Branching vs. Linear Time: Semantical Perspective [*]

Moshe Y. Vardi [†]

Rice University

[*]Updated version of paper posted on home page.

[†]Joint work with Sumit Nain

Formal Verification Today

Verification as debugging: *Failure* of verification identifies bugs.

- Both specifications and programs attempt to formalize informal requirements.
- Verification contrasts two independent formalizations.
- Failure of verification reveals inconsistency between the two formalizations.

Model checking: uncommonly effective debugging tool

- a systematic exploration of the design state space
- good at catching difficult “corner cases”

Designs are Labeled Graphs

Key Idea: Designs can be represented as transition systems (finite-state machines)

State-Transition System: $M = (W, I, R, F, \pi)$

- W: states
- $I \subseteq W$: initial states
- $R \subseteq W \times W$: transition relation
- $F \subseteq W$: fair states
- $\pi : W \rightarrow Powerset(Prop)$: Observation function

Fairness: An assumption of “reasonableness” – restrict attention to computations that visit F infinitely often, e.g., “the channel will be up infinitely often”.

Specifications

Behavioral Specifications: properties of computations.

Examples:

- "No two processes can be in the critical section at the same time." – *safety*
- "Every request is eventually granted." – *liveness*
- "Every continuous request is eventually granted." – *liveness*
- "Every repeated request is eventually granted." – *liveness*

Desideratum: A formal logic to specify requirements – a *temporal logic*.

Temporal Logic

Linear Temporal logic (LTL): logic of temporal sequences (Pnueli, 1977)

Main feature: time is implicit

- *next* φ: φ holds in the next state.
- *eventually* φ: φ holds eventually
- *always* φ: φ holds from now on
- φ *until* ψ: φ holds until ψ holds.

- $\pi, w \models next\ \varphi$ if w $$\begin{tikzcd}\bullet \arrow[r] & \underset{\varphi}{\bullet} \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \ldots\end{tikzcd}$$

- $\pi, w \models \varphi\ until\ \psi$ if w $$\begin{tikzcd}\underset{\varphi}{\bullet} \arrow[r] & \underset{\varphi}{\bullet} \arrow[r] & \underset{\varphi}{\bullet} \arrow[r] & \underset{\psi}{\bullet} \arrow[r] & \bullet \ldots\end{tikzcd}$$

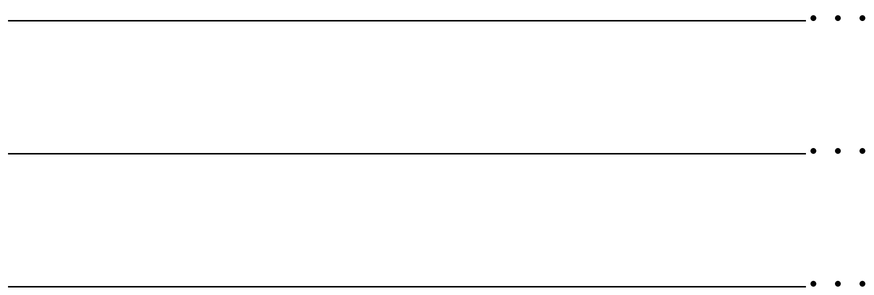
Examples

- always not (CS$_1$ and CS$_2$): mutual exclusion (safety)
- always (Request implies eventually Grant): liveness
- always (Request implies (Request until Grant)): liveness
- always (always eventually Request) implies eventually Grant: liveness

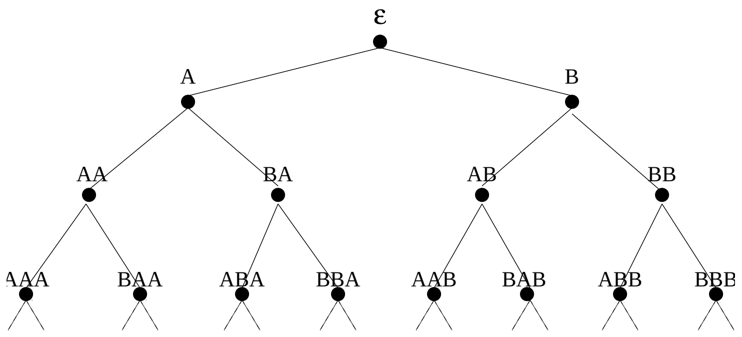
Linear vs. Branching Time

Lamport, 1980: "'Sometimes' is sometimes 'Not Never"'

- **Linear time**: a system induces a set of traces
- *Specs*: describe traces



- **Branching time**: a system induces a trace tree
- *Specs*: describe trace trees



Linear vs. Branching Time: Logics

- **Linear time**: a system generates a set of computations
- *Specs*: describe computations
- LTL: $always\ (\mathsf{request} \rightarrow eventually\ \mathsf{grant})$

- **Branching time**: a program generates a computation tree
- *Specs*: describe computation trees
- CTL: $\forall always\ (\mathsf{request} \rightarrow \forall eventually\ \mathsf{grant})$

Linear vs. Branching Time: Long Debate

- Pnueli, 1977: linear time (LTL)
- Lamport, 1980: linear time better than branching time
- Ben-Ari, Pnueli, and Manna, 1981: branching time (UB)
- Emerson and Clarke, 1981: branching time (CTL)
- Pnueli, 1985: linear time
- Emerson and Halpern, 1986: branching time (CTL*)
- Emerson and Lei, 1985: branching time (CTL*)

Conclusion: Philosophically and technically – a draw.

Linear vs. Branching Time: Prehistory

- A.N. Prior’s first lecture on tense logic, Wellington University, 1954: linear time.

- Prior’s “Time and modality”, 1957: relationship between linear tense logic and modal logic.

- Sep. 1958, letter from Kripke: “[I]n an indetermined system, we perhaps should not regard time as a linear series, as you have done. Given the present moment, there are several possibilities for what the next moment may be like – and for each possible next moment, there are several possibilities for the moment after that. Thus the situation takes the form, not of a linear sequence, but of a ’tree”’. (Kripke was a high-school student, not quite 18, in Omaha, Nebraska.)

- Kripke became interested in modal logic through reading Prior’s ‘Modality and Quantification in S5” in 1956. His letter was in response to Prior’s “Time and Modality”.

Linear vs. Branching Time: Philosophy

Philosophical Conundrum

- Prior:

– Nature of course of time – branching

– Nature of course of events – linear

- Rescher:

– Nature of time – linear

– Nature of course of events – branching

– "We have 'branching *in* time', not 'branching *of* time'".

Linear vs. Branching Time: “Final Showdown”

V., 2001: a pragmatic examination (motivated by hardware verification)

- Expressiveness
- Computational Complexity
- Compositionality

Conclusion: Linear time wins for pragmatic reasons!

The Pragmatic Argument

- Verification engineers find branching time unintuitive. (IBM: “nontrivial CTL equations are hard to understand and prone to error.”)
- Branching time was designed for reasoning about *closed* rather than *open* systems. Branching-time modular reasoning is exceedingly hard.
- Combining formal and dynamic validation (i.e., simulation) techniques for branching-time model checking is essentially impractical, as dynamic validation is inherently linear.

Industry-Standard Languages:
- *PSL*: Essentially linear time
- *SVA*: Linear time

Concurrency-Theoretical Critique

Critique 2001-2: "Linear-time semantics is too weak; two processes may be trace equivalent, but one may deadlock in a context where the other one would not!"

Classical Example: 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 $b!0$.

But: Branching-time semantics distinguishes between the two processes – they are not *bisimilar*.

V., 2002: "It's time to start learning about concurrency semantics."

Concurrency Semantics

N.&V., 2006: State of art is highly unsatisfying!

- **Wide Agreement**: Process equivalence is the most fundamental notion in concurrency semantics.
- **No Agreement**: When are two processes equivalent? Profusion of answers (“The next 700 ... Syndrome”)!

van Glabbeek, 1990: “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 application”

Semantics: Processes vs. Programs

- **Process semantics**:

– *Wide Agreement*: Two processes are equivalent if they pass the same tests – *testing equivalence*.

– *No Agreement*: What is a test?

- **Program semantics**:

– *Wide Agreement*: Two programs are equivalent if they behave the same in all contexts – *contextual equivalence*.

– *Wide Agreement*: Behavior refers to I/O behavior.

N.&V., 2006: Why does *equivalence* not mean the same thing in process semantics and program semantics?

Our Approach: Three fundamental principles.

Contextual Equivalence

Principle of Contextual Equivalence: Two processes are equivalent if they behave the same in all contexts.

Contexts:

- *Question*: What is a context?
- *Answer*: A context is a process with a “hole”.

Excluded: Button-pushing experiments [Milner, 1980]

Why?: Experiments can be too weak or too strong (cf. [De Nicola-Hennessy, 1987])

Bisimulation: Pro

Definition: Processes P_1 and P_2 are *bisimilar* if there is behavior-preserving relation between the states of P_1 and the states of P_2.

Bisimulation occupies a special place of honor in concurrency theory:

- Stirling, 1998: “The joys of bisimulation”
- Mathematical elegance (e.g., colagebraic formulation)
- Ubiquity in logic (e.g., modal logic, non-well-founded set theory)
- Finest process equivalence relation (almost)

Bisimulation: Cons

N.&V.: None of those makes bisimulation a reasonable notion of process equivalence!

- Bisimulation is *not* a contextual equivalence relation – tests that correspond to bisimulation are too powerful, not realizable as processes.

- Bisimulation is a *structural* similarity relation, not behavioral; not a reasonable relation to expect in general between implementation and specification.

- Elegance does not guarantee usefulness; the latter requires pragmatic evaluation, e.g., is bisimulation-based minimization cost effective? (Fisler-V., 2002: not in the context of model checking).

What Is Process Behavior?

Principle of Contextual Equivalence: Two processes are equivalent if they *behave* the same in all contexts.

Old Problem: *Non-observable behavior* – communication traces do not model all behavior, e.g., deadlocks and livelocks.

Hoare, Brookes, Roscoe, 1984: enrich the theory – *failure semantics*, *failure-divergence semantics*, etc.

N.&V., 2006: In digital circuits, all relevant behavior is observable, including deadlock and livelock!

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

Why: Relevant behavior should be captured by the description of the process, not by theoreticians!

Receptiveness

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

Problem: Process is not *receptive* to communication on channel b, when it is in the left branch

Popular requirement: Processes need to be receptive [Dill, 1989, Lynch-Tuttle, 1989, Abadi-Lamport 1993].

Problem: CSP/CCS formalisms are *under-specified*; do not tell us what happens when receptiveness fail (so are also state-transition systems)!

Digression: Nonmonotonic Logic

Basic Idea: Draw conclusions from absence of premises. [Clarke, 1978, McCarthy, 1980].

- *Example*: “If Tweety is a bird, and I don’t know that Tweety is a penguin, then I can infer that Tweety can fly.”

Today: 700 nomonotonic logics! – circumscription, default reasoning, negation by failure, stable-model semantics, well-founded semantics, . . .

Why?: No universally accepted way to draw conclusions from absence of premises – numerous interpretations!

Comprehensiveness and Abstraction

Question: Is there a contradiction between comprehensiveness and abstraction? Does the Comprehensive-Modeling Principle push us to very low-level detailed models?

Answer: The principle required *relevant* details to be modeled; relevance is determined by abstraction level.

- Digital circuits: analog aspects *irrelevant*.
- Metastable circuit states: analog aspects *relevant*.

Example: Should normal termination be distinguished from deadlocked termination? It depends on the abstraction!

- Reasoning about termination: deadlocks *irrelevant*.
- Reasoning about deadlocks: deadlocks *relevant*

Who decides?
N.&V.: The modeler, not the theoretician!

What is Process Behavior

Principle of Contextual Equivalence: Two processes are equivalent if they *behave* the same in all contexts.

Question: What does “behave” mean?

Answer: It means that the same behavior is being *observed*.

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

Conclusion: The observable result of a test is an I/O trace.

Nondeterminism: The observable result of a test is a *set* of I/O traces.

- Unifies *may* and *must* testing.

An Opposing View

van Glabbeek, 2001:

> "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."

Our perspective:

- Consider *all* possible contexts.
- A context can decide what behaviors to *observe*.
- The modeler decide what behaviors are *observable*.

Crux: "doubt and difference of opinion" is the result of formalism under-specificity!

Properties and Observability

Principle of Observable properties: Behavioral properties ought to be observable.

Definition: A property φ is *observable* if whenever $P_1 \models \varphi$ and $P_2 \not\models \varphi$, then there is a context C such that $C[P_1]$ and $C[P_2]$ have different sets of I/O traces.

Lemma: the CTL property

$$\forall\, always\, \exists\, eventually\; p$$

is not observable.

Conclusion: The CTL property $\forall\, always\, \exists\, eventually\; p$ is a *structural*, rather than *behavioral* property.

Case Study: Back to Transducers

Moore, 1956, Milner, 1975:

Transducer:
$M = (Q, q_0, I, O, \sigma, \lambda, \pi)$
- Q: states
- $q_0 \in Q$: start state
- I: input channels
- O: output channels
- $\sigma : I \cup O \rightarrow 2^{\Sigma}$: alphabet assignment
- $\lambda : Q \times O \rightarrow \Sigma$: output function
- $\delta : Q \times \sigma(i_1) \times \ldots \sigma(i_n) \rightarrow 2^{Q}$: transition function.

Notes:

- States are always observable (via output function) and receptive.
- Inputs at time k take effect at time $k+1$ (no causal loops).
- Deadlocks needs to be modeled explicitly, e.g., $output = \text{``}nothing''^{\omega}$
- Nondeterminism.

Parallel Composition

Connection: For a set $\mathcal{M}$ of transducers, a connection C consists of a set of pairs (i_A, o_B) of "compatible" input/output channels.

Synchronous Parallel Composition: $\|_C \mathcal{M}$ runs all transducers in $\mathcal{M}$ in parallel, binding input to output channels per C.

Composition Theorem: Given a set $\mathcal{M}$ of transducers, a connection C and a single transducer A in $\mathcal{M}$, we can express $\|_C \mathcal{M}$ as a binary composition of A with a single transducer.

In words: A context with a hole is equivalent to a single transducer – "testing process".

Transducer Equivalence

Trace: infinite sequence of input/output pairs, corresponding to an execution.

Trace equivalence $A \equiv_T B$: A and B have the same set of traces.

Contextual Equivalence $A \equiv_C B$: for all transducers M, we have that $A \| M \equiv_T B \| M$.

Note:

- Observe whole trace; no “test success”.
- Observe all possible traces; no “may” or “must”.

Theorem: Trace equivalence and contextual equivalence coincide (*adequacy* and *full abstraction*).

Conclusion: A very robust notion of equivalence (hinted at in Vaandrager, 1991).

What Is a Trace

But: van Glabbeek is partly right; there is no agreement on what is a trace!

- Finite traces
- Infinite traces
- fair traces
- $\ldots$

Compositional Traces:

- Define a binary composition operation $\|$ on sets of traces.
- Require that $traces(M_1)\|traces(M_2) \approx traces(M_1\|M_2)$

Theorem: Adequacy and full abstraction apply to all compositional notion of traces.

Robust result: Linear-time semantics provide the "right" notion of process equivalence.

What Is Linear-Time Logic?

Given: Property φ, Observables O, alphabet $\Sigma = 2^O$.

Standard Approach:

- $models(\varphi) \subseteq \Sigma^\omega$ (set of traces).
- $M \models \varphi$ if $traces(M) \subseteq models(\varphi)$ (trace containment).

New Approach:

- $model(\varphi) \subseteq 2^{\Sigma^\omega}$ (family of sets of traces).
- $M \models \varphi$ if $traces(M) \in models(\varphi)$ (trace-set membership)

Reason for Change: "Promptness" cannot be captured by trace containment!

- $promptly\ p$ – p will occur within a bounded amount of time.
- $eventually\ p$ does not guarantee boundness.

Summary: Three Principles

1. **Principle of Contextual Equivalence**: Two processes are equivalent if they behave the same in all contexts.

2. **Principle of Comprehensive Modeling**: Process description should model all relevant aspects of process behavior.

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

Response to Criticism

Possible criticism: Principle of Comprehensive Modeling forces a very low-level view of systems [Vaandrager, 1991, wrt I/O automata].

Response 1: You cannot focus solely on the “high-level view” if the low-level details are relevant. Make up your mind on what is relevant!

Response 2: To avoid clutter, have high-level language features, as well as low-level language features:

- Let user chose whether deadlock is observable or not.

Response 3: Accept that high-level model is not fully specified, let theoreticians determine observables, and allow (700) different interpretations.

What is New Here?

- Well known that bisimulation is not testable!
  - But bisimulation is still widely acceptable as a reasonable notion of process equivalence (e.g., van Glabbeek, 2001).
- *Decorated trace equivalence* - enrich traces with observables.
  - We do not add observables to traces; process description includes all relevant observables.
- Brookes, 2002: “We do not augment traces with extraneous book-keeping information, or impose complex closure conditions. Instead we incorporate the crucial information about blocking directly in the internal structure of traces. ”
  - We agree. Our principles elaborate on Brookes’ approach.

In Conclusion

Provocative Thesis: Concurrency theory went off the charted path, into the wild forest.

- Obvious definition of contextual equivalence abandoned.
- Under-specificity of formalisms led to numerous interpretations.

Next:

- Re-examine more complex settings, e.g., probabilistic behavior.
- Let the debate over the *principles* of process equivalence begin.

Postscript: Branching Time In Verification

Agreed: Linear-time verification can be hard!

● Trace containment of finite transition systems is PSPACE-complete.

Reasonable Approaches:

● Use branching-time relations as an approximation, e.g., simulation under-approximates trace containment [Kesten-Piterman-Pnueli, 2003].

● Use structural relations to prove trace containment [Abadi-Lamport,1991].

Furthermore: Sometime implementation is obtained from specification via a structural refinement, e.g., pipelined vs. non-pipelined architectures.

● "Flushing simulation" - flushed pipelined states correspond to non-pipelined states [Burch-Dill, 1994].

● Symbolic simulation [Jones, 1999]

● Well-founded equivalence bisimulation [Manolios, 2000].

Branching-Time Logic

Wrong Corollary: Branching-time logics are needed to specify systems.

- Structural relations used here to prove trace containment. Branching time in the service of linear time!