# System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

## 11: Formal Specification:
## Temporal logic
## CTL


Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

# (A philosophical note)

- Your dreams, aspirations, goals in life: liveness
- Your fears: safety

# BRANCHING-TIME PROPERTIES

# Linear-Time vs. Branching-Time Properties

So far we have been talking about properties of **linear** behaviors (sequences).

But some properties are not linear, e.g.:

*"it is possible to recover from any fault"*

or

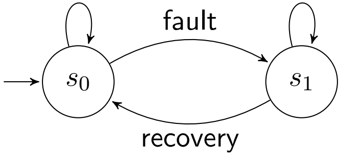
*"we can get back to the initial state from any reachable state"*

# Linear-Time vs. Branching-Time Properties

*"it is possible to recover from any fault"*

Based on *one* (linear) behavior alone,[1] we cannot conclude whether our system satisfies the property.

E.g., the following system satisfies the property, although it contains a behavior that stays forever in state $s_1$:




[1] if we had *all* linear behaviors of a system, we could in principle reconstruct its branching behavior as well – how?

# Linear-Time vs. Branching-Time Behaviors

Linear-time behavior = infinite sequence.

Branching-time behavior = infinite **tree**.

Hence the name "Computation Tree Logic" – CTL.

# Defining the semantics of CTL

We could:

1. define the semantics of CTL on trees,
2. define the "unfolding" of a transition system into a tree (or forest of trees, in case there are many initial states),
3. define what it means for a transition system to satisfy a CTL formula: its forest satisfies the formula.

Instead:

- we will simplify and define the semantics of CTL directly on the transition system (Kripke structure).

# CTL (Computation Tree Logic) – Syntax

There are two kinds of CTL formulas: state formulas and path formulas. When we just say "CTL formula" we mean CTL state formula.

- CTL **state formulas** are defined by the following grammar:

$$\begin{aligned} \phi \quad ::= \quad & p \mid q \mid ..., \text{ where } p, q, ... \in \mathsf{AP} \\ & \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \mathbf{E}\psi \mid \mathbf{A}\psi \end{aligned}$$

  where $\psi$ must be a path formula, and $\phi_1, \phi_2$ must be state formulas.

- CTL **path formulas** are defined by the following grammar:

$$\psi \quad ::= \quad \mathbf{X}\phi \quad \mid \quad \phi_1 \, \mathbf{U} \, \phi_2$$

  where $\phi, \phi_1, \phi_2$ must all be state formulas.

# CTL Syntax: Notes

- $\mathbf{E}$ ("there exists a path") and $\mathbf{A}$ ("for all paths") are called **path quantifiers**.
- As usual, we can use any Boolean operator $\vee, \rightarrow, \leftrightarrow$, etc., as abbreviation / syntactic sugar.
- Similarly, we can also use the temporal operators $\mathbf{G}$ and $\mathbf{F}$ in CTL path formulas.
  For example, $\mathbf{EF}p \equiv \mathbf{E}(true\,\mathbf{U}\,p)$, $\mathbf{AF}p \equiv$

# CTL Syntax: Notes

- $\mathbf{E}$ ("there exists a path") and $\mathbf{A}$ ("for all paths") are called **path quantifiers**.
- As usual, we can use any Boolean operator $\vee, \rightarrow, \leftrightarrow$, etc., as abbreviation / syntactic sugar.
- Similarly, we can also use the temporal operators $\mathbf{G}$ and $\mathbf{F}$ in CTL path formulas.
  For example, $\mathbf{EF}p \equiv \mathbf{E}(\mathit{true}\,\mathbf{U}\,p)$, $\mathbf{AF}p \equiv \mathbf{A}(\mathit{true}\,\mathbf{U}\,p)$,
  $\mathbf{AG}p \equiv$

# CTL Syntax: Notes

- $\mathbf{E}$ ("there exists a path") and $\mathbf{A}$ ("for all paths") are called **path quantifiers**.
- As usual, we can use any Boolean operator $\vee, \rightarrow, \leftrightarrow$, etc., as abbreviation / syntactic sugar.
- Similarly, we can also use the temporal operators $\mathbf{G}$ and $\mathbf{F}$ in CTL path formulas.
  For example, $\mathbf{EF}p \equiv \mathbf{E}(\mathit{true}\,\mathbf{U}\,p)$, $\mathbf{AF}p \equiv \mathbf{A}(\mathit{true}\,\mathbf{U}\,p)$,
  $\mathbf{AG}p \equiv \neg\mathbf{EF}\neg p$, $\mathbf{EG}p \equiv$

# CTL Syntax: Notes

- $\mathbf{E}$ ("there exists a path") and $\mathbf{A}$ ("for all paths") are called **path quantifiers**.
- As usual, we can use any Boolean operator $\vee, \rightarrow, \leftrightarrow$, etc., as abbreviation / syntactic sugar.
- Similarly, we can also use the temporal operators $\mathbf{G}$ and $\mathbf{F}$ in CTL path formulas.
  For example, $\mathbf{EF}p \equiv \mathbf{E}(\mathit{true}\,\mathbf{U}\,p)$, $\mathbf{AF}p \equiv \mathbf{A}(\mathit{true}\,\mathbf{U}\,p)$,
  $\mathbf{AG}p \equiv \neg\mathbf{EF}\neg p$, $\mathbf{EG}p \equiv \neg\mathbf{AF}\neg p$, etc.
- Alternative syntax: $\forall\Box$ instead of $\mathbf{AG}$, $\exists\Diamond$ instead of $\mathbf{EF}$, etc.

# CTL (Computation Tree Logic) – Syntax

Examples of (syntactically correct) CTL formulas:

$$\mathbf{AG}p$$

$$\mathbf{EF}q$$

$$\mathbf{AGEF}(p \rightarrow q)$$

# CTL (Computation Tree Logic) – Syntax

Examples of (syntactically correct) CTL formulas:

$$\mathbf{AG}p$$

$$\mathbf{EF}q$$

$$\mathbf{AGEF}(p \rightarrow q)$$

Syntactically incorrect CTL formulas:

$$\mathbf{G}p, \quad \mathbf{AGF}p, \quad (\mathbf{AG}p) \wedge \mathbf{F}q, \quad \mathbf{AEG}p, \quad \mathbf{A}p, \quad \mathbf{A}\neg\mathbf{F}p$$

# CTL – Semantics: Intuition

Let $s$ be a state of the Kripke structure.

Then $s$ satisfies the CTL formula $\mathbf{EG}\phi$, written

$$s \models \mathbf{EG}\phi$$

iff **there exists** an infinite path starting from $s$ and satisfying $\mathbf{G}\phi$.

# CTL – Semantics: Intuition

Let $s$ be a state of the Kripke structure.

Then $s$ satisfies the CTL formula $\mathbf{EG}\phi$, written

$$s \models \mathbf{EG}\phi$$

iff **there exists** an infinite path starting from $s$ and satisfying $\mathbf{G}\phi$.

$$s \models \mathbf{AG}\phi$$

iff

# CTL – Semantics: Intuition

Let $s$ be a state of the Kripke structure.

Then $s$ satisfies the CTL formula $\mathbf{EG}\phi$, written

$$s \models \mathbf{EG}\phi$$

iff **there exists** an infinite path starting from $s$ and satisfying $\mathbf{G}\phi$.

$$s \models \mathbf{AG}\phi$$

iff **every** infinite path starting from $s$ satisfies $\mathbf{G}\phi$.

# Examples

Let's construct transition systems (Kripke structures) satisfying or violating the following CTL formulas:

$$\mathbf{AG}p$$

# Examples

Let's construct transition systems (Kripke structures) satisfying or violating the following CTL formulas:

$$\mathbf{AG}p$$

$$\mathbf{AF}p$$

# Examples

Let's construct transition systems (Kripke structures) satisfying or violating the following CTL formulas:

$$\mathbf{AG}p$$

$$\mathbf{AF}p$$

$$\mathbf{EG}p$$

# Examples

Let's construct transition systems (Kripke structures) satisfying or violating the following CTL formulas:

$$\mathbf{AG}p$$

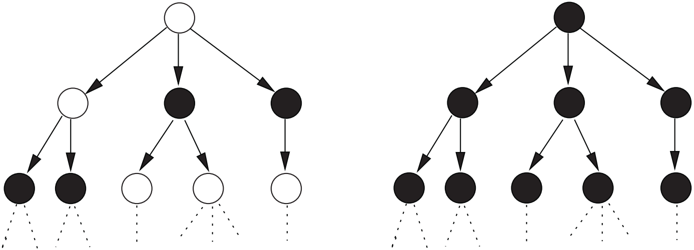
$$\mathbf{AF}p$$

$$\mathbf{EG}p$$

$$\mathbf{EF}p$$

# CTL Semantics – Illustration

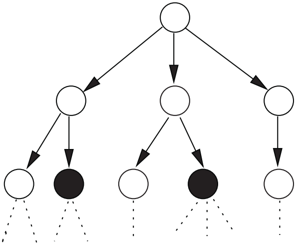
Figures taken from [Baier and Katoen, 2008]



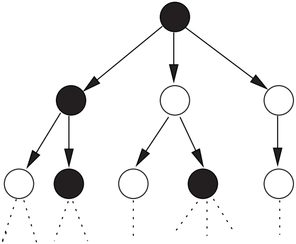
$\forall\Diamond\, black$ $\qquad$ $\forall\Box\, black$

# CTL Semantics – Illustration

Figures taken from [Baier and Katoen, 2008]



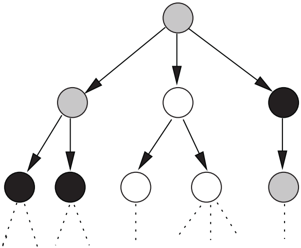
$\exists\Diamond\, black$



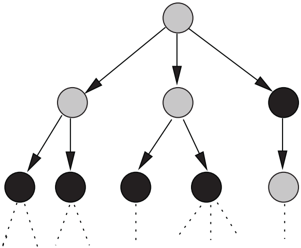
$\exists\Box\, black$

# CTL Semantics – Illustration

Figures taken from [Baier and Katoen, 2008]



$\exists(gray \, \mathsf{U} \, black)$



$\forall(gray \, \mathsf{U} \, black)$

# CTL – Formal Semantics

The satisfaction relation $\models$ for CTL depends on the kind of CTL formula:

- CTL state formulas are evaluated on states: if $s$ is a state of the transition system, and $\phi$ is a CTL state formula, we must define what $s \models \phi$ means.
- CTL path formulas are evaluated on infinite paths (similar to LTL): if $\pi$ is an infinite path in the transition system, and $\psi$ is a CTL path formula, we must define what $\pi \models \psi$ means.

# CTL – Formal Semantics

The satisfaction relation $\models$ for CTL depends on the kind of CTL formula:

- CTL state formulas are evaluated on states: if $s$ is a state of the transition system, and $\phi$ is a CTL state formula, we must define what $s \models \phi$ means.
- CTL path formulas are evaluated on infinite paths (similar to LTL): if $\pi$ is an infinite path in the transition system, and $\psi$ is a CTL path formula, we must define what $\pi \models \psi$ means.

Let $(\mathsf{AP}, S, S_0, L, R)$ be a Kripke structure and let $s \in S$.

- Recall: a path $\pi$ starting from $s$ is an infinite sequence of states and transitions: $\pi = s \to s_1 \to s_2 \to \cdots$
- $\pi(i)$ denotes the $i$-th state in the path, $s_i$, with $\pi(0) = s$.
- Let $Paths(s)$ denote the set of all paths starting from $s$.

# CTL – Formal Semantics

Let $(\mathsf{AP}, S, S_0, L, R)$ be a Kripke structure and let $s \in S$.

Satisfaction relation for CTL state formulas:

$$
\begin{array}{lll}
s \models p & \text{iff} & p \in L(s) \\
s \models \phi_1 \wedge \phi_2 & \text{iff} & s \models \phi_1 \text{ and } s \models \phi_2 \\
s \models \neg\phi & \text{iff} & s \not\models \phi \\
s \models \mathbf{E}\psi & \text{iff} & \exists \pi \in Paths(s) : \pi \models \psi \\
s \models \mathbf{A}\psi & \text{iff} & \forall \pi \in Paths(s) : \pi \models \psi
\end{array}
$$

Satisfaction relation for CTL path formulas (similar to LTL):

$$
\begin{array}{lll}
\pi \models \mathbf{X}\phi & \text{iff} & \pi(1) \models \phi \\
\pi \models \phi_1 \, \mathbf{U} \, \phi_2 & \text{iff} & \exists i \geq 0 : \pi(i) \models \phi_2 \wedge \forall 0 \leq j < i : \pi(j) \models \phi_1
\end{array}
$$

# CTL – Examples

How to express these properties in CTL?

"$p$ *holds at all reachable states*"

# CTL – Examples

How to express these properties in CTL?

"*$p$ holds at all reachable states*" $\mathbf{AG}p$

# CTL – Examples

How to express these properties in CTL?

*"$p$ holds at all reachable states"* $\mathbf{AG}p$

*"there exists a way to get back to the initial state from any reachable state"*

# CTL – Examples

How to express these properties in CTL?

*"$p$ holds at all reachable states"* $\mathbf{AG}p$

*"there exists a way to get back to the initial state from any reachable state"* $\mathbf{AG}\ \mathbf{EF}\ \textit{init}$

# CTL – Examples

How to express these properties in CTL?

*"$p$ holds at all reachable states"* $\mathbf{AG}p$

*"there exists a way to get back to the initial state from any reachable state"* $\mathbf{AG}\ \mathbf{EF}\ \mathit{init}$

*"$p$ is inevitable"*

# CTL – Examples

How to express these properties in CTL?

*"$p$ holds at all reachable states"* $\mathbf{AG}p$

*"there exists a way to get back to the initial state from any reachable state"* $\mathbf{AG}\ \mathbf{EF}\ \mathit{init}$

*"$p$ is inevitable"* $\mathbf{AF}\ p$

# CTL – Examples

How to express these properties in CTL?

*"$p$ holds at all reachable states"* $\mathbf{AG}p$

*"there exists a way to get back to the initial state from any reachable state"* $\mathbf{AG}\ \mathbf{EF}$ *init*

*"$p$ is inevitable"* $\mathbf{AF}\ p$

*"$p$ is possible"*

# CTL – Examples

How to express these properties in CTL?

"$p$ *holds at all reachable states*" $\mathbf{AG}p$

"*there exists a way to get back to the initial state from any reachable state*" $\mathbf{AG}\ \mathbf{EF}\ \mathit{init}$

"$p$ *is inevitable*" $\mathbf{AF}\ p$

"$p$ *is possible*" $\mathbf{EF}\ p$

# CTL – Examples

How to express these properties in CTL?

"$p$ *holds at all reachable states*" $\mathbf{AG}p$

"*there exists a way to get back to the initial state from any reachable state*" $\mathbf{AG}\ \mathbf{EF}\ \mathit{init}$

"$p$ *is inevitable*" $\mathbf{AF}\ p$

"$p$ *is possible*" $\mathbf{EF}\ p$

How would you express the last two in LTL?

# CTL – Examples

How to express these properties in CTL?

*"$p$ holds at all reachable states"* $\mathbf{AG}p$

*"there exists a way to get back to the initial state from any reachable state"* $\mathbf{AG}\ \mathbf{EF}\ \mathit{init}$

*"$p$ is inevitable"* $\mathbf{AF}\ p$

*"$p$ is possible"* $\mathbf{EF}\ p$

How would you express the last two in LTL?

We will see that when we compare LTL and CTL.

# THE MODEL-CHECKING PROBLEM FOR CTL

# The verification problem for CTL: CTL model checking

The **CTL model checking problem**: does a given transition system (Kripke structure) $M$ satisfy a given CTL (state) formula $\phi$?

Let $M = (\mathsf{AP}, S, S_0, L, R)$.
$S_0$ is a set, so $M$ generally has many initial states.

We want **every initial state** of $M$ to satisfy $\phi$:

$$\forall s \in S_0 : s \models \phi$$

We write this as:

$$M \models \phi$$

(same notation as in LTL model-checking, but here $\phi$ is a CTL formula).

# LTL vs CTL: EXPRESSIVENESS COMPARISON

# Formula equivalence

- Recall: When are two formulas $\phi_1, \phi_2$ in the same logic, say LTL, **equivalent**?

# Formula equivalence

- Recall: When are two formulas $\phi_1, \phi_2$ in the same logic, say LTL, **equivalent**?
  Multiple ways to define this, all equivalent:
  - When the formula $\phi_1 \leftrightarrow \phi_2$ is valid.
  - When $\forall \sigma \in \Sigma^\omega : \sigma \models \phi_1 \Leftrightarrow \sigma \models \phi_2$.
  - ...
- Can we compare LTL and CTL formulas for equivalence?
  What would it even mean, since LTL is linear-time and CTL is branching-time?

# Formula equivalence

- Recall: When are two formulas $\phi_1, \phi_2$ in the same logic, say LTL, **equivalent**?
  Multiple ways to define this, all equivalent:
  - When the formula $\phi_1 \leftrightarrow \phi_2$ is valid.
  - When $\forall \sigma \in \Sigma^\omega : \sigma \models \phi_1 \Leftrightarrow \sigma \models \phi_2$.
  - ...
- Can we compare LTL and CTL formulas for equivalence?
  What would it even mean, since LTL is linear-time and CTL is branching-time?
  Idea: compare the transition systems that satisfy these formulas!
- Let $\phi_1$ be an LTL formula and $\phi_2$ be a CTL formula.
  We say that $\phi_1$ and $\phi_2$ are equivalent if **for any Kripke structure** $TS$**:** $TS \models \phi_1 \Leftrightarrow TS \models \phi_2$.

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | |

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
| --- | --- |
| $p$ | $p$ |
| $\mathbf{G}p$ | |

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | $p$ |
| $\mathbf{G}p$ | $\mathbf{AG}p$ |
| $\mathbf{F}p$ | |

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | $p$ |
| $\mathbf{G}p$ | $\mathbf{AG}p$ |
| $\mathbf{F}p$ | $\mathbf{AF}p$ |
| $\mathbf{X}p$ | |

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | $p$ |
| $\mathbf{G}p$ | $\mathbf{AG}p$ |
| $\mathbf{F}p$ | $\mathbf{AF}p$ |
| $\mathbf{X}p$ | $\mathbf{AX}p$ |
| $p\,\mathbf{U}\,q$ | |

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | $p$ |
| $\mathbf{G}p$ | $\mathbf{AG}p$ |
| $\mathbf{F}p$ | $\mathbf{AF}p$ |
| $\mathbf{X}p$ | $\mathbf{AX}p$ |
| $p\,\mathbf{U}\,q$ | $\mathbf{A}(p\,\mathbf{U}\,q)$ |
| $\mathbf{GF}p$ | |

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | $p$ |
| $\mathbf{G}p$ | $\mathbf{AG}p$ |
| $\mathbf{F}p$ | $\mathbf{AF}p$ |
| $\mathbf{X}p$ | $\mathbf{AX}p$ |
| $p\,\mathbf{U}\,q$ | $\mathbf{A}(p\,\mathbf{U}\,q)$ |
| $\mathbf{GF}p$ | $\mathbf{AGAF}p$ |
| $\mathbf{FG}p$ | |

# Examples of equivalent formulas

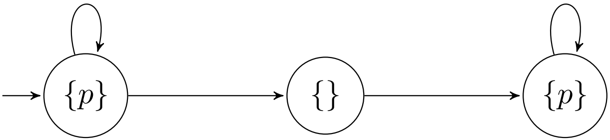
| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | $p$ |
| $\mathbf{G}p$ | $\mathbf{AG}p$ |
| $\mathbf{F}p$ | $\mathbf{AF}p$ |
| $\mathbf{X}p$ | $\mathbf{AX}p$ |
| $p\,\mathbf{U}\,q$ | $\mathbf{A}(p\,\mathbf{U}\,q)$ |
| $\mathbf{GF}p$ | $\mathbf{AGAF}p$ |
| $\mathbf{FG}p$ | $\mathbf{AFAG}p$ ??? |

# Examples of equivalent formulas

| LTL formula | Equivalent CTL formula |
|---|---|
| $p$ | $p$ |
| $\mathbf{G}p$ | $\mathbf{AG}p$ |
| $\mathbf{F}p$ | $\mathbf{AF}p$ |
| $\mathbf{X}p$ | $\mathbf{AX}p$ |
| $p\,\mathbf{U}\,q$ | $\mathbf{A}(p\,\mathbf{U}\,q)$ |
| $\mathbf{GF}p$ | $\mathbf{AGAF}p$ |
| $\mathbf{FG}p$ | $\mathbf{AFAG}p$ ??? **NO! Argh!** |

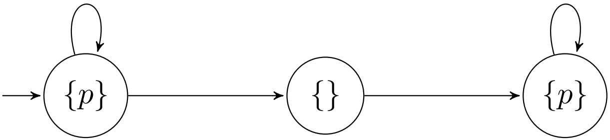
# $\mathbf{FG}p$ and $\mathbf{AFAG}p$ are **not** equivalent

Here's a transition system that distinguishes them:



# $\mathbf{FG}p$ and $\mathbf{AFAG}p$ are **not** equivalent

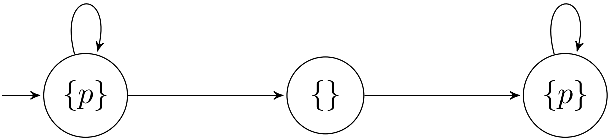
Here's a transition system that distinguishes them:




The above transition system satisfies $\mathbf{FG}p$ but violates $\mathbf{AFAG}p$.

# $\mathbf{FG}p$ and $\mathbf{AFAG}p$ are **not** equivalent

Here's a transition system that distinguishes them:




The above transition system satisfies $\mathbf{FG}p$ but violates $\mathbf{AFAG}p$.

**Homework**: Is there a transition system that satisfies $\mathbf{AFAG}p$ but violates $\mathbf{FG}p$?

# LTL and CTL are incomparable in terms of expressiveness

**Theorem**

*There is no CTL formula equivalent to the LTL formula* $\mathbf{FG}p$.

**Theorem**

*There is no LTL formula equivalent to the CTL formula* $\mathbf{AGEF}p$.

Proofs: on whiteboard.

# CTL: historical and other remarks

- Introduced by [Emerson and Clarke, 1981]
- Long intellectual "fights" over which logic is better!
  - *Sometimes is Sometimes "Not Never"– on the temporal logic of programs* [Lamport, 1980]
  - *What good is temporal logic?* [Lamport, 1983]
  - *Modalities for Model Checking: Branching Time Logic Strikes Back* [Emerson and Lei, 1985]
  - *"Sometimes" and "Not Never" revisited: On branching versus linear time temporal logic* [Emerson and Halpern, 1986]
  - *Branching versus linear logics yet again* [Carmo and Sernadas, 1990]
  - *Sometimes and not never re-revisited: on branching versus linear time* [Vardi, 1998]
  - *Branching vs. Linear Time: Final Showdown* [Vardi, 2001]
- More powerful logics:
  - CTL*: a combination of CTL and LTL, e.g., can write things like $\mathbf{AFG}p$.
  - The $\mu$-calculus [Kozen, 1983]
  - ...

# CTL and LTL in nuXmv

# CTL and LTL in nuXmv

```
-- transition system from lemma 6.19 of Baier-Katoen
MODULE TransitionSystem3
VAR   state : { s0, s1, s2 };
INIT state = s0
TRANS (state = s0 -> (next(state) = s0 | next(state) = s1))
        &
        (state = s1 -> next(state) = s2)
        &
        (state = s2 -> next(state) = s2)

MODULE main
VAR
-- this illustrates the difference between FGp and AFAGp:
  ts3: TransitionSystem3;

LTLSPEC         F G(ts3.state=s0 | ts3.state=s2)
CTLSPEC         AF AG (ts3.state=s0 | ts3.state=s2)
```

# Bibliography I


Baier, C. and Katoen, J.-P. (2008).
*Principles of Model Checking*.
MIT Press.

Carmo, J. and Sernadas, A. (1990).
Branching versus linear logics yet again.
*Formal Aspects of Computing*, 2(1):24–59.

Clarke, E., Grumberg, O., and Peled, D. (2000).
*Model Checking*.
MIT Press.

Emerson, E. and Clarke, E. (1981).
Design and synthesis of synchronization skeletons using branching-time temporal logic.
In *Workshop on Logic of Programs*. LNCS 131.

Emerson, E. and Halpern, J. (1986).
"sometimes" and "not never" revisited: On branching versus linear time temporal logic.
*ACM journal*, 33(1):151–178.

Emerson, E. and Lei, C. (1985).
Modalities for model checking: Branching time logic strikes back.
In *12th ACM Symp. POPL*.

Huth, M. and Ryan, M. (2004).
*Logic in Computer Science: Modelling and Reasoning about Systems*.
Cambridge University Press.

Kozen, D. (1983).
Results on the propositional $\mu$-calculus.
*Theoretical Computer Science*, 27(3):333–354.

# Bibliography II


Lamport, L. (1980).
Sometimes is sometimes "not never"– on the temporal logic of programs.
In *7th ACM Symp. POPL*, pages 174–185.

Lamport, L. (1983).
What good is temporal logic?
In Mason, R., editor, *Information Processing 83: Proceedings of the Ninth IFIP World Computer Congress*, pages 657–668. Elsevier Science Publishers.

Vardi, M. (1998).
Sometimes and not never re-revisited: on branching versus linear time.
In *Concurrency Theory, CONCUR 1998*, volume 1466 of *Lecture Notes in Computer Science*.

Vardi, M. (2001).
Branching vs. linear time: Final showdown.
In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001*, volume 2031 of *Lecture Notes in Computer Science*.