterference is defined in terms of state machine runs which are considerably harder to deal with than state transitions. Goguen and Meseguer [GM84] present the theorem for their initial noninterference formulation and Haigh and Young [HY87] extend it to channel control policies. Haigh and Young's work was later revised by Rushby [Rus92].

Noninterference and security properties in general are often known to exhibit the *composition paradox*. Two secure (by notion of noninterference) programs, when composed, might result in an insecure program. Various alternate information flow formulations have been proposed to address this issue including forward correct-ability [JT88], restrictiveness [McC90], nondeducibility on strategies [WJ90], and the perfect security property [ZL97].

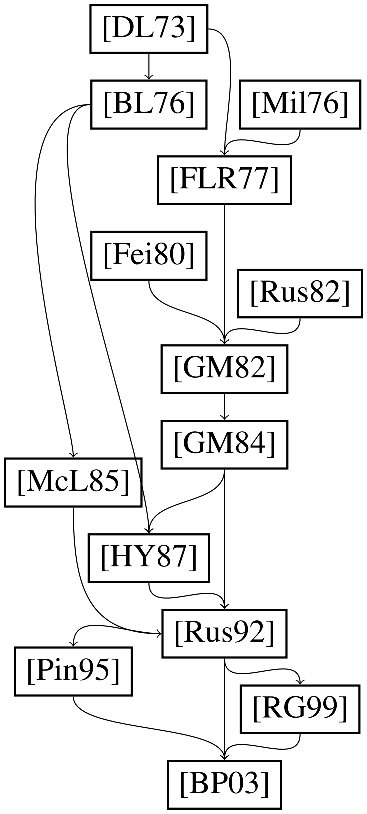


Figure 3: Non-Interference Literature Overview

Concurrent Game Structures

A novel tool for the analysis of open systems has been described by Alur, Henzinger, and Kupferman [AHK98] which explicitly models actions taken by different *agents*. A *concurrent game structure* [CGS] models agents with each contributing to the action at each transition (that is, concurrently) and is central to Alur, Henzinger, and Kupferman's definition of alternating-time temporal logic which they create for use in specification and model-checking of distributed system properties. Concurrent game structures are especially well suited for modeling *open systems* in which the environment and its effects can be modeled by its contribution to transitions of the system along with the contributions of other agents.

We propose a noninterference definition for systems modeled by CGS's with the use of equivalences on runs as opposed to purging of runs. We propose to replace the assertion that the removal of noninterfering actions has no effect with the assertion that two action sequences that are *indistinguishable* by a certain agent have indistinguishable outcomes (see Section 2 for more details). To complete the formulation we also propose an unwinding theorem in this new context and consider the impact of our new formulation on previous investigations of noninterference on non-transitivity.

Motivation

The existing formulations of noninterference that we present in this thesis (that of Goguen & Meseguer and that of Rushby) both establish noninterference as a statement relating the outputs to agents across a pair of runs of state machines (specifically a run and the same run purged of certain actions). There is an overarching intuition over such formulation: agents should not distinguish between observations of a system if they cannot distinguish between runs that got them to those observations. A run and the run purged are examples of action sequences which are meant to be *indistinguishable* for some agent and the outputs on the last states (or upon transitions as in Rushby) are examples of *indistinguishable* observations (that is if they are equal for both runs).

We believe that this is the core of noninterference and the formulation we present in our thesis is based directly on this realization. We will not deal (in general) with purging of runs or outputs at states. Instead we will make use of indistinguishability of runs of a system and indistinguishability of states of that system.

In Section 2.4.3 we will present some issues with the handling of intransitivity in existing noninterference formulations (specifically Rushby's). We believe that the intransitive purge (Rushby's solution to the issues) is a patch for the noninterference formulation based specifically on purge. Purging runs translates to removing various actions performed in those runs which results in a completely different action sequence traversing different states. We believe that this is what leads to the requirement (of Goguen & Meseguer and Rushby formulations of state machines) that transition functions need to be totally defined (for every action at every state). This in turn prevents the systems from concisely modeling situations in which there is some correct sequence of events but more importantly some sequences just cannot happen.

If we allow ourselves to conclude that the cause of the problem is purge we can tackle the issue at its root. That is, we could perhaps sidestep the need for special considerations of intransitive policies if we can define action distinguishability in a nondestructive way, allowing us, in turn, to not require total transition functions, and thus allowing us to model response-like systems (that specify validity of runs) naturally within the state machine itself. This is exactly what we present in this thesis.

2 Background

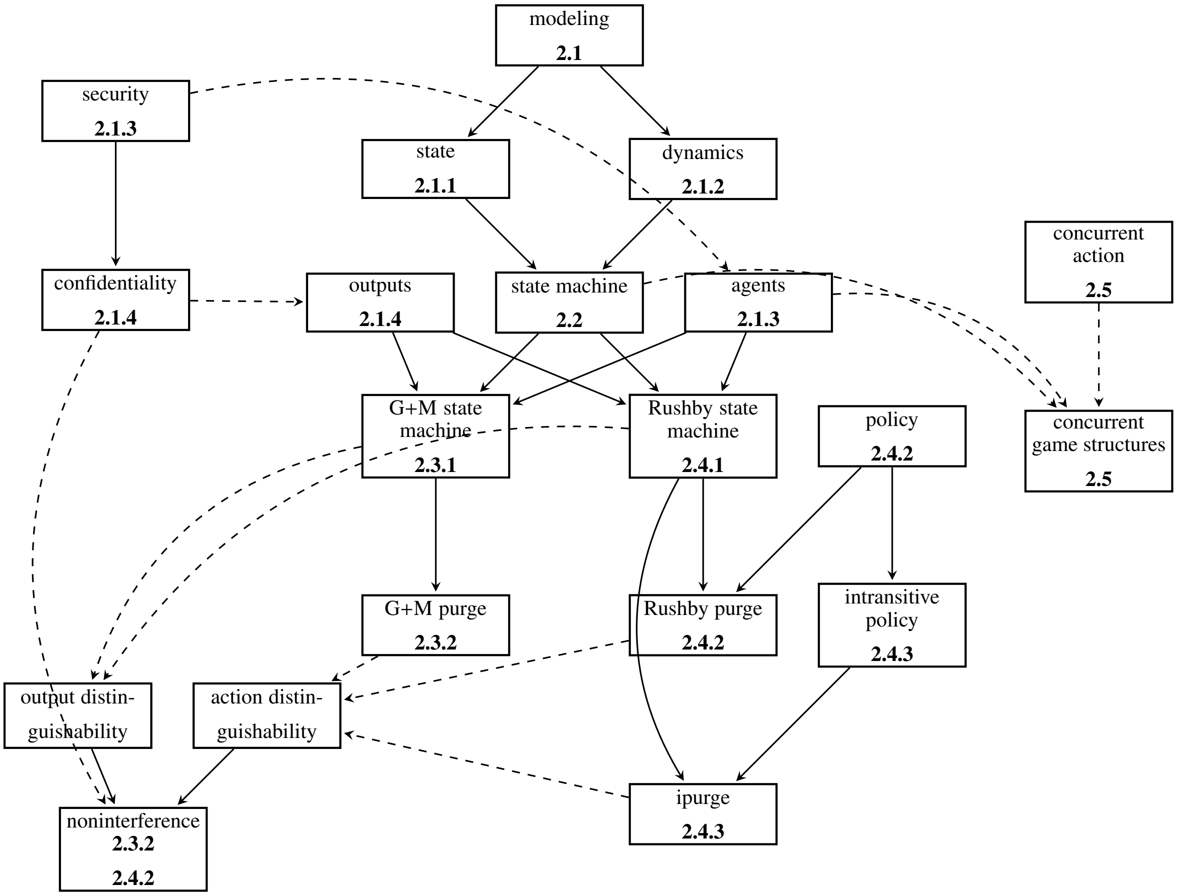


Figure 4: Background Concepts Dependency Graph

We begin with a quick overview of modeling as an introduction to the ideas presented further in this section. To demonstrate the relationship between our work and the originating works on noninterference we present here first the initial noninterference formulation of Goguen & Meseguer and a second, slightly refined, noninterference formulation of Rushby. We also present the concurrent game structure background necessary to introduce our noninterference work and provide some motivation for the use of concurrent game structures.

2.1 Modeling

Rarely is one capable of reasoning about real world systems in their full detail and intricate complexity. Even if it were possible we are most often interested in only a fraction of the system. We thus create models and factor in only the details that we need to reason about to make needed conclusions. As the first ingredient of the model, a sufficient description of the system or its state is required. If our system changes over time or we need to reason about such changes, we need to model the change as well. In many modeling tasks we also need to consider entities that are responsible for the changes in the system and finally when reasoning about issues of confidentiality, it is necessary to model visibility of information to the acting entities.

2.1.1 State

Every bit of information necessary to reason about a system comprises a state. If we are modeling a simple algorithm, we might consider a set of variable=value pairs as the state. Sometimes the intent of our model lets us abstract away facts that are not especially necessary to the questions we want to ask of the model. For example if we only care whether a certain integer variable is odd or even, we have no need to consider its value in the state but rather whether it is odd or even. Reasoning about systems in this manner has been widely studied in the field of *abstract interpretation* [CC77]. This basic notion is present in every single model described in this section.

2.1.2 Dynamics

If we need to reason about changes of the system, we must model this change. Such changes are referred to as *transitions* and are (in all models in this document) are given labels.

The state and dynamics modeled in this thesis are represented by *state machines*. Noninterference, however, require is to consider agents and outputs. The basics of the state machine are carried over to the more structured models described further in this section.

2.1.3 Agents

In the area of security, almost every issue depends on what is allowable (or observable) for a certain entity and not for another. What use of security is there if the world exhibits only a single entity? Thus we consider *agents*. If those agents contribute to the transitions of the model we also model the association between the transitions and the agents. In the model of Goguen & Meseguer (see Section 2.3.1), agents are provided in the form of a set of *users*. The associated between state transitions is made by attaching a user to each transition. In the model of Rushby, agents are represented in the form of a set of *security domains* with each domain being associated with a set of transition labels.

2.1.4 Outputs

The issue of confidentiality (of which noninterference is an example) deals centrally with what an agent sees. To aid in such reasoning about the observations made by agents, we model the *outputs* to agents. Although the issue here is what an agent observes and not necessarily what is output, we adapt the terminology of *outputs* as it is widely used in the noninterference literature. The representation of outputs varies. In the model of Goguen & Meseguer (see Section 2.3.1), outputs are provided to each agent at each state. In the model of Rushby (see Section 2.3.1), the outputs are provided at transitions only to the agent performing the transition. In our own noninterference formulations output is described by distinguishability relations.

Example 1 (Bird-song Propagation). We wish to model the propagation of a bird song among several (or three in our simple example) birds. We theorize that the nature of the propagation is of the following manner:

- Each bird except one merely copies what it hears from one other bird. Such birds are attuned to the songs of their one companion who they copy and hear no one else. The one exceptional bird is the one exhibiting the creative capacity to sing the original bird song.
- The mimicking birds are sometimes impatient and will repeat notes of the song

they hear without having heard new notes but they will never repeat a note that differs from the one they last heard. Also, each bird initially assumes having heard a dull note if they have not heard anything yet.

We derive elements of a simple model:

- We will call the creative bird, a and the mimicking birds as b and c. We assume that b is attuned to a's singing and c is attuned to b's. These comprise our agents.
- We will consider only two bird song notes, 0 and 1 where 0 refers to the dull note. We will also label the act of bird x singing note n as x_n. Thus bird b singing note 1 is represented by b_1.
- We will represent our state as XY where $X, Y \in \{0, 1\}$. The X represents a's last sung note and Y represents b's. We do not care what c sings since no one is listening to him.

The dynamics of the model are of the following nature:

- Bird a sings an original song which is exhibited by actions a_0 or a_1. If the world is on state XY ($X, Y \in \{0, 1\}$) the action a_x causes the world to transition to state xY.
- Bird b sings the song he hears from a which is represented by the X component of state XY. If the world is in state XY, the action b_y can only occur if $X = y$ and causes the world to transition to state Xy.
- Bird c sings the song he hears from b which is represented by the Y component of state XY. If the world is in state XY, the action c_z can only occur if $Y = z$ and does not cause the world to change state.

The idea of an output is exhibited in the example not only in the state but can also be clearly seen in the original description of the situation which we are modeling. That is, each bird *hears* the notes of one other bird. Such an output is indirectly present in the state and dynamics. We see that bird b *hears* the X component of state XY and is only capable of singing a note if it has heard it recently from a.

This example is elaborated further in the various models throughout this section.

2.2 State Machines

The nature of our work is based on the dynamic progression of a system modeled in a sufficiently simple manner for manipulation and reasoning. The models used throughout this thesis all derive from the very basic state machines consisting of states and transitions (dynamics). We describe these here.

Definition 2 (State Machine). *A* state machine $S = \langle Q, q_0, \Sigma, \delta \rangle$ *is composed of*

- *A set of* states Q *and an* initial state $q_0 \in Q$.
- *A set of* actions *(or alphabet)* Σ.
- *A* transition function $\delta : Q \times \Sigma \rightarrow Q$.

The dynamics of the machine involve starting at some initial state (q_0) and making transitions to other states as specified by the transition function δ.

Definition 3 (State Machine Run). *A run r in system S is a (valid) finite sequence actions. That is $r \in R(S) \subseteq \Sigma^*$ where $R(S)$ is the set of runs of state machine S. Note that although a run technically composed only of actions, we can reason about states at various points in the run. We assume that the actions are applied at successive states starting with q_0 following transitions defined by the δ function. We will often write runs in the form $q_0, a_0, q_1, ..., a_{n-1}, q_n$ where q_i are states and a_i are actions with the property that $\delta q_i, a_i = q_{i+1}$. Note that only valid runs can be written in this way whereas arbitrary action sequences cannot necessarily satisfy the condition $\delta q_i, a_i = q_{i+1}$. We will write $\lceil r \rceil$ as the last state reached by the run r.*

Example 4 (Bird-song Propagation as a State Machine). Consider a state machine S for the Bird-song Propagation model (as in Example 1) as follows.

$$Q = \{00, 01, 10, 11\}$$

The states of our system correspond to the last notes sung by birds a and b. That is, state XY represents a state in which a's last note was X and b's last note was Y.

$$q_0 = 00$$

The initial state corresponds to the "initial recollection" of the dull note by every bird as described in Example 1.

$$\Sigma = \{a_0, a_1, b_0, b_1, c_0, c_1\}$$

Our alphabet of transitions correspond exactly to the bird-song notes. b_0 represents bird b singing note 0. Note, however, that agents (or birds) are not modeled here. We have to wait for a more structure representation to handle the agents (as in the model of Goguen & Meseguer in Section 2.3.1).

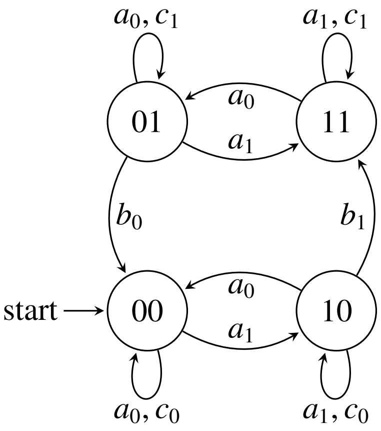


Figure 5: State Machine Example

And δ is defined according to Figure 5. Circles in our figure denote states. An arrow labeled with "start" points to the starting state q_0. An arrow from state q to state q' labeled with actions $s_1, ..., s_n$ defines $\delta(q, s_i) = q'$ for all $i \in \{1, ..., n\}$.

We note that not every action sequence in Σ^* is a run, or specifically $R(S) \neq \Sigma^*$. In our example we see that a_1, b_1 is a run but a_0, b_1 is not. We could write the first of the runs as $00, a_1, 10, b_1, 11$ but for the second run, we see that there is no transition labeled b_1 from state 00 which is reached after taking action a_0. A high-level characterization of the runs of our example would be of the form: b_0 can be taken only if the last a_i action was a_0 and similarly b_1 can only be taken if the last a_i action was a_1 – a restatement of the Bird-song Propagation dynamics.

2.3 Goguen & Meseguer

The noninterference formalization as described by Goguen & Meseguer makes use of more structure placed upon the standard state machine. Specifically, to reason about noninterference it is first necessary to include some representation of agents or users along with an association between actions (transitions) and those agents. We describe these additions first and then describe the noninterference definition.

2.3.1 State Machine

Definition 5 (Goguen & Meseguer's State Machine). *A* state machine $S = \langle U, S, C, Out, out, do, s_0 \rangle$ *is composed of*

- *A set of* users U.
- *A set of* states S.
- *A set of* state commands C.
- *A set of* outputs Out.
- *An* output function $out : S \times U \to Out$.
- *A (total)* do function $do : S \times U \times C \to S$.
- *An initial state* s_0.

The definition describes a fairly standard notion of a state machine with some additional structure. We have transitions composed of two portions (a user and a command) instead of the usual generalized transition which provide a means of associating users with transitions. We could reduce the such transitions by setting Σ in our simple state machine to be equivalent to $U \times C$. The transition function is renamed to *do* (from the original δ). Also present is an output function necessary for formulation of distinguishability in noninterference. Finally the transition function is defined completely, that is, for every state and every user/command pair. This makes the definition of a run less involved.

Definition 6 (Goguen & Meseguer' State Machine Run). *A run r in system S is a finite sequence of user and state command pairs. That is $r \in R(S) = (U \times C)^*$. We* *assume that the user/command pairs are applied at successive states starting with* s_0 *following transitions defined by the do function. We will write* $\lceil r \rceil_u$ *as the output to user u at the last state of run r or specifically* $\lceil r \rceil_u = output(\lceil r \rceil, u)$*.*

Example 7 (Bird-song Propagation as a Goguen & Meseguer State Machine). Consider a state machine S as follows.

$$\begin{array}{rcl}
U & = & \{a, b, c\} \\
S & = & \{00, 01, 10, 11, \epsilon\} \\
s_0 & = & 00 \\
C & = & \{0, 1\} \\
Out & = & \{0, 1, \epsilon\} \\
output: \ \langle xy, a \rangle & \mapsto & 0 \ \text{ for every } x, y \in \{0, 1\} \\
\langle xy, b \rangle & \mapsto & x \ \text{ for every } x, y \in \{0, 1\} \\
\langle xy, c \rangle & \mapsto & y \ \text{ for every } x, y \in \{0, 1\} \\
\langle \epsilon, u \rangle & \mapsto & \epsilon \ \text{ for every user } u
\end{array}$$

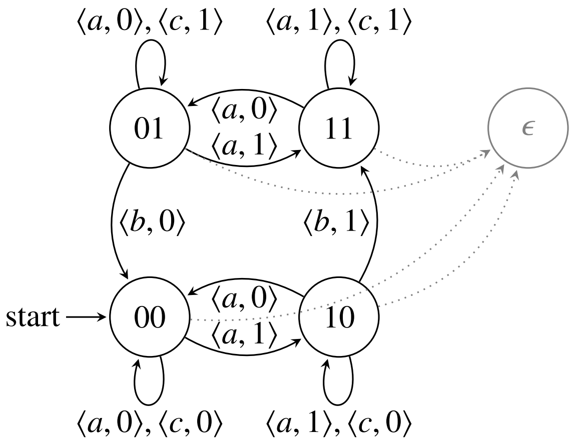


Figure 6: Goguen & Meseguer State Machine Example

And *do* is defined according to Figure 6. The gray dotted arrow denotes all the transitions that are necessary to make *do* total. Also, the x denotes any command.

Having the additional user and output structure in place we can provide a more satisfying interpretation of the example. That is we see that user a performs either 0 or 1 and user b observes (via output) a's last action and repeats it. We could say no such things in the basic state machine system we described in Example 4. User c observes what b performed last. Any behavior outside this interpretation leads to the special "error" state ϵ which exists merely to make *do* total.

The fact that *do* must be defined for every state, action, and user prevents us from defining the validity of runs directly. We have, as an alternative, defined a special error state ϵ and let every action that lies outside of the specification of the Bird-song Propagation example to transition the system to the error state.

2.3.2 Noninterference

The Goguen & Meseguer definition of noninterference is based on *purging* commands from runs.

Definition 8 (Goguen & Meseguer' Purge). *Given a run r, $p_G(r)$ is a run with the commands of agents in G* purged*. Specifically,*

$$\begin{array}{rcl} p_G(\epsilon) & = & \epsilon \\ p_G(\langle u, c\rangle.r) & = & \begin{cases} \langle u, c\rangle.p_G(r) & \textit{if } u \in G \\ p_G(r) & \textit{otherwise.} \end{cases} \end{array}$$

For example if we have a run $r = \langle a, c_1\rangle, \langle b, c_2\rangle, \langle b, c_3\rangle, \langle c, c_4\rangle, \langle b, c_5\rangle, \langle a, c_6\rangle, ...$ and purge it of a and c's actions, we will have $p_{\{a,c\}}(r) = \langle b, c_2\rangle, \langle b, c_3\rangle, \langle b, c_5\rangle, ...$ (see Figure 7). Note that the resulting run is just as valid as the original since every run is valid in the Goguen & Meseguer state machine model. Also note that there is no correspondence between the states visited in the original run and the purged run except for the first state which is always q_0.

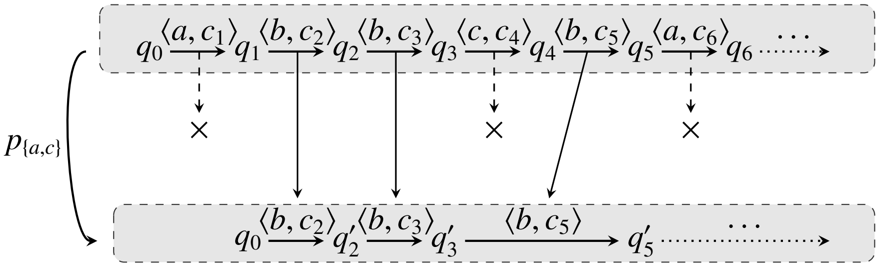


Figure 7: Purge

Definition 9 (Goguen & Meseguer's Noninterference). *A set of users $G \subseteq U$ does not interfere with a set of users $G' \subseteq U$, or $G \not\Rightarrow G'$ iff for every run $r \in (U \times C)^*$ and every user $u \in G'$, we have*

$$\lceil r \rceil_u = \lceil p_G\,(r) \rceil_u$$

If not $G \not\Rightarrow G'$ then G does interfere *with G' or $G \Rightarrow G'$.* [1]

Example 10 (Interference in Bird-song Propagation as a Goguen & Meseguer State Machine). Consider the system described in Example 6. We can make the following interference assertions.

- $\{a\} \Rightarrow \{b\}$ as demonstrated by run $\langle a, 1\rangle, \langle b, 0\rangle$. That is:

  $$p_{\{a\}}\,(\langle a, 1\rangle, \langle b, 0\rangle) = \langle b, 0\rangle$$

  The last state in the original run is ϵ at which the output to b is ϵ. The last state in the purged run is 00 and the output for b there is 0. The outputs do not match and hence the run is sufficient to prove that $\{a\} \Rightarrow \{b\}$.

- $\{a\} \Rightarrow \{c\}$ as demonstrated by run $\langle a, 1\rangle, \langle b, 1\rangle$ which purges to $\langle b, 1\rangle$.

- $\{a\} \Rightarrow \{b, c\}$. This is equivalent to the first two assertions combined.

- $\{b\} \Rightarrow \{c\}$ via run $\langle b, 1\rangle$ which purges to the empty run ϵ.

[1] It may seem unusual to define the lack of interference and from that define its negation, the presence of interference but we note that we the same approach is taken by Goguen & Meseguer.

- $\{a, b\} \Rightarrow \{c\}$ via the same run $\langle b, 1\rangle$.
- $\{c\} \not\Rightarrow \{a, b\}$. We see that all of the actions performed by user c do not change the state of the machine. Removing such actions would therefore have no effect on the output as seen by the other two agents. This also lets us conclude $\{c\} \not\Rightarrow \{a\}$ and $\{c\} \not\Rightarrow \{b\}$.

2.4 Rushby

We present here the formalization derived from Rushby in [Rus92] as it is in many respects cleaner than the original definitions of Goguen & Meseguer. We introduce slight notation differences to allow for an easier transition to the second system definition further in this document. Rushby's formulation is equivalent to that of Goguen & Meseguer [Rus92] but allows for easier formulation of intransitive purge presented further in the section.

2.4.1 State Machine

Definition 11 (Rushby's State Machine). *A* state machine system *is composed of*

- *a set of* states S *and an initial state* $s_0 \in S$*,*
- *a set of* actions *A,*
- *a set of* outputs *O,*
- *a* step function*,* $step : S \times A \to S$*, and*
- *an* output function*,* $output : S \times A \to O$*.*

The actions in this model are inputs from the usual definition of an automata and the step function corresponds to the transition function. The model also includes outputs. An agent that performs a certain action at a certain state observes an output as encoded in the output function. This is in contrast to the Goguen & Meseguer state machines in which the output was observed by each user at each state. In further contrast, note that the users are missing from this definition. These are replaced by security domains later.

The function $run : S \times A^* \to S$ is defined inductively from the step function.

2.4.2 Noninterference

Definition 12 (Security Domain). *We define* security domains *D and associate each action with a domain by a function $dom : A \to D$. The domain here is the equivalent of the user in the Goguen & Meseguer state machine.*

Note that the use of security domains exhibits differences with the use of users as in Goguen & Meseguer. We no longer have actions associated with a user at each transition. We have each action associated with a user (domain) globally (over all transitions).

Definition 13 (Noninterference Policy). *A* noninterference policy *is a reflexive relation on D and denoted by $\rightsquigarrow$. Also, we denote $\not\rightsquigarrow$ as complement of $\rightsquigarrow$ or $(D \times D) - \rightsquigarrow$.*

Policies serve to capture the intuitive notion of information flow. Examples of such policies are discussed in Section 1 and include multi-level security (see Figure 1) and channel control (see Figure 2). The formalism is of course general enough to allow a range of other security needs such as isolation in which agents are not allowed to communicate.

Definition 14 (Rushby's Purge). *Given a security domain $v \in D$ and an action sequence $\alpha \in A^*$, define $p_v(\alpha)$ as the sequence with actions associated with any domain u with $v \not\rightsquigarrow u$ removed. That is,*

$$\begin{array}{rcl} p_v(\epsilon) & = & \epsilon \\ p_v(a.\alpha) & = & \begin{cases} a.p_v(\alpha) & \text{if } dom(a) \rightsquigarrow v \\ p_v(\alpha) & \text{otherwise.} \end{cases} \end{array}$$

Definition 15 (Rushby's Policy Satisfaction). *A system is* secure *for policy $\rightsquigarrow$ (or* satisfies *the policy) if*

$$output(\lceil r \rceil, a) = output(\lceil p_{dom(a)}(r) \rceil, a)$$

Example 16 (Bird-song Propagation as a Rushby State Machine). Consider the state machine defined as follows:

$$
\begin{array}{rcl}
S & = & \{00, 01, 10, 11, \epsilon\} \\
s_0 & = & 00 \\
A & = & \{a_0, a_1, b_0, b_1, c_0, c_1\} \\
O & = & \{0, 1, \epsilon\} \\
output: \ \langle xy, a_z \rangle & \mapsto 0 & \text{for every } x, y, z \in \{0, 1\} \\
\langle xy, b_z \rangle & \mapsto x & \text{for every } x, y, z \in \{0, 1\} \\
\langle xy, c_z \rangle & \mapsto y & \text{for every } x, y, z \in \{0, 1\} \\
\langle \epsilon, x \rangle & \mapsto \epsilon & \text{for every action } x
\end{array}
$$

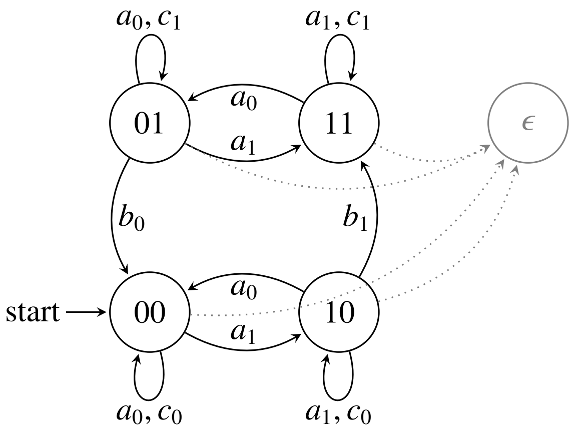


Figure 8: Rushby State Machine Example

And *step* is defined according to Figure 8. Also let us define security domains and a security policy.

$$\begin{aligned}
D &= \{a, b, c\} \\
dom: \quad a_x &\mapsto a \;\text{ for every } x \in \{0, 1\} \\
b_x &\mapsto b \;\text{ for every } x \in \{0, 1\} \\
c_x &\mapsto c \;\text{ for every } x \in \{0, 1\} \\
\rightsquigarrow &= \{(a, b), (b, c)\} \;\text{ (see Figure 9)}
\end{aligned}$$

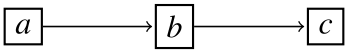


Figure 9: A Noninterference Policy

We can quickly see that our system does not satisfy the policy $\rightsquigarrow$ by presenting run $r = a_1, b_1$. We see $r' = p_c(r) = b_1$. The last state in r is 11 whereas the last state in r' is ϵ. Finally $0 = output(00, x) \neq output(\epsilon, x) = \epsilon$ for every domain x. If we listen to our intuitive perspective on the machine, we might disagree with the verdict. The machine exemplifies a situation in which domain a takes actions 0 or 1 which are "observed" by and repeated by domain b and then observed by domain c. In this regard we could expect the information flow depicted in the policy in Figure 9 to be exemplified by the example system. The issue here is the fact that our example policy was not transitive. Special consideration is given to policies of this form.

2.4.3 Non-transitivity

The policy in Figure 9 is not transitive. That is, $a \not\rightsquigarrow c$ even though $a \Rightarrow b$ and $b \Rightarrow c$. The example was designed to model a situation in which agent b listens to agent a and repeats what it hears. It does not seem to make sense for b to repeat something it did not hear. The specifications of a state machine for use in Rushby's (and Goguen & Meseguer) noninterference requires is to define state transitions for such nonsensical actions. We did so casually by sending the system into an error state by such transitions. Being in an error state, however, was "observable" by every agent and thus agent *a* is capable of interfering with *c* by virtue of not taking actions (ie, not saying 1 before *b* utters a 1 in spite of the situation being modeled).

Such problems are addressed by modifying the purge function. Intuitively we do not purge actions of an agent if they could be allowed to influence another agent via some intermediate agents. The function *sources* is first defined to represent the agents whose actions are not to be deleted at various points in a run.

Definition 17 (Sources). *A function* $sources : A^* \times D \rightarrow P(D)$ *identifies actions in a run which should not be removed by purge.*

$$\begin{array}{rcl} sources(\epsilon, u) & = & \{u\} \\ sources(a.\alpha, u) & = & \begin{cases} sources(\alpha, u) \cup \{dom(a)\} & \textit{if } \exists v \textit{ with } v \in sources(\alpha, u) \textit{ and } dom(a) \rightsquigarrow v \\ sources(\alpha, u) & \textit{otherwise} \end{cases} \end{array}$$

Using this, an alternate definition of purge is provided:

Definition 18 (Rushby's Intransitive Purge). *Given a security domain* $v \in D$ *and an action sequence* $\alpha \in A^*$*, define* $ip_v(\alpha)$ *as the sequence with actions associated with any domain u with u not in the sources for v at the various points in the run removed. That is,*

$$\begin{array}{rcl} ip_v(\epsilon) & = & \epsilon \\ ip_v(a.\alpha) & = & \begin{cases} a.ip_v(\alpha) & \textit{if } dom(a) \in sources(a.\alpha, u) \\ ip_v(\alpha) & \textit{otherwise.} \end{cases} \end{array}$$

With the purge definition for intransitive policies as above we can see that the problematic run from Example 16 is no longer an issue. That is a_1, b_1 purges for c to a_1, b_1 and thus the output to c as a resolution of an action after the run and the purged run are the same. The solution to the in-transitivity presented here is, however, problematic on an intuitive level. The definition of *sources* lets us prevent the purging of actions that *could* have influenced future observation of an agent but not necessarily *did* influence. In our example, as long as *b* performs *any* action after an action *a*, this action will not be purged. What if *b*'s actions are not at all based on *a*'s actions? This problem is especially apparent in the next example:

Example 19 (Non-Transitive Interference Without Intermediate Interference). Consider a Rushby state machine S of the following form:

$$\begin{array}{rcl}
S & = & \{s, 1, 1', 2, 2'\} \\
s_0 & = & s \\
A & = & \{a_1, a_2, b, c\} \\
O & = & \{1, 2, \epsilon\} \\
output: \ \langle x, a_z \rangle & \mapsto & \epsilon \text{ for every } x \in S \text{, and every } z \in \{1, 2\} \\
\langle x, b \rangle & \mapsto & \epsilon \text{ for every } x \in S \\
\langle x, c \rangle & \mapsto & \epsilon \text{ for every } x \in \{s, 1, 2\} \\
\langle 1', c \rangle & \mapsto & 1 \\
\langle 2', c \rangle & \mapsto & 2
\end{array}$$

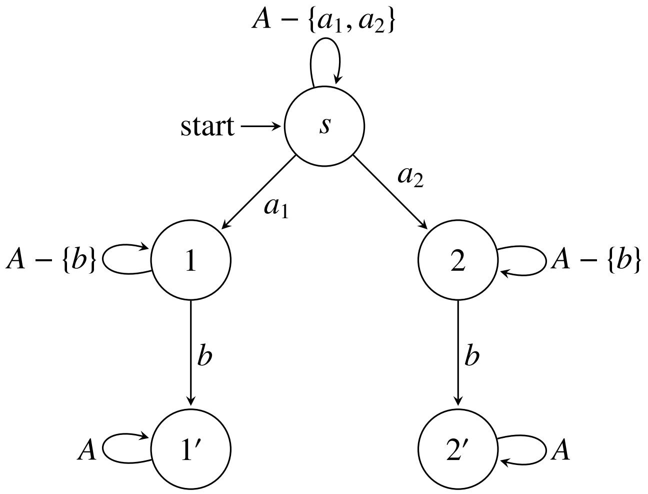
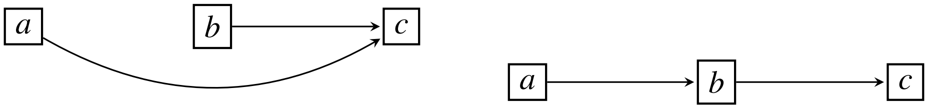


Figure 10: Intransitivity State Machine Example

And *step* is defined according to Figure 10. Also let us define security domains.

$$\begin{aligned} D &= \{a, b, c\} \\ dom: \quad a_x &\mapsto a \ \text{ for every } x \in \{1, 2\} \\ b &\mapsto b \\ c &\mapsto c \end{aligned}$$

The high-level workings of this system are thus: no one receives any distinguishable outputs except domain c and for c to receive different outputs, domain b must first perform an action following an action of domain a. If a does a_1 then b will "enable" c to see this fact after performing action b but no sooner (similarly for a_2).



(a)

(b)

Figure 11: Two Noninterference Policies

We can easily see that both policies in Figure 11 are satisfied by this system but this fact is somewhat unsettling. Firstly, Figure 11a would suggest to us that no information is flowing from a to b (specifically that a does not interfere with b). Yet Figure 11b tells us that information is flowing from a to c through b (or specifically that a interferes with c through b). How can we accept both of these statements at the same time?

2.5 Concurrent Game Structures

Our notion of noninterference is centered on systems modeled as *concurrent game structures*. The definition of a concurrent game structure below is an adaptation from one described in [AHK98].

Definition 20 (Concurrent Game Structure). *A* concurrent game structure *[c.g.s.] is a tuple $S = \langle A, Q, q_0, Act, act, \delta \rangle$:*

- *A set A of agents* [2]*. We write the agents of A as $a_1, a_2, ..., a_k$. We will also often use k to denote the number of agents ($|A|$).*
- *A set Q of* states *and* initial state q_0.
- *A set Act of* actions [3].
- *For each agent $a \in A$ and each state $q \in Q$, a set of possible actions $act(q, a) \subseteq Act$ for that agent at that state. Also, for any state $q \in Q$, a* move vector at q *is a tuple $\begin{bmatrix} i_{a_1} \\ \vdots \\ i_{a_k} \end{bmatrix}$ (sometimes we will write it as $\left[i_{a_1}, \cdots, i_{a_k}\right]$ instead) with $i_{a_j} \in act(q, a_j)$ for every agent $a_j \in A$.*
  - *We denote the set of possible move vectors at state $q \in Q$ as $D(q) = act(q, a_1) \times act(q, a_2) \times \cdots \times act(q, a_k)$.*
- *For any state $q \in Q$ and a move vector $[i_1, ..., i_k] \in D(q)$, the next state resulting from the combined actions of all the agents as represented by the move vector is $\delta(q, [i_1, ..., i_k]) \in Q$.*

Definition 21 (Concurrent Game Structure Run). *A* run *of a c.g.s. S is a finite sequence of move vectors $\hat{i_0}, \hat{i_1}, \cdots, \widehat{i_{n-1}}$. A* valid *run is one in which the move vectors are applied (and thus applicable) to successive states starting with the initial state q_0.*

We will often write valid runs as sequences of move vectors and states of the form $q_0, \hat{i_0}, q_1, \hat{i_1}, \cdots, \widehat{i_{n-1}}, q_n$. The fact that states are present here is merely a book-keeping measure as the information presented by them is completely redundant. The validity of a run written in this way is demonstrated by the fact that $\hat{i_j} \in D(q_j)$ and $q_{j+1} = \delta(q_j, \hat{i_j})$ for all j.

Oftentimes it is cumbersome to specify concurrent game structures and/or their transitions. We use several shorthand notations for transition specification and a cross

[2] This is a slight departure from the original definition of a concurrent game structure in [AHK98] which merely notes the existence of a certain number of agents k. We will require in our later discussion the association between agents in two different c.g.s.'s. For this reason we specify agents as a set.

[3] This is yet another departure from the original definition and is made for the same reason: the need to equate actions of agents at different points in a run or across multiple runs. The original definition merely notes that each agent has a certain number of actions at each state.

product construction for specification of c.g.s.'s. These are described in Appendix B and Appendix C respectively.

2.5.1 State Machines as Concurrent Game Structures

Note that a concurrent game structure is a more elaborate state machine. We can think of a usual state machine (such as the one described for Goguen & Meseguer's noninterference) as a (severely) limited concurrent game structure. Further in our thesis we will demostrate reductions of noninterference and systems specified in Goguen & Meseguer's model to those specified in concurrent game structures and our own noninterference formulation. The first step in the reduction will be the representation of Goguen & Meseguer state machines as concurrent game structures which we demonstrate here. The reductions presented here and further in the thesis include:

- R_0 - From a state machine to a concurrent game structure.
- R_1 - From a Goguen & Meseguer state machine (ignoring outputs) to a concurrent game structure.
- R_1' - From a Goguen & Meseguer state machine to a concurrent interference system (Definition 40).

State Machines without Agents

Remark 22. *A state machine with states and labeled transitions is a concurrent game structure with a single agent. We demonstrate this via a reduction R_0 to restructure state machines into concurrent game structures.*

Definition 23 (State Machine to Concurrent Game Structure Reduction). *Given a state machine $S = \langle Q, q_0, \Sigma, \delta \rangle$, the reduction of $R_0(S)$ is a concurrent game structure $S' = \langle A, Q, q_0, Act, act, \delta' \rangle$ defined as follows:*

- *The set of states Q is the same as in the state machine.*
- *The initial state q_0 is the same as in the state machine.*
- *The set of agents includes only a single agent a. That is $A = \{a\}$.*

- *The set of actions Act is equal to the set of actions* Σ.
- *The set of actions available to a at state q is exactly the set of actions available at q in S. That is,* $act(q, a) = \{c \mid \delta$ *is defined for* $(q, c)\}$.
- *For any state* $q \in Q$ *and action c available at q,* $\delta'(q, [c]) = \delta(q, c)$.

We see that the difference between a state machine and a single-agent concurrent game structure is merely the notation.

Example 24 (State Machine to Concurrent Game Structure Reduction). Let us demonstrate Example 4 as a concurrent game structure via an R_0 reduction.

$$\begin{aligned} A &= \{a\} \\ Q &= \{00, 01, 10, 11\} \\ q_0 &= 00 \\ Act &= \{a_0, a_1, b_0, b_1, c_0, c_1\} \end{aligned}$$

The specification of *act* and δ can be derived from Figure 12.

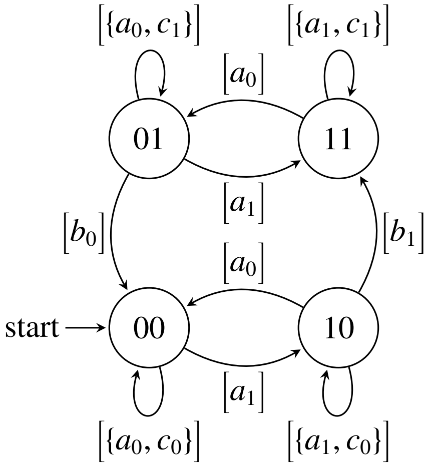


Figure 12: Concurrent Game Structure Version of Example 4

State Machines with Agents

Having state machines with agents (like the Goguen & Meseguer type), we can perform a slightly more involved reduction to concurrent game structures. Additionally, in this case, we need to establish some further correspondence between the original machine and the concurrent game structure to aid us in reasoning about the relationship of noninterference assertions across these two models further in this thesis.

Remark 25. *A state machine with states and transitions labeled with actions and agents is a concurrent game structure with multiple agents but where valid move vectors are those in which only one agent performs an action other than the designated null action. We demonstrate this via reduction R_1.*

Definition 26 (State Machine with Agents to Concurrent Game Structure Reduction). *Given a state machine (of the Goguen & Meseguer type) $S = \langle U, S, C, Out, out, do, s_0 \rangle$, the reduction $R_1(S)$ is a concurrent game structure $S' = \langle A, Q, q_0, Act, act, \delta \rangle$ defined as follows:*

- *The set of agents A is equal to the set of users U.*
- *The set of states Q is equal to set of states S with one additional "null" state which we write as ϵ here.*
- *The initial state q_0 is equal to the initial state s_0.*
- *The set of actions Act is equal to the set of commands C with one additional action ϵ.*
- *All actions in Act are available for each agent in every state. That is, $act(q, a) = Act$ for every $q \in Q$ and every $a \in A$. This assumes that the transition function in the original machine was defined for every state, user, and command.*
- *If $do(q, \langle a_i, c \rangle) = q'$ then $\delta(q, {\scriptstyle [\epsilon,..,c,...,\epsilon]}) = q'$ where ${\scriptstyle [\epsilon,...,c,...,\epsilon]}$ denotes a move vector in which agent a_i (the i^{th} agent) takes action c and every other agent does ϵ. For every other move vector $\hat{\imath}$, we have $\delta(q, \hat{\imath}) = \epsilon$.*

Example 27 (State Machine with Agents to Concurrent Game Structure Reduction). Let us demonstrate Example 7 as a concurrent game structure using the reduction R_1. We designate the null state by λ since the original already had a state named ϵ.

$$\begin{array}{rcl} A & = & \{a, b, c\} \\ Q & = & \{00, 01, 10, 11, \epsilon, \lambda\} \\ q_0 & = & 00 \\ Act & = & \{0, 1, \epsilon\} \\ act : \langle x, y \rangle & \mapsto & Act \;\; \text{for every state } x \in Q \text{ and every } y \in A \end{array}$$

The specification δ can be derived from Figure 13. Note that the only transitions leading into state λ are those that have multiple non-null actions in the move vectors while those leading to ϵ are those that were specified by the original state machine's transition function.

The definition of concurrent game structures forced us to add an additional state and a special null action. It is not as clear whether the reduction is completely valid. The validity, however, can only be argued in respect to the uses of the two models. This, for our thesis, is the specification of noninterference assertions. We develop here a few definitions that will aid us in our later correlations between noninterference assertions across the state machine and the corresponding concurrent game structure. We will demonstrate the validity of our reduction in respect to noninterference assertions in Section 3.7.

Definition 28 (Sequential Run). *A run r in a concurrent game structure with a special designated null action ϵ is* sequential *if every move vector in the run contains exactly one non-null action.*

Such runs correspond exactly to runs in a Goguen & Meseguer state machine. We now define purging of sequential runs.

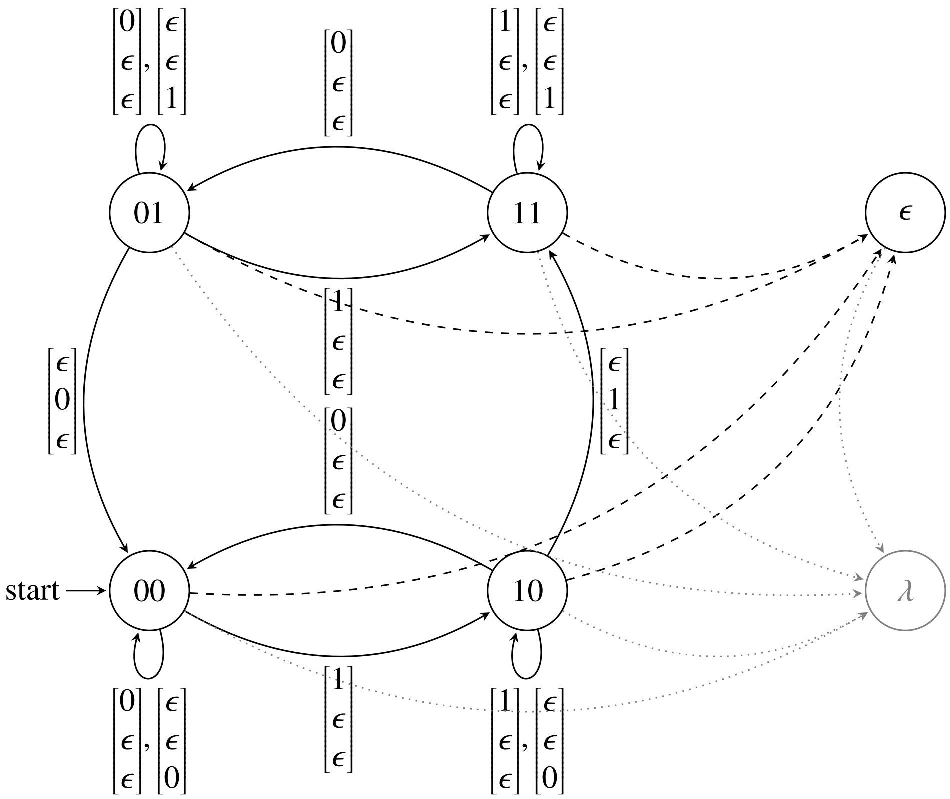


Figure 13: Concurrent Game Structure Version of Example 7

Definition 29 (Sequential Run Purge). *Let us define $p_G(r)$ for a any set of agents $G \in A$ and any sequential run r. We do so by first* flattening *the runs to runs that are of the form used in Goguen & Meseguer state machines.*

$$\begin{aligned} flat(\epsilon) &= \epsilon \qquad \textit{note: } \epsilon \textit{ here refers to the empty run} \\ flat([\epsilon,...,\epsilon,i_j,\epsilon,...,\epsilon].r) &= \langle i_j, a_j \rangle .flat(r) \qquad \textit{where } a_j \textit{ refers to the } j^{th} \textit{ agent} \end{aligned}$$

We thus define purging of sequential runs by first flattening them:

$$p_G(r) = p_G(flat(r))$$

Note that the purge on the right hand side refers to the purge as described in Definition 8. Also note that the purging of a sequential run of a concurrent game structure results in a run composed not of move vectors but rather action/agent pairs (the kind of runs present in $Goguen\&Meseguer$ state machines).

It will be necessary further in this thesis to perform the opposite of $flat$, that is, take a Goguen & Meseguer state machine run and raise it to a concurrent game structure run. We define *raise* in the following manner:

$$\begin{aligned} raise(\epsilon) &= \epsilon \\ raise(\langle i_j, a_j\rangle.r) &= [\epsilon,...,\epsilon,i_j,\epsilon,...,\epsilon].raise(r) \qquad \text{where } a_j \text{ refers to the } j^{th} \text{ agent} \end{aligned}$$

Note that *raise* produces only sequential runs.

3 Noninterference

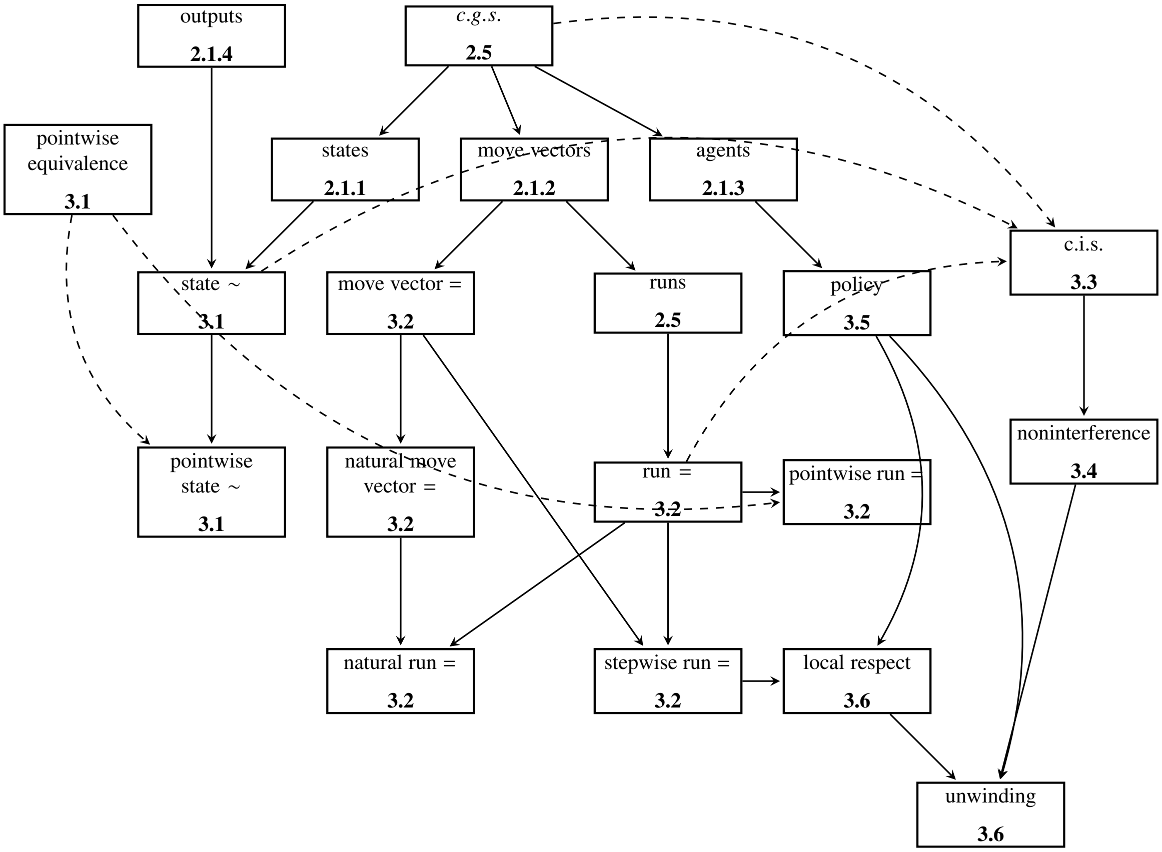


Figure 14: Dependency Graph

To establish our definition of noninterference it is necessarily first to formalize the knowledge or perception of agents. The central issue, after all, is the ability of agents to discern differences in the state of the system after no differentiating actions of their own. We need some way to describe the perceived difference in the eyes of an agent. We do this by introducing equivalence relations on the states of a c.g.s. as means to describe perception, and an equivalence relation on runs to establish the lack of differentiating actions.

We establish our definition of noninterference and demonstrate it using several examples. We then describe a theorem with which we can prove noninterference assertions and finally demonstrate that our notion of noninterference is general enough to describe the original formulation of Goguen & Meseguer.

3.1 State Equivalence

We establish agent perception via a relation on the states of a c.g.s..

Definition 30 (State Equivalence). *A* view partition *(or* state equivalence *) $\sim_G$ in to some set of agents $G \subseteq A$, is an equivalence relation on the states Q. We will refer to states equivalent under $\sim_G$ as $\sim_G$-equivalent states. We require that $q \sim_\emptyset q'$ for every pair of states.*

We assume the following axiom. Intuitively, we cannot gain distinguishability be considering a smaller set of agents (as in Axiom 1). Alternatively we cannot lose distinguishability by considering a larger set of agents (Remark 31).

Axiom 1 (State Equivalence).

$$\frac{q \sim_G q' \quad G^- \subseteq G}{q \sim_{G^-} q'}$$

Proposition 31. *From Axiom 1, we can derive its dual:*

$$\frac{q \not\sim_G q' \quad G \subseteq G^+}{q \not\sim_{G^+} q'}$$

Proof. Assume $q \not\sim_G q'$ but $q \sim_{G^+} q'$. By Axiom 1, $q \sim_G q'$ since $G \subseteq G^+$ contradicting the assumption. □

The intention of the view partition is to specify states that are indistinguishable to a set of agent. Note that this notion is a replacement for the idea of an *output* as in [Rus92]. More interestingly, view partitions are used by Rushby for demonstrating unwinding theorems. In our formalism, however, the view partitions take a more central role.

Though it may potentially be useful to define the partitions for groups of agents arbitrarily we will sometimes make assumptions on the structure of state equivalence relations. The most common of these assumptions is the requirement that state equivalence for groups of agents is directly derived from that of single agents as characterized by Remark 32.

Remark 32. *The collective set of state-equivalences $\sim$ follows* pointwise state equivalence *iff for any set of agents $G \subseteq A$ and states $q, q' \in Q$, it is the case that $q \sim_G q'$ iff $\forall a \in G \;\; q \sim_{\{a\}} q'$.*

If we think of states q and q' as being indistinguishable for the group G and we assume the notion of the groups discernibility as being cooperative, then we must assume that none of the agents in the group can discern the difference between q and q' as otherwise they would all as a group be able to discern the difference. Likewise, a group not being able to distinguish states results from none of the agents in the group being able to make the distinction.

3.2 Run Equivalence

To establish lack of differentiating actions, we define a relations on runs of a c.g.s..

Definition 33 (Run Equivalence). *A* run equivalence *in respect to a group of agents G in a c.g.s. S, written $=_G$ is an equivalence relation on runs in S. If r, r' are two runs in S with $r =_G r'$, we call the runs $=$-equivalent. We require that $r =_\emptyset r'$ for every pair of runs.*

We also assume the following axiom. The nature of its intuitive basis is the same as that of the state equivalence axiom.

Axiom 2 (Run Equivalence).

$$\frac{r =_G r' \quad G^- \subseteq G}{r =_{G^-} r'}$$

Proposition 34. *From Axiom 2, we derive its dual:*

$$\frac{r \neq_G r' \quad G \subseteq G^+}{r \neq_{G^+} r'}$$

Proof. Assume $r \neq_G r'$ but $r =_{G^+} r'$. By Axiom 2, $r \sim_G r'$ since $G \subseteq G^+$ contradicting the assumption. □

As we have done for state equivalence, we will sometimes make the following assumption concerning the run-equivalence for groups in relation to individuals.

Remark 35. *A run-equivalence $=$ is a* pointwise run equivalence *if for any set of agents $G \subseteq A$ and runs r, r' in a c.g.s. S, it is the case that $r =_G r'$ iff $\forall a \in G \;\; r =_{\{a\}} r'$.*

Working specifically in concurrent game structures has the benefit of a natural specification of run-equivalence that we will use most often throughout this work; one that is based on equivalence of move vectors (as the building blocks of a run).

Definition 36 (Move Vector Equivalence). *We define a* move vector equivalence *as an equivalence relation on move vectors. Also we define the* natural equivalence of move vectors *in respect to a group of agents. If $\hat{i}, \hat{i'}$ are move vectors and $G \subseteq A$ is a set of agents, $\hat{i} \doteq_G \hat{i'}$ iff the move vectors truncated to only the actions of agents in G are the equal.*

As we noted for runs, the equivalence of move vectors may be specified arbitrary for groups of agents but we make a remark about pointwise defined move vectors.

Remark 37. *A move vector equivalence $\doteq$ is a* pointwise move vector equivalence *if for any set of agents $G \subseteq A$ and move vectors $\hat{i}$, $\hat{i'}$, it is the case that $\hat{i'} \doteq_G \hat{i'}$ iff $\forall \in G \;\; \hat{i} \doteq_{\{a\}} \hat{i'}$.*

Note that there is nothing in our definition of run equivalence to require the equivalence to be defined based on the equivalence of all the move vectors comprising the runs. We do, however, define a class of run equivalences that are specified in such a way.

Definition 38 (Stepwise Run Equivalence). *A* stepwise run equivalence *with basis $\doteq$ is a run equivalence defined in terms of a move vector equivalence $\doteq$ such that $r = r'$ iff (r, r' are both of length n and) $r[j] \doteq r'[j]$ for all $j \in \{1, ..., n\}$ (this has an implied requirement that both runs are of length n).*

Note that if a stepwise run equivalence is defined in terms of a pointwise move vector equivalence, then the run equivalence is itself pointwise.

We extend our definition of a natural move vector equivalence to a natural run equivalence which will be used most often throughout this thesis.

Definition 39 (Natural Run Equivalence). *The* natural run equivalence *is the step-wise run equivalence defined the natural move vector equivalence.*

3.3 Concurrent Interference System

We will combine a concurrent game structure with the accompanying equivalence relations and refer to them jointly as a system.

Definition 40 (Concurrent Interference System). *A* concurrent interference system *S (or just* system*) is a triple $\langle C, \sim, = \rangle$ where,*

- *C is a concurrent game structure,*
- *$\sim_G$ is a state equivalence relation for every set of agents $G \subseteq A$ of C, and*
- *$=_G$ is a run equivalence relation for every set of agents $G \subseteq A$ of C.*

3.4 Noninterference

With notions of perception and differentiating actions, we have the necessary structure to define noninterference.

Definition 41 (Noninterference). *Agents $G' \subseteq A$ do not* interfere *with agents $G \subseteq A$ (or $G' \not\Rightarrow G$) iff $=_{A-G'}$-equivalent runs end in $\sim_G$-equivalent states. That is, any two runs r, r' with $r =_{A-G'} r'$ end at $\sim_G$-equivalent states. Note that all runs we are considering start with q_0 or the initial state.*

Proposition 42. *Given our axioms for state and run equivalences, we can derive tow following two rules for interference assertions.*

$$\frac{G' \Rightarrow G \quad G' \subseteq G'^{+}}{G'^{+} \Rightarrow G}$$

$$\frac{G' \Rightarrow G \quad G \subseteq G^{+}}{G' \Rightarrow G^{+}}$$

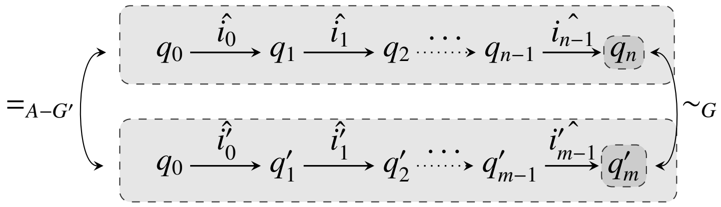

$q_0 \xrightarrow{\hat{i_0}} q_1 \xrightarrow{\hat{i_1}} q_2 \overset{\cdots}{\dashrightarrow} q_{n-1} \xrightarrow{\widehat{i_{n-1}}} q_n$
$=_{A-G'}$
$\sim_G$
$q_0 \xrightarrow{\hat{i'_0}} q'_1 \xrightarrow{\hat{i'_1}} q'_2 \overset{\cdots}{\dashrightarrow} q'_{m-1} \xrightarrow{\widehat{i'_{m-1}}} q'_m$

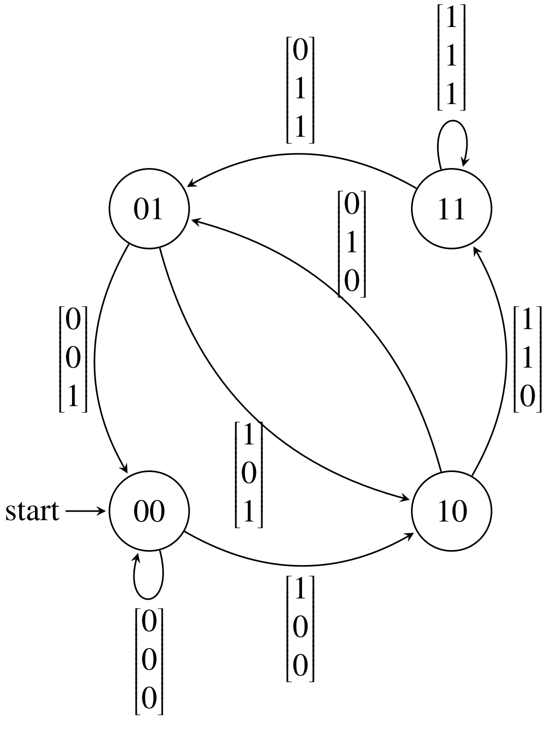
Figure 15: Noninterference

Proof. If $G' \Rightarrow G$, then we have two runs r, r' with $r =_{A-G} r'$ ending in non $\sim_G$-equivalent states. For a larger set of agents G'' (with $G' \subseteq G''$), we have $(A - G'') \subseteq (A - G')$ and thus $r =_{A-G''} r'$ by the axioms of run equivalence, giving us $G'' \Rightarrow G$ as the fact that r, r' end in non $\sim_G$-equivalent states has not changed. Alternatively for a bigger set G'' (with $G \subseteq G''$) we get that r, r' end in non $\sim_{G''}$-equivalent states by the axioms of state equivalence, and thus $G' \Rightarrow G''$. □

The definition as shown serves to capture the notion that agents cannot discern differences if they took up no differentiating actions where differences are described by state equivalence and differentiating actions are described by run equivalence. If an agent or a set of agents can tell the difference between final states in two runs in which these agents made the choose the same actions (as in the natural run-equivalence definition), then we can conclude that the agents were interfered with.

Example 43 (Bird-song Propagation as a Concurrent Interference System). Consider a concurrent interference system composed as follows:

$$
\begin{array}{rcl}
A & = & \{a, b, c\} \\
Q & = & \{00, 01, 10, 11\} \\
q_0 & = & 00 \\
Act & = & \{0, 1\} \\
act : \quad \langle x, a \rangle & \mapsto & Act \;\; \text{for every state } x \in Q \\
\langle XY, b \rangle & \mapsto & \{X\} \;\; \text{for every state } XY \in Q \\
\langle XY, c \rangle & \mapsto & \{Y\} \;\; \text{for every state } XY \in Q
\end{array}
$$



$\begin{bmatrix}1\\1\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\\1\end{bmatrix}$
01
11
$\begin{bmatrix}0\\1\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\\1\end{bmatrix}$
$\begin{bmatrix}1\\1\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\\1\end{bmatrix}$
start
00
10
$\begin{bmatrix}0\\0\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\\0\end{bmatrix}$

Figure 16: Concurrent Interference System Example: Transitions

Let transitions be defined according to Figure 16. Also, let us define the state equivalence relations pointwise based off of the following:

$$\begin{aligned}
\sim_{\{a\}} &= \{(X, Y) \mid X, Y \in Q\} \\
\sim_{\{b\}} &= \{(X_1Y_1, X_2Y_2) \mid X_1Y_1, X_2Y_2 \in Q \text{ and } X_1 = X_2\} \\
\sim_{\{c\}} &= \{(X_1Y_1, X_2Y_2) \mid X_1Y_1, X_2Y_2 \in Q \text{ and } Y_1 = Y_2\}
\end{aligned}$$

Finally let the run equivalence be the natural run equivalence. We can thus evaluate some noninterference assertions in this system:

- $\{a\} \Rightarrow \{b\}$. We can see this via runs $00, \begin{bmatrix}0\\0\\0\end{bmatrix}, 00$ and $00, \begin{bmatrix}1\\0\\0\end{bmatrix}, 10$. We see that $\begin{bmatrix}0\\0\\0\end{bmatrix} \doteq_{\{b,c\}} \begin{bmatrix}1\\0\\0\end{bmatrix}$ as the move vector components for agents b and c are the same in all the move vectors in both runs. The first run, however, ends at state 00 whereas the second ends at 10 with $00 \not\sim_{\{b\}} 10$.

- $\{a, b\} \Rightarrow \{c\}$. This can be demonstrated by runs $00, \begin{bmatrix}0\\0\\0\end{bmatrix}, 00, \begin{bmatrix}0\\0\\0\end{bmatrix}, 00$ and $00, \begin{bmatrix}1\\0\\0\end{bmatrix}, 10, \begin{bmatrix}0\\1\\0\end{bmatrix}, 01$. All corresponding move vectors are equivalent for $\{c\}$ so the runs are $=_{\{c\}}$-equivalent but they end in non $\sim_{\{c\}}$-states.

- $\{a\} \not\Rightarrow \{c\}$. We will prove this fact once we show the Little Unwinding Theorem but the statement itself is of some interest. To show $\{a\} \Rightarrow \{c\}$ we need to find two runs whose corresponding move vector components are equal for agents b and c (that is $A - \{a\}$) but which end in non $\sim_{\{c\}}$-equivalent states. We see, however, that it is agent b's last action that directly impacts the distinguishability (the state equivalence) for agent c. Thus a alone does not interfere with c but rather does this through b.

- $\{b\} \not\Rightarrow \{c\}$. This is another fact of interest but unlike the previous, we cannot use our unwinding theorem to prove it. To show $\{b\} \not\Rightarrow \{c\}$ we need to find $=_{\{c,b\}}$-equivalent runs that end in non $=_{\{c\}}$-equivalent states. We see, however, that the actions of b are determined directly from the actions of a, giving c a distinguishability of runs (based on a's actions) that prevents c to get to non $\sim_{\{c\}}$ states via indistinguishable runs.

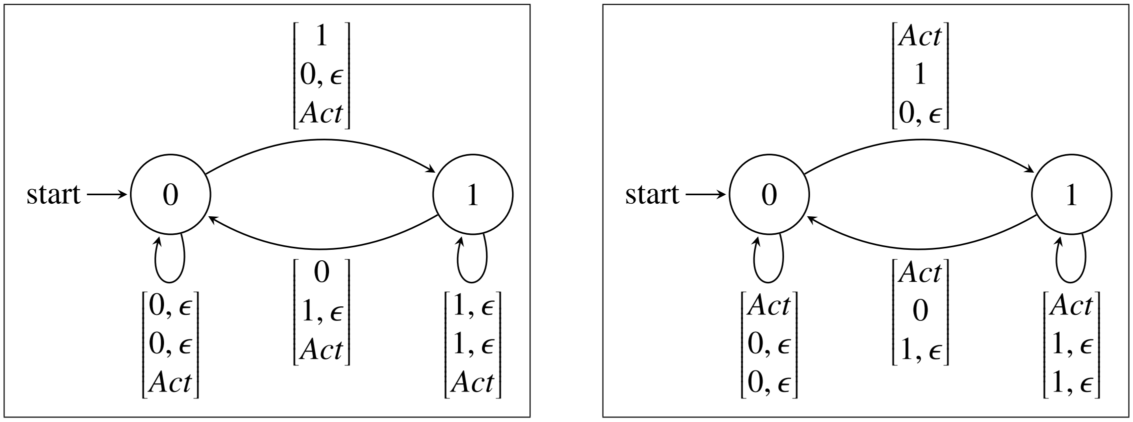
Example 44 (Bird-song Propagation as Concurrent Interference System with Free Choice). In our previous example, we did not allow agents b and c any choice at all. That is, for every state, each of these agents had exactly one allowable action. This resulted in some surprising noninterference assertions (see Example 43). Here we will restructure the system so that each agent does not necessarily have to repeat the last thing it hears; it can wait instead. This is closer to our original motivation behind the example (see Example 1). The specification gets too complex so we will specify the system via a cross-product construction.

$$\begin{array}{rcl}
A & = & \{a, b, c\} \\
Act & = & \{0, 1, \epsilon\} \\
Q_1 & = & \{0, 1\} \\
Q_2 & = & \{0, 1\} \\
q_1 & = & 0 \\
q_2 & = & 0 \\
act_1 : & \langle x, a \rangle & \mapsto Act \text{ for every state } x \in Q_1 \\
& \langle x, b \rangle & \mapsto \{x, \epsilon\} \text{ for every state } x \in Q_1 \\
& \langle x, c \rangle & \mapsto Act \text{ for every state } x \in Q_1 \\
act_2 : & \langle x, a \rangle & \mapsto Act \text{ for every state } x \in Q_2 \\
& \langle x, b \rangle & \mapsto Act \text{ for every state } x \in Q_2 \\
& \langle x, c \rangle & \mapsto \{x, \epsilon\} \text{ for every state } x \in Q_2
\end{array}$$

Let δ_1 and δ_2 be defined according to Figure 17a and Figure 17b respectively.

Let Q, q_0, act, and δ be defined according to the cross product construction from $Q_1, Q_2, q_1, q_2,$, act_1, act_2, and δ_1, δ_2 respectively.

Finally let us define the state equivalence relations pointwise based off of the following:



(a) (b)

Figure 17: Concurrent Interference System Example with Choice: Transitions

$$\begin{aligned}
\sim_{\{a\}} &= \{(X, Y) \mid X, Y \in Q\} \\
\sim_{\{b\}} &= \{(\langle X_1, Y_1\rangle, \langle X_2, Y_2\rangle) \mid \langle X_1, Y_1\rangle, \langle X_2, Y_2\rangle \in Q \text{ and } X_1 = X_2\} \\
\sim_{\{c\}} &= \{(\langle X_1, Y_1\rangle, \langle X_2, Y_2\rangle) \mid \langle X_1, Y_1\rangle, \langle X_2, Y_2\rangle \in Q \text{ and } Y_1 = Y_2\}
\end{aligned}$$

The state equivalence relations describe the following situation: agent a cannot tell any states apart, agent b can tell the difference between states that differ in their first component (which corresponds exactly to what agent a performed last), and agent c can tell the difference between states that differ in the second component (corresponding to what agent b performed last).

Finally let the run equivalence be the natural run equivalence. We can thus evaluate some noninterference assertions in this revised system. Note that any run in the system described in Example 43 is also a run in this revised system. We have, however, additional runs possible here. Specifically, agents b and c have a choice of acting out (as in Example 43) or performing the null operation ϵ.

- $\{a\} \Rightarrow \{b\}$ and $\{a, b\} \Rightarrow \{c\}$. The interference claims do not change from Example 43 as the witness runs are still valid here.

- $\{a\}\not\Rightarrow\{c\}$. This fact can be shown using the Little Unwinding Theorem further in this thesis.

- $\{b, c\}\not\Rightarrow\{a\}$. No group of agents can possibly interfere with a since a cannot tell apart any two states.

- $\{c\}\not\Rightarrow\{a, b\}$ (equivalent to the conjunction of $\{c\}\not\Rightarrow\{a\}$ and $\{c\}\not\Rightarrow\{b\}$). Actions of c have absolutely no impact on the state thus c cannot interfere with any other agent (and thus with the group).

- $\{b\} \Rightarrow \{c\}$. We see that this last claim differs from the original example. Consider the following two runs:

  $$r = \langle 0, 0\rangle, \begin{bmatrix} 1 \\ \epsilon \\ \epsilon \end{bmatrix}, \langle 1, 0\rangle, \begin{bmatrix} \epsilon \\ 1 \\ \epsilon \end{bmatrix}, \langle 1, 1\rangle$$

  $$r = \langle 0, 0\rangle, \begin{bmatrix} 1 \\ \epsilon \\ \epsilon \end{bmatrix}, \langle 1, 0\rangle, \begin{bmatrix} \epsilon \\ \epsilon \\ \epsilon \end{bmatrix}, \langle 1, 0\rangle$$

  The runs are $=_{\{a,c\}}$-equivalent but end in non $\sim_{\{c\}}$-equivalent states. Thus by giving agent b a choice, we caused it to interfere with c whereas without the choice (Example 43), the interference could not be shown.

3.5 Policies

Having a definition of noninterference, we can aggregate a set of noninterference assertions in a noninterference policy:

Definition 45 (Noninterference Policy). *A* noninterference policy $\rightsquigarrow$ *is a reflexive relation on A* [4]*. For agents $a, a' \in A$, $a \rightsquigarrow a'$ denotes that agent a can interfere with agent a'. If not $a \rightsquigarrow a'$ (or $a \not\rightsquigarrow a'$) then agent a is meant to not interfere with agent a'.*

[4] Note that this refers to a certain set of agents and thus is applicable to any system with that set of agents. Also note that this definition only differs from the one provided by Rushby in that agents are involved instead of security domains.

We denote $a\downarrow = \{a' \mid a' \rightsquigarrow a\}$ as the set of agents in A that are allowed to interfere with a and $a^{\perp} = \{a' \mid a' \not\rightsquigarrow a\}$ as the set of those agents which are not allowed to interfere with a.

Definition 46 (Policy Satisfaction). *A system S* satisfies *a pointwise noninterference policy $\rightsquigarrow$ if for every agent $a \in A$, $a^{\perp} \not\Rrightarrow \{a\}$.*

Example 47 (Policy Satisfaction). The system described in Example 44 satisfies the policy in Figure 18.

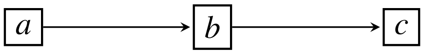


Figure 18: Policy for the Concurrent Interference System in Example 44

To show the fact we need to establish several noninterference assertions: $\{b, c\} \not\Rrightarrow \{a\}$, $\{c\} \not\Rrightarrow \{b\}$, and $\{a\} \not\Rrightarrow \{c\}$. All of these have been established in Example 44.

3.6 Unwinding

The Unwinding Theorem serves to reduce the question of noninterference which is based on equivalence of runs and states to remarks about individual transitions of a system. We require, however, that the run equivalence is defined stepwise.

$$\begin{tikzcd}
q_0 \arrow[r, "\hat{\imath}_0"{name=A0}] & q_1 \arrow[r, "\hat{\imath}_1"{name=A1}] & q_2 \arrow[r, dotted, "\cdots"] & q_{n-1} \arrow[r, "\hat{\imath}_{n-1}"{name=A2}] & \boxed{q_n} \arrow[d, leftrightarrow, bend left, "\sim_G"] \\
q_0 \arrow[r, "\hat{\imath}'_0"'{name=B0}] & q'_1 \arrow[r, "\hat{\imath}'_1"'{name=B1}] & q'_2 \arrow[r, dotted, "\cdots"] & q'_{n-1} \arrow[r, "\hat{\imath}'_{n-1}"'{name=B2}] & \boxed{q'_n}
\arrow[from=A0, to=B0, leftrightarrow, "\doteq_{A-G'}"]
\arrow[from=A1, to=B1, leftrightarrow, "\doteq_{A-G'}"]
\arrow[from=A2, to=B2, leftrightarrow, "\doteq_{A-G'}"]
\end{tikzcd}$$

Figure 19: Noninterference with Step-wise Run Equivalence

Definition 48 (Local Respect of an Assertion). *A system S with a stepwise run equivalence locally respects a noninterference assertion $G' \not\Rrightarrow G$ iff for every pair of states $q, q' \in Q$ with $q \sim_G q'$, and every pair of move vectors $\hat{\imath} \in D(q), \hat{\imath}' \in D(q')$ with*

$\hat{i} \doteq_{A-G'} \hat{i'}$ (where $\doteq$ here refers to the stepwise basis of the run equivalence) we have $\delta(q, \hat{i}) \sim_G \delta(q', \hat{i'})$.

We can now present our version of the Unwinding Theorem (originally demonstrated by Goguen & Meseguer in [GM84] for their noninterference formulation).

Theorem 49 (Little Unwinding Theorem). *A system S with a step-wise run equivalence $=$ satisfies a noninterference assertion $G' \not\rightsquigarrow G$ if S locally respects the assertion.*

Proof. We need to show that G' does not interfere with G which, by our definition, means that all $=_{A-G'}$-equivalent runs end in $\sim_G$-equivalent states. We begin with a lemma:

Lemma 50. *For any two runs r, r' with $r = q_0, \hat{i_0}, q_1, ..., q_n$, $r' = q'_0, \hat{i'_0}, q'_1, ..., q'_n$, and $r =_{A-G'} r'$ we have $q_j \sim_G q'_j$ for any $j \in \{0, ..., n\}$.*

Proof. Let us show this by induction on j:

- $j = 0$: $q_0 = q'_0$ as all runs begin with the same state and thus $q_0 \sim_G q'_0$ as $\sim_G$ is an equivalence relation.
- $j - 1 \leq n$: Let us assume that $q_{j-1} \sim_G q'_{j-1}$.
- $j \leq n$: By inductive assumption, $q_{j-1} \sim_G q'_{j-1}$ and by initial assumption $\widehat{i_{j-1}} \doteq_{A-G'} \widehat{i'_{j-1}}$. Since S locally respects $\rightsquigarrow$, we know $\delta(q_{j-1}, \widehat{i_{j-1}}) \sim_G \delta(q'_{j-1}, \widehat{i'_{j-1}})$ or more specifically, $q_j \sim_G q'_j$.

□

Consider any two runs r, r' with $r =_{A-G'} r'$. Lemma 50 tells us that the runs end in $\sim_G$-equivalent states and thus $G' \not\rightsquigarrow G$. Note that the definition of a stepwise defined run equivalence requires any equivalent runs to be of the same length. □

We also demonstrate the main Unwinding Theorem as a tool for showing satisfaction of an entire noninterference policy (in line with the Unwinding Theorem presented by Rushby in [Rus92]). We begin with an alternate definition of local respect.

Definition 51 (Local Respect of a Policy). *A system S with a stepwise run equivalence locally respects a policy $\rightsquigarrow$ iff for every agent $a \in A$, states $q, q' \in Q$ with $q \sim_{\{a\}} q'$, and move vectors $\hat{\imath} \in D(q), \hat{\imath}' \in D(q')$ with $\hat{\imath} \doteq_{A-a^{\perp}} \hat{\imath}'$ (where $\doteq$ here refers to the stepwise basis of the run equivalence) we have $\delta(q, \hat{\imath}) \sim_{\{a\}} \delta(q', \hat{\imath}')$.*

Proposition 52. *A system S with a stepwise run equivalence locally respects a policy $\rightsquigarrow$ iff for every agent $a \in A$, S locally respects the assertion $a^{\perp} \not\Rightarrow \{a\}$.*

The proposition is merely a substitution of local respect of an assertion into the definition of local respect of a noninterference policy.

Theorem 53 (Unwinding Theorem). *A system S with a step-wise run equivalence $=$ satisfies a policy $\rightsquigarrow$ if S locally respects $\rightsquigarrow$.*

Proof. Given local respect of $\rightsquigarrow$ we know S locally respects $a^{\perp} \not\Rightarrow \{a\}$ for every agent a by Proposition 52. By the Little Unwinding Theorem, we know that $a^{\perp} \not\Rightarrow \{a\}$. □

Note that the Unwinding Theorem presented here could just as well be labeled as a corollary of the Little Unwinding Theorem. We call it a theorem, however, to remain consistent with the naming of the equivalent Unwinding Theorem presented by Rushby in [Rus92].

Example 54 (Unwinding Theorem Application). In Example 43 (and Example 44) we a noninterference claim without providing sufficient proof. Having the Little Unwinding Theorem lets us now prove the claim:

- $\{a\} \not\Rightarrow \{c\}$ (as in Example 43)

  We need to check that the system locally respects this assertion which in turn requires us to examine the transitions coming out from any two $\sim_{\{c\}}$-equivalent states. Let us consider the equivalence classes in turn:

  - $\{00, 10\}$ - There are no $\doteq_{\{b,c\}}$-equivalent move vectors coming out of a pair of these states.
  - $\{01, 11\}$ - Similarly to the above, there are no $\doteq_{\{b,c\}}$-equivalent move vectors coming out of any pair of these states.

Thus we see that the local respect condition for the assertion has been satisfied vacuously.

- $\{a\}\not\Rightarrow\{c\}$ (as in Example 44)

  We need to check that the system locally respects this assertion which in turn requires us to examine the transitions coming out from any two $\sim_{\{c\}}$-equivalent states. Let us consider the equivalence classes in turn:

  - $\{\langle 0,0\rangle, \langle 1,0\rangle\}$ - We see that the only transitions that lead from these states to states with the second component being equal to 1 are $\begin{bmatrix}1,\epsilon\\ 1\\ 0,\epsilon\end{bmatrix}$ from $\langle 1,0\rangle$ to $\langle 1,1\rangle$ and $\begin{bmatrix}0\\ 1\\ 0,\epsilon\end{bmatrix}$ from $\langle 1,0\rangle$ to $\langle 0,1\rangle$. Every other transition has agent b doing action 0 or ϵ and leads to states with 0 as the second component. Thus it is b's action that differentiates transitions that lead to states with different second components. Therefore we have no $\doteq_{\{b,c\}}$-equivalent move vectors that leads to different second state components (that is, non $\sim_{\{c\}}$-equivalent states).

  - $\{\langle 0,1\rangle, \langle 1,1\rangle\}$ - Similarly to the above, it is b's action that determines whether the state we transition to has 0 or 1 as the second component. Since we are considering $\doteq_{\{b,c\}}$-equivalent move vectors, we see there is no $\doteq_{\{b,c\}}$-equivalent move vectors that lead us from the members of the equivalence class to non $\sim_{\{c\}}$-equivalent states.

  Thus we see that the local respect for the assertion has been satisfied vacuously.

3.6.1 Limitations

Though the Unwinding Theorem can be used for proving noninterference assertions, it is not always applicable. The theorem is too weak as it can be used to prove assertions using a stronger noninterference definition. Specifically if we were to allow our systems to start in any state and likewise define valid runs as starting at any state, our proof of the Unwinding Theorems would still apply but yet noninterference in such alternate systems is much harder to satisfy (as runs do not necessarily start in the same state).

Specifically in Example 43, we see that the assertion $\{b\} \not\Rightarrow \{c\}$ cannot be proven using the Little Unwinding Theorem as indeed the local respect of the assertion is not satisfied. We see that states 00 and 10 are $\sim_{\{a,c\}}$-equivalent, but they contain $\doteq_{\{a,c\}}$ transitions $[1,0,0]$ and $[1,1,0]$ from 00 and 10 respectively that lead to states 10 and 11 which are not $\sim_{\{c\}}$-equivalent. We see, however, that states 00 and 10 cannot be reached from the initial state by following only $\doteq_{\{a,c\}}$-equivalent move vectors. Thus we cannot construct runs to demonstrate $\{b\} \Rightarrow \{c\}$ though if we defined valid runs to start in any state, we easily could.

3.7 Goguen & Meseguer's Noninterference

We note that our definition of noninterference is general enough to express the familiar Goguen & Meseguer and Rushby noninterference assertions in its framework. We present here the relationship between the Goguen & Meseguer formulation and our own by expressing it in our more general scheme.

Given a state machine S, we present here the construction of a concurrent interference system $S' = \langle C, \sim, = \rangle$ such that any Goguen noninterference assertion about S is exactly exhibited by a noninterference assertion about S'. The state machine construction here is based on Remark 25 and the reduction of presented in Definition 26.

Definition 55 (State Machine with Agents and Outputs to Concurrent Interference System Reduction). *Given a state machine (of the Goguen & Meseguer type) $S = \langle U, S, C, Out, out, do, s_0 \rangle$, the reduction $R_1'(S)$ is a concurrent interference system $S' = \langle R_1(S), \sim, = \rangle$ defined as follows:*

- *$R_1(S)$ is the reduction of S as described in Definition 26.*
- *$out(q, u) = out(q', u)$ iff $q \sim_{\{u\}} q'$. The equivalence for every other set of agents is defined pointwise, that is $\sim$ is a pointwise state equivalence.*
- *$p_G(r) = p_G(r')$ iff $r =_{A-G} r'$ for every pair of sequential runs r, r'. If either of r, r' is not sequential then $r \not=_G r'$ for any group of agents G without condition*

(unless of course $r = r'$).

To demonstrate the validity of such a reduction, we would like to establish that noninterference assertions are equivalent in both the original system and the concurrent interference systems.

Theorem 56 (Noninterference-wise Validity of Reduction). *A noninterference assertion $G' \not\Rightarrow G$ is true in a Goguen & Meseguer system S iff the assertion is true in the concurrent interference system $S' = R'_1(S)$.*

Proof. Let us consider both sides of the theorem in turn.

($\Rightarrow$) We are given that $G' \not\Rightarrow G$ is true in S. Let us assume that $G' \Rightarrow G$ in S'. Thus there are two runs r, r' with $r =_{A-G'} r'$ that end states q, q' with $q \not\sim_G q'$. The definition of the reduction tells is that any non-sequential runs end in the special ϵ state. Since $r =_{A-G'} r'$, we know that either both runs are sequential or neither is (recall that $r \not= r'$ for any non sequential runs $r \neq r'$). If both were to be non-sequential then both would end in the ϵ and thus not end in non $\sim_G$-equivalent states. It must be, therefore, that both r, r' are sequential. Thus we know $p_{G'}(r) = p_{G'}(r')$ (via our construction of run equivalence). The definition of purge tells us that the flattened runs purge to the same run. Let us say $f = flat(r)$ and $f' = flat(r')$. We thus know $p_{G'}(f) = p_{G'}(f')$. Let us call this "central" run $c = p_{G'}(f) = p_{G'}(f')$. By definition of the state equivalence in our reduction, we know that the output to the last states of f and f' (which we previously labeled as q and q') differs for some agent in G. The pointwise state equality does not let us have $q \not\sim_G q'$ while $q \sim_{\{u\}} q'$ for all $u \in G$. Let us call this agent u. Thus we have $\lceil f \rceil_u \neq \lceil f' \rceil_u$. Let o be the output to u at the end of the central run, that is, $o = \lceil c \rceil_u$. We can now see that either $o \neq \lceil f \rceil_u$ or $o \lceil f' \rceil_u \neq \lceil f' \rceil_u$. Let us say it is the first without the loss of generality. Thus we have $\lceil p_{G'}(f) \rceil_u \neq \lceil f \rceil_u$ which lets us conclude that $G' \Rightarrow \{u\}$ in the original Goguen & Meseguer state machine, contradicting the given fact that $G' \not\Rightarrow G$ in that state machine.

($\Leftarrow$) Going the other direction we have $G' \not\Rightarrow G$ in S'. Let as assume that $G' \Rightarrow G$ in S. That is, there is an agent $u \in G$, and a run r such that $\lceil p_{G'}(r) \rceil_u \neq \lceil r \rceil_u$. Let us refer to the last states of r and $p_{G'}(r)$ as q and q' respectively. By our definition of state equivalence, we have $q \not\sim_{\{u\}} q'$ and thus $q \not\sim_{G} q'$ by Proposition 31. By raising the two runs, we still arrive at the same ending states, that is $q = \lceil raise(p_{G'}(r)) \rceil$ and $q' = \lceil raise(r) \rceil$. Since $p_{G'}(r)$ purges to itself (it is already purged), we know that $p_{G'}(raise(p_{G'}(r))) = p_{G'}(raise(r))$ which implies (by our definition of run equivalence in the reduction) that $raise(p_{G'}(r)) =_{A-G'} raise(r)$ (recall that *raise* always results in sequential runs). Thus we have two $=_{A-G'}$-equivalent runs that end in non $\sim_G$-equivalent states. Thus $G' \Rightarrow G$ in S', contradicting our assumption. □

The reduction of Rushby state machines is more involved. We do not provide it here, however.

4 Composability

The issue of composability of security properties is a commonly noted problem [McL94]. Specifically if a security assertion is true in some components, we would naturally like to preserve the property upon combining the components into a larger whole. Alternatively if we are analyzing a complex system via decomposition into simpler components, we would like to make conclusions about security properties of the whole by solving the issue of their presence in the simpler components.

The security property relevant to our thesis, of course, is noninterference. Like for security properties in general, noninterference is cited to not be preserved under composition [ZL95]. We present a variation of a problematic composition example described by McCullough [McC88] but note its lack of formal specification (we provide such a specification in Appendix D). Though such examples often lack formal specification, often specific guidelines are provided to avoid the situations exemplified (in the example presented it is the presence of finite buffers). We, alternatively, attack the problem from the different level by addressing an reasoning behind the intuitive notion that composition *should* preserve properties.

4.1 Problematic Composition

We present here a high-level description of a variation of the composition example presented by McCullough in [McC88]. An attempt at modeling the situation in a concurrent game structure is presented in Appendix D. It is unclear to us that this is a valid example of the non-composability of a security property though we present it here and let the reader judge.

Consider a system composed of three components: *high* (or h), *low0* (or l_0), and *low1* (or l_1). Present in the system are also two 1-bit buffers which we label $0B$, $1B$. $0B$ connects the output of the low0 component to high and $1B$ connects the output of the low1 component to high. Specifically what a connection and what an output means is omitted here but we attempt to model the informal specification in Appendix D. Finally an input is fed into high and both low0 and low1 send output to a common output buffer (again, this is not modeled here). The buffers $0B$ and $1B$ are 1-bit and blocking. If a component attempts to write to the buffer while it is filled, the component is blocked (until the buffer is emptied). The situation is described by Figure 22.

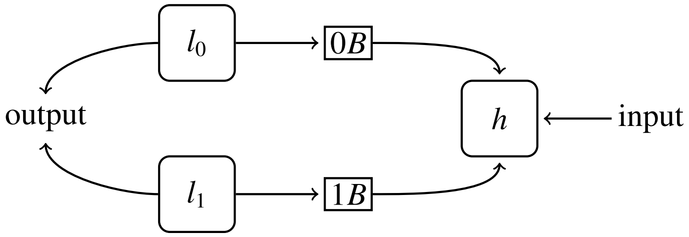


Figure 20: Composability Example Overview

The behavior of the components is as follows:

- h: The h component reads a bit from the input buffer and clears buffer $0B$ if the read bit was 0 or clears buffer $1B$ if the read bit was 1.
- l_0: The l_0 component first fills the $0B$ buffer and proceeds to loop involving writing to $0B$ (and potentially blocking) followed by writing a 0 to the output buffer.
- l_1: Similar to l_0 except it fills and writes to $1B$ buffer instead. Also it writes a 1 to the output buffer as opposed to a 0.

The argument that this is an example of the failure of composition involves two elements: firstly that some security property is satisfied in the components, and secondly, that the same property is not satisfied in the complete system. The security property relevant here is absence of a high level input leaking to low level output. The input to h is high whereas the output from l_0 and l_1 is low.

The property is satisfied in l_0 and in l_1 as they do not read anything but write to the output or to the $0B$ and $1B$ buffers. The property is satisfied by h as it does not produce any outputs. Note that these arguments are indeed very flimsy as the exact nature of communication involving the buffers has not been formally described.

Consider now the system as a whole. The l_0 and l_1 components fill their respective buffers and begin to block while attempting to write another bit. At this point, h receives either 0 or 1 from the input. If 0 is received, h reads from buffer $0B$ and unblocks l_0 which then writes a 0 to the output buffer. Alternatively 1 is read and the end result is that 1 is written to the output by l_1. Such a situation certainly describes the high input leaking to the low output and if one accepts the arguments that no such occurrence was present in the individual components, then one must admit that the composition failed to preserve the property of lack of leakage.

Our attempts at modeling the situation, however (see Appendix D), suggest that it is not entirely clear that the components satisfy the property or more specifically whether the property can even be formulated in the components. Specifically it might not be possible to reasonably consider the components individually and separate from the entire system given their connections through the buffers to other components. It might even be argued that the diagram we provided (Figure 22) of the system is deceiving in that the arrow between l_0 and $0B$ points towards $0B$ and not inversely, suggesting that *information* only flows from l_0 to $0B$. The fact that h can unblock l_0 through the buffer suggest some kind of influence in the reverse direction.

Though we might have some arguments against the example being a valid demonstration of the lack of preservation of a security policy under composition, it is not bring us at all to any conclusion on composition. McCullough's partial resolution on the problem is a conclusion that finite (and specifically blocking) buffers need to be not included in the system composition to preserve noninterference [McC88]. Though such conclusions might be useful on a certain implementation level as guidelines for designers against specific practices (like uses of finite blocking buffers), it is not particularly enlightening on a more abstract and general level. We strive for in this section for such more abstract and general conclusions about composability of systems.

4.2 Composition and Projection

The question of preservation of non-interference is naturally tied with the preservation of some structure in the composition process. If one makes the claim that noninterference is preserved in a composed system, or more specifically that this is something that *should* happen, one really notes that there is (or ought to be) a close structural connection between the subcomponents and the resulting composed system. Specifically to non-interference the connection that is relevant is between runs and perception ($\sim$) in subcomponents and the composition. We begin our composability discussion with the formalization of these connections.

Definition 57 (Run Projection). *A* run projection *(denoted by π) is a function transforming a run in system S to to a run in system S_2. What we are interested about in terms of run projections is their potential preservation properties:*

- ***validity-preservation**: A projection π* preserves validity *if for every run r in S, the run $\pi(r)$ is a valid run in S_2.*
- ***=-preservation**: A projection π* preserves = *if for every group of agents $G \subseteq A$ (of S), every $=_G$-equivalent runs r, r' in S are mapped to two $=_G$-equivalent runs in S_2. The final bit of subtlety here is that, once more, $=_G$ are potentially different in the two systems.*

We will write a composition function as C. Such a composition function will take systems S_1 and S_2 to produce system S. The agents present in each subsystem are assumed to be present in S. Since state and run equivalences are required for our non-interference definitions, we require that composition C also determines the appropriate = and $\sim$ in the composed system (usually based closely on the = and $\sim$ in the subsystems).

One final bit of structure connection we might require is a logical preservation of knowledge or $\sim$ by a pair of projections in the context of system composition C.

Definition 58 (State Equivalence Preservation). *If C composes systems S_1 and S_2 to system S, the projections π_1 (from S to S_1), π_2 (from S to S_2)* preserve $\sim$ *iff*

For every pair of runs r, r' in S, which are mapped to $r_1 = \pi_1(r), r_1' = \pi_1(r')$ and to $r_2 = \pi_1(r), r_2' = \pi_1(r')$, with $r, r', r_1, r_1', r_2, r_2'$ ending in states $q, q', q_1, q_1', q_2, q_2'$ respectively, we have:

- *For every group of agents $G \subseteq A$ (in S) $q \sim_G q'$ iff both $q_1 \sim_{1_G} q_1'$ and $q_2 \sim_{2_G} q_2'$.*

We must be careful here as some agents might not be present in some subsystems. That is, if we are composing two systems that both contain agent a but do not share agents a_1 and a_2 which are present in systems S_1 and S_2 respectively, the state equivalence of system S_1 does not by definition handle equivalence of states for agents $\{a, a_2\}$. To alleviate this problem we make the following remark:

Remark 59. *In context of system composition we assume that $\sim$ and $=$ are extended to handle all agents of both systems. Given a set of agents G_d not originally present in system S, we assume $q \sim_{G \cup G_d} q'$ iff $q \sim_G q'$ for every pair of states $q, q' \in Q$ of S and for every group of agents G of S. Likewise $r =_{G \cup G_d} r'$ iff $r =_G r'$ for every pair of runs in S. Note that this means $q \sim_{G_d} q'$ and $r =_{G_d} r'$ for every pair of states and runs. We refer to such agents as* dummy agents *where dummy refers to not only the fact that such an agent serves as placeholder but also for the fact that the dummy cannot tell the difference between any states and any runs.*

We note that Remark 59 gives us the ability to reason about non-present agents together with present ones. For example, consider a system S_1 with agents $\{a, a_1\}$ and system S_2 with agents $\{a, a_2\}$. The definition of $\sim$ preservation would state:

- $q \sim_{\{a_1\}} q'$ iff $q_1 \sim_{1_{\{a_1\}}} q'_1$. Definition of $\sim$ preservation would require also $q_2 \sim_{2_{\{a_1\}}} q'_2$ but the statement is always true for any two states of S_2 as a_1 was not originally present in S_2 via Remark 59.
- $q \sim_{\{a,a_1\}} q'$ iff both $q_1 \sim_{1_{\{a,a_1\}}} q'_1$ and $q_2 \sim_{2_{\{a\}}} q'_2$ as the statement $q_2 \sim_{2_{\{a\}}} q'_2$ is equivalent to $q_2 \sim_{2_{\{a,a_1\}}} q'_2$ by Remark 59.

Having such formalism, we can properly describe the somewhat intuitive belief that non-interference is (or should) be preserved via connections in the structure of subcomponents and the composition. First let us describe what we mean by preservation of non-interference.

Definition 60 (Noninterference Preservation). *A composition C* preserves non-interference *if every non-interference assertion $G' \not\Rightarrow G$ true in each subsystem is true in the composed system.*

Theorem 61 (Projected Noninterference). *Any composition C function taking in systems S_1 and S_2 and producing composed system S, preserves non-interference if there exist two run projections π_1 and π_2 such that for all i, π_i run-preserves and =-preserves from S to S_i, and that the projections preserve $\sim$ ($\sim$-preservation is a property of the pair of projections).*

Proof. Assume the aforementioned projections exist but C does not preserve non-interference. That is, there is a non-interference assertion $G' \not\Rightarrow G$ true in both subsystem but $G' \not\Rightarrow G$ is not true in the composed system S. By definition of non-interference, there exist two $=_{A-G'}$-equivalent runs r, r' in S that end in non $\sim_G$-equivalent states.

Consider run pairs projected to both systems, $r_1 = \pi_1(r)$, $r'_1 = \pi_1(r')$, $r_2 = \pi_2(r)$, and $r'_2 = \pi_2(r')$. Their validity as runs and their =-equivalence can be made in their respective subsystems via the validity and = preservation properties of the projections. Since $\sim$ is preserved by the pair of projections and since r, r' end in non $\sim_G$-equivalent states, we know that either r_1,r'_1 end in non $\sim_G$-equivalent states or r_2,r'_2 do. Thus we have $G' \Rightarrow G$ in at least one of the subsystems contradicting our assumption. □

4.3 Composition $C_{\neg cc,i}$

We demonstrate the application of Theorem 61 with an example of a simple type of composition of two systems which we refer to as non-concurrent, independent composition (or $C_{\neg cc,i}$). We consider two systems both modeling two users, h and l (for high and low). A noninterference policy $h \not\Rightarrow l$ is to be assumed in place for both systems. We describe the structure of this composition based on some motivating factors:

- *Two users interact with two systems*: Users h and l are modeled both in system S_1 and system S_2.
- *The systems model different kinds actions*: The set of available actions to users in S_1 is different from that in S_2. We refer to the actions available as Act_i for system S_i. Note that the special null action ϵ needs to be available to both agents in both systems (this will be described shortly).

- *The systems do not interact with each other*: Composed state is of the form $Q = Q_1 \times Q_2$ where Q_i is understood to be the set of states of S_i.
- *Agents interact with one system at a time*: Transitions of the composed system $S = C_{\neg cc,i}(S_1, S_2)$ are 2-tuples (actions of each of the two agents). To make sure we can define the composed transition function, we make the following restrictions on the transition functions of the component systems:

  For any state $q \in Q_i$ and for any action a of agent h at q (and similarly for l), the move vector (a, ϵ) (similarly (ϵ, a)) must be available at q. Also, the move vector (ϵ, ϵ) must be available at any state.

We formalize the system assumptions discussed above (with a few additions) in the following definition.

Definition 62 (Composability). *A system $S = \langle\langle A, Q, q, Act, act, \delta\rangle, \sim, =\rangle$ is $C_{\neg cc,i}$-*composable *iff*

- *$A = \{h, l\}$,*
- *$\epsilon \in act(q, a)$ for any agent a and at any state q, and*
- *$=$ is a natural run-equivalence.*

*Also, two systems $S_1 = \langle\langle A_1, Q_1, q_1, Act_1, act_1, \delta_1\rangle, \sim_1, =_1\rangle$, $S_2 = \langle\langle A_2, Q_2, q_2, Act_2, act_2, \delta_2\rangle, \sim_2, =_2\rangle$ are together $C_{\neg cc,i}$-*composable *iff*

- *S_1 and S_2 are individually $C_{\neg cc,i}$-composable, and*
- *$Act_1 \cap Act_2 = \emptyset$.*

Given two $C_{\neg cc,i}$-composable systems $S_1 = \langle\langle A_1, Q_1, q_1, Act_1, act_1, \delta_1\rangle, \sim_1, =_1\rangle$, $S_2 = \langle\langle A_2, Q_2, q_2, Act_2, act_2, \delta_2\rangle, \sim_2, =_2\rangle$ let use define $C_{\neg cc,i}(S_1, S_2) = S = \langle\langle A, Q, q, Act, act, \delta\rangle, \sim, =\rangle$:

- $A = A_1 = A_2 = \{h, l\}$,
- $Q = Q_1 \times Q_2$, and the starting state $q = \langle {q_1 \atop q_2} \rangle$,
- $\langle {q_1 \atop q_2} \rangle \sim_G \langle {q_1' \atop q_2'} \rangle$ iff both $q_1 \sim_{1_G} q_1'$ and $q_2 \sim_{2_G} q_2'$ for any group of agents $G \subseteq A$,

- $=$ is the natural run equivalence,
- $Act = Act_1 \cup Act_2$,
- $act(\langle {q_1 \atop q_2} \rangle, a) = act_1(q_1, a) \cup act_2(q_2, a)$ for every agent a, and
- $\delta\left(\langle {q_1 \atop q_2} \rangle, [a_1,a_2]\right) = \langle {\delta_1(q_1, d_1([a_1,a_2])) \atop \delta_2(q_2, d_2([a_1,a_2]))} \rangle$ where d_i is defined as follows:

$$d_i\left([a_1,a_2]\right) = \begin{cases} [a_1,a_2] & a_1 \in Act_i, a_2 \in Act_i \\ [a_1,\epsilon] & a_1 \in Act_i, a_2 \notin Act_i \\ [\epsilon,a_2] & a_1 \notin Act_i, a_2 \in Act_i \\ [\epsilon,\epsilon] & a_1 \notin Act_i, a_2 \notin Act_i \end{cases}$$

Preservation of Noninterference

Theorem 63 (Noninterference Preservation). *$C_{\neg cc,i}$ preserves noninterference.*

Proof. Let us define a project π_1 (and similarly π_2) for runs in the following manner:

$$\begin{aligned} \pi_1\left(\epsilon\right) &= \epsilon \\ \pi_1\left(r.[a_1,a_2]\right) &= \pi_1\left(r\right).d_1\left([a_1,a_2]\right) \end{aligned}$$

Claim. *If r in S ends at state $\langle {q_1 \atop q_2} \rangle$ then $\pi_1(r)$ (and similarly $\pi_2(r)$) ends at state q_1 (similarly q_2).*

Proof. We show this by induction on the length of the run r. We also consider π_1 without loss of generality.

- $|r| = 0$: The composed start state is defined as $\langle {q_1 \atop q_2} \rangle$ where q_1 is the start state in S_1.
- $|r| = n$: Assume that for a run r of length n ending in $\langle {q_1 \atop q_2} \rangle$, the run projected to S_1, $\pi_1(r)$ ends in q_1.
- $|r| = n + 1$: Let us say that after n steps ($r' = r[1..n]$), the shorter run r' ends at $\langle {q_1 \atop q_2} \rangle$. By assumption, the projection $\pi_1(r')$ of this run ends at q_1. By definition of

the composed δ, the next state reached by r is $\langle{\delta_1(q_1,d_1([a_1,a_2])) \atop \delta_2(q_2,d_2([a_1,a_2]))}\rangle$ (assuming the last move vector in r is $[a_1,a_2]$) or specifically the first component of the next state is $\delta_1(q_1, d_1([a_1,a_2]))$. We see that the projection also makes us take the move vector $d_1([a_1,a_2])$ and thus the next state in S_1 via the projected run is $\delta_1(q_1, d_1([a_1,a_2]))$.

□

Claim. *π_i preserve validity.*

Proof. Let us consider π_1 without the loss of generality. The validity of a run can be claimed if the starting state is correct and that the move vectors at each point are actually allowable at that point. Let us consider $r = q_0, \widehat{i_0}, q_1, \cdots, \widehat{i_{n-1}}, q_n$ and $r' = \pi_1(r) = q'_0\widehat{i'_0}, q'_1, \cdots, \widehat{i'_{n-1}}, q'_n$.

- **start state**: Any run r in S begins with $\langle{q_1 \atop q_2}\rangle$ where q_1 is the start state of S_1.
- **move vectors**: Consider a move vector $[a_1,a_2]$ in r taken at state $\langle{q_1 \atop q_2}\rangle$. Without loss of generality let us consider the action by the first agent. If $a_1 \notin Act_1$ then the first component of $d_1([a_1,a_2])$ is ϵ which is a valid action in S_1 for the agent given the assumption that S_1 is $C_{\neg cc,i}$-composable. Otherwise $a_1 \in Act_1$ and thus d_1 leaves the action unchanged and we can see it is allowable since by definition of the composed actions $act(\langle{q_1 \atop q_2}\rangle, a) = act_1(q_1, a) \cup act_2(q_2, a)$ (remember that the set of actions is disjoint between the two systems).

  In the projection, d_1 $([a_1,a_2])$ is taken at state q_1. The ϵ components of the move vector are allowable at any state so we only need to worry about non-ϵ actions.

□

Claim. *$\pi_i\,(r_1)$ preserves $=$.*

Proof. Consider S_1 without the loss of generality. We already established that $\pi_1\,(r_1)$ and $\pi_1\,(r_2)$ start in the starting state of S_1. Next we need to show that for two move vectors $[a_j,a'_j]$, $[b_j,b'_j]$ (where j refers to the j^{th} move vector in the run), we have $d_1([a_j,a'_j]) =_G d_1([b_j,b'_j])$. Since $r_1 =_G r_2$ (for some group of users G) in S, we know $[a_j,a'_j] =_G [b_j,b'_j]$. Since d_1 takes any action outside of Act_1 to ϵ, both or none of a_j, b_j (and/or a'_j, b'_j

depending on agents in G) must be in Act_1. If both, then d_1 does not affect G's components and thus they remain equal for G. If neither is in Act_1 then both are changed to ϵ by d_1 and thus also equal. □

Claim. *π_1 and π_2 preserve $\sim$.*

Proof. This comes directly from the first claim and the definition of the composed $\sim$. □

This gives us everything necessary to claim that $C_{\neg cc,i}$ preserves noninterference by Theorem 61. □

5 Conclusions

We presented a new formulation of noninterference based directly to the intuitive basis of the interplay between distinguishability of action and output. Our notion was at the same time general enough to express existing definitions in the same framework but also intricate enough to simplify the handling of intransitive information flow. We also explored the issue of composability of systems and described a sufficient condition for proving the preservation of noninterference under composition of systems.

5.1 Potential Future Work

Though the work presented explores a wide range of issue and makes conclusions on some, there are many avenues for further work.

- The Unwinding Theorem we present for our noninterference assertions is too weak. That is, it requires too much of a system to make a conclusion about noninterference. We saw in Example 43 that a noninterference assertion is true but cannot be proven using the Unwinding Theorem. It would be of great use to develop an alternate and stronger theorem capable of proving noninterference assertions in larger number of situations.
- The composition example presented in Section 4.3 is perhaps a little too simplistic. To ascertain the usability of the Projected Noninterference theorem, we would almost certainly need a notion of composition that is sufficiently complex for modeling more realistic situations.

References

[AHK98] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Lecture Notes in Computer Science*, 1536:23–60, 1998.

[Bis03] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, Reading, MA, USA, 2003.

[BL76] D.E. Bell and L. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. *MITRE technical report, MITRE Corporation, Bedford Massachusetts*, 2997:ref A023 588, 1976.

[BP03] Michael Backes and Birgit Pfitzmann. Intransitive non-interference for cryptographic purposes. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 140, Washington, DC, USA, 2003. IEEE Computer Society.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, New York, NY, USA, 1977. ACM Press.

[DL73] D.E.Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations. *MITRE technical report, MITRE Corporation, Bedford Massachusetts*, 1973.

[Fei80] R. J. Feiertag. A tecnique for proving specifications are multilevel secure. Technical Report CSL-109, Computer Science Laboratory, SRI International, Menlo Park, CA, Jan 1980.

[FLR77] R. J. Feiertag, K. N. Levitt, and L. Robinson. Proving multilevel security of a system design. In *SOSP '77: Proceedings of the sixth ACM symposium on Operating systems principles*, pages 57–65, New York, NY, USA, 1977. ACM Press.

[GM82] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[GM84] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pages 75–86, 1984.

[HY87] J. Thomas Haigh and William D. Young. Extending the noninterference version of MLS for SAT. *IEEE Trans. Softw. Eng.*, 13(2):141–150, 1987.

[JT88] Dale M. Johnson and F. Javier Thayer. Security and the composition of machines. In *CSFW '88: Proceedings of the IEEE Computer Security Foundations Workshop*, pages 72–89, 1988.

[McC88] Daryl McCullough. Noninterference and the composability of security properties. *1988 IEEE Symposium on Security and Privacy*, 00:177, 1988.

[McC90] Daryl McCullough. A hookup theorem for multilevel security. *IEEE Trans. Softw. Eng.*, 16(6):563–568, 1990.

[McL85] John McLean. A comment on the 'basic security theorem' of bell and lapadula. *Information Processing Letters*, 20(2):67–70, 1985.

[McL94] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 79–93, 1994.

[Mil76] Jonathan K. Millen. Security kernel validation in practice. *Commun. ACM*, 19(5):243–250, 1976.

[Pin95] S. Pinsky. Absorbing covers and intransitive non-interference. In *SP '95: Proceedings of the 1995 IEEE Symposium on Security and Privacy*, page 102, Washington, DC, USA, 1995. IEEE Computer Society.

[RG99] Roscoe and Goldsmith. What is intransitive noninterference? In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[RMMG01] Peter Ryan, John D. McLean, Jonathan K. Millen, and Virgil D. Gligor. Non-interference: Who needs it? In *CSFW '01: Proceedings of the IEEE Computer Security Foundations Workshop*, page 237, 2001.

[Rus81] J. Rushby. Design and verification of secure systems. In *Proceedings of the 8th ACM Symposium on Operating Systems Principles (SOSP)*, volume 15, pages 12–21, 1981.

[Rus82] John M. Rushby. Proof of separability: A verification technique for a class of a security kernels. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 352–367, London, UK, 1982. Springer-Verlag.

[Rus92] John Rushby. Noninterference, Transitivity, and Channel-Control Security Policies. Technical report, Compuster Science Laboratory, SRI International, Menlo Park, CA, Dec 1992.

[WJ90] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161, 1990.

[ZL95] A. Zakinthinos and E. S. Lee. The composability of non-interference [system security]. In *CSFW '95: Proceedings of The Eighth IEEE Computer Security Foundations Workshop*, page 2, Washington, DC, USA, 1995. IEEE Computer Society.

[ZL97] Aris Zakinthinos and E. Stewart Lee. A general theory of security properties. In *Proceedings of the 18th IEEE Computer Society Symposium on Research in Security and Privacy*, 1997.

Index

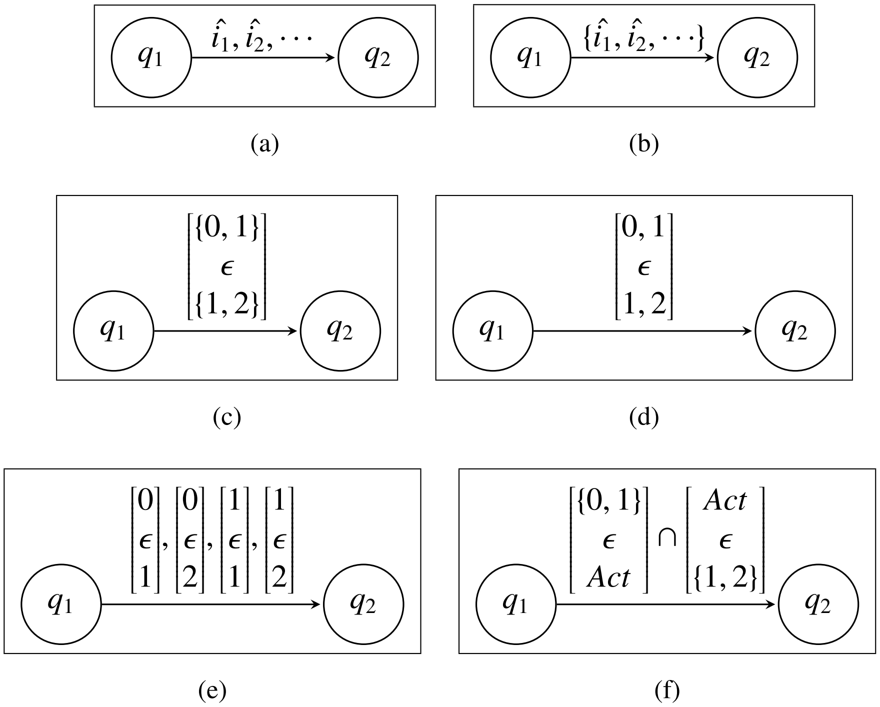
Appendix

A Notation

- $\sim$ - State equivalence (see Definition 30)
- $\doteq$ - Move vector equivalence (see Definition 36)
- $=$ - Run equivalence (see Definition 33)
- $r, r', r_x, ...$ - Runs of a state machine.
- $q, q', q_x, ...$ - States of a state machine.
- $\hat{i}, \hat{i'}, \hat{i_x}, ...$ - Move vectors.
- $\frac{S_1 \quad S_2 \quad \cdots \quad S_n}{C}$ - Expresses that if statements S_1, S_2, ..., and S_n are all true then so is C.
- G^+ - Any set G' such that $G \subseteq G'$. Inside a statement, $S(G^+)$ holds is equivalent to $\forall G'$ with $G \subseteq G'$, $S(G')$ holds.
- G^- - Any set G' such that $G' \subseteq G$.
- $|r|$ - The length of a run r.
- $r[j]$ - The j^{th} move vector in run r.
- $\lceil r \rceil$ - The last state reached in run r (see Definition 6).
- $r[1..n]$ - A slice of a run r containing the first n move vectors.
- $\rightsquigarrow$ - Noninterference policy (see Definition 45).
- $G' \Rightarrow G$ (or $G' \not\Rightarrow G$) - Agents G' do (or not) interfere with agents G (see Definition 41).
- $a \downarrow$ - Agents that are allowed to interfere with a based on some policy (see Definition 45).
- $a^{\perp}$ - Agents that are not allowed to interfere with a based on some policy (see Definition 45).
- ϵ - A special null action, a special error state, or an empty run (of no actions).
- $a.r$ - A concatenation of an action or move vector a with a run r.
- $r.a$ - A concatenation of an action or move vector a onto the end of run r.

B c.g.s. Transition Specification

We describe the various transition specification conventions in Figure 21.


q_1
$\hat{i}_1, \hat{i}_2, \cdots$
q_2
q_1
$\{\hat{i}_1, \hat{i}_2, \cdots\}$
q_2
q_1
$\begin{bmatrix}\{0,1\}\\ \epsilon \\ \{1,2\}\end{bmatrix}$
q_2
q_1
$\begin{bmatrix}0,1\\ \epsilon \\ 1,2\end{bmatrix}$
q_2
q_1
$\begin{bmatrix}0\\ \epsilon \\ 1\end{bmatrix}, \begin{bmatrix}0\\ \epsilon \\ 2\end{bmatrix}, \begin{bmatrix}1\\ \epsilon \\ 1\end{bmatrix}, \begin{bmatrix}1\\ \epsilon \\ 2\end{bmatrix}$
q_2
q_1
$\begin{bmatrix}\{0,1\}\\ \epsilon \\ Act\end{bmatrix} \cap \begin{bmatrix}Act\\ \epsilon \\ \{1,2\}\end{bmatrix}$
q_2

(a)

(b)

(c)

(d)

(e)

(f)

Figure 21: Concurrent Game Structure Shorthand Transition Specifications

If we include a set of move vectors on a transition as in Figure 21b, we mean that same transition is taken upon any of the move vectors in the set. Sometimes we omit the set brackets as in Figure 21a (thus Figure 21a and Figure 21b demonstrate the same thing). We may also include sets of actions inside move vectors as in Figure 21c. This has the same meaning as specifying a set of move vectors where in each, a single action is taken from each of the action sets in the original specification as in Figure 21e (that is the transitions in Figure 21c and Figure 21e are equivalent). Sometimes we will leave off the set brackets for action set specification as in Figure 21d. Since we allow specification of move vector sets on transitions, we can also perform some set operations on such sets as in Figure 21f which is equivalent to Figure 21d.

C c.g.s. Cross-Product Construction

Specification of complex systems or problematic. We will sometimes use a cross-product construction to first describe two pieces of a state machine and then take the *cross product* of the pieces to create our full machines.

Definition 64 (Cross Product Production). *The* cross product construction *specifies a concurrent game structure from two components that include a state space, the allowable actions, and the transition function.*

Given a set of agents A, a set of actions Act, a pair of state sets Q_1, Q_2, a pair of starting states q_1, q_2, a pair of allowable action functions act_1, act_2 (where $act_1 : Q_1 \times A \rightarrow P(Act)$ and $act_2 : Q_2 \times A \rightarrow P(Act)$), and a pair of transition functions δ_1, δ_2 (where $\delta_1 : Q_1 \times Act^k \rightarrow Q_1$ and $\delta_2 : Q_2 \times Act^k \rightarrow Q_2$), the cross product of the pairs is a concurrent game structure $S = \langle A, Q, q_0, Act, act, \delta \rangle$ defined as follows:

- $Q = Q_1 \times Q_2$.
- $q_0 = \langle q_1, q_2 \rangle$.
- $act(\langle q_1, q_2 \rangle, a) = act_1(q_1, a) \cap act_2(q_2, a)$ *for every pair of states* $q_1 \in Q_1, q_2 \in Q_2$ *and every agent* $a \in A$.
- $\delta(\langle q_1, q_2 \rangle, \hat{\imath}) = \langle \delta_1(q_1, \hat{\imath}), \delta_2(q_2, \hat{\imath}) \rangle$ *for every pair of states* $q_1 \in Q_1, q_2 \in Q_2$ *and every move vector* $\hat{\imath} \in D(\langle q_1, q_2 \rangle)$.

An example of such a construction can be seen in Example 44.

D Composition Example Specification

One of the most commonly cited examples of composition problems is the one using blocking buffers. The following example is derived from [McC88]. The situation, however, is never described formally as a state machine. We do construct a state machine for the situation below as an exercise.

We begin by considering a system that appears to be a composition of several sub-systems. It might be odd to attack the problem of composition by considering an already composed system instead of starting with the pieces first but as we will eventually see, it is not obvious how to define the sub-systems individually.

Consider a system composed of three smaller components, and four 1-bit buffers with the following general behaviors (see Figure 22):

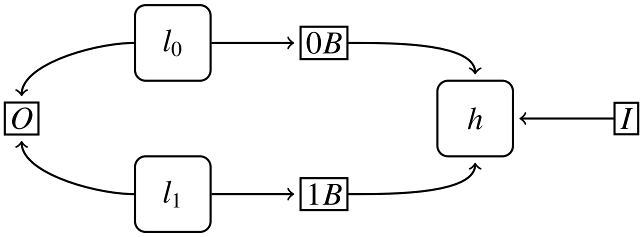


Figure 22: Composability Example

- h: The h component reads a bit from buffer I and clears buffer $0B$ if the read bit was 0 or clears buffer $1B$ if the read bit was 1.
- l_0: The l_0 component first fills the $0B$ buffer and proceeds to loop involving writing to $0B$ (and potentially blocking) followed by writing a 0 to buffer O.
- l_1: Similar to l_0 except it fills and writes to $1B$ buffer instead. Also it writes a 1 to O buffer as opposed to a 0.

The overall system is designed to accept input (via the I buffer) and provide output via the O buffer. The relevant question is whether information flows from I to O. If the input is designated as **high** and the output as **low**, we can consider the assertion that **high** cannot interfere with **low**.

D.1 Model

We model the situation with a state-space composed of a cross product of several component state spaces. The components of the states should make it easier to ascertain the system components and to consider them outside of the composed system. We will assume that the three subsystems contain each one agent and contain a local state with that agent having effect on that local state. In addition we let the buffers be agents with additional local state. We do this to resolve the issue of placement of the buffer-relate state components: does $0B$ buffer-related state belong to l_0 or h? We opt for neither and let the buffers be handled by their own agent and having their own state space.

The issue of state space association to system is only partially resolved as there is still a need, for example, to share some state space between l_0 and $0B$. We introduce small state space pieces for all such interfaces (see Figure 23). Using such a structure, we may be able to reason about subsystems on their own in a consistent fashion and explore the real flow information in the system.

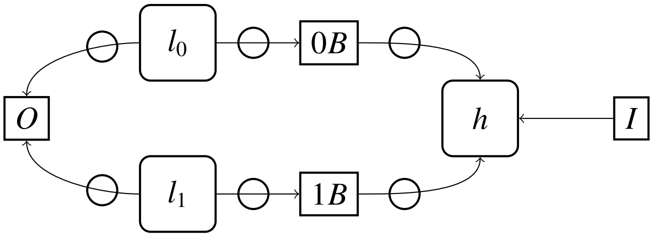


Figure 23: Composability Example with State Components

We begin the formal model by first describing precisely the state space and the actions available for each agent.

Let us define various state space components:

- $Q_I = \{I = 0, I = 1\}$: Local state of buffer I. The values refer to the content of the buffer and should be self-explanatory.
- $Q_O = \{O = 0, O = 1\}$: local state of buffer O.
- $Q_{0B} = \{b_0, \neg b_0\}$: local state of buffer $0B$. The values here denote whether or not the buffer is filled. We do not care of the content of the buffer at all for this example. b_0 means the buffer

is *blocked* or filled and $\neg b_0$ denotes that the buffer is not blocked or empty.

- $Q_{1B} = \{b_1, \neg b_1\}$: local state of buffer $1B$.

- $Q_{l_0} = \{i_{00}, i_{01}, i_{02}\} \times \{R_0 (O = 0), R_0 (b_0), R_0 (\epsilon)\}$: local state of component l_0. The first sub-component refers to the state of the three-state state machine that governs the general behavior of this component (described Figure 24). The second component is the interface between l_0 and $0B$. It describes l_0's request (R_0) for a specific change in $0B$ where ϵ denotes no request.

- $Q_{l_1} = \{i_{10}, i_{11}, i_{12}\} \times \{R_1 (O = 1), R_1 (b_1), R_1 (\epsilon)\}$: local state of component l_1. The structure is similar to that of l_0 except the requests now refer to the interface with $1B$ instead (see Figure 24).

- $Q_{i_h} = \{i_h\} \times \{R_h (\neg b_0), R_h (\neg b_1)\}$: local state of component h. There is only one state in the state machine describing h's behavior. In addition h has an interface to the $0B$ and $1B$ buffers (namely requests to empty them).

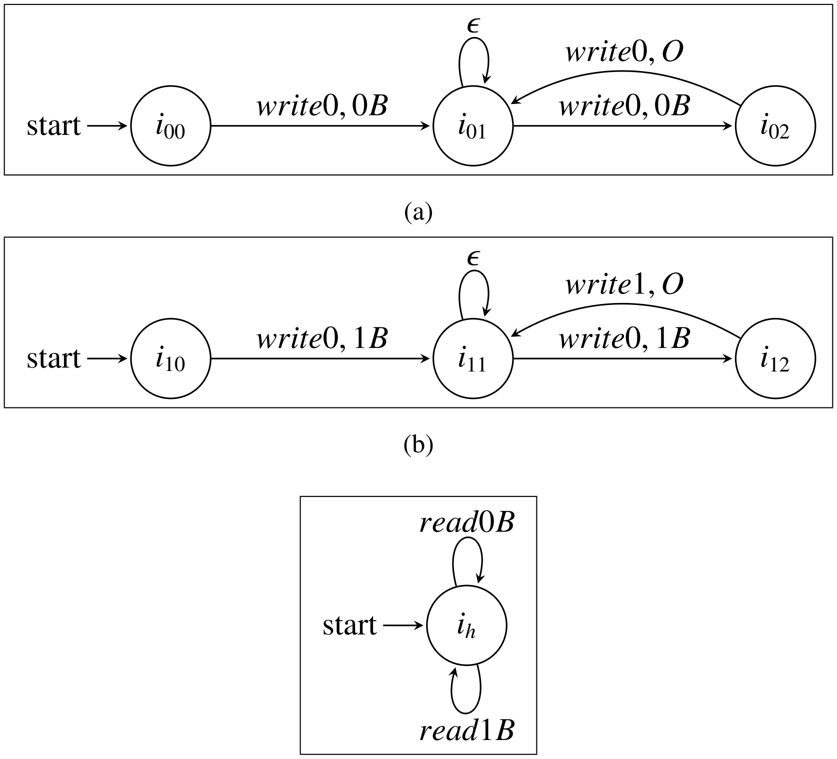
We will thus consider our system consisting of global state $Q = Q_I \times Q_O \times Q_{0B} \times Q_{1B} \times Q_{l_0} \times Q_{l_1} \times Q_{i_h}$.

Next we describe the availability of actions, the transitions, and their effect on the various state spaces in a high-level manner.

agent	availability	action	effect
l_0	(i_{00} or i_{01}) and $R_0\,(\epsilon)$	$write\ 0, 0B$	$R_0\,(b_0)$
	i_{02} and $R_0\,(\epsilon)$	$write\ 0, O$	$R_0\,(O = 0)$
	$R_0\,(b_0)$ and b_0	ϵ	none
	$R_0\,(b_0)$ and $\neg b_0$	λ	$R_0\,(\epsilon)$
	$R_0\,(O = 0)$	λ'	$R_0\,(\epsilon)$
l_1	(i_{10} or i_{11}) and $R_1\,(\epsilon)$	$write\ 0, 1B$	$R_1\,(b_1)$
	i_{12} and $R_1\,(\epsilon)$	$write\ 1, O$	$R_1\,(O = 1)$
	$R_1\,(b_1)$ and b_1	ϵ	none
	$R_1\,(b_1)$ and $\neg b_1$	λ	$R_1\,(\epsilon)$
	$R_1\,(O = 1)$	λ'	$R_1\,(\epsilon)$
h	$I = 0$	$read\ 0B$	$R_h\,(\neg b_0)$
	$I = 1$	$read\ 1B$	$R_h\,(\neg b_1)$
I	always	$S\ (I = 0)$	$I = 0$
	always	$S\ (I = 1)$	$I = 1$
O	$R_0\,(O = 0)$ and not $R_1\,(O = 1)$	$S\ (O = 0)$	$O = 0$
	$R_1\,(O = 1)$	$S\ (O = 1)$	$O = 1$
$0B$	$R_0\,(b_0)$ and not $R_h\,(\neg b_0)$	$S\ (b_0)$	b_0
	$R_h\,(\neg b_0)$ and not $R_0\,(b_0)$	$S\ (\neg b_0)$	$\neg b_0$
	$R_0\,(b_0)$ and $R_h\,(\neg b_0)$	$S\ (b_0)$ and $S\ (\neg b_0)$	$\neg b_0$
$1B$	$R_1\,(b_1)$ and not $R_h\,(\neg b_1)$	$S\ (b_1)$	b_1
	$R_h\,(\neg b_1)$ and not $R_1\,(b_1)$	$S\ (\neg b_1)$	$\neg b_1$
	$R_1\,(b_0)$ and $R_h\,(\neg b_1)$	$S\ (b_1)$ and $S\ (\neg b_1)$	$\neg b_1$

The meaning of the various actions is as follows:

- ϵ - This represents a null action that has no effects (that is, the state component affected does not change).
- *write* $0, 0B$ - This represents writing a 0 to buffer $0B$. The effect of such an action is the request to write to the buffer. Similarly for writing 1 and to buffer $1B$.
- λ - This represents an buffer filling request being processed. The result is that the request is emptied.



(a)

(b)

(c)

Figure 24: Component Transitions and Effect on State Components

- λ' - Similarly as above but for the output buffer.
- *read* $0B$ - This represents reading from buffer OB and similarly for $1B$. The effect is that the a request is made to empty the buffers.
- $S(I = 0)$ - This represents the setting of buffer I to 0. Similarly for $S(I = 1), S(O = 0), S(O = 1)$.
- $S(b_0)$ - This represents the filling of buffer $0B$. Similarly for $S(b_1)$.
- $S(\neg b_0)$ - The emptying of buffer $0B$ and similarly for $S(\neg b_1)$.

The effects on the table describe the value of state components after each transition. The effects described are always local to the state space of the component owning each action or a shared component (such as some components owned by the buffers). For example consider agent h. We see that the action *read* $0B$ is available as long as the system is in the state with the Q_I component equal to $I = 0$. The effect of such an action is that a request for emptying the buffer is that the value of the Q_{i_h} is set to $R_h(\neg b_0)$.

Having a c.g.s. model for the system, it becomes apparent that considering the components separately from the whole system is not straightforward. Though we presented the component state spaces in an attempt to make this task easier, significant amount of transitions for the components depend on state of other components especially involving the buffers. For example the ϵ and λ actions of l_0 depend on the $\{b_0, \neg b_0\}$ component of the buffer's state space. Not only this, the buffers' actions depend on state space of two different components. For example the $S(b_0)$ and $S(\neg b_0)$ actions of $0B$ depend on state space of both l_0 *and* h (specifically $R_0(b_0)$ and $R_h(\neg b_0)$). These dependencies prevent us from separating the whole system into individual components.