The Rise and Fall of Linear Temporal Logic

Moshe Y. Vardi

Rice University

Thread I: Entscheidungsproblem

Entscheidungsproblem (**The Decision Problem**) [Hilbert-Ackermann, 1928]: Decide if a given first-order sentence is *valid* (dually, *Satisfiable*).

Church-Turing Theorem, 1936: The Decision Problem is unsolvable.

Classification Project: Identify decidable fragments of first-order logic.

- *Monadic Class*
- *Bernays-Schönfinkel Class*
- *Ackermann Class*
- Gödel Class (w/o $=$)

Monadic Logic, I

Monadic Class: First-order logic with $=$ and monadic predicates – captures *syllogisms*.

- $(\forall x)P(x), (\forall x)(P(x) \rightarrow Q(x)) \models (\forall x)Q(x)$

[Löwenheim, 1915]: The Monadic Class is decidable.

- *Proof*: Bounded-model property – if a sentence is satisfiable, it is satisfiable in a structure of bounded size.
- *Proof technique*: quantifier elimination.

Monadic Logic, II

Monadic Second-Order Logic: Allow second-order quantification on monadic predicates.

> [Skolem, 1919]: Monadic Second-Order Logic is decidable – via bounded-model property and quantifier elimination.

Question: $=$ allowed. What about $<$?

Thread II: Logic and Automata

Two paradigms in logic:

- **Paradigm I**: *Logic* – declarative formalism
  - Specify properties of mathematical objects, e.g., $(\forall x, y, z)(mult(x, y, z) \leftrightarrow mult(y, x, z))$ – commutativity.
- **Paradigm II**: *Machines* – imperative formalism
  - Specify computations, e.g., Turing machines, finite-state machines.

Surprising Phenomenon: Intimate connection between logic and machines

Nondeterministic Finite Automata

$A = (\Sigma, S, S_0, \rho, F)$

- *Alphabet*: Σ
- *States*: S
- *Initial states*: $S_0 \subseteq S$
- *Nondeterministic transition function*: $\rho : S \times \Sigma \to 2^S$
- *Accepting states*: $F \subseteq S$

Input word: $a_0, a_1, \ldots, a_{n-1}$

Run: $s_0, s_1, \ldots, s_n$

- $s_0 \in S_0$
- $s_{i+1} \in \rho(s_i, a_i)$ for $i \geq 0$

Acceptance: $s_n \in F$

Recognition: $L(A)$ – words accepted by A.

Logic of Finite Words

View finite word $w = a_0, \ldots, a_{n-1}$ over alphabet Σ as a mathematical structure:

- Domain: $0, \ldots, n-1$
- Binary relation: $<$
- Unary relations: $\{P_a : a \in \Sigma\}$

First-Order Logic (FO):

- Unary atomic formulas: $P_a(x)$ ($a \in \Sigma$)
- Binary atomic formulas: $x < y$

Example: $(\exists x)((\forall y)(\neg(x < y)) \wedge P_a(x))$ – last letter is a.

Monadic Second-Order Logic (MSO): monadic second-order quantifier – $\exists Q$ new unary atomic formulas – $Q(x)$.

NFA vs. MSO

Theorem [Büchi, Elgot, Trakhtenbrot, 1957-8 (independently)]: MSO $\equiv$ NFA
- Both MSO and NFA define the class Reg.

- From NFA to MSO ($A \mapsto \varphi_A$)
  - Existence of run – existential monadic quantification
  - Proper transitions and acceptance - first-order formula

- From MSO to NFA ($\varphi \mapsto A_\varphi$): closure of NFAs under
  - *Union* – disjunction
  - *Projection* – existential quantification
  - *Complementation* – negation

NFA Nonemptiness

Nonemptiness Problem: Decide if $L(A) \neq \emptyset$.

Directed Graph $G_A = (S, E)$ of NFA $A = (\Sigma, S, S_0, \rho, F)$:
- *Nodes*: S
- Edges: $E = \{(s,t) : t \in \rho(s,a)$ for some $a \in \Sigma\}$

Lemma: A is nonempty iff there is a path in G_A from S_0 to F.

- Decidable in time linear in size of A, using *breadth-first search* or *depth-first search*.

MSO Satisfiability – Finite Words

Satisfiability: $models(\psi) \neq \emptyset$

Satisfiability Problem: Decide if given ψ is satisfiable.

Lemma: ψ is satisfiable iff A_ψ is nonnempty.

Corollary: MSO satisfiability is decidable.

- Translate ψ to A_ψ.
- Check nonemptiness of A_ψ.

MSO Satisfiability – Computational Complexity

- *Upper Bound*: Nonelementary Growth

$$2^{\cdot^{\cdot^{\cdot^{2^n}}}}$$

(tower of height $O(n)$)

- *Lower Bound* [Stockmeyer, 1974]: Satisfiability of FO over finite words is nonelementary (no bounded-height tower).

Sequential Circuits

Church, 1957: Use MSO to specify sequential circuits.

Sequential circuits: $C = (I, O, R, f, g, R_0)$

- I: input signals
- O: output signals
- R: sequential elements
- $f : 2^I \times 2^R \rightarrow 2^R$: transition function
- $g : 2^R \rightarrow 2^O$: output function
- $R_0 \in 2^R$: initial assignment

Trace: element of $(2^I \times 2^R \times 2^O)^\omega$

$t = (I_0, R_0, O_0), (I_1, R_1, O_1), \ldots$

- $R_{j+1} = f(I_j, R_j)$, $O_j = g(R_j)$

Specifying Traces

View infinite trace $t = (I_0, R_0, O_0), (I_1, R_1, O_1), \ldots$ as a mathematical structure: Domain: *N*, binary relation: $<$, unary relations: $I \cup R \cup O$.

First-Order Logic (FO):

- Unary atomic formulas: $P(x)$ ($P \in I \cup R \cup O$)
- Binary atomic formulas: $x < y$

Example: $(\forall x)(\exists y)(x < y \wedge P(y))$ – P holds i.o.

Monadic Second-Order Logic (MSO): Monadic 2nd-order quantifier: $\exists Q$, new unary atomic formulas: $Q(x)$.

Model Checking Sequential Circuits

Model-Checking Problem: Given circuit C and MSO formula φ; does φ hold in all traces of C?

Easy Observation: MSO model-checking problem reducible to MSO satisfiability problem – use FO to encode the "logic" (i.e., f, g) of the circuit C.

Büchi Automata, I

Büchi Automaton: $A = (\Sigma, S, S_0, \rho, F)$

- *Alphabet*: Σ
- *States*: S
- *Initial states*: $S_0 \subseteq S$
- *Transition function*: $\rho : S \times \Sigma \to 2^S$
- *Accepting states*: $F \subseteq S$

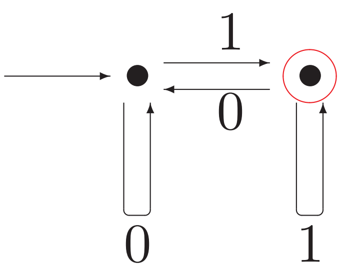
Büchi Automata, II

Input word: $a_0, a_1, \ldots$

Run: $s_0, s_1, \ldots$

- $s_0 \in S_0$
- $s_{i+1} \in \rho(s_i, a_i)$ for $i \geq 0$

Acceptance: F visited infinitely often



– infinitely many 1's

Fact: Büchi automata define the class ω-*Reg* of ω-regular languages.

Logic vs. Automata

Paradigm: Compile high-level logical specifications into low-level finite-state language

Compilation Theorem: [Büchi,1960] Given an MSO formula φ, one can construct a Büchi automaton A_φ such that a trace σ satisfies φ if and only if σ is accepted by A_φ.

MSO Satisfiability Algorithm:

- φ is satisfiable iff $L(A_\varphi) \neq \emptyset$
- $L(\Sigma, S, S_0, \rho, F) \neq \emptyset$ iff there is a path from S_0 to a state $f \in F$ and a cycle from f to itself.

Corollary [Church, 1960]: Model checking sequential circuits wrt MSO specs is decidable.

Thread III: Temporal Logic

Prior, 1914–1969, Philosophical Preoccupations:

- *Religion*: Methodist, Presbytarian, atheist, agnostic
- *Ethics*: "Logic and The Basis of Ethics", 1949
- *Free Will, Predestination, and Foreknowledge*:
  - "The future is to some extent, even if it is only a very small extent, something we can make for ourselves".
  - "Of what will be, it has now been the case that it will be."
  - "There is a deity who infallibly knows the entire future."

> Mary Prior: "I remember his waking me one night [in 1953], coming and sitting on my bed, . . ., and saying he thought one could make a formalised tense logic."

- 1957: "Time and Modality"

Linear vs. Branching Time, A

- 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 Saul Kripke: "In 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.)

Linear vs. Branching Time, B

- **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, C

- Prior developed the idea into Ockhamist and Peircean theories of branching time (branching-time logic *without* path quantifiers)

  Sample formula: $CKMpMqAMKpMqMKqMp$

- Burgess, 1978: "Prior would agree that the determinist sees time as a line and the indeterminist sees times as a system of forking paths."

Linear vs. Branching Time, D

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 time: Hans Kamp, Dana Scott and others continued the development of linear time during the 1960s.

Temporal and Classical Logics

Key Theorem:

- Kamp, 1968: Linear temporal logic with past and binary temporal connectives ("until" and "since"), over the integers, has precisely the expressive power of FO.

The Temporal Logic of Programs

- Prior: "There are practical gains to be had from this study too, for example in the representation of time-delay in computer circuits"
- Rescher & Urquhart, 1971: applications to processes ("a programmed sequence of states, deterministic or stochastic")

"Big Bang 1" [Pnueli, 1977]:

- Future linear temporal logic (LTL) as a logic for the specification of non-terminating programs
- Temporal logic with "eventually" and "always" (later, with "next" and "until")
- Model checking via reduction to MSO and automata

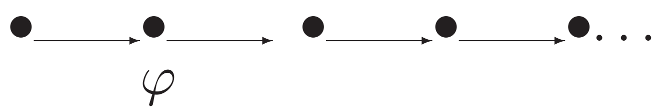
Crux: Need to specify *ongoing behavior* rather than *input/output relation*!

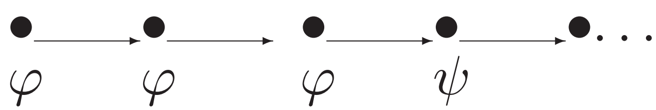
Linear Temporal Logic

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 

- $\pi, w \models \varphi\ until\ \psi$ if w 

Examples

Psalm 34:14: "*Depart from evil and do good*"

- always not (CS_1 and CS_2): mutual exclusion (safety)
- always (Request implies eventually Grant): liveness
- always (Request implies (Request until Grant)): liveness

Expressive Power

- Gabbay, Pnueli, Shelah & Stavi, 1980: Propositional LTL over the naturals has precisely the expressive power of FO.
- Thomas, 1979: FO over naturals has the expressive power of star-free ω-regular expressions

Summary: LTL=FO=star-free ω-RE $<$ MSO=ω-RE

Meyer on LTL, 1980, in "Ten Thousand and One Logics of Programming":

"The corollary due to Meyer – I have to get in my controversial remark – is that that [GPSS'80] makes it theoretically uninteresting."

Computational Complexity

Recall: Satisfiability of FO over traces is non-elementary

Contrast with LTL:

- Wolper, 1981: LTL satisfiability is in EXPTIME.
- Halpern&Reif, 1981, Sistla&Clarke, 1982: LTL satisfiability is PSPACE-complete.

Basic Technique: *tableau* (influenced by branching-time techniques)

PastLTL

Lichtenstein, Pnueli,&Zuck, 1985: past-time connectives useful in LTL:

- yesterday q: q was true in the previous state
- past p: q was true sometime in the past
- p since q: p has been true since q was true

Example: always (rcv $\rightarrow$ past snt)

Theorem

- Expressively equivalent to LTL [LPZ'85]
- Satisfiability of PastLTL is PSPACE-complete [LPZ'85]
- PastLTL is exponentially more succinct than LTL [Markey, 2002]

Model Checking

"Big Bang 2" [Clarke & Emerson, 1981, Queille & Sifakis, 1982]: Model checking programs of size m wrt CTL formulas of size n can be done in time mn.

Linear-Time Response [Lichtenstein & Pnueli, 1985]: Model checking programs of size m wrt LTL formulas of size n can be done in time $m2^{O(n)}$ (*tableau*-based).

Seemingly:

- *Automata*: Nonelementary
- Tableaux: exponential

Back to Automata

Exponential-Compilation Theorem: [V. & Wolper,1983–1986]

Given an LTL formula φ of size n, one can construct a Büchi automaton A_φ of size $2^{O(n)}$ such that a trace σ satisfies φ if and only if σ is accepted by A_φ.

Automata-Theoretic Algorithms:

- *LTL Satisfiability*:
  φ is satisfiable iff $L(A_\varphi) \neq \emptyset$ (PSPACE)
- *LTL Model Checking*:
  $M \models \varphi$ iff $L(M \times A_{\neg\varphi}) = \emptyset$ ($m2^{O(n)}$)

Vardi, 1988: Also with past.

Reduction to Practice

Practical Theory:

- Courcoubetis, V., Yannakakis & Wolper, 1989: Optimized search algorithm for explicit model checking
- Burch, Clarke, McMillan, Dill & Hwang, 1990: Symbolic algorithm for LTL compilation
- Clarke, Grumberg & Hamaguchi, 1994: Optimized symbolic algorithm for LTL compilation
- Gerth, Peled, V. & Wolper, 1995: Optimized explicit algorithm for LTL compilation

Implementation:

- Spin [Holzmann, 1995]: Promela w. LTL:
- SMV [McMillan, 1995]: SMV w. LTL

Satisfactory solution to Church's problem?
Almost, but not quite, since LTL$<$MSO$=\omega$-RE.

Enhancing Expressiveness

- Wolper, 1981: Enhance LTL with grammar operators, retaining EXPTIME-ness (PSPACE [SC'82])
- V. & Wolper, 1983: Enhance LTL with automata, retaining PSPACE-completeness
- Sistla, V. & Wolper, 1985: Enhance LTL with 2nd-order quantification, losing elementariness
- V., 1989: Enhance LTL with fixpoints (as in Kozen's μ-calculus), retaining PSPACE-completeness

Bottom Line: ETL (LTL w. automata) $=$ μTL (LTL w. fixpoints) $=$ MSO, and has exponential-compilation property.

Dynamic and Branching-Time Logics

Dynamic Logic [Pratt, 1976]:

- The $\Box\varphi$ of modal logic can be taken to mean "φ holds after an execution of a program step".
- Dynamic modalities:
  - $[\alpha]\varphi$ – φ holds after all executions of α.
  - $\psi \to [\alpha]\varphi$ corresponds to Hoare triple $\{\psi\}\alpha\{\varphi\}$.

Propositional Dynamic Logic [Fischer & Ladner, 1977]: *Boolean* propositions, programs – *regular expressions* over *atomic* programs.

Satisfiability [Pratt, 1978]: EXPTIME – using *tableau*-based algorithm

Extensions to nonterminating programs [Streett 1981, Harel & Sherman 1981, Kozen 1982] – less suitable for temporal properties.

Branching-Time Logic

From dynamic logic back to temporal logic: The dynamic-logic view is clearly branching; what is the analog for temporal logic?

- Emerson & Clarke, 1980: correcteness properties as fixpoints over computation trees
- Ben-Ari, Manna & Pnueli, 1981: branching-time logic UB; saistisfiability in EXPTIME using tablueax
- Clarke & Emerson, 1981: branching-time logic CTL; efficient model checking
- Emerson & Halpern, 1983: branching-time logic CTL* – ultimate branching-time logic

Key Idea: Prior missed *path quantifiers*

- $\forall\, eventually\ p$: on all possible futures, p eventually happen.

Linear vs. Branching Temporal Logics

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

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

Combining Dynamic and Temporal Logics

Two distinct perspectives:

- Temporal logic: *state based*, Dynamic logic: *action based*

Symbiosis:

- Harel, Kozen & Parikh, 1980: Process Logic (branching time)
- V. & Wolper, 1983: Yet Another Process Logic (branching time)
- Harel and Peleg, 1985: Regular Process Logic (linear time)
- Henriksen and Thiagarajan, 1997: Dynamic LTL (linear time)

Tech Transfer:

- Beer, Ben-David & Landver, IBM, 1998: RCTL (branching time)
- Beer, Ben-David, Eisner, Fisman, Gringauze, Rodeh, IBM, 2001: Sugar (branching time)

From LTL to PSL

Model Checking at Intel

Prehistory:

- 1990: successful feasibility study using Kurshan's COSPAN
- 1992: a pilot project using CMU's SMV
- 1995: an internally developed (linear time) property-specification language

History:

- 1997: Development of 2nd-generation technology started (engine and language)
- 1999: BDD-based model checker released
- 2000: SAT-based model checker released
- 2000: *ForSpec* (language) released

Dr. Vardi Goes to Intel

1997: (w. Fix, Hadash, Kesten, & Sananes)

V.: How about LTL? FHKS: Not expressive enough.
V.: How about ETL? μTL? FHKS: Users will object.

1998 (w. Landver)

V.: How about ETL? L.: Users will object.
L.: How about regular expressions? V.: They are equivalent to automata!

RELTL: LTL plus dynamic modalities, interpreted linearly – $[e]\varphi$ E.g.: $[\mathsf{true}^*, \mathsf{send}, !\mathsf{cancel}]\mathsf{sent}$

Easy: RELTL=ETL=ω-RE

ForSpec: RELTL + hardware features (clocks and resets) [Armoni et al.]

From ForSpec to PSL

Industrial Standardization:

- Process started in 2000
- Four candidates: IBM's Sugar, Intel's ForSpec, Mororola's CBV, and Verisity's E.
- Fierce debate on linear vs. branching time

Outcome:

- Big political win for IBM (see references to PSL/Sugar)
- Big technical win for Intel
  - PSL is LTL + RE + clocks + resets
  - Branching-time extension as an acknowledgement to Sugar
- Major influence on the design of *SVA* (another industrial standard)

Bottom Line: *Huge* push for model checking in industry.

What about the Past?

- Avoided in industrial languages due to implementation challenges
- Less important in model checking; if past events are important, then program would keep track of them; but, important in specification (LPZ'85)!

Dax, Klaedtke, &Lange, 2010 (cf., Leucker & Sánchez, 2010): *Regular Temporal Logic* (RTL): PastLTL, Dynamic modalities: $[e]\varphi$, Past Dynamic modalities: $[e]^{-}\varphi$

Theorem [DKL'10]

- Expressively equivalent to RELTL.
- Exponentially more succinct that RELTL.
- Satisfiability is PSPACE-complete

Linear Dynamic Logic (LDL)

Observations:

- Dynamic modalities subsume temporal connectives, e.g., always q is equivalent to $[\texttt{true}^*]q$
- To capture past, add *reverse* operator to REs.
  - a: "consume" a and move forward.
  - a^-: "consume" a and move backward.

Inspiration:
- PDL+converse [Pratt, 1976]
- Two-way navigation in XPath

Example: $[\text{true}^*, \text{rcv}]\langle(\text{true}^-)^*\rangle\text{sent}$

Theorem: (1) Expressively equivalent to RELTL; (2) Exponentially more succinct than RELTL; (3) Satisfiability is PSPACE-complete.

LTL is Dead, Long Live LDL!

What was important about PLTL?

- Linear time
- Simple syntax
- Exponential-compilation property
- *Equivalence to FO*

What is important about LDL?

- Linear time
- Extremely simply syntax: REs (with *reverse*) and dynamic modalities
- Exponential-compilation property
- *Equivalence to MSO*

Also: easy to pronounce :-)

From PSL to SVA

"SystemVerilog Assertions (SVA) is essentially a language construct that provides a powerful alternate way to write constraints, checkers and cover points for your design. It lets you express rules (i.e., english sentences) in the design specification in a SystemVerilog format that tools can understand."

```
// Check if ack arrives 3 clocks after a request
assert property (@(posedge clk) req |-> ##3 ack);

// Check if cnt increments everytime vld is set
assert property ((@posedge clk) vld |-> (cnt == ($past(cnt) + 1));
```

Figure 1: SVA Assertions

Postscript

- 2007 Turing Award: E.M. Clarke, E.A. Emerson, and Joseph Sifakis: For their role in developing Model-Checking into a highly effective verification technology that is widely adopted in the hardware and software industries.

- Clarke+Emerson+Sifakis, CACM, 2009: *Model checking – algorithmic verification and debugging*

- Ed Clark, July 23, 2009, email: "I no longer want to be considered a proponent of Branching Time Logic."