“Sometimes” and “not never” revisited: on branching versus linear time temporal logic

By E. Allen Emerson, Joseph Y. Halpern

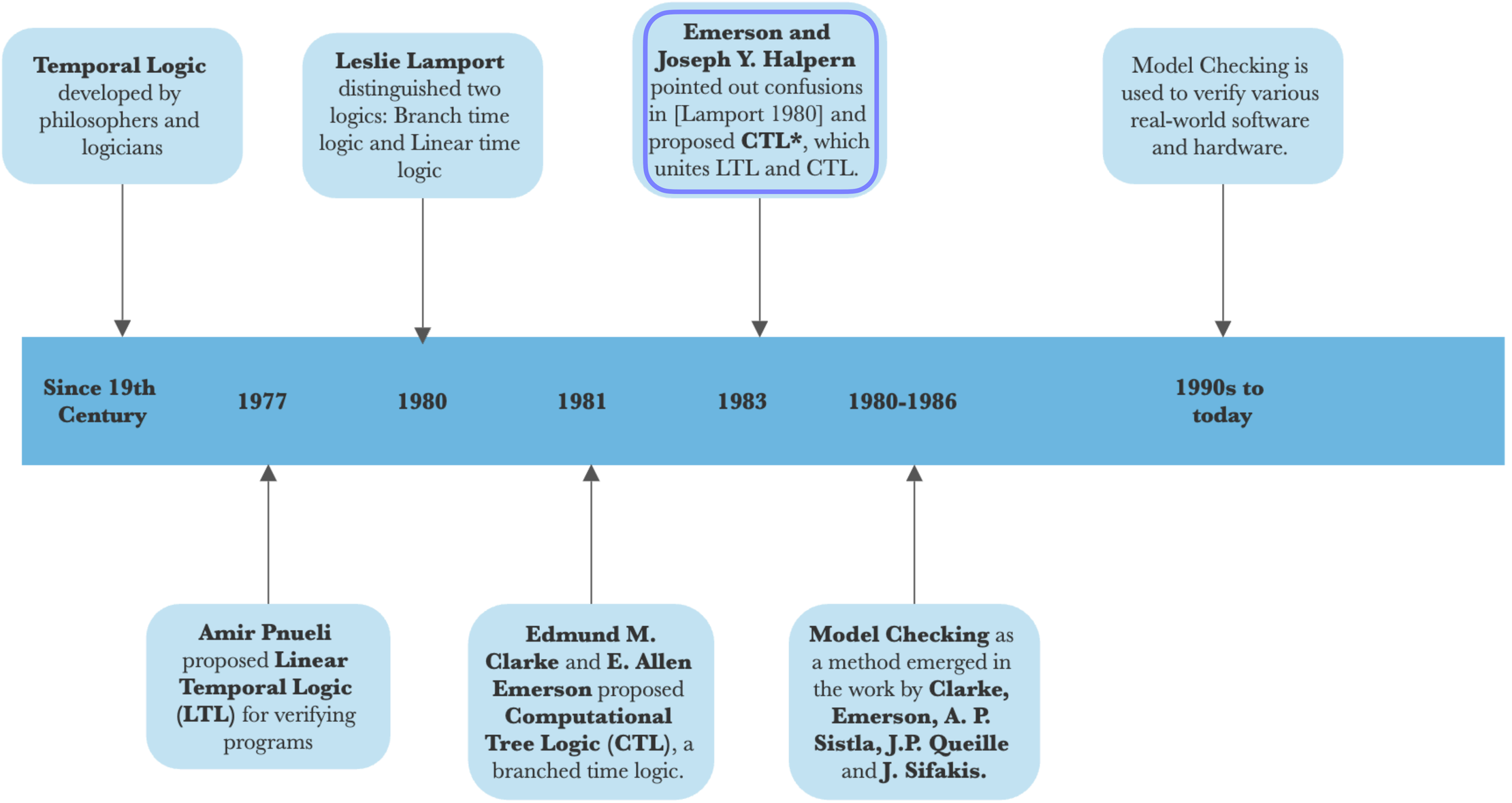
Jialu Bao Feb. 14th 2022 Great Works in PL

Temporal Logic

- Formulas:

$$p, q ::= ap \in \mathscr{AP} \mid \top \mid \bot \mid p \wedge q \mid p \vee q \mid p \Rightarrow q \mid Fp \mid Gp \mid Xp \mid pUq$$

- Informal semantics of temporal operators:
  - Fp — p holds sometime in the **future**.
  - Gp — p **globally** holds / p holds always.
  - Xp — p holds **next** time.
  - pUq — p holds **until** q time.


Temporal Logic developed by philosophers and logicians
Leslie Lamport distinguished two logics: Branch time logic and Linear time logic
Emerson and Joseph Y. Halpern pointed out confusions in [Lamport 1980] and proposed **CTL***, which unites LTL and CTL.
Model Checking is used to verify various real-world software and hardware.
Since 19th Century
1977
1980
1981
1983
1980-1986
1990s to today
Amir Pnueli proposed **Linear Temporal Logic (LTL)** for verifying programs
Edmund M. Clarke and **E. Allen Emerson** proposed **Computational Tree Logic (CTL)**, a branched time logic.
Model Checking as a method emerged in the work by **Clarke, Emerson, A. P. Sistla, J.P. Queille** and **J. Sifakis.**

Excerpts from paper

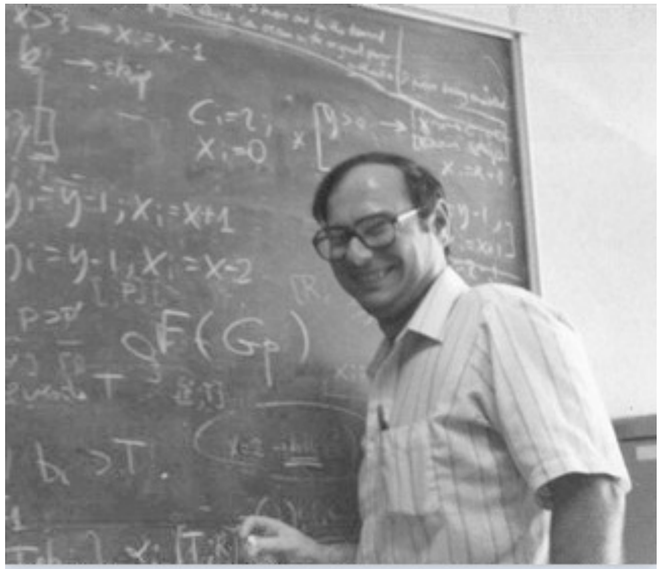
From “The Temporal Logic of Programs”

By Amir Pnueli, 1977

- “The prevalent notions of what constitutes a correctness of a program can all be reduced to two main concepts:

  a. The concept of <u>invariance</u>, i.e. a property holding continuously throughout the execution of a program.

  b. The second and even more important concept is that of <u>eventuality</u> (or temporal implication). In its full generality this denotes a dependence in time in the behavior of the program. We write $\varphi \Rightarrow \psi$, read as: "ψ eventually follows φ" or "φ temporally implies ψ", if whenever the situation described by φ arises in the program, it is guaranteed that eventually the situation described by ψ will be attained.



Amir Pnueli
1941-2009

- Do you agree with his characterization?
  Ex. Which does "*p* holds at program point *l*" belong to?

From “The Temporal Logic of Programs”

By Amir Pnueli, 1977

- “Is the notion of **temporality** really needed in order to discuss intelligently and usefully the behavior of programs?”
  - For invariance properties: **not needed!**
  - For eventuality properties:
    - For deterministic, sequential, structured programs: **not essential**
      - “We can pinpoint exactly where we are in the execution based on program location and loop counters. ”
    - For cyclic/non-deterministic/concurrent programs:
      - **“Some temporal device is necessary.”**

- This reminds me of the distinction Dijkstra made in “Goto statement considered harmful. ” Any thoughts on the implication of that connection?

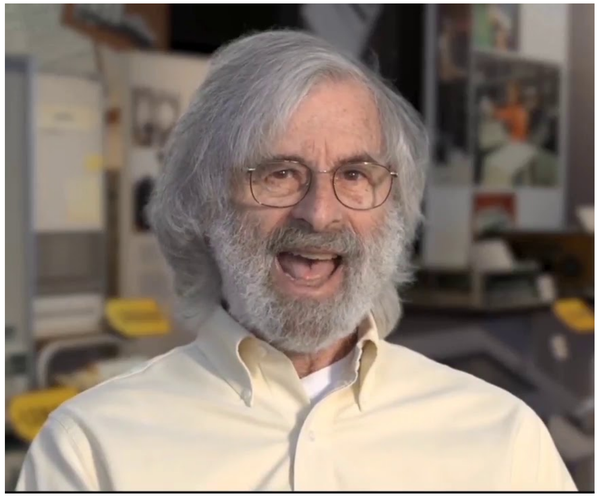
From “The Temporal Logic of Programs”

By Amir Pnueli, 1980

- Example usage of temporality:
  - Liveness: “Something good must **eventually** happen.”
  - Safety: “Something bad must **always not** happen.”
  - Responsiveness: When getting a request p, **eventually** q will happen.
- Encoding “q eventually follows p” using temporal operators F, G, X, U?
  - $p \implies Fq$

From “‘Sometime’ is sometimes ‘not never’”

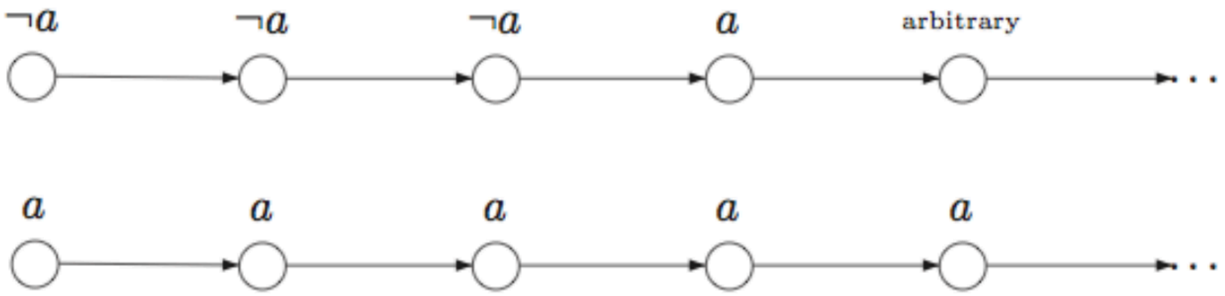
By Leslie Lamport, 1981



The logic of linear time was used by Pnueli in [15], while the logic of branching time seems to be the one used by most computer scientists for reasoning about temporal concepts. We have found this to cause some confusion among our colleagues, so one of our goals has been to clarify the formal foundations of Pnueli's work.

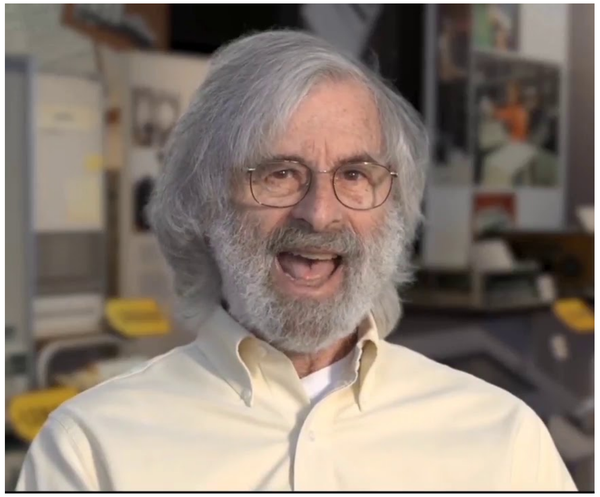
Linear Time Interpretation

- Fa
- Ga



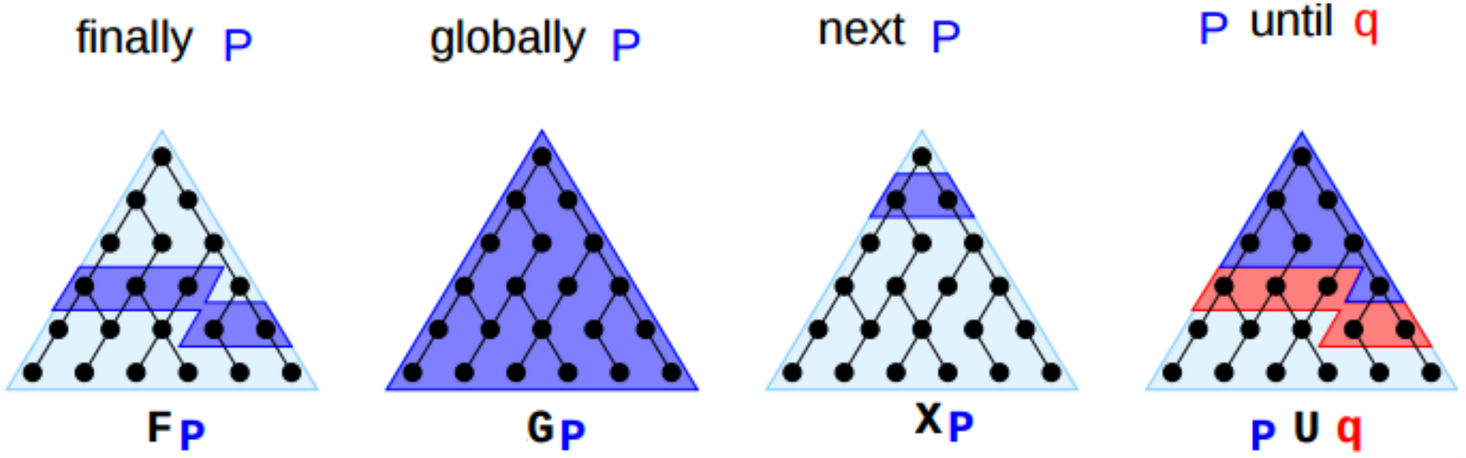
From “‘Sometime’ is sometimes ‘not never’”

By Leslie Lamport, 1981



The logic of linear time was used by Pnueli in [15], while the logic of branching time seems to be the one used by most computer scientists for reasoning about temporal concepts. We have found this to cause some confusion among our colleagues, so one of our goals has been to clarify the formal foundations of Pnueli's work.

Branching Time Interpretation

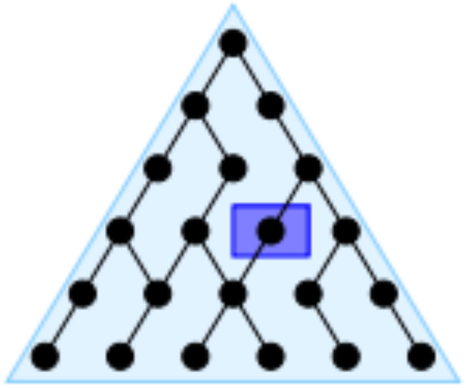


From "'Sometime' is sometimes 'not never'"

By Leslie Lamport, 1981

- Is "sometime" the same as "not never"?
  - Sometime: Fa
  - Not never: $\neg G \neg a$
- In Linear time interpretation

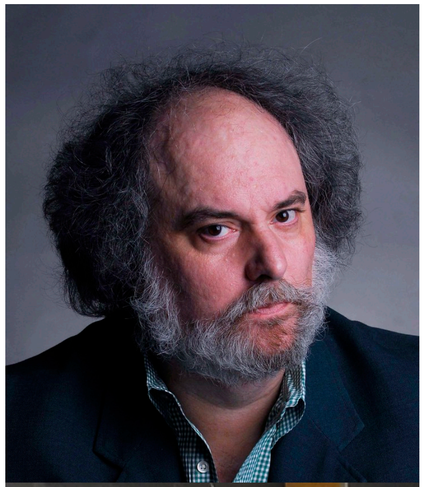
- In Branching time interpretation,

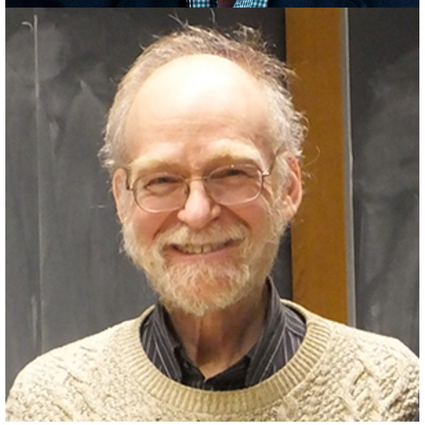


satisfies $\neg G \neg a$ but not Fa

From "'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic"

By E. Allen Emerson and Joseph Y. Halpern , 1983





We now provide our critique of Lamport's approach. Although we do have a few minor criticisms regarding some peculiar technical features and limitations of Lamport's formalism, we would like to emphasize, before we begin, that Lamport's formal results are technically correct—that is, they follow via sound mathematical arguments from his definitions. Our main criticisms instead center around

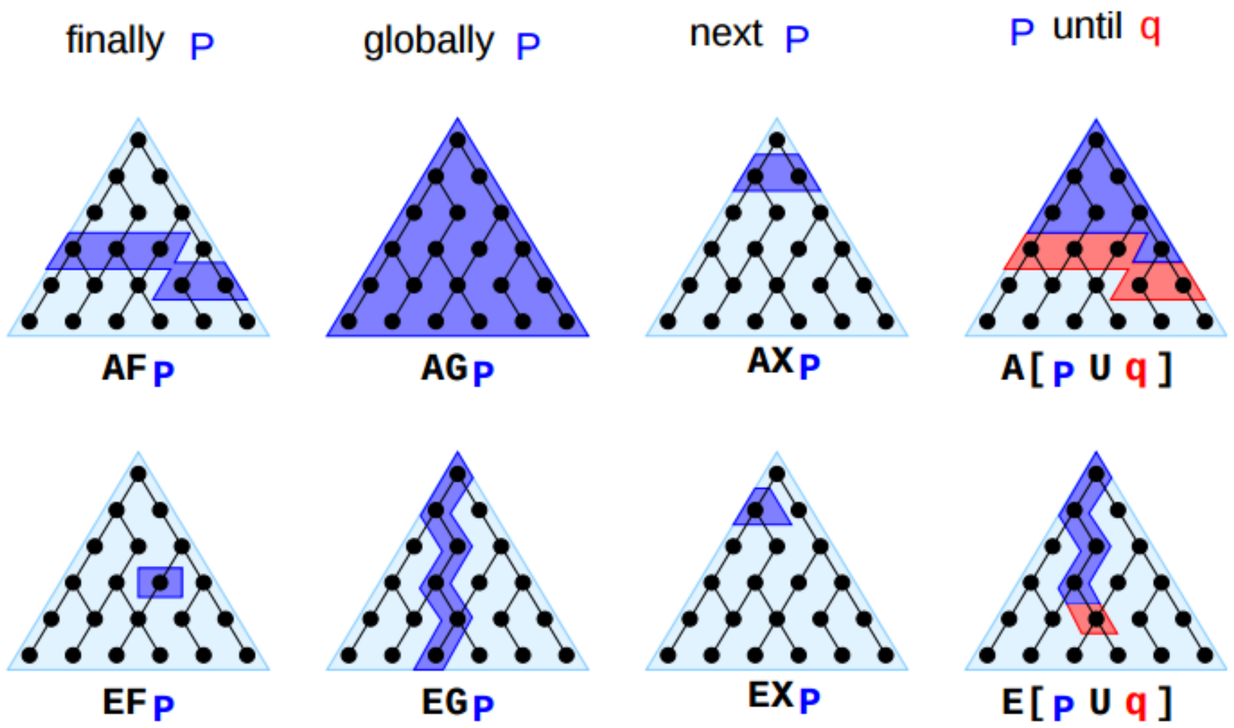
(1) Lamport's basic definitions and underlying assumptions, and
(2) the informal conclusions regarding the application of temporal logic to reasoning about concurrent programs that Lamport infers from his technical results.

Our chief disagreement is, of course, with Lamport's conclusion that linear time logic is superior to branching time logic for reasoning about concurrent programs.

From "'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic"

By E. Allen Emerson and Joseph Y. Halpern , 1983

"basic definitions":



Observation:

- $Fp \vee G\neg p$ interpreted in Lamport's Branching time logic is the same as $AFp \vee AG\neg p$ interpreted in Computation Tree Logic (CTL).
- In CTL, though $AFp \vee AG\neg p$ is not valid, $A(Fp \vee G\neg p)$ is valid.

From "'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic"

By E. Allen Emerson and Joseph Y. Halpern , 1983

- CTL*: A unification of Linear Temporal Logic (LTL) and Computation Tree Logic (CTL)
  - Assert on both states and paths

S1. $s \models P$ iff $P \in \mathbf{L}(s)$ where P is an atomic proposition.
S2. $s \models p \wedge q$ iff $s \models p$ and $s \models q$ where p, q are state formulas.
$s \models \neg p$ iff not $(s \models p)$ where p is a state formula.
S3. $s \models Ap$ iff for every path $x \in \mathbf{X}$ with first$(x) = s$, $x \models p$ where p is a path formula.
$s \models Ep$ iff for some path $x \in \mathbf{X}$ with first$(x) = s$, $x \models p$ where p is a path formula.
P1. $x \models p$ iff first$(px) \models p$ where p is a state formula.
P2. $x \models p \wedge q$ iff $x \models p$ and $x \models q$ where p, q are path formulas.
$x \models \neg p$ iff not $(x \models p)$ where p is a path formula.
P3a. $x \models Fp$ iff for some $i \geq 0$, first$(x^i) \models p$ where p is a state formula.
P3b. $x \models Fp$ iff for some $i \geq 0$, $x^i \models p$ where p is a path formula.
P4a. $x \models Xp$ iff $|x| \geq 1$ and first$(x^1) \models p$ where p is a state formula.
P4b. $x \models Xp$ iff $|x| \geq 1$ and $x^1 \models p$ where p is a path formula.
P5a. $x \models (p\ U\ q)$ iff for some $i \geq 0$, first$(x^i) \models q$ and for all $j \geq 0$ [$j < i$ implies first$(x^j) \models p$].
P5b. $x \models (p\ U\ q)$ iff for some $i \geq 0$, $x^i \models q$ and for all $j \geq 0$ [$j < i$ implies $x^j \models p$].
P6a. $x \models \overset{\infty}{F} p$ iff for infinitely many distinct i, first$(x^i) \models p$ where p is a state formula.
P6b. $x \models \overset{\infty}{F} p$ iff for infinitely many distinct i, $x^i \models p$ where p is a path formula.

From "'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic"

By E. Allen Emerson and Joseph Y. Halpern , 1983

"assumptions":

Lamport correctly observes that "the future behavior [of a concurrent program] depends only upon the current state, and not upon how that state was reached." With this motivation, Lamport requires that the set of paths **X** be *suffix closed*, that is, if $x \in \mathbf{X}$ then $x^{succ} \in \mathbf{X}$. As observed in [5], however, suffix closure is not sufficient to guarantee that a program's future behavior depends only on its current state. We also need to require that, at least, **X** be *fusion closed* (cf. [26]), meaning that if $x_1 s y_1, x_2 s y_2 \in \mathbf{X}$ then $x_1 s y_2 \in \mathbf{X}$.[2] Moreover, there are some additional properties that the set **X** of all computations of a concurrent program can be expected to satisfy. We say that **X** is *limit closed* (cf. [1]) if whenever each of the infinite sequence of paths $x_1 y_1, x_1 x_2 y_2, x_1 x_2 x_3 y_3, \ldots$ is in **X**, then the infinite path $x_1 x_2 x_3 \cdots$ (which is the "limit" of the prefixes $(x_1, x_1 x_2, x_1 x_2 x_3, \ldots)$) is also in **X**. We say that a set **X** of paths is **R**-*generable* iff there exists a total, binary relation **R** on **S** such that **X** consists precisely of the infinite sequences $(s_0, s_1, s_2, \ldots)$ of states from **X** for which $(s_i, s_{i+1}) \in \mathbf{R}$.

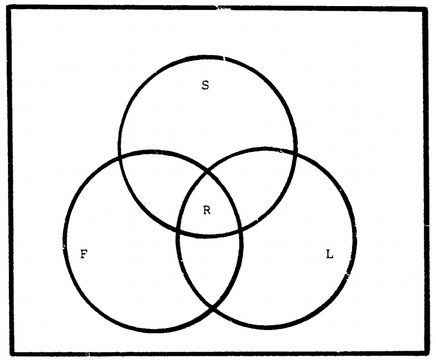


Fig. 1. *R* is the class of all *R*-generable path sets, *S* is the class of all suffix closed path sets, *F* is the class of all fusion closed path sets, and *L* is the class of all limit closed path sets. All regions shown are nonempty.

- What do you think of the assumption of *R*-generable?

From "'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic"

By E. Allen Emerson and Joseph Y. Halpern , 1983

Is linear temporal logic superior to branching temporal logic?

1. Cannot express $\neg Ap$.

> Linear time logic also suffers from the problem that, when we view a linear time logic L as the branching time logic B(L) (i.e., all formulas of the form Aq where q is a formula of L), it is not closed under negation. Although it may be possible to

2. Cannot explicitly assert alternative computation paths.
   - Ex. EG(Processor i is in non-critical section) $\wedge$ EF(Processor i is in the "trying region")
     - $A(G$(Processor i is in non-critical section) $\vee$ F(Processor i is in the "trying region"))
3. Usage in model checking

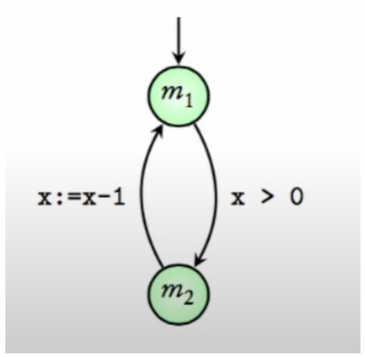
> Model checking for large subclasses of CTL* can be done very efficiently, and, in general, it seems that model checking is easier for branching time than for linear time logics (cf. [4, 11]).

Model Checking

Properties specified in temporal logic formula p

Ex. $F(x \leq 0)$

Systems represented as transition systems T



Automata A_1 that accepts the state/path satisfying $\neg p$

Automata A_2 with location l as start state and accepts paths possible in T

If the intersection of $L(A_1)$ and $L(A_2)$ is empty, then the location l in the system T satisfies p

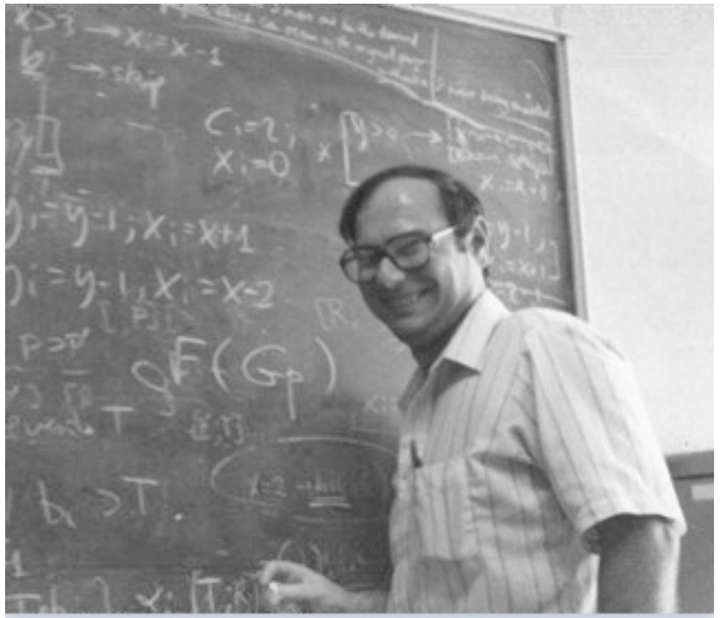
- What are considerations for logic used for specification?
  - Later, in model checking, most work use LTL or CTL, not CTL*. Why is that the case?

A more meta comment excerpted from paper:

From “The Temporal Logic of Programs”

By Amir Pnueli, 1977

- Two trends in the program verification research:
  - “The first is towards **unification** of the basic notions and approaches to program verification, be they sequential or concurrent programs.”
  - “The second is the continuous search for proof methods which will **approximate** more and more the **intuitive reasoning** that a programmer employs in designing and implementing his programs.”



Amir Pnueli
1941-2009