RICE UNIVERSITY

Linear vs. Branching Time: A Semantical Perspective

by

Sumit Nain

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE **Master of Science**

APPROVED, THESIS COMMITTEE:

[illegible]

Professor Moshe Y. Vardi, Chair
Karen Ostrum George Professor
Department of Computer Science

[illegible]

Professor Keith Cooper
Department of Computer Science

[illegible]

Assistant Professor Walid Taha
Department of Computer Science

HOUSTON, TEXAS

MARCH 2009

Abstract

Linear vs. Branching Time: A Semantical Perspective

by

Sumit Nain

The discussion of the relative merits of linear versus branching-time goes back to early 1980s. The dominating belief has been that the linear-time framework is not expressive enough semantically, marking linear-time logics as weak. Here we examine this issue from the perspective of process equivalence, one of the most fundamental notions in concurrency theory. We postulate three principles that we view as fundamental to any discussion of process equivalence. First, we take contextual equivalence as the primary notion of equivalence. Second, we require the description of a process to fully specify all relevant behavioral aspects of the process. Finally, we require observable process behavior to be reflected in input/output behavior. Under these postulates the distinctions between the linear and branching semantics tend to evaporate. Applying them to the framework of transducers, we show that our postulates result in a unique notion of process equivalence, which is trace based, rather than tree based.

Acknowledgments

I am very grateful for the support and guidance of my advisor, Dr. Moshe Vardi. I would like to thank the other members of my thesis committee, Dr. Walid Taha and Dr. Keith Cooper. Finally, I must also thank my office-mates, Deian Tabakov and Seth Fogarty.

Contents

Chapter 1

Introduction

One of the most significant recent developments in the area of formal design verification is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* systems [19, 44, 55, 65]. In temporal-logic *model checking*, we verify the correctness of a finite-state system with respect to a desired property by checking whether a labeled state-transition graph that models the system satisfies a temporal logic formula that specifies this property (see [21]). Model-checking tools have enjoyed a substantial and growing use over the last few years, showing ability to discover subtle flaws that result from extremely improbable events. While early on these tools were viewed as of academic interest only, they are now routinely used in industrial applications [31].

A key issue in the design of a model-checking tool is the choice of the temporal language used to specify properties, as this language, which we refer to as the *temporal property-specification language*, is one of the primary interfaces to the tool. (The other primary interface is the modeling language, which is typically the hardware description language used by the designers). One of the major aspects of all temporal languages is their underlying model of time. Two possible views regarding the nature of time induce two types of temporal logics [43]. In *linear* temporal logics, time is treated as if each moment in time has a unique possible future. Thus, linear temporal logic formulas are interpreted over linear sequences and we regard them as describing the behavior of a single computation of a program. In *branching* temporal logics, each moment in time may split into various possible futures. Accordingly, the structures over which branching temporal logic formulas are interpreted can be viewed as infinite computation trees, each describing the behavior of the possible computations of a nondeterministic program.

In the linear temporal logic LTL, formulas are composed from the set of atomic propositions using the usual Boolean connectives as well as the temporal connectives G ("always"), F ("eventually"), X ("next"), and U ("until"). The branching temporal logic CTL* augments LTL by the path quantifiers E ("there exists a computation") and A ("for all computations"). The branching temporal logic CTL is a fragment of CTL* in which every temporal connective is preceded by a path quantifier. Note that LTL has implicit universal path quantifiers in front of its formulas. Thus, LTL is essentially the linear fragment of CTL*.

The discussion of the relative merits of linear versus branching temporal logics in the context of system specification and verification goes back to the 1980s [43, 26, 7, 54, 28, 27, 57, 18, 16, 62, 63]. As analyzed in [54], linear and branching time logics correspond to two distinct views of time. It is not surprising therefore that LTL and CTL are expressively incomparable [18, 27, 43]. The LTL formula FGp is not expressible in CTL, while the CTL formula $AFAGp$ is not expressible in LTL. On the other hand, CTL seems to be superior to LTL when it comes to algorithmic verification, as we now explain.

Given a transition system M and a linear temporal logic formula φ, the model-checking problem for M and φ is to decide whether φ holds in all the computations of M. When φ is a branching temporal logic formula, the problem is to decide whether φ holds in the computation tree of M. The complexity of model checking for both linear and branching temporal logics is well understood: suppose we are given a transition system of size n and a temporal logic formula of size m. For the branching temporal logic CTL, model-checking algorithms run in time $O(nm)$ [19], while, for the linear temporal logic LTL, model-checking algorithms run in time $n2^{O(m)}$ [44]. Since LTL model checking is PSPACE-complete [56], the latter bound probably cannot be improved.

The difference in the complexity of linear and branching model checking has been viewed as an argument in favor of the branching paradigm. In particular, the computational advantage of CTL model checking over LTL model checking made CTL a popular choice, leading to efficient model-checking tools for this logic [20]. Through the 1990s, the dominant temporal specification language in industrial use was CTL. This dominance stemmed from the phenomenal success of SMV, the first symbolic model checker, which was CTL-based, and its follower VIS, also originally CTL-based, which served as the basis for many industrial model checkers.

In [64] we argued that in spite of the phenomenal success of CTL-based model checking, CTL suffers from several fundamental limitations as a temporal property-specification language, all stemming from the fact that CTL is a branching-time formalism: the language is unintuitive and hard to use, it does not lend itself to compositional reasoning, and it is fundamentally incompatible with semi-formal verification. In contrast, the linear-time framework is expressive and intuitive, supports compositional reasoning and semi-formal verification, and is amenable to combining enumerative and symbolic search methods. Indeed, the trend in the industry during this decade has been towards linear-time languages, such as ForSpec [6], PSL [25], and SVA [66].

In spite of the pragmatic arguments in favor of the linear-time approach, one still hears the arguments that this approach is not expressive enough, pointing out that in semantical analyses of concurrent processes, e.g., [60], the linear-time approach is considered to be the weakest semantically. In this dissertation we address the semantical arguments against linear time and argue that even from a semantical perspective the linear-time approach is quite adequate for specifying systems.

The gist of our argument is that branching-time-based notions of process equivalence are *not* reasonable notions of process equivalence, as they distinguish between processes that are not contextually distinguishable. In contrast, the linear-time view does yield an appropriate notion of contextual equivalence.

Chapter 2

A Principled Approach to Process Semantics

2.1 The Case Against Linear Time

The most fundamental approach to the semantics of programs focuses on the notion of equivalence. Once we have defined a notion of equivalence, the semantics of a program can be taken to be its equivalence class. In the context of concurrency, we talk about process equivalence. The study of process equivalence provides the basic foundation for any theory of concurrency [51], and it occupies a central place in concurrency-theory research, cf. [60].

The linear-time approach to process equivalence focuses on the traces of a process. Two processes are defined to be *trace equivalent* if they have the same set of traces. It is widely accepted in concurrency theory, however, that trace equivalence is too weak a notion of equivalence, as processes that are trace equivalent may behave differently in the same context [50]. An an example, using CSP notation, the two processes

$$\mathbf{if}(\mathbf{true} \rightarrow a?x; h!x)\Box(\mathbf{true} \rightarrow b?x; h!x)\mathbf{fi}$$

$$\mathbf{if}(a?x \rightarrow h!x)\Box(b?x \rightarrow h!x)\mathbf{fi}$$

have the same set of communication traces, but only the first one may deadlock when run in parallel with a process such as $b!0$.

In contrast, the two processes above are distinguished by *bisumulation*, highly popular notion of process equivalence [51, 53, 58]. It is known that CTL characterizes bisimulation, in the sense that two states in a transition system are bisimilar iff they satisfy exactly the same CTL formulas [15] (see also [39]). This is sometime mentioned as an important feature of CTL.

This contrast, between the pragmatic arguments in favor of the adequate expressiveness of the linear-time approach [64] and its accepted weakness from a process-equivalence perspective, calls for a re-examination of process-equivalence theory.

2.2 Process Equivalence Revisited

While the study of process equivalence occupies a central place in concurrency-theory research, the answers yielded by that study leave one with an uneasy feeling. Rather than providing a definitive answer, this study yields a profusion[1] of choices [3]. This situation led to statement of the form "It is not the task of process theory to find the 'true' semantics of processes, but rather to determine which process semantics is suitable for which applications" [60]. This situation should be contrasted with the corresponding one in the study of sequential-program equivalence. It is widely accepted that two programs are equivalent if they behave the same in all contexts, this is referred to as *contextual* or *observational* equivalence, where behavior refers to input/output behavior [67]. In principle, the same idea applies to processes: two processes are equivalent if they pass the same tests, but there is no agreement on what a test is and on what it means to pass a test.

We propose to adopt for process-semantics theory precisely the same principles accepted in program-semantics theory.

Principle of Contextual Equivalence: Two processes are equivalent if they behave the same in all contexts, which are processes with "holes".

As in program semantics, a context should be taken to mean a process with a "hole", into which the processes under consideration can be "plugged". This agrees with the point of view taken in *testing equivalence*, which asserts that tests applied to processes need to themselves be defined as processes [23]. Furthermore, *all* tests defined as processes should be considered. This excludes many of the "button-pushing experiments" of [50]. Some of these experiments are too strong–they cannot be defined as processes, and some are too weak–they consider only a small family of tests [23].

[1]This is referred to as the "Next '700 . . .' Syndrome." [3]

In particular, the tests required to define bisimulation equivalence [2, 50] are widely known to be too strong [10, 11, 12, 32].

In spite of its mathematical elegance [5, 58] and ubiquity in logic [8, 4], bisimulation is *not* a reasonable notion of process equivalence, as it makes distinctions that *cannot* be observed. Bisimulation is a structural similarity relation between states of the processes under comparison, rather than an observational comparison relation.

The most explicit advocacy of using bisimulation-based equivalence (in fact, *branching bisimulation*) appears in [61], which argues in favor of using equivalence concepts that are based on internal structure because of their context independence: "if two processes have the same internal structure they surely have the same observable behavior." It is hard to argue with the last point, but expecting an implementation to have the same internal structure as a specification is highly unrealistic and impractical, as it requires the implementation to be too close to the specification. In fact, it is clear from the terminology of "observational equivalence' used in [51] that the intention there was to formulate a concept of equivalence based on observational behavior, rather than on internal structure. Nevertheless, the terms "observational equivalence" for bisimulation-based equivalence in [51] is, perhaps, unfortunate, as weak-bisimulation equivalence is in essence a notion of *structural* similarity.

Remark 1 *One could argue that bisimulation equivalence is not only a mathematically elegant concept; it also serves as the basis for useful sound proof techniques for establishing process equivalence, cf. [39]. The argument here, however, is not against bisimulation as a useful mathematical concept; such usefulness ought to be evaluated on its own merits, cf.*

[30]. Rather, the argument is against viewing bisimulation-based notions of equivalence as reasonable notions of process equivalence.

The Principle of Contextual Equivalence does not fully resolve the question of process equivalence. In additional to defining the tests to which we subject processes, we need to define the observed behavior of the tested processes. It is widely accepted, however, that linear-time semantics results in important behavioral aspects, such as deadlocks and livelocks, being non-observable [50]. It is this point that contrasts sharply with the experience that led to the adoption of linear time in the context of hardware model checking [64]; in today's synchronous hardware all relevant behavior, including deadlock and livelock is observable (observing livelock requires the consideration of infinite traces). Compare this with our earlier example, where the process

$$\mathbf{if}(\mathbf{true} \rightarrow a?x; h!x)\Box(\mathbf{true} \rightarrow b?x; h!x)\mathbf{fi}$$

may deadlock when run in parallel with a process such as $b!0$. The problem here is that the description of the process does not tell us what happens when the first guard is selected in the context of the parallel process $b!0$. The deadlock here is not described explicitly; rather it is implicitly inferred from a lack of specified behavior. This leads us to our second principle.

Principle of Comprehensive Modeling: A process description should model all relevant aspects of process behavior.

The rationale for this principle is that relevant behavior, where relevance depends on the application at hand, should be captured by the description of the process, rather than inferred from lack of behavior by a semantical theory proposed by a concurrency theorist. It is the usage of inference to attribute behavior that opens the door to numerous interpretations, and, consequently, to numerous notions of process equivalence.

Remark 2 *It is useful to draw an analogy here to another theory, that of* nonmonotonic logic, *whose main focus is on inferences from absence of premises. The field started with some highly influential papers, advocating, for example "negation as failure" [17] and "circumscription" [48]. Today, however, there is a profusion of approaches to nonmonotonic logic, including numerous extensions to negation as failure and to circumscription [47]. One is forced to conclude that there is no universally accepted way to draw conclusions from absence of premises. (Compare also to the discussion of negative premises in transition-system specifications [12, 32].)*

Going back to our problematic process

$$\mathbf{if}(\mathbf{true} \rightarrow a?x; h!x)\Box(\mathbf{true} \rightarrow b?x; h!x)\mathbf{fi}$$

The problem here is that the process is not *receptive* to communication on channel b, when it is in the left branch. The position that processes need to be receptive to all allowed inputs from their environment has been argued by many authors [1, 24, 45]. It can be viewed as an instance of our Principle of Comprehensive Modeling, which says that the behavior that results from a write action on channel b when the process is in the left branch needs to be specified explicitly. From this point of view, process-algebraic formalisms such as CCS [50] and CSP [40] are *underspecified*, since they leave important behavioral aspects unspecified. For example, if the distinction between normal termination and deadlocked termination is relevant to the application, then this distinction ought to be explicitly modeled.

Rather, in CCS and CSP there is no observable distinction between normal and deadlocked termination, as both situations are characterized only by the absence of outgoing transitions. (The formalism of Kripke structures, often used in the model-checking literature [21], also suffers from lack of receptiveness, as it does not distinguish between inputs and outputs.)

It is interesting to note that *transducers*, which were studied in an earlier work of Milner [49], which led to [50], are receptive. Transducers are widely accepted models of hardware. We come back to transducers in the next section.

Remark 3 *The Principle of Comprehensive Modeling is implicit in a paper by Halpern on modeling game-theoretic situations [36]. The paper shows that a certain game-theoretic paradox is, in fact, a consequence of deficient modeling, in which states of agents do not capture all relevant aspects of their behavior. Once the model is appropriately enriched, the paradox evaporates away. For extensive discussions on modeling multi-agent systems, see Chapters 4 and 5 in [29] and Chapter 6 in [35].*

The Principle of Comprehensive Modeling can be thought of as the "Principle of Appropriate Abstraction". Every model is an abstraction of the situation being modeled. A good model necessarily abstracts away irrelevant aspects, but models explicitly relevant aspects. The distinction between relevant and irrelevant aspects is one that can be made only by the model builder and users. For example, a digital circuit is a model of an analog circuit in which only the digital aspects of the circuit behavior are captured [33]. Such a model should not be used to analyze non-digital aspects of circuit behavior, such as timing issues or issues of metastable states. Such issues require richer models. The Principle of Comprehensive Modeling does not call for infinitely detailed models; such models are useless as they offer no abstraction. Rather, the principle calls for models that are rich enough, but not too rich, dependning on the current level of abstraction. Whether or not deadlocked termination should be considered distinct from normal termination depends on the the current level of abstraction; at one level of abstraction this distinction is erased, but at a finer level of abstraction this distinction is material. For further discussion of abstraction see [42].

The Principle of Comprehensive Modeling requires a process description to model all relevant aspects of process behavior. It does not spell out how such aspects are to be modeled. In particular, it does not address the question of what is observed when a process is being tested. Here again we propose to follow the approach of program semantics theory and argue that only the input/output behavior of processes is observable. Thus, observable relevant aspects of process behavior ought to be reflected in its input/output behavior.

Principle of Observable I/O: The observable behavior of a tested process is precisely its input/output behavior.

Of course, in the case of concurrent processes, the input/output behavior has a temporal dimension. That is, the input/output behavior of a process is a trace of input/output actions. The precise "shape" of this trace depends of course on the underlying semantics, which would determine, for example, whether we consider finite or infinite traces, the temporal granularity of traces, and the like. It remains to decide how nondeterminism is observed, as, after all, a nondeterministic process does not have a unique behavior. This leads to notions such as *may testing* and *must testing* [23]. We propose here to finesse this issue by imagining that a test is being run several times, eventually exhibiting *all* possible behaviors. Thus, the input/output behavior of a nondeterministic test is its full set of input/output traces.

(One could argue that by allowing a test to observe *all* input/output traces, our notion of test is too strong, resulting in an overly fine notion of process equivalence. Since our focus here is on showing that trace equivalence is not too coarse, we do not pursue this point further here.)

It should be noted that the approach advocated here is diametrically opposed to that of [61], who argues *against* contextual equivalence: "In practice, however, there appears to be doubt and difference of opinion concerning the observable behaviour of systems. Moreover, what is observable may depend on the nature of the systems on which the concept will be applied and the context in which they will be operating." In contrast, our guiding principles say that (1) by considering *all* possible contexts, one need not worry about identifying specific contexts or testing scenarios, and (2) process description ought to describe the observable behavior of the process precisely to remove doubts about that behavior. In our opinion, the "doubt and difference of opinion" about process behavior stem from the underspecificity for formalisms such as CCS and CSP.

Remark 4 *In the same way that bisimulation is not a contextual equivalence relation, branching-time properties are not necessarly contextually observable. Adapting our principles to property observability we should expect behavioral properties to be observable in the following sense. If two processes are distinguished by a property φ, that is, P_1 satisfies φ, but P_2 does not satisfy φ, there has to be a context C such that the set of input-output traces of $C[P_1]$ is different than that of $C[P_2]$. Consider, however, the CTL property $AGEFp$, which says that from all given states of the process it is possible to reach a state where p holds. It is easy to construct processes P_1 and P_2, one satisfying $AGEFp$ and one falsifying it, such that $C[P_1]$ and $C[P_2]$ have the same set of input-output traces for all contexts C. Thus, $AGEFp$ is a structural property rather than an observable property.*

In the next section we apply our approach to transducers; we show that once our three principles are applied we obtain that trace-based equivalence is adequate and fully abstract; that is, it is precisely the unique observational equivalence for transducers.

We believe that this holds in general; that is, under our three principles, trace-based equivalence provides the "right" notion of process equivalence.

Chapter 3

Nondeterministic Transducers

Transducers constitute a fundamental model of discrete-state machines with input and output channels [37]. They are still used as a basic model for sequential computer circuits [33]. We use nondeterministic transducers as our model for processes. We define a synchronous composition operator for such transducers, which provides us a notion of context. We then define linear observation semantics and give adequacy and full-abstraction results for trace equivalence in terms of it.

3.1 Definition of Transducers

A nondeterministic transducer is a state machine with input and output channels. The state-transition function depends on the current state and the input, while the output depends solely on the current state (thus, our machines are Moore machines [37]).

Definition 1 *A transducer is a tuple,* $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$*, where*

- *Q is a countable set of states.*

- *q_0 is the start state.*

- *I is a finite set of input channels.*

- *O is a finite set of output channels.*

- *Σ is a finite alphabet of actions (or values).*

- *$\sigma : I \cup O \rightarrow 2^{\Sigma} - \{\emptyset\}$ is a function that allocates an alphabet to each channel.*

- *$\lambda : Q \times O \rightarrow \Sigma$ is the output function of the transducer. $\lambda(q, o) \in \sigma(o)$ is the value that is output on channel o when the transducer is in state q.*

- *$\delta : Q \times \sigma(i_1) \times \cdots \times \sigma(i_n) \rightarrow 2^Q$, where $I = \{i_1, \ldots, i_n\}$, is the transition function, mapping the current state and input to the set of possible next states.*

Both I and O can be empty. In this case δ is a function of state alone. This is important because the composition operation that we define usually leads to a reduction in the number of channels. Occasionally, we refer to the set of allowed values for a channel as the channel alphabet. This is distinct from the total alphabet of the transducer (denoted by Σ).

We represent a particular input to a transducer as an assignment that maps each input channel to a particular value. Formally, an *input assignment* for $(Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ is a function $f : I \rightarrow \Sigma$, such that for all $i \in I$, $f(i) \in \sigma(i)$. The entire input can then, by a slight abuse of notation, be succinctly represented as $f(I)$. The set of all input assignments of transducer M is denoted $In(M)$. Similarly, an *output assignment* is a mapping $g : O \rightarrow \Sigma$ such that there exists $q \in Q$, for all $o \in O$, $g(o) = \lambda(q, o)$. The set of all output assignments of M is denoted $Out(M)$. The *output mapping* of M is the function $h : Q \rightarrow Out(M)$ that maps a state to the output produced by the machine in that state: for all $q \in Q, o \in O, h(q)(o) = \lambda(q, o)$.

We point to three important features of our definition. First, note that transducers are receptive. That is, the transition function $\delta(q, f)$ is defined for all states $q \in Q$ and input assignments f. There is no implicit notion of deadlock here. Deadlocks need to be modeled explicitly, e.g., by a special sink state d whose output is, say, "deadlock". Second, note that inputs at time k take effect at time $k + 1$. This enables us to define composition without worrying about causalilty loops, unlike, for example, in Esterel [9]. Thirdly, note that the internal state of a transducer is observable only through its output function. How much of the state is observable depends on the output function.

3.2 Synchronous Parallel Composition

In general there is no canonical way to compose machines with multiple channels. In concrete devices, connecting components requires as little as knowing which wires to join. Taking inspiration from this, we say that a composition is defined by a particular set of desired connections between the machines to be composed. This leads to an intuitive and flexible definition of composition.

A connection is a pair consisting of an input channel of one transducer along with an output channel of another transducer. We require, however, sets of connections to be well formed. This requires two things:

- no two output channels are connected to the same input channel, and
- an output channel is connected to an input channel only if the output channel alphabet is a subset of the input channel alphabet.

These conditions guarantee that connected input channels only receive well defined values that they can read. We now formally define this notion.

Definition 2 (Connections) *Let $\mathcal{M}$ be a set of transducers. Then*

$$Conn(\mathcal{M}) = \{X \subseteq \mathcal{C}(\mathcal{M}) | (a,b) \in X, (a,c) \in X \Rightarrow b = c\}$$

where $\mathcal{C}(\mathcal{M}) = \{(i_A, o_B) \ |\{A,B\} \subseteq \mathcal{M}, i_A \in I_A, o_B \in O_B, \sigma_B(o_B) \subseteq \sigma_A(i_A)\}$ is the set of all possible input/output connections for $\mathcal{M}$. Elements of $Conn(\mathcal{M})$ are valid connection sets.

Given a set of connections between a set of transducers, we can obtain a composite transducer in a natural way using the cartesian product. The state space is just the cartesian product of the states of the individual transducers. Every channel involved in a connection is removed, and the remaining channels become channels of the composite.

Definition 3 (Composition) *Let $\mathcal{M} = \{M_1, \ldots, M_n\}$, $M_k = (Q_k, q_0^k, I_k, O_k, \Sigma_k, \sigma_k, \lambda_k, \delta_k)$, be a set of transducers, and $C \in Conn(\mathcal{M})$. Then the* composition *of $\mathcal{M}$ with respect to C, denoted by $||_C(\mathcal{M})$, is a transducer $(Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ defined as follows:*

- $Q = Q_1 \times \ldots \times Q_n$
- $q_0 = q_0^1 \times \ldots \times q_0^n$
- $I = \bigcup_{k=1}^{n} I_k - \{i \mid (i, o) \in C\}$
- $O = \bigcup_{k=1}^{n} O_k - \{o \mid (i, o) \in C\}$
- $\Sigma = \bigcup_{k=1}^{n} \Sigma_k$
- $\sigma(u) = \sigma_k(u)$, *where* $u \in I_k \cup O_k$
- $\lambda(q_1, \ldots, q_n, o) = \lambda_k(q_k, o)$ *where* $o \in O_k$
- $\delta(q_1, \ldots, q_n, f(I)) = \Pi_{k=1}^{n}(\delta_k(q_k, g(I_k)))$

  where $g(i) = \lambda_j(q_j, o)$ *if* $(i, o) \in C,\ o \in O_j$, *and* $g(i) = f(i)$ *otherwise.*

Definition 4 (Binary Composition) *The binary composition of* M_1 *and* M_2 *with respect to* $C \in Conn(\{M_1, M_2\})$ *is* $M_1 ||_C M_2 = ||_C(\{M_1, M_2\})$.

The following theorem shows that a general composition can be built up by a sequence of binary compositions. Thus binary composition is as powerful as general composition and henceforth we switch to binary composition as our default composition operation.

Theorem 1 (Composition Theorem) *Let* $\mathcal{M} = \{M_1, \ldots, M_n\}$ *be a set of transducers, where* $M_k = (Q_k, q_0^k, I_k, O_k, \Sigma_k, \sigma_k, \lambda_k, \delta_k)$, *and* $C \in Conn(\mathcal{M})$. *Let* $\mathcal{M}' = \mathcal{M} - \{M_n\}$, $C' = \{(i, o) \in C | i \in I_j, o \in O_k, j < n, k < n\}$ *and* $C'' = C - C'$. *Then*

$$||_C(\mathcal{M}) = ||_{C''}(\{||_{C'}(\mathcal{M}'), M_n\}).$$

Proof. Let

$$M = ||_C(\mathcal{M}) = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$$

$$M' = ||_{C'}(\mathcal{M}') = (Q', q_0', I', O', \Sigma', \sigma', \lambda', \delta')$$

$$M'' = ||_{C''}(\{M', M_n\}) = (Q'', q_0'', I'', O'', \Sigma'', \sigma'', \lambda'', \delta'')$$

To prove that $M'' = M$ we need to show that each component of M'' is identical to the corresponding component of M. Below we give such a proof for each separate component.The proofs depend entirely on Definition 3.

- $Q'' = Q' \times Q_n = (Q_1 \times \ldots \times Q_{n-1}) \times Q_n = Q$ (using Defn. 3).
- $q_0'' = q_0' \times q_0^n = (q_0^1 \times \ldots \times q_0^{n-1}) \times q_0^n = q_0$ (using Defn. 3).
- $I'' = I' \cup I_n - \{i|(i,o) \in C''\} = (\bigcup_{k=1}^{n-1} I_k - \{i \mid (i,o) \in C'\}) \cup I_n - \{i|(i,o) \in C''\} = (\bigcup_{k=1}^{n-1} I_k) \cup I_n - \{i \mid (i,o) \in C'\} - \{i|(i,o) \in C''\} = \bigcup_{k=1}^{n} I_k - \{i \mid (i,o) \in C' \cup C''\} = I$.
- $O'' = O$. Proof is identical to the input case, because of the symmetry between the definition of inputs and outputs of a composition (see Defn. 3).
- $\Sigma'' = \Sigma' \cup \Sigma_n = (\bigcup_{k=1}^{n-1} \Sigma_k) \cup \Sigma_n = \bigcup_{k=1}^{n} \Sigma_k = \Sigma$.
- $\sigma'' = \sigma$. This is true because composition does not change any channel alphabet.
- $\lambda'' = \lambda$. Composition simply projects the outputs of the individual automata on the remaining output channels.

- $\delta'' = \delta$.

$\delta''(q_1, \ldots, q_n, f(I)) = \delta'(q_1, \ldots, q_{n-1}, g(I')) \times \delta_n(q_n, g(I_n))$, where

$$g(i) = \begin{cases} \lambda'(q_1, \ldots, q_{n-1}, o) \text{ if } (i, o) \in C'', o \in O' \\ \lambda_n(q_n, o) \text{ if } (i, o) \in C'', o \in O_n \\ f(i) \text{ otherwise.} \end{cases}$$

Now $\lambda'(q_1, \ldots, q_{n-1}, o) = \lambda_k(q_k, o)$ for $o \in O_k$ and $1 \leq k < n$. We use this fact to rewrite g as follows:

$$g(i) = \begin{cases} \lambda_k(q_k, o) \text{ if } (i, o) \in C'', o \in O_k, k \leq n \\ f(i) \text{ otherwise.} \end{cases}$$

Next we see that $\delta'(q_1, \ldots, q_{n-1}, g(I')) = \Pi_{k=1}^{n-1}(\delta_k(q_k, h(I_k)))$, where

$$h(i) = \begin{cases} \lambda_j(q_j, o) \text{ if } (i, o) \in C', o \in O_j, j < n \\ g(i) \text{ otherwise.} \end{cases}$$

Now we can simply expand $g(i)$ in the definition of $h(i)$ and we get

$$h(i) = \begin{cases} \lambda_j(q_j, o) \text{ if } (i, o) \in C', o \in O_j, j < n \\ \lambda_j(q_j, o) \text{ if } (i, o) \in C'', o \in O_j, j \leq n \\ f(i) \text{ otherwise.} \end{cases}$$

We can write δ'' and δ as follows:

$$\delta''(q_1, \ldots, q_n, f(I)) = (\Pi_{k=1}^{n-1}(\delta_k(q_k, h(I_k)))) \times \delta_n(q_n, g(I_n))$$

$$\delta(q_1, \ldots, q_n, f(I)) = \Pi_{k=1}^{n}(\delta_k(q_k, e(I_k)))$$

where

$$e(i) = \begin{cases} \lambda_j(q_j, o) \text{ if } (i, o) \in C, o \in O_j, j \leq n \\ f(i) \text{ otherwise.} \end{cases}$$

Finally, to prove that δ'' and δ are the same function, it suffices to show that $e(i)$ agrees with $h(i)$ on I' and agrees with $g(i)$ on I_n.

$\boxtimes$

The upshot of Theorem 1 is that in the framework of transducers a general context, which is a network of transducers with a hole, is equivalent to a single transducer. Thus, for the purpose of contextual equivalence it is sufficient to consider testing transducers.

3.3 Executions and Traces

Definition 5 (Execution) *An* execution *for transducer $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ is a countable sequence of pairs $\langle s_i, f_i \rangle_{i=0}^{l}$ such that $s_0 = q_0$, and for all $i \geq 0$,*

- $s_i \in Q$.
- $f_i : I \rightarrow \Sigma$ *such that for all* $u \in I$, $f(u) \in \sigma(u)$.
- $s_i \in \delta(s_{i-1}, f_{i-1}(I))$.

If $l \in N$, the execution is finite and its length is l. If $l = \infty$, the execution is infinite and its length is defined to be ∞. The set of all executions of transducer M is denoted $exec(M)$.

Definition 6 (Trace) *Let $\alpha = \langle s_i, f_i \rangle_{i=0}^{l} \in exec(M)$. The* trace *of α, denoted by $[\alpha]$, is the sequence of pairs $\langle \omega_i, f_i \rangle_{i=0}^{l}$, where for all $i \geq 0$, $\omega_i : O \rightarrow \Sigma$ and for all $o \in O$, $\omega_i(o) = \lambda(s_i, o)$. The set of all traces of a transducer M, denoted by $Tr(M)$, is the set $\{[\alpha] | \alpha \in exec(M)\}$. An element of $Tr(M)$ is called a trace of M.*

Thus a trace is a sequence of pairs of output and input actions. While an execution captures the real underlying behavior of the system, a trace is the observable part of that behavior. The length of a trace α is defined to be the length of the underlying execution and is denoted by $|\alpha|$.

Definition 7 (Trace Equivalence) *Two transducers M_1 and M_2 are trace equivalent, denoted by $M_1 \sim_T M_2$, if $Tr(M_1) = Tr(M_2)$. Note that this requires that they have the same set of input and output channels.*

We now study the properties of trace equivalence with respect to composition. In order to do so, we need a way to match traces of a composition to traces of its components. We first define the composition of executions and traces.

Definition 8 *Given* $\alpha = \langle s_i, f_i \rangle_{i=0}^{n} \in exec(M_1)$ *and* $\beta = \langle r_i, g_i \rangle_{i=0}^{n} \in exec(M_2)$*, we define the composition of* α *and* β *w.r.t* $C \in Conn(\{M_1, M_2\})$ *as follows*

$$\alpha||_C\beta = \langle (s_i, r_i), h_i \rangle_{i=0}^{n}$$

where $h_i(u) = f_i(u)$ *if* $u \in I_1 - \{i|(i, o) \in C\}$ *and* $h_i(u) = g_i(u)$ *if* $u \in I_2 - \{i|(i, o) \in C\}$*.*

Definition 9 *Given* $t = \langle \omega_i, f_i \rangle_{i=0}^{n} \in Tr(M_1)$ *and* $u = \langle \nu_i, g_i \rangle_{i=0}^{n} \in Tr(M_2)$*, we define the composition of t and u w.r.t* $C \in Conn(\{M_1, M_2\})$ *as follows*

$$t||_C u = \langle \mu_i, h_i \rangle_{i=0}^{n}$$

where $\mu_i(o) = \omega_i(o)$ *if* $o \in O_1 - \{o|(i, o) \in C\}$ *and* $\mu_i(o) = \nu_i(o)$ *if* $o \in O_2 - \{o|(i, o) \in C\}$*, and* h_i *is as defined in Definition 8 above.*

Note that the composition operation defined on traces is purely syntactic. There is no guarantee that the composition of two traces is a trace of the composition of the transducers generating the individual traces. The following simple property is necessary and sufficient to achieve this.

Definition 10 (Compatible Traces) *Given* $C \in Conn(\{M_1, M_2\})$*,* $t_1 = \langle \omega_i^1, f_i^1 \rangle_{i=0}^{n} \in$

$Tr(M_1)$ and $t_2 = \langle \omega_i^2, f_i^2 \rangle_{i=0}^n \in Tr(M_2)$, we say that t_1 and t_2 are compatible *with respect to C if for all $(u, o) \in C$ and for all $i \geq 0$, we have*

- *If $u \in I_j$ and $o \in O_k$ then $f_i^j(u) = \omega_i^k(o)$, for all $i \geq 0$ and for $j, k \in \{1, 2\}$.*

Lemma 1 *Let $C \in Conn(\{M_1, M_2\})$, $t \in Tr(M_1)$ and $u \in Tr(M_2)$. Then $t||_C u \in Tr(M_1||_C M_2)$ if and only if t and u are compatible with respect to C.*

We now extend the notion of trace composition to sets of traces.

Definition 11 *Let $T_1 \subseteq Tr(M_1)$, $T_2 \subseteq Tr(M_2)$ and $C \in Conn(\{M_1, M_2\})$. We define*

$$T_1||_C T_2 = \{t_1||_C t_2 \,|\, t_1 \in Tr(M_1), t_2 \in Tr(M_2), |t_1| = |t_2|\}$$

The next theorem is an important intermediate result on the way to proving the congruency w.r.t. composition of not just trace equivalence, but of a general class of linear-time semantics derived from trace equivalence. The result can be thought of as an invariance theorem. Suppose M_1 and M_2 are transducers, T_1 is a subset of the traces of M_1, T_2 is a subset of the traces of M_2 and $C \in Conn(\{M_1, M_2\})$. Then the theorem says that those elements of $T_1||_C T_2$ which are also valid traces of $M_1||_C M_2$, only depend on T_1 and T_2, and are independent of M_1 and M_2.

Theorem 2 (Syntactic theorem of traces) *Let $T_1 \subseteq Tr(M_1) \cap Tr(M_3)$ and $T_2 \subseteq Tr(M_2) \cap Tr(M_4)$, and $C \in Conn(\{M_1, M_2\}) \cap Conn(\{M_3, M_4\})$. Then*

$$(T_1||_C T_2) \cap Tr(M_1||_C M_2) = (T_1||_C T_2) \cap Tr(M_3||_C M_4)$$

Proof. Let $t \in (T_1||_C T_2) \cap Tr(M_1||_C M_2)$. Then $t = t_1||_C t_2$, where $t_1 \in T_1$ and $t_2 \in T_2$. Since $t_1||_C t_2 \in Tr(M_1||_C M_2)$, by Lemma 1, t_1 and t_2 are compatible with respect to C. Since $T_1 \subseteq Tr(M_3)$ and $T_2 \subseteq Tr(M_4)$, again by Lemma 1, $t_1||_C t_2 \in Tr(M_3||_C M_4)$. Therefore $(T_1||_C T_2) \cap Tr(M_1||_C M_2) \subseteq (T_1||_C T_2) \cap Tr(M_3||_C M_4)$. By symmetry, set inclusion, and thus equality, holds in the reverse direction too. ⊠

Using Theorem 2, we show now that any equivalence defined in terms of sets of traces is automatically a congruence with respect to composition, if it satisfies a certain natural property.

Definition 12 (Trace-based equivalence) *Let $\mathcal{M}$ be the set of all transducers. Let $R : \mathcal{M} \to \{A \subseteq Tr(M) | M \in \mathcal{M}\}$ such that for all $M \in \mathcal{M}$, $R(M) \subseteq Tr(M)$. Then R defines an equivalence relation on $\mathcal{M}$, denoted by $\sim_R$, such that for all $M_1, M_2 \in \mathcal{M}$, $M_1 \sim_R M_2$ if and only if $R(M_1) = R(M_2)$. Further, the function R is called an equivalence-based invariance, and the relation $\sim_R$ is called a trace-based equivalence.*

Trace-based equivalences enable us to relativize trace equivalence to "interesting" traces. For example, one may want to consider finite traces only, infinite traces only, fair traces only, and the like. Of course, not all such relativizations are appropriate.

We require traces to be *compositional*, in the sense described below. This covers finite, infinite, and fair traces.

Definition 13 (Compositionality) *Let $\sim_R$ be a trace-based equivalence. We say that $\sim_R$ is compositional if given transducers M_1, M_2 and $C \in Conn(\{M_1, M_2\})$, the following* *hold:*

1. $R(M_1||_C M_2) \subseteq R(M_1)||_C R(M_2)$.

2. *If* $t_1 \in R(M_1)$, $t_2 \in R(M_2)$, *and* t_1, t_2 *are compatible w.r.t.* C, *then* $t_1||_C t_2 \in R(M_1||_C M_2)$.

The two conditions in Definition 13 are, in a sense, soundness and completeness conditions, as the first ensures that no inappropriate traces are present, while the second ensures that all appropriate traces are present. That is, the first condition ensures that the trace set captured by R is not too large, while the second ensures that it is not too small.

Note, in particular, that trace equivalence itself is a compositional trace-based equivalence. We are now in a position to obtain full abstraction results for our notion of compositional trace-based equivalence

3.4 Full Abstraction

There are two aspects to full abstraction. The first lies in showing that the semantics makes all the needful distinctions, and the second in showing that it makes no unnecessary ones. Thus we want to show that if two transducers are equivalent by our semantics, then no context can distinguish between them. Here we prove the stronger condition that trace semantics is a congruence with respect to composition. Then we next show that if two machines are inequivalent under trace semantics, then some context (i.e., composition with a transducer) will be able distinguish between the two. The following theorem asserts that

$\sim_R$ is a *congruence* with respect to composition.

Theorem 3 (Congruence Theorem) *Let $\sim_R$ be a compositional trace-based equivalence. Let $M_1 \sim_R M_3$, $M_2 \sim_R M_4$, and $C \in Conn(\{M_1, M_2\}) = Conn(\{M_3, M_4\})$. Then $M_1||_C M_2 \sim_R M_3||_C M_4$.*

Proof. We prove this by showing $R(M_1||_C M_2) = (R(M_1)||_C R(M_2)) \cap Tr(M_1||_C M_2) = (R(M_3)||_C R(M_4)) \cap Tr(M_3||_C M_4) = R(M_3||_C M_4)$. We prove the first equality by showing set inclusion from both directions. The second equality is an instance of Theorem 2. The third equality follows from the first by symmetry.

- $R(M_1||_C M_2) \subseteq (R(M_1)||_C R(M_2)) \cap Tr(M_1||_C M_2)$, because by Definition 13, $R(M_1||_C M_2) \subseteq R(M_1)||_C R(M_2)$, and by Definition 12, $R(M_1||_C M_2) \subseteq Tr(M_1||_C M_2)$.
- $R(M_1||_C M_2) \supseteq (R(M_1)||_C R(M_2)) \cap Tr(M_1||_C M_2)$,

  because if $t_1||_C t_2 \in (R(M_1)||_C R(M_2)) \cap Tr(M_1||_C M_2)$ then, by Lemma 1, t_1 and t_2 are compatible w.r.t C, and, by Definition 13, $t_1||_C t_2 \in R(M_1||_C M_2)$.

$\boxtimes$

An immediate corollary of Theorem 3 is the fact that no context can distinguish between two trace-based equivalent transducers. The corollary is fact a special case of the theorem, obtained by setting $M_2 = M_4$.

Corollary 1 *Let M_1 and M_2 be transducers, R be a compositional trace-based equivalence and $M_1 \sim_R M_2$. Then for all transducers M and all $C \in Conn(\{M, M_1\}) = Conn(\{M, M_2\})$, we have that $M||_C M_1 \sim_R M||_C M_2$.*

Finally, it is also the case that some context can always distinguish between two inequivalent transducers. If we choose a composition with an empty set of connections, all original traces of the composed transducers are present in the traces of the composition. If $M_1 \not\simeq_R M_2$, then $M_1||_\emptyset M \not\simeq_R M_2||_\emptyset M$. We claim the stronger result that given two inequivalent transducers with the same interface, we can always find a third transducer that distinguishes between the first two, when it is *maximally* connected with them.

But first we need to slightly restrict the form that the semantics R can take by imposing an additional naturalness condition, that essentially says that R should not be able to discriminate between identical traces produced by machines with the same interface.

Definition 14 (Interface-respecting Semantics) *Let M_1 and M_2 be such that $In(M_1) = In(M_2)$ and $Out(M_1) = Out(M_2)$, and R be a trace-based equivalence. We say that R is interface-respecting, if $t \in R(M_1)$ and $t \in Tr(M_2)$ necessarily imply that $t \in R(M_2)$.*

Definition 15 (Tester) *Given transducers M and M', we say that M' is a tester for M, if there exists $C \in Conn(\{M, M'\})$ such that $M||_C M'$ has no input channels and exactly one output channel o with $o \in O'_M$. We also say M' is a tester for M w.r.t. C.*

Theorem 4 *Let M_1 and M_2 be transducers with $In(M_1) = In(M_2)$ and $Out(M_1) = Out(M_2)$, R be a compositional interface-respecting trace-based equivalence and $M_1 \not\simeq_R M_2$. Then there exists a transducer M and $C \in Conn(\{M, M_1\}) = Conn(\{M, M_2\})$, such that M is a tester for M_1 and M_2 w.r.t. C, and $M||_C M_1 \not\simeq_R M||_C M_2$.*

Proof. Let $M_1 = (Q_1, q_0^1, I_1, O_1, \Sigma, \sigma_1, \lambda_1, \delta_1)$ and $M_2 = (Q_2, q_0^2, I_2, O_2, \Sigma, \sigma_2, \lambda_2, \delta_2)$. Since $M_1 \not\sim_R M_2$, we assume without loss of generality that there exists $\tau \in R(M_1) \setminus R(M_2)$. Let $\tau = \langle \omega_i, f_i \rangle_{i=0}^n \in Tr(M_1)$. We define $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ as follows:

- $Q = \{q_i : i \in \mathbb{N}\} \cup \{q_f\}$, is a countable set of states with a special failure state.
- For each $o \in O_1$, we create an input channel in_o in I and assign alphabet $\sigma(in_o) = \sigma_1(o)$ to it.
- For each $in \in I_1$, we create an output channel o_{in} in O and assign alphabet $\sigma(o_{in}) = \sigma_1(in)$ to it.
- An extra output channel o_t, with alphabet $\{a, b\} \subseteq \Sigma$, that will be the only visible channel remaining after composition.
- $\lambda(q_i, o_{in}) = f_i(in)$, $\lambda(q_i, o_t) = a$ and $\lambda(q_f, o_t) = b$. In all other cases, we don't care what output M produces, and λ can be assumed to be arbitrary.
- For state $q \in Q$, and input assignment $g : I \longrightarrow \Sigma$,

$$\delta(q, g(I)) = \begin{cases} q_{i+1}, \text{ if } q = q_i \text{ and } \forall\, in_o \in I, g(in_o) = \omega_i(o), \\ \\ q_f, \text{ otherwise.} \end{cases}$$

We define the set of connections $C \in Conn(\{M, M_1\})$ as follows: for all $in \in I_1, o \in O_1$, $(in, o_{in}) \in C$ and $(in_o, o) \in C$, and nothing else is in C. Now $M||_C M_1$ has exactly one channel, which is the output channel o_t belonging to M, and so M is a tester for M_1

w.r.t. C. The transducer M is deterministic and designed to follow the execution of the distinguishing trace τ. As soon as the computation of the machine being tested diverges from this trace, M will enter its failure state and switch its visible output from a to b. Thus if M_2 does not produce the trace τ, then we can clearly distinguish it from M_1 using M. The only remaining case to consider is when M_2 does produce this trace but it does not fall under the set distinguished by R. That is, $\tau \in Tr(M_2)$ and $\tau \notin R(M_2)$. But this is impossible as R is interface-respecting by definition. ⊠

Chapter 4

Probabilistic Transducers

4.1 Preliminaries

In order to rigorously construct a probabilistic model of transducer behavior, we will require certain concepts from measure theory and its application to the space of infinite sequences over some alphabet (i.e., Cantor and Baire spaces). This is because our probabilistic notion of behavior will be defined by probability distributions, which are measures, over Q^{ω}, the set of infinite sequences of states. We briefly cover the required mathematical background in this section. All lemmas and theorems in this section are stated without proof. The interested reader should consult any standard text in measure theory ([34], [22]).

4.1.1 Measure and Probability

Intuitively, a probability distribution over some set X should satisfy the following properties: the probability of any event (a subset of X) should be non-negative, the probability of the entire set X as an event should be 1, and the probability of the union of two disjoint events should be the sum of the probabilities of the events. For technical reasons, the third condition is actually replaced by a stronger condition requiring countable additivity. It turns out that this combination of desired properties cannot always be achieved if the events are allowed to be arbitrary subsets of X. For the properties to hold simultaneously, the set of events has to be restricted to a subset of 2^X that is closed under complements and countable unions. Such a subset of the power set is called a σ-algebra.

Definition 16 (σ-algebra) *Let X be a set and $\mathcal{F}$ be a set of subsets of X. We say that $\mathcal{F}$ is an* algebra *over X if it is closed under taking complements and finite unions. A σ-algebra over X is an algebra that is closed under countable unions. Given a subset $\mathcal{A}$ of 2^X, the σ-algebra* generated *by $\mathcal{A}$ is the smallest σ-algebra containing $\mathcal{A}$, and can be obtained as the intersection of all σ-algebras containing $\mathcal{A}$.*

Definition 17 (Measure) *Let X be a set and $\mathcal{F}$ be a σ-algebra over X. A* measure *over $(X, \mathcal{F})$ is a function $\mu : \mathcal{F} \longrightarrow [0, \infty]$ from $\mathcal{F}$ to the extended positive reals, that satisfies the following conditions:*

Nullity. $\mu(\emptyset) = 0$.

Countable additivity. $\mu(\bigcup_{i \in I} A_i) = \sum_{i \in I} \mu(A_i)$ *for every countable set of pairwise disjoint sets $A_i \in \mathcal{F}$.*

The triple $(X, \mathcal{F}, \mu)$ is called a measure space. *If $\mu(X) = 1$ then μ is a* probability *measure. A* probability space *is a measure space with a probability measure.*

Frequently, when there is some relation between sets X and Y, we can use a measure defined on X to obtain a measure on Y. The rest of this subsection deals with two such instances.

Given a function from X to Y that preserves measurable subsets in the inverse, we can use it to generate a measure on Y from any measure on X. Such a function is called a *measurable function*. In particular, the function mapping Q^ω to $Out(M)^\omega$, which is a generalization of the output mapping of a transducer, is measurable. Later, we crucially exploit this fact while defining probabilistic analogues of executions and traces.

Definition 18 (Measurable function) *Let X, Y be sets and $\mathcal{F}$, $\mathcal{G}$ be σ-algebras over X and Y, respectively. A function $f : X \rightarrow Y$ is called measurable, if for all $A \in \mathcal{G}$, $f^{-1}(A) \in \mathcal{F}$.*

Lemma 2 *If $\mu : \mathcal{F} \rightarrow [0, \infty]$ is a measure over $\mathcal{F}$, and $f : X \rightarrow Y$ is a measurable function, then $\mu_f : \mathcal{G} \rightarrow [0, \infty]$, defined as $\mu_f(A) = \mu(f^{-1}(A))$ for all $A \in \mathcal{G}$, is a measure over $\mathcal{G}$.*

Finally, a measure on the product of spaces can be defined in the natural way as the product of the measures on the individual spaces. This *product measure* will be used in the composition of probabilistic transducers.

Theorem 5 (Product Measure) *Let $(X_i, \mathcal{F}_i, \mu_i)$ be a measure space for $i \in I$. Then the product space $(\prod_{i \in I} X_i, \prod_{i \in I} \mathcal{F}_i, \prod_{i \in I} \mu_i)$, defined as follows, is a measure space.*

- *$\prod_{i \in I} X_i$ is the cartesian product of sets.*

- $\prod_{i \in I} \mathcal{F}_i = \{\prod_{i \in I} B_i : \forall i \in I, B_i \in \mathcal{F}_i\}$ *is the product σ-algebra, .*

- $(\prod_{i \in I} \mu_i)(\{x_i : i \in I\}) = \prod_{i \in I}(\mu_i(x_i))$ *for $x_i \in X_i$, is the product measure.*

If the μ_i are probability measures, then the product measure is also a probability measure.

4.1.2 Measure on Infinite Words

In the previous subsection we dealt with measures on arbitrary spaces. However, in defining the behavior of probabilistic transducers, we will have to work with a highly structured set: the space of infinite sequences over some alphabet. This is because, when the transition function of the transducer is probabilistic instead of nondeterministic, a sequence of inputs induces a probability distribution over the set of state sequences of the same length, which in turn defines a distribution over the set of output sequences. In this subsection we briefly review some useful properties of such spaces.

In order to define a measure on the space of infinite sequences over some alphabet Σ, we must first choose a suitable σ-algebra. The natural choice here is to use the σ-algebra generated by the basic open sets of the natural topology on Σ^ω. The basic open set is called a *cylinder* and is defined as the set of all possible infinite extensions of a given finite word. Intuitively, if we view Σ^ω as an infinite tree, then a cylinder is a finite path followed by a complete infinite subtree.

Definition 19 (Cylinders) *Given an alphabet Σ, and a finite word $\beta \in \Sigma^*$, the cylinder C_β is defined as the set $\{\beta \cdot \alpha : \alpha \in \Sigma^\omega\}$, where Σ^ω is the set of all infinite words over Σ. The finite word generating a cylinder is called the* handle *of the cylinder.*

Definition 20 (Borel σ-algebra) *Given an alphabet Σ, the Borel σ-algebra over Σ^ω, denoted by $\mathcal{B}(\Sigma)$, is the σ-algebra generated by the set of cylinders of Σ^ω.*

We want to define a probability measure on Σ^ω. Consider what such a measure μ would look like, and the value it would take on cylinders. Given a cylinder C_β, we can write it as a disjoint union of cylinders $C_\beta = \bigcup_{x\in\Sigma} C_{\beta\cdot x}$. Then, by countable additivity, $\mu(C_\beta) = \sum_{x\in\Sigma} \mu(C_{\beta\cdot x})$. Now, we can interpret the function μ on cylinders as a function f on finite words, since there is a one to one correspondence between cylinders and finite words. Turning things around, such a function $f : \Sigma^* \to [0,1]$ can be used to define the measure on cylinders. The value that the measure takes on cylinders can in turn define the value it takes on other sets in the σ-algebra. This intuition is captured by the next definition and the theorem following it.

Definition 21 (Prefix function) *Let Γ be a countable alphabet and Γ^* be the set of all finite words over Γ. A prefix function over Γ is a function $f : \Gamma^* \to [0,1]$ that satisfies the following properties:*

- $f(\epsilon) = 1$.
- $f(\alpha) = \sum_{x\in\Gamma} f(\alpha\cdot x)$ *for all* $\alpha \in \Gamma^*$.

Theorem 6 *Given an alphabet Σ, and a prefix function f over Σ, there is a unique probability measure $\mu : \mathcal{B}(\Sigma) \to [0,1]$ such that for every cylinder C_β of Σ^ω, $\mu(C_\beta) = f(\beta)$.*

4.2 Definition of Probabilistic Transducers

We would like to extend the results of the nondeterministic case to the case where the transition function of the machine is probabilistic, that is, the transitions that the machine takes have probabilities associated with them. We do this by associating each distinct input and state combination with a probability measure on the set of states.

Definition 22 (Probabilistic Transducer) *A probabilistic transducer is a tuple,* $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ *where*

- Q *is a countable set of states.*
- q_0 *is the start state.*
- I *is a finite set of input channels.*
- O *is a finite set of output channels.*
- Σ *is a finite alphabet of actions (or values).*
- $\sigma : I \cup O \to 2^{\Sigma}$ *is a function that allocates a channel alphabet to each channel.*
- $\lambda : Q \times O \to \Sigma$ *is the output function of the machine.* $\lambda(q, o) \in \sigma(o)$ *is the value that is output on channel* o *when the machine is in state* q.
- $\delta : Q \times \sigma(i_1) \times \ldots \times \sigma(i_n) \to \Omega$, *where* $I = \{i_1, \ldots, i_n\}$ *and* Ω *is the set of all probability measures on* Q, *is the transition function mapping the current state and input to a probability distribution on the set of states.*

Input assignments, output assignments, output mapping, $In(M)$ and $Out(M)$ are defined just as for the nondeterministic case (Section 3.1).

Note that the only difference between a probabilistic transducer and a non-deterministic one is in the definition of the transition function δ. Also note that in Definition 3 in Section 3.2, the transition function of the composition is defined as the cartesian product of the transition functions of the component transducers. So if we can define a cartesian product operation for the transition function of probabilistic transducers, then the definitions for general and binary composition, as well as the composition theorem and its proof, which equates the two, will carry over in their entirety without any change from the non-deterministic case. Such a product operation is provided by the product measure (Theorem 5). Intuitively, a transition of a composite machine can be viewed as multiple independent transitions of its components, one for each component. Then the probability of making such a composite transition must be the same as the probability of the multiple independent transitions occurring at the same time, which is just the product of the individual probabilities. This is formally captured by the product measure construction.

We will not restate the definitions for general and binary composition, as well as the composition theorem. From here on, transducer will mean probabilistic transducer and composition will mean binary composition of probabilistic transducers. In the next section, we define appropriate notions of probabilistic behavior for transducers.

4.3 Probabilistic Executions and Traces

A single input assignment $f(I)$ to a transducer M in state q_0, induces a probability distribution on the set of states Q, given by $\delta(q_0, f(I))$. Similarly, a pair of input assignments $f(I), g(I)$ applied in sequence should give a probability distribution on the set of all pairs of states Q^2. Intuitively, the probability assigned to the pair (q_1, q_2) should be the probability that M steps through q_1 and q_2 in sequence as we input $f(I)$ followed by $g(I)$, which is $\delta(q_0, f(I))(q_1) \times \delta(q_1, g(I))(q_2)$. If we assign such a probability to each pair of states, we find that the resultant distribution turns out to be a probability measure. A similar procedure can be applied to any finite length of input sequence. Thus, given an input sequence of finite length n, we can obtain a probability distribution on the set Q^n, where the probability assigned to an element of Q^n can be intuitively interpreted as the probability of the transducer going through that sequence of states in response to the input sequence.

This procedure breaks down when we consider an infinite sequence of inputs, because Q^ω, the set of infinite sequences over Q, is uncountable and defining the probability for singleton elements is not sufficient to define a distribution. In fact, the probability of each individual infinite sequence of states could very well be zero (similar to the case of the uniform distribution over a finite interval of the real line). In order to obtain a distribution, we need to define the probability for all measurable subsets of Q^ω. We know from Section 4.1.2 that the suitable σ-algebra to use here is the Borel σ-algebra over Q^ω.

Theorem 6 is the bridge between the case of finite sequences of states, which we intuitively know how to handle, and the infinite case where the procedure of looking at individual sequences breaks down. The theorem tells us that if we can obtain a prefix function on the set of states Q, then we can use it to obtain a measure on Q^ω. Note that a prefix function deals only with finite sequences, and essentially captures the idea that the probability of visiting a particular state q must be the same as the probability of visiting q and then going to some arbitrary state. In a similar vein, the probability of heads in a single toss of a coin must be the same as the probability of heads in the first of two tosses, when we do not care about the results of the second toss. We use the transition function of the transducer to define the prefix function on Q.

Definition 23 *Let $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ be a transducer, and $\pi = \langle f_i \rangle_{i=0}^{\infty} \in In(M)^\omega$ be an infinite sequence of inputs. Then we can inductively define a prefix function $\rho(M, \pi)$ over Q as follows:*

- $\rho(M, \pi)(\epsilon) = 1$.
- $\rho(M, \pi)(q) = \delta(q_0, f_0(I))(q)$ *for* $q \in Q$.
- $\rho(M, \pi)(\alpha \cdot p \cdot q) = \rho(M, \pi)(\alpha \cdot p) \times \delta(p, f_{|\alpha \cdot p|}(I))(q)$ *for* $q \in Q$.

Proposition 1 $\rho(M, \pi)$ *is a prefix function over Q.*

Proof. Let $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ and $\pi = \langle f_i \rangle_{i=0}^{\infty} \in In(M)^\omega$. By Definition 23, $\rho(M, \pi)(\epsilon) = 1$. Also, $\sum_{q \in Q} \rho(M, \pi)(\epsilon \cdot q) = \sum_{q \in Q} \delta(q_0, f_0(I))(q) = 1$, because $\delta(q_0, f_0(I))$ is a probability measure on Q. So the definition of prefix function is satisfied for the case of the empty word. Now let $\alpha \in Q^*$ such that $\alpha \neq \epsilon$. Then $\alpha = \beta \cdot p$ for some $\beta \in Q^*$ and $p \in Q$. Then, by Definition 23, for any $q \in Q$, $\rho(M, \pi)(\alpha \cdot q) =$

$\rho(M,\pi)(\beta \cdot p \cdot q) = \rho(M,\pi)(\beta \cdot p) \times \delta(p, f_{|\beta \cdot p|}(I))(q)$. Therefore $\sum_{q \in Q} \rho(M,\pi)(\alpha \cdot q) = \rho(M,\pi)(\alpha) \times \sum_{q \in Q} \delta(p, f_{|\beta \cdot p|}(I))(q)$. Since $\delta(p, f_{|\beta \cdot p|}(I))$ is a probability measure over Q, its total measure over Q must be 1. Hence we have, $\sum_{q \in Q} \rho(M,\pi)(\alpha \cdot q) = \rho(M,\pi)(\alpha)$, and so $\rho(M,\pi)$ is a prefix function over Q. ⊠

So given any infinite sequence of inputs, we can obtain a prefix function on the set of states and thus obtain a unique probability measure on $\mathcal{B}(Q)$. We call such a measure an *execution measure*, since it plays the same role in defining the behavior of the transducer that executions did in the non-deterministic case.

Definition 24 (Execution Measure) *Let $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ be a transducer, and $\pi \in In(M)^\omega$ be an infinite sequence of inputs. The execution measure of π over M, denoted $\mu(M,\pi)$, is the unique probability measure on $\mathcal{B}(Q)$ such that for every cylinder C_β of Q^ω, $\mu(M,\pi)(C_\beta) = \rho(M,\pi)(\beta)$.*

Since the output of a transducer depends only on its state, each state q maps to an output assignment $h(q) : O \rightarrow \Sigma$ such that $h(q)(o) = \lambda(q, o)$ for all $o \in O$. Then we can extend $h : Q \rightarrow Out(M)$ to a mapping from sequences of states to sequences of output assignments in the natural way: for $\alpha, \beta \in Q^*$, $h(\alpha \cdot \beta) = h(\alpha) \cdot h(\beta)$. We can also extend it to the case of infinite sequences. Since an infinite sequence of states is just a mapping $g : \mathbb{N} \rightarrow Q$ from the natural numbers to the set of states, then $h \circ g : \mathbb{N} \rightarrow Out(M)$ is a mapping from the naturals to the set of outputs. We now show that $h : Q^\omega \rightarrow Out(M)^\omega$ is a *measurable function*, that is h^{-1} maps measurable subsets of $Out(M)^\omega$ to measurable subsets of Q^ω.

Lemma 3 *The extended output mapping, $h : Q^\omega \to Out(M)^\omega$, of a transducer M is a measurable function.*

Proof. It suffices to show that h^{-1} maps cylinders of $Out(M)^\omega$ to measurable subsets of Q^ω. Let $\alpha \in Out(M)^*$, and consider $h^{-1}(C_\alpha)$. Now $h^{-1}(C_\alpha) = \{\beta \in Q^\omega : h(\beta) \in C_\alpha\} = \{\beta_1 \cdot \beta_2 : \beta_1 \in Q^*, h(\beta_1) = \alpha, \beta_2 \in Q^\omega, h(\beta_2) \in Out(M)^\omega\} = \{\beta_1 \cdot \beta_2 : \beta_1 \in Q^*, h(\beta_1) = \alpha, \beta_2 \in Q^\omega\} = \bigcup_{\gamma \in A} C_\gamma$, where $A = \{\beta \in Q^* : h(\beta) = \alpha\}$. Therefore h^{-1} maps a cylinder to a union of cylinders, which is a measurable set, and thus h is a measurable function. $\boxtimes$

The above result allows us to use h to translate a measure on Q^ω into a measure on $Out(M)^\omega$. So for each execution measure, we can define a *trace* measure, which is the analog of a trace in the non-deterministic case.

Definition 25 (Trace Measure) *Let $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ be a transducer, π be an infinite sequence of inputs, and $h : Q \to Out(M)$ be the output mapping. The trace measure of π over M, denoted by $\mu_T(M, \pi)$, is the unique probability measure on $\mathcal{B}(Out(M))$ defined as follows: for all $A \in \mathcal{B}(Out(M))$, $\mu_T(M, \pi)(A) = \mu(M, \pi)(h^{-1}(A))$.*

The trace measures of a transducer are the observable part of its behavior. We define the probabilistic version of trace semantics in terms of trace measures.

Definition 26 (Trace Equivalence) *Two transducers M_1 and M_2 are trace equivalent, denoted by $M_1 \sim_T M_2$, if*

- *$In(M_1) = In(M_2)$ and $Out(M_1) = Out(M_2)$.*

- *For all* $\pi \in In(M_1)^\omega$, $\mu_T(M_1, \pi) = \mu_T(M_2, \pi)$.

The first condition is purely syntactic, and is essentially the requirement that the two transducers have the same input/output interface. The second condition says that they must have identical trace measures.

In contrast to the the non-deterministic case, instead of linear traces and executions, the basic semantic object here is a probability distribution over the set of all infinite words over some alphabet (in other words, an infinite tree). Before attempting to obtain full abstraction results, we show that the semantics defined above has an equivalent formulation in terms of *finite* linear traces and executions. The key insight involved in reducing an infinitary semantics to a finitary one is that each trace and execution measure is defined completely by the value it takes on cylinders, and the cylinders have a one-to-one correspondence with the set of finite words. Each cylinder is in some sense equivalent to its handle.

Definition 27 (Execution) *Let* $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ *be a probabilistic transducer. An* execution *of* M *is a sequence of pairs* $\langle f_i, s_i \rangle_{i=0}^n$ *such that* $n \in \mathbb{N}$*, and for all* $i \geq 0$, $s_i \in Q$ *and* $f_i \in In(M)$*. The set of all executions of machine* M *is denoted* $exec(M)$.

Note that in contrast to the non-deterministic case, the definition of execution does not depend on the transition function δ. Also, all executions are finite in length.

Definition 28 (Likelihood of an execution) *Let* $\alpha = \langle f_i, s_i \rangle_{i=0}^n \in exec(M)$*. Then the* likelihood *of* α*, denoted by* $\chi_M(\alpha)$*, is defined as follows:*

$$\chi_M(\alpha) = \delta(q_0, f_0(I))(s_0) \times \Pi_{i=1}^n(\delta(s_{i-1}, f_i(I))(s_i))$$

where the product $\Pi_{i=1}^{n}$ is defined to have value 1 *for* $n = 0$.

Definition 29 (Trace) *Let* $\alpha = \langle f_i, s_i \rangle_{i=0}^{n} \in exec(M)$. *The trace of* α, *denoted by* $[\alpha]$, *is a sequence of pairs* $\langle f_i, h(s_i) \rangle_{i=0}^{n}$, *where* $h : Q \rightarrow Out(M)$ *is the output mapping of* M. *The set of all traces of machine* M, *denoted by* $Tr(M)$, *is the set* $\{[\alpha] | \alpha \in exec(M)\}$. *An element of* $Tr(M)$ *is called a trace of* M.

Definition 30 (Likelihood of a Trace) *Let* $t \in Tr(M)$ *be a finite trace of* M. *Then the likelihood of* t, *denoted by* $\chi_M(t)$, *is defined as follows:*

$$\chi_M(t) = \sum_{\alpha \in Exec(M), [\alpha]=t} \chi_M(\alpha)$$

Note that in our definition of trace, we ignore $h(q_0)$, since the initial state of a transducer is unique.

The length of a trace α is defined to be the length of the underlying execution and is denoted by $|\alpha|$. Once again, the transition function is not needed to define traces, and so a trace is a purely syntactic object. The semantical nature of a trace is now completely captured by the likelihood of the trace. Note that if two transducers have the same interface, they have the same set of traces: $Tr(M_1) = Tr(M_2)$ if and only if $In(M_1) = In(M_2)$ and $Out(M_1) = Out(M_2)$.

The next theorem offers a simpler definition of trace equivalence. We need the following propositions for its proof.

Proposition 2 *Let* $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$*,* $\pi = \langle f_i \rangle_{i=0}^{\infty} \in In(M)^{\omega}$*,* $\alpha = \langle f_i, s_i \rangle_{i=0}^{n} \in exec(M)$*, and* $\beta = \langle s_i \rangle_{i=0}^{n} \in Q^*$*. Then* $\chi_M(\alpha) = \rho(M, \pi)(\beta)$*.*

Proof. We prove the desired equality by induction on the length of the execution. If $n = 0$, then by Definitions 28 and 23, $\chi_M(\alpha) = \delta(q_0, f_0(I))(s_0) = \rho(M, \pi)(s_0)$. Let $n > 0$, $\alpha = \gamma \cdot (f_{n-1}, s_{n-1}) \cdot (f_n, s_n)$, $\beta = \eta \cdot s_{n-1} \cdot s_n$. Then, by Definition 28, $\chi_M(\alpha) = \chi_M(\gamma \cdot (f_{n-1}, s_{n-1})) \times \delta(s_{n-1}, f_n(I))(s_n)$, and by the induction hypothesis, $\chi_M(\gamma \cdot (f_{n-1}, s_{n-1})) = \rho(M, \pi)(\eta \cdot s_{n-1})$. So $\chi_M(\alpha) = \rho(M, \pi)(\eta \cdot s_{n-1}) \times \delta(s_{n-1}, f_n(I))(s_n) = \rho(M, \pi)(\eta \cdot s_{n-1} \cdot s_n) = \rho(M, \pi)(\beta)$ (the second equality follows from Definition 23). ⊠

Proposition 3 *Let* $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$*,* $\pi = \langle f_i \rangle_{i=0}^{\infty} \in In(M)^{\omega}$*,* $t = \langle f_i, w_i \rangle_{i=0}^{n} \in Tr(M)$*, and* $\beta = \langle w_i \rangle_{i=0}^{n} \in Out(M)^*$*. Then* $\chi_M(t) = \mu_T(M, \pi)(C_\beta)$*.*

Proof. Let $h : Q \to Out(M)$ be the output mapping of M. Then, by Definition 30 and Proposition 2, $\chi_M(t) = \sum_{\alpha \in exec(M), [\alpha] = t} \chi_M(\alpha) = \sum_{\gamma \in h^{-1}(\beta)} \rho(M, \pi)(\gamma)$. Also, by Definition 25, $\mu_T(M, \pi)(C_\beta) = \mu(M, \pi)(h^{-1}(C_\beta)) = \mu(M, \pi)(\bigcup_{\gamma \in h^{-1}(\beta)} C_\gamma)$. Since cylinders with handles of the same length are necessarily disjoint, and $\mu(M, \pi)$ is a measure, using countable additivity we get $\mu(M, \pi)(\bigcup_{\gamma \in h^{-1}(\beta)} C_\gamma) = \sum_{\gamma \in h^{-1}(\beta)} \mu(M, \pi)(C_\gamma) = \sum_{\gamma \in h^{-1}(\beta)} \rho(M, \pi)(\gamma)$ (the second equality follows from Definition 24). Therefore, $\chi_M(t) = \mu_T(M, \pi)(C_\beta)$. ⊠

Theorem 7 *Let* M_1 *and* M_2 *be probabilistic transducers with* $Tr(M_1) = Tr(M_2)$*. Then* $M_1 \sim_T M_2$ *if and only if, for all* $t \in Tr(M_1)$*,* $\chi_{M_1}(t) = \chi_{M_2}(t)$*.*

Proof.

If: Let $M_1 \sim_T M_2$ and $t = \langle f_i, w_i \rangle_{i=0}^{n} \in Tr(M_1)$. Let $\pi = \langle f_i \rangle_{i=0}^{\infty} \in In(M_1)^{\omega}$ and $\beta = \langle w_i \rangle_{i=0}^{n} \in Out(M_1)^*$. Since $M_1 \sim_T M_2$, then the trace measure induced by π must be the same for both transducers, i.e., $\mu_T(M_1, \pi) = \mu_T(M_2, \pi)$. In particular, $\mu_T(M_1, \pi)(C_\beta) = \mu_T(M_2, \pi)(C_\beta)$. By Proposition 3, we have $\chi_{M_1}(t) = \chi_{M_2}(t)$.

Only If: Let $Tr(M_1) = Tr(M_2)$, and for all $t \in Tr(M_1)$, $\chi_{M_1}(t) = \chi_{M_2}(t)$. Given any $\pi = \langle f_i \rangle_{i=0}^{\infty} \in In(M_1)^{\omega}$, $\beta = \langle w_i \rangle_{i=0}^{n} \in Out(M_1)^*$, and $u = \langle f_i, w_i \rangle_{i=0}^{n} \in Tr(M_1)$, we have by assumption, $\chi_{M_1}(u) = \chi_{M_2}(u)$, and therefore, by Proposition 3, $\mu_T(M_1, \pi)(C_\beta) = \mu_T(M_2, \pi)(C_\beta)$. Since the measures are completely determined by their value on cylinders, we have $\mu_T(M_1, \pi) = \mu_T(M_2, \pi)$ for all $\pi \in In(M_1)^{\omega}$ and so $M_1 \sim_T M_2$.

⊠

The theorem above allows us to reason in terms of single finite traces. This is a significant reduction in complexity from the original definition in terms of probability distributions on infinite trees. In particular this simplifies the proof of the full abstraction results to follow.

In the next section, we use this alternative characterization of trace equivalence to show that it is fully abstract with respect to contextual equivalence. First we need to be able to calculate the likelihoods of traces of a composition from the likelihoods of traces of its components. In the propositions that follow, composition of traces and executions is defined exactly as for the non-deterministic case (see Definitions 8 and 9 in Section 3.3).

Proposition 4 *Let M_1 and M_2 be transducers, $C \in Conn(\{M_1, M_2\})$, $\alpha \in exec(M_1)$ and $\beta \in exec(M_2)$ such that $\alpha||_C\beta \in exec(M_1||_C M_2)$. Then*

$$\chi_{M_1||_C M_2}(\alpha||_C\beta) = \chi_{M_1}(\alpha) \times \chi_{M_2}(\beta)$$

Proof. Let $M_k = (Q_k, q_0^k, I_k, O_k, \Sigma_k, \sigma_k, \lambda_k, \delta_k)$, $k \in \{1, 2\}$, and $M = (Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta) = M_1||_C M_2$, where $C \in Conn(\{M_1, M_2\})$. Let $\alpha = \langle f_i, s_i \rangle_{i=0}^n \in exec(M_1)$, $\beta = \langle g_i, r_i \rangle_{i=0}^n \in exec(M_2)$, and $\alpha||_C\beta \in exec(M)$. We define $e_i : I \to \Sigma_1 \cup \Sigma_2$ as $e_i(in) = f_i(in)$, if $in \in I_1$, and $e_i(in) = g_i(in)$, otherwise.

By the definition of composition, $\delta((s_j, r_j), e_{j+1}(I)) = \delta_1(s_j, f_{j+1}(I_1)) \times \delta_2(r_j, g_{j+1}(I_2))$. Applying this to the expansion of $\chi_M(\alpha||_C\beta)$, given by Definition 28, and then rearranging the terms in the product, we obtain the desired equality.

$$\begin{aligned}
&\chi_M(\alpha||_C\beta) \\
&= \delta((q_0^1, q_0^2), e_0(I))(s_0, r_0) \times \Pi_{i=1}^n(\delta((s_{i-1}, r_{i-1}), e_i(I))(s_i, r_i)) \\
&= \delta_1(q_0^1, f_0(I_1)) \times \delta_2(q_0^2, g_0(I_2)) \times \Pi_{i=1}^n(\delta_1(s_{i-1}, f_i(I_1)) \times \delta_2(r_{i-1}, g_i(I_2))) \\
&= (\delta_1(q_0^1, f_0(I_1)) \times \Pi_{i=1}^n\delta_1(s_{i-1}, f_i(I_1))) \times (\delta_2(q_0^2, g_0(I_2)) \times \Pi_{i=1}^n\delta_2(r_{i-1}, g_i(I_2))) \\
&= \chi_{M_1}(\alpha) \times \chi_{M_2}(\beta)
\end{aligned}$$

⊠

Proposition 5 *Let M_1 and M_2 be transducers, $C \in Conn(\{M_1, M_2\})$ and $t \in Tr(M_1||_C M_2)$. Then $\chi_{M_1||_C M_2}(t) = \sum_{u,v} \chi_{M_1}(u) \times \chi_{M_2}(v)$ where $u \in Tr(M_1)$, $v \in Tr(M_2)$ such that*

$u||_C v = t$.

Proof.

$$\begin{aligned}
&\sum_{u||_C v=t} \chi_{M_1}(u) \times \chi_{M_2}(v) \\
&= \sum_{u||_C v=t} ((\sum_{[\alpha]=u} \chi_{M_1}(\alpha)) \times (\sum_{[\beta]=v} \chi_{M_2}(\beta))) && \text{(using Dfn. 30)} \\
&= \sum_{u||_C v=t} (\sum_{[\alpha]=u,[\beta]=v} (\chi_{M_1}(\alpha) \times \chi_{M_2}(\beta))) && \text{(rearranging terms)} \\
&= \sum_{u||_C v=t} (\sum_{[\alpha]=u,[\beta]=v} (\chi_{M_1||_C M_2}(\alpha||_C\beta))) && \text{(using Prop. 4)} \\
&= \sum_{[\alpha||_C\beta]=t} \chi_{M_1||_C M_2}(\alpha||_C\beta) && \text{(rearranging terms)} \\
&= \chi_{M_1||_C M_2}(t)
\end{aligned}$$

⊠

4.4 Full Abstraction

As in the nondeterministic case, here again we want to show that our semantics recognizes exactly the distinctions that can be detected by some context and vice versa. The two sides of this property are often called, resp., observational congruence and adequacy. Here we first prove the stronger condition that trace semantics is a congruence with respect to the composition operation. Then the property of observational congruence with respect to contexts automatically follows as a corollary.

Theorem 8 (Congruence Theorem) *Let $M_1 \sim_T M_3$, $M_2 \sim_T M_4$ and $C \in Conn(\{M_1, M_2\})$. Then $M_1||_C M_2 \sim_T M_3||_C M_4$. We say that $\sim_T$ is congruent with respect to composition.*

Proof. Let $t \in Tr(M_1||_C M_2)$. Since $Tr(M_1) = Tr(M_3)$ and $Tr(M_2) = Tr(M_4)$, we have $\{(u,v) : u \in Tr(M_1), v \in Tr(M_2), u||_C v = t\} = \{(u,v) : u \in Tr(M_3), v \in Tr(M_4), u||_C v = t\}$. Then, by Proposition 5 and Theorem 7,

$\chi_{M_1||_C M_2}(t) = \sum_{\{(u,v):u||_C v=t\}} \chi_{M_1}(u) \times \chi_{M_2}(v) = \sum_{\{(u,v):u||_C v=t\}} \chi_{M_3}(u) \times \chi_{M_4}(v) = \chi_{M_3||_C M_4}(t)$. Again, by Theorem 7, we have $M_1||_C M_2 \sim_T M_3||_C M_4$. $\boxtimes$

Similar to the nondeterministic case, an immediate corollary of Theorem 8 is the fact that no context can distinguish between two trace-based equivalent transducers.

Corollary 2 *Let M_1 and M_2 be transducers, and $M_1 \sim_T M_2$. Then for all transducers M and all $C \in Conn(\{M, M_1\}) = Conn(\{M, M_2\})$, we have that $M||_C M_1 \sim_T M||_C M_2$.*

We can easily complete the other requirement of showing full abstraction of trace semantics with respect to contextual equivalence, by demonstrating a trivial context that makes a distinction between trace inequivalent transducers. Let M_1 and M_2 be transducers such that $M_1 \not\sim_T M_2$. Now we can simply choose an empty set of connections C, and a completely deterministic transducer M, as the basis of our testing context. In this case the trace measures of the composition $M_1||_C M$ will be the same as the trace measures of M_1 alone, and full abstraction would be trivially achieved. Here we give a stronger result, similar to that already described for the nondeterministic case. We show that given two inequivalent transducers with the same interface, we can always find a third transducer that is a *tester* (see Section 3.4) for them and that distinguishes between the first two, when it is *maximally* connected with them.

Theorem 9 *Let M_1 and M_2 be transducers with $Tr(M_1) = Tr(M_2)$ and $M_1 \not\simeq_T M_2$. Then there exists a transducer M and $C \in Conn(\{M, M_1\}) = Conn(\{M, M_2\})$, such that M is a tester for M_1 and M_2 w.r.t. C, and $M||_C M_1 \not\simeq_T M||_C M_2$.*

Proof. Let $M_1 = (Q_1, q_0^1, I', O', \Sigma, \sigma_1, \lambda_1, \delta_1)$ and $M_2 = (Q_2, q_0^2, I', O', \Sigma, \sigma_2, \lambda_2, \delta_2)$. Since $M_1 \not\simeq_T M_2$, by Theorem 7, there exists $t \in Tr(M_1) = Tr(M_2)$, such that $\chi_{M_1}(t) \neq \chi_{M_2}(t)$. Let $t = \langle f_i, \omega_i \rangle_{i=0}^n$ for finite n. We define the testing transducer $(Q, q_0, I, O, \Sigma, \sigma, \lambda, \delta)$ as follows:

- $Q = \{q_0, q_1, \ldots, q_{n+1}\} \cup \{q_f\}$ is a finite set of states, with q_f being a special sink state.
- For each $o \in O'$, we create an input channel in_o in I and assign alphabet $\sigma(in_o) = \sigma_1(o)$ to it.
- For each $in \in I'$, we create an output channel o_{in} in O and assign alphabet $\sigma(o_{in}) = \sigma_1(in)$ to it.
- An extra output channel o_t, with alphabet $\{a, b\} \subseteq \Sigma$, that will be the only visible channel remaining after composition.
- $\lambda(q_i, o_{in}) = f_i(in)$, $\lambda(q_i, o_t) = a$ and $\lambda(q_f, o_t) = b$. In all other cases, we don't care what output M produces, and λ can be assumed to be arbitrary.

- The transition function δ is defined as follows

  - $\delta(q_i, h(I))(q_{i+1}) = 1$, if for all $in_o \in I$, $h(in_o) = \omega_i(o)$.

  - $\delta(q_i, h(I))(q) = 0$, if $q \neq q_{i+1}$ and for all $in_o \in I$, $h(in_o) = \omega_i(o)$.

  - $\delta(q, h(I))(q_f) = 1$, if for some $in_o \in I$, $h(in_o) \neq \omega_i(o)$.

  - $\delta(q, h(I))(q') = 0$, if $q' \neq q_f$ and for some $in_o \in I$, $h(in_o) \neq \omega_i(o)$.

We define the set of connections $C \in Conn(\{M, M_1\}) = Conn(\{M, M_2\})$ as follows: for all $in \in I'$, $o \in O'$, $(in, o_{in}) \in C$ and $(in_o, o) \in C$, and nothing else is in C. Now both $M||_C M_1$ and $M||_C M_2$ have exactly one channel each, which is the output channel o_t belonging to M, and so M is a tester for M_1 and M_2 w.r.t. C.

The transducer M simulates a deterministic transducer in that from each state and input combination, a single transition has probability 1 and all other transitions have zero probability. Further it is designed to follow the execution of the distinguishing trace t. As soon as the computation of the machine being tested diverges from this trace, M will enter its sink state and switch its visible output from a to b. When the machine being tested undergoes an execution corresponding to the trace t, the composition will output the trace a^{n+1}. We now show that the likelihood of this trace is different for $M||_C M_1$ and $M||_C M_2$, and this will complete the proof. By Proposition 5, we have $\chi_{M||_C M_1}(a^{n+1}) = \sum_{u,v} \chi_M(u) \times \chi_{M_1}(v)$ where $u \in Tr(M)$, $v \in Tr(M_1)$ such that $u||_C v = a^{n+1}$. Now, by design, there is only a single such $u \in Tr(M)$, and a single such $v \in Tr(M_1)$, and we also have $\chi_M(u) = 1$, and $v = t$. So $\chi_{M||_C M_1}(a^{n+1}) = \chi_{M_1}(t)$. But since, by symmetry, this argument applies to M_2 as well, we have $\chi_{M||_C M_2}(a^{n+1}) = \chi_{M_2}(t)$, and therefore $\chi_{M||_C M_1}(a^{n+1}) \neq$

$\chi_{M||_C M_2}(a^{n+1})$. Thus the testing transducer M can distinguish between M_1 and M_2. ☒

The previous two theorems, taken together, show that trace equivalence is fully abstract with respect to contextual equivalence.

Chapter 5

Conclusion

It could be fairly argued that the arguments raised here have been raised before.

- Testing equivalence, introduced in [23], is clearly a notion of contextual equivalence. Their answer to the question, "What is a test?", is that a test is any process that can be expressed in the formalism. So a test is really the counterpart of a context in program equivalence. (Though our notion of context in Section 3.2, as a network of transducers, is, a priori, richer.) At the same time, bisimulation equivalence has been recognized as being too fine a relation to be considered as contextual equivalence [10, 11, 12, 32].

- Furthermore, it has also been shown that many notions of process equivalence studied in the literature can be obtained as contextual equivalence with respect to appropriately defined notions of directly observable behavior [13, 41, 46, 52]. These notions fall under the title of *decorated trace equivalence*, as they all start with trace semantics and then endow it with additional observables. These notions have the advantage that, like bisimulation equivalence, they are not blind to issues such as deadlock behavior.

With respect to the first point, it should be noted that despite the criticisms leveled at it, bisimulation equivalence still enjoys a special place of respect in concurrency theory as a reasonable notion of process equivalence [3, 60]. In fact, the close correspondence between bisimulation equivalence and the branching-time logic CTL has been mentioned as an advantage of CTL. Thus, it is not redundant, in our opinion, to reiterate the point that bisimulation and its variants are *not* contextual equivalences.

With respect to the second point we note that our approach is related, but quite different, than that taken in decorated trace equivalence. In the latter approach, the "decorated" of traces is attributed by concurrency theorists. As there is no unique way to decorate traces, one is left with numerous notions of equivalence and with the attitude quoted above that "It is not the task of process theory to find the 'true' semantics of processes, but rather to determine which process semantics is suitable for which applications" [60]. In our approach, only the modelers know what the relevant aspects of behavior are in their applications and only they can decorate traces appropriately, which led to our Principles of Comprehensive Modeling and Observable I/O. In our approach, there is only one "right" of contextual equivalence, which is trace-based equivalence.

Admittedly, the comprehensive-modeling approach is not wholly original, and has been foretold by Brookes [14], who said: "We do not augment traces with extraneous bookkeeping information, or impose complex closure conditions. Instead we incorporate the crucial information about blocking directly in the internal structure of traces. " Still, we believe that it is valuable to carry Brookes's approach further, substantiate it with our three guiding principles, and demonstrate it in the framework of transducers.

An argument that may be leveled at our comprehensive-modeling approach is that it requires a low-level view of systems, one that requires modeling all relevant behavioral aspects. This issue was raised by Vaandrager in the context of I/O Automata [59]. Our response to this criticism is twofold. First, if these low level details (e.g., deadlock behavior) are relevant to the application, then they better be spelled out by the modeler rather than by the concurrency theorist.

As discussed earlier, whether deadlocked termination should be distinguished from normal termination depends on the level of abstraction at which the model operates. It is a pragmatic decision rather than a theoretical decision. Second, if the distinction between normal termination and deadlocked termination is important to some users but not others, one could imagine language features that would enable explicit modeling of deadlocks when such modeling is desired, but would not force users to apply such explicit modeling. The underlying semantics of the language, say, in terms of structured operational semantics [38], can expose deadlocked behavior for some language features and not for others. In other words, Vaandrager's concerns about users being force to adopt a low-level view should be addressed by designing more flexible languages, and not by introducing new notions of process equivalence.

Note that the alternative to our approach is to accept formalisms for concurrency that are not fully specified and admit a profusion of different notions of process equivalence.

In conclusion, this dissertation puts forward an admittedly provocative thesis, which is that process-equivalence theory allowed itself to wander in the "wilderness" for lack of accepted guiding principles. The obvious definition of contextual equivalence was not scrupulously adhered to, and the underspecificity of the formalisms proposed led to too many interpretations of equivalence. While one may not realistically expect a single dissertation to overwrite about 30 years of research, a more modest hope would be to stimulate a lively discussion on the basic principles of process-equivalence theory.

Bibliography

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languagues and Systems*, 15(1):73–132, 1993.

[2] S. Abramsky. Observation equivalence as a testing equivalence. *Theor. Comput. Sci.*, 53:225–241, 1987.

[3] S. Abramsky. What are the fundamental structures of concurrency?: We still don't know! *Electr. Notes Theor. Comput. Sci.*, 162:37–41, 2006.

[4] P. Aczel. Non-well-founded sets. Technical report, CSLI Lecture Notes, no. 14, Stanford University, 1988.

[5] P. Aczel and N.P. Mendler. A final coalgebra theorem. In *Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 1989.

[6] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 296–211. Springer, 2002.

[7] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[8] J. F. A. K. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1983.

[9] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[10] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.

[11] B. Bloom and A. R. Meyer. Experimenting with process equivalence. *Theor. Comput. Sci.*, 101(2):223–237, 1992.

[12] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. *J. ACM*, 43(5):863–914, 1996.

[13] M. Boreale and R. Pugliese. Basic observables for processes. *Information and Computation*, 149(1):77–98, 1999.

[14] S.D. Brookes. Traces, pomsets, fairness and full abstraction for communicating processes. In *Proc. 13th Int'l Conf. on Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 466–482. Springer, 2002.

[15] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[16] J. Carmo and A. Sernadas. Branching vs linear logics yet again. *Formal Aspects of Computing*, 2:24–59, 1990.

[17] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[18] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 1988.

[19] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languagues and Systems*, 8(2):244–263, 1986.

[20] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Decade of Concurrency – Reflections and Perspectives (Proceedings of REX School)*, volume 803 of *Lecture Notes in Computer Science*, pages 124–175. Springer, 1993.

[21] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[22] D.L. Cohn. *Measure Theory*. Birkhäuser Boston, 1994.

[23] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

[24] D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits.* MIT Press, 1989.

[25] C. Eisner and D. Fisman. *A Practical Introduction to PSL.* Springer, 2006.

[26] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th Int. Colloq. on Automata, Languages, and Programming*, pages 169–181, 1980.

[27] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[28] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 84–96, 1985.

[29] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, Mass., 1995.

[30] K. Fisler and M.Y. Vardi. Bisimulation minimization and symbolic model checking. *Formal Methods in System Design*, 21(1):39–78, 2002.

[31] R. Goering. Model checking expands verification's scope. *Electronic Engineering Today*, February 1997.

[32] J.F. Groote. Transition system specifications with negative premises. *Theor. Comput. Sci.*, 118(2):263–299, 1993.

[33] G.D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[34] P.R. Halmos. *Measure Theory*. Springer Verlag, 1978.

[35] J. Y. Halpern. *Reasoning About Uncertainty*. MIT Press, Cambridge, Mass., 2003.

[36] J.Y. Halpern. On ambiguities in the interpretation of game trees. *Games and Economic Behavior*, 20:66–96, 1997.

[37] J. Hartmanis and R.E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice Hall, 1966.

[38] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[39] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[40] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[41] B. Jonsson. A fully abstract trace model for dataflow networks. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 155–165, 1989.

[42] J. Kramer. Is abstraction the key to computing? *Comm. ACM*, 50(4):36–42, 2007.

[43] L. Lamport. "Sometimes" is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symp. on Principles of Programming Languages*, pages 174–185, 1980.

[44] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

[45] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[46] M.G. Main. Trace, failure and testing equivalences for communicating processes. *Int'l J. of Parallel Programming*, 16(5):383–400, 1987.

[47] W.W. Marek and M. Trusczynski. *Nonmonotonic Logic: Context-Dependent Reasoning*. Springer, 1997.

[48] J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artif. Intell.*, 13(1-2):27–39, 1980.

[49] R. Milner. Processes: a mathematical model of computing agents. In *Logic Colloquium*, pages 157–173. North Holland, 1975.

[50] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[51] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[52] E.R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Inf.*, 23(1):9–66, 1986.

[53] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. 5th GI Conf. on Theoretical Computer Science*, Lecture Notes in Computer Science, Vol. 104. Springer, Berlin/New York, 1981.

[54] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proc. 12th Int. Colloq. on Automata, Languages, and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 15–32. Springer, 1985.

[55] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 8th ACM Symp. on Principles of Programming Languages*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[56] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32:733–749, 1985.

[57] C. Stirling. Comparing linear and branching time temporal logics. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398, pages 1–20. Springer, 1987.

[58] C. Stirling. The joys of bisimulation. In *23th Int. Symp. on Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Science*, pages 142–151. Springer, 1998.

[59] F.W. Vaandrager. On the relationship between process algebra and input/output automata. In *Proc. 6th IEEE Symp. on Logic in Computer Science*, pages 387–398, 1991.

[60] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001.

[61] R.J. van Glabbeek. What is branching time and why to use it? In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science; Entering the 21st Century*, pages 469–479. World Scientific, 2001.

[62] M.Y. Vardi. Linear vs. branching time: A complexity-theoretic perspective. In *Proc. 13th IEEE Sym.. on Logic in Computer Science*, pages 394–405, 1998.

[63] M.Y. Vardi. Sometimes and not never re-revisited: on branching vs. linear time. In D. Sangiorgi and R. de Simone, editors, *Proc. 9th Int'l Conf. on Concurrency Theory*, Lecture Notes in Computer Science 1466, pages 1–17, 1998.

[64] M.Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[65] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

[66] S. Vijayaraghavan and M. Ramanathan. *A Practical Guide for SystemVerilog Assertions*. Springer, 2005.

[67] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.