The Rise and Fall of LTL

Moshe Y. Vardi

Rice University
Monadic Logic

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 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: What about <?
Nondeterministic Finite Automata

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

- **Alphabet:** \( \Sigma \)
- **States:** \( S \)
- **Initial states:** \( S_0 \subseteq S \)
- **Nondeterministic transition function:**
  \( \rho : S \times \Sigma \rightarrow 2^S \)
- **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 \( A \).

**Example:**

\[
\begin{array}{c}
\bullet & \xrightarrow{0} & \bullet \\
\uparrow & & \uparrow \\
0 & & 1
\end{array}
\]

– ends with 1’s

**Fact:** NFAs define the class \( Reg \) of regular languages.
Logic of Finite Words

View finite word \( w = a_0, \ldots, a_{n-1} \) over alphabet \( \Sigma \) 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: \( (\exists x)((\forall y)(\neg(x < y)) \land P_a(x)) \) – 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.

**Proof**: Effective

- 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**: $L(A) \neq \emptyset$

**Nonemptiness Problem**: Decide if given $A$ is nonempty.

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

**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**: \( \text{models}(\psi) \neq \emptyset \)

**Satisfiability Problem**: Decide if given \( \psi \) is satisfiable.

**Lemma**: \( \psi \) is satisfiable iff \( A_\psi \) is nonempty.

**Corollary**: MSO satisfiability is decidable.
- Translate \( \psi \) to \( A_\psi \).
- Check nonemptiness of \( A_\psi \).

**Complexity**:
- **Upper Bound**: Nonelementary Growth
  \[
  2 \cdot 2^n
  \]
  (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 logic to specify sequential circuits.

**Sequential circuits**: \( C = (I, O, R, f, g, R_0) \)
- \( I \): input signals
- \( O \): output signals
- \( R \): sequential elements
- \( f : 2^I \times 2^R \to 2^R \): transition function
- \( g : 2^R \to 2^O \): 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**: $\mathbb{N}$
- **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**: $(\forall x)(\exists y)(x < y \land P(y))$ — $P$ holds i.o.

**Monadic Second-Order Logic (MSO):**

- **Monadic second-order quantifier**: $\exists Q$
- **New unary atomic formulas**: $Q(x)$

**Model-Checking Problem**: Given circuit $C$ and formula $\varphi$; does $\varphi$ hold in all traces of $C$?

**Easy Observation**: Model-checking problem reducible to satisfiability problem — use FO to encode the “logic” (i.e., $f, g$) of the circuit $C$. 
Büchi Automata

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

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

**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

$\xrightarrow{1} \bullet \xrightarrow{0} \bigcirc$ – infinitely many 1’s

**Fact:** Büchi automata define the class $\omega$-$Reg$ of $\omega$-regular languages.
**Logic vs. Automata II**

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

**Compilation Theorem**: [Büchi, 1960] Given an MSO formula $\varphi$, one can construct a Büchi automaton $A_\varphi$ such that a trace $\sigma$ satisfies $\varphi$ if and only if $\sigma$ is accepted by $A_\varphi$.

**MSO Satisfiability Algorithm**:

1. $\varphi$ is satisfiable iff $L(A_\varphi) \neq \emptyset$

2. $L(\Sigma, S, S_0, \rho, F) \neq \emptyset$ 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.

Church, 1960: “Algorithm not very efficient” *(nonelementary complexity, [Stockmeyer, 1974])*.
Prior, 1914–1969, Philosophical Preoccupations:

- **Religion**: Methodist, Presbyterian, 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: “[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.)
Linear vs. Branching Time, B

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

  ![Trace examples]

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

```
ε
A
  \_A
  \ |\ A
  | \B
  |   \B
  \   \B
  A   B
  A   B
  A   B
  A   B
  A   B
  A   B
  A   B
  A   B
```
Linear vs. Branching Time, C

- Prior developed the idea into Ockhamist and Peircean theories of branching time (branching-time logic \textit{without} path quantifiers)

Sample formula: $CKMpMqAMKpMqMqMp$

- 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.
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

Precursors:

• 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** $\varphi$: $\varphi$ holds in the next state.
- **eventually** $\varphi$: $\varphi$ holds eventually
- **always** $\varphi$: $\varphi$ holds from now on
- **$\varphi$ until $\psi$:** $\varphi$ holds until $\psi$ holds.

\[ \pi, w \models \text{next } \varphi \text{ if } w \bullet \bullet \bullet \varphi \bullet \bullet \bullet \bullet \bullet \ldots \]

\[ \pi, w \models \varphi \text{ until } \psi \text{ if } w \bullet \bullet \bullet \varphi \bullet \bullet \bullet \varphi \bullet \psi \bullet \bullet \bullet \bullet \bullet \ldots \]
Examples

- always not (CS₁ and CS₂): 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 \( \omega \)-regular expressions

**Summary:** \( \text{LTL} = \text{FO} = \text{star-free } \omega \text{-RE} < \text{MSO} = \omega \text{-RE} \)

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.”
Recall: Satisfiability of FO over traces is non-elementary

Contrast with LTL:
- Wolper, 1981: LTL satisfiability is in EXPTIME.

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

Lichtenstein, Pnueli, & Zuck, 1985: past-time connectives are 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 $(rcv \rightarrow \text{past snt})$

**Theorem**

- Expressively equivalent to LTL [LPZ’85]
- Satisfiability of PLTL is PSPACE-complete [LPZ’85]
- PLTL 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
Exponential-Compilation Theorem:

[V. & Wolper, 1983–1986]

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

Automata-Theoretic Algorithms:

1. **LTL Satisfiability**: 
   $\varphi$ is satisfiable iff $L(A_\varphi) \neq \emptyset$ (PSPACE)

2. **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:

- COSPAN [Kurshan, 1983]: deterministic automata specs
- Spin [Holzmann, 1995]: Promela w. LTL
- SMV [McMillan, 1995]: SMV w. LTL

Satisfactory solution to Church’s problem? Almost, but not quite, since $\text{LTL} \prec \text{MSO} = \omega\text{-RE.}$
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 $\mu$-calculus), retaining PSPACE-completeness

**Bottom Line:** ETL (LTL w. automata) = $\mu$TL (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 “$\varphi$ holds after an execution of a program step”.
- Dynamic modalities:
  - $[\alpha] \varphi$ – $\varphi$ holds after all executions of $\alpha$.
  - $\psi \rightarrow [\alpha] \varphi$ 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

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: correctness properties as fixpoints over computation trees
- Ben-Ari, Manna & Pnueli, 1981: branching-time logic UB; satisfiability in EXPTIME using tableau
- 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**: $\text{always(request } \rightarrow \text{ eventually grant)}$

- **Branching time**: a system generates a computation tree

- **Specs**: describe computation trees

- **CTL**: $\forall\text{always (request } \rightarrow \forall\text{ eventually grant)}$
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
1997: (w. Fix, Hadash, Kesten, & Sananes)

V.: How about LTL?
F., H., K., & S.: Not expressive enough.

V.: How about ETL? $\mu$TL?
F., H., K., & S.: 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.: [true*, send, !cancel]sent

**Easy:** RELTL=ETL=$\omega$-RE

**ForSpec:** RELTL + hardware features (clocks and resets) [Armoni, Fix, Flaisher, Gerth, Ginsburg, Kanza, Landver, Mador-Haim, Singerman, Tiemeyer, V., Zbar]
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
  - Some evolution over time in hardware features
- 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, the past is important in specification (LPZ’85)!

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

**Theorem [DKL’10]**
- Expressively equivalent to RELTL.
- Exponentially more succinct than RELTL.
- Satisfiability is PSPACE-complete
Linear Dynamic Logic (LDL)

Observations:

- Dynamic modalities subsume temporal connectives, e.g., always $q$ is equivalent to $[true^*]q$
- 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: $[true^*, rcv]\langle (true^{-})^*\rangle sent$

Theorem:

- Expressively equivalent to RELTL.
- Exponentially more succinct than RELTL.
- 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
- Exponential-compilation property
- *Equivalence to MSO*
- Extremely simply syntax: REs (with *reverse*) and dynamic modalities

Also: easy to pronounce :-)
