# A survey on Amir Pnueli's work on

# Temporal logic, proof rules and Model Checking

Orna Grumberg
Technion
Haifa, Israel

Haifa Verification Conference, 2010

# Program Verification

Given

- a (hardware or software) system and
- a formal specification

**does the system satisfy the specification?**

- For sequential programs, input-output specification is often sufficient
- For reactive systems, temporal specification is needed

# Early works

- Needed to convince that temporal logic is suitable for program specification
- Decided which operators are needed
  - F and G
  - U and X
- Provided proof rules for verifying properties – deductive rules, non-automated
- Developed automated verification technique for the full logic
  - LTL model checking

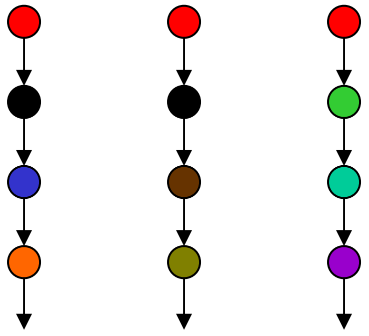
# Overview

- Temporal logic and its use for specifications
- Deductive verification methods
- LTL model checking

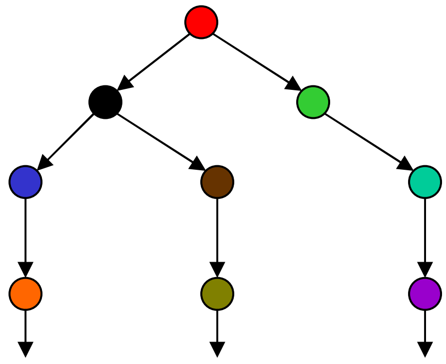
# Temporal Logics

- Temporal Logics
  - Express properties of event orderings in time

- Linear Time
  - Every moment has a unique successor
  - Infinite sequences (words)
  - Linear Time Temporal Logic (LTL)



- Branching Time
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)



# Early works

- In FOCS 1977, Pnueli introduced
  - linear-time temporal logic
  - promoted temporal logic as a specification language

- In POPL 1981, Pnueli, Ben-Ari and Manna introduced
  - branching-time temporal logic

# Temporal logic

AP - a set of atomic formulas

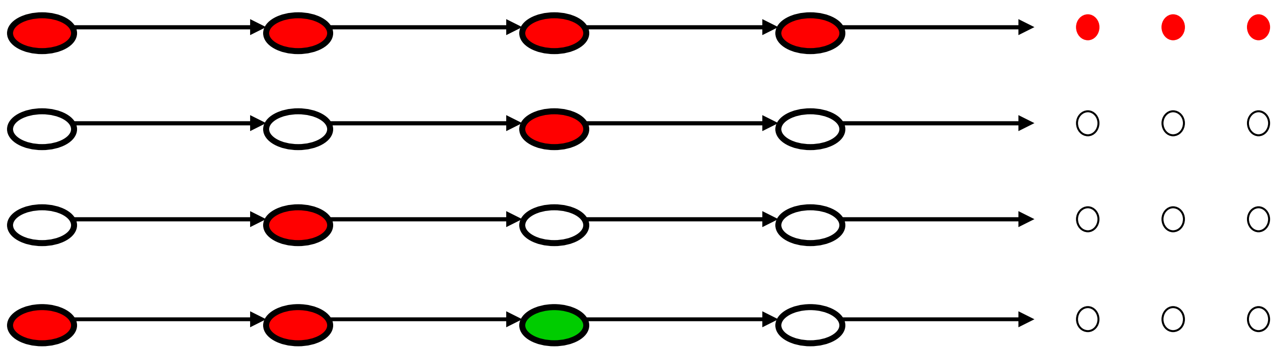
**Temporal operators:**

Gp

Fp

Xp

pUq



**Path quantifiers:** A for **all** path

E there **exists** a path

# Atomic formulas

First-ordered formulas defined over program variables

- Examples:
  - $x > 0$
  - $x = y+z$
  - $\exists y\ (\ x = 2y\ ) \equiv even(x)$

# Alternative notations

- $\Diamond\varphi$ = $\mathbf{F}\varphi$
- $\Box\varphi$ = $\mathbf{G}\varphi$
- $\mathrm{O}\varphi$ = $\mathbf{X}\varphi$

# When is temporal logic useful?

For specifying the behavior of reactive systems

- Non-terminating
- Respond to requests from the environment

# Properties - examples

- Invariance, safety
  - Partial correctness

    $(at\_start \wedge ( x>0 \wedge y\geq 0) \rightarrow G\,[\, at\_halt \rightarrow z' = y/x \,]$

  - Mutual exclusion

    $G\ \neg ( CS1 \wedge CS2 )$

  - Clean behavior, i.e., no division by zero

  - Deadlock freedom

# More properties

- Non-starvation
  $G\ (request \rightarrow F\ granted\ )$

- A resource will not be granted if not requested
  $(F\ grant) \rightarrow (\neg grant\ U\ request\ )$

- Communication protocols
  $(\neg\ receive\text{-}message)\ U\ send\text{-}message$

# Proprties (cont.)

- p1 happens before p2
  $Pr(p1, p2) = (F\ p2) \rightarrow (\neg p2\ U\ p1)$

- Preserving order of events
  $Pr(req_1, req_2) \rightarrow Pr(grant_1, grant_2)$

# Possible extensions of the logic

- Past and future operators
  - previously / next
  - since / until

- Real-time

# Deductive methods

Proof rules for proving that a program satisfies a temporal property

- Non-algorithmic
- Parts can be automated
  - e.g. finding invariants

Reduces temporal reasoning to first-order reasoning

# Deductive methods

Weakness:

- Different rules for different properties
- Non-automatic in the general case

Strength:

- Can handle infinite-state programs
- Recent methods for software model checking are an automation of such deductive rules

# We show proof rules for safety properties

- Invariance: Gp
  - p holds forever
- Precedence: pWq
  - p holds as long as q does not hold
  - unlike U, q is not guaranteed to eventually hold

# Example

- Program with variables $\{x, y\}$
  $P :: \text{if } x > 0 \text{ then } y := x+y \text{ else } y := 0$

- Transitions:
  $\tau_1$: $x > 0$ and $y := x+y$
  $\tau_2$: $x \leq 0$ and $y := 0$

- Formulas in first-order logic:
  $\rho_{\tau 1}$: $x > 0 \wedge (y' = x+y \wedge x' = x)$
  $\rho_{\tau 2}$: $x \leq 0 \wedge (y' = 0 \wedge x' = x)$

Notation: $\{pre\}\ \tau\ \{post\}$

If the transition $\tau$ is executed from a state that satisfies pre then after the execution post holds

Verification condition:

$\rho_\tau \wedge pre \rightarrow post'$

# Example (cont.)

To prove:

- $\{ y>0 \}\ \tau_1\ \{ y > 0 \}$

- Prove the verification condition:
  $(y > 0 \wedge (x > 0 \wedge y'=x+y \wedge x'=x)) \rightarrow y' > 0$

Everything is translated to first-order logic!

# Basic invariance rule

- To show that assertion $\varphi$ always holds:

$$\frac{\begin{array}{l} \theta \rightarrow \varphi \\ \{\varphi\}\ \tau\ \{\varphi\} \quad \text{for all } \tau \in P \end{array}}{G\ \varphi}$$

- $\theta$ - initial condition
- $\tau$ - transition

# Additional invariance rules

$$\frac{Gp,\ \ p \to q}{Gq}$$

monotonicity

$$\frac{Gp,\ \ Gq}{G(q \wedge Gq)}$$

conjunction

- All the proof requirements are translated to first-order logic

- proved manually or sent to a theorem prover

# Precedence $pWq$

- If p initially holds then p holds as long as q does not hold

$$\frac{\{p\}\ \tau\ \{p \vee q\} \quad \text{for all } \tau \text{ in } P}{p \Rightarrow pWq}$$

- Can be extended to nested W

# Algorithmic verification – Model Checking

# Algorithmic verification – Model Checking

An efficient procedure that receives:

- A finite-state model describing a system
- A formula in propositional temporal logic, describing a property

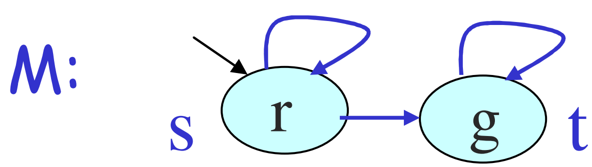
It always terminates and returns

yes, if the system has the property

no + Counterexample, otherwise

# Finite models over atomic propositions

- Kripke structure $M = (S, I, R, L)$




AP = {r, g}

# LTL model checking

To check if $M \models \varphi$

- construct **tableau** $T_{\neg\varphi}$:
  - A finite model
  - contains all paths that satisfy $\neg\varphi$

# Example

To model check $M \models G\,(req \rightarrow F\, grant)$:

- Construct a tableau $\mathbf{T}_{\neg\varphi}$ for the negated formula

  $\neg\varphi = F(\, req \wedge G\, \neg grant\,)$

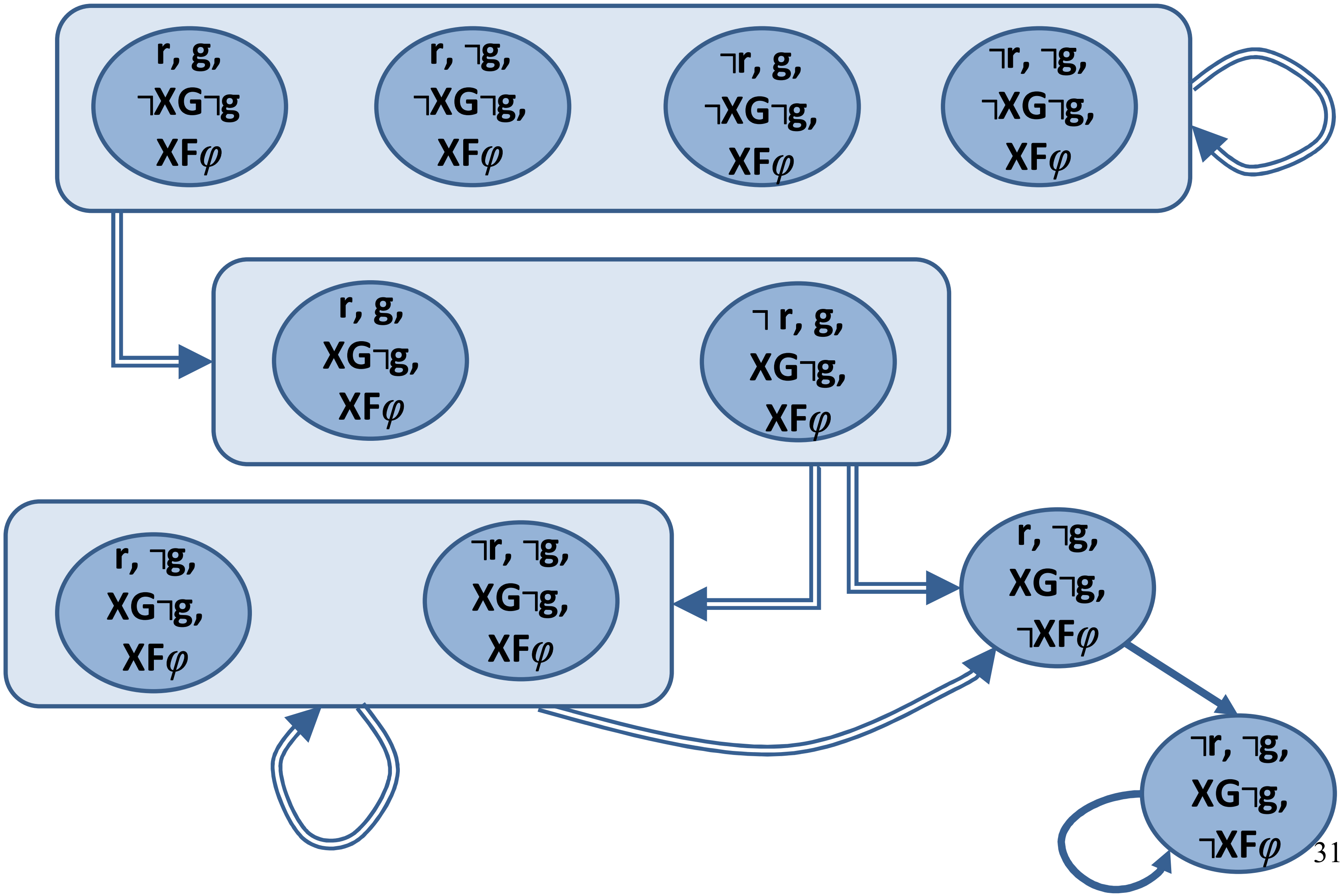
"There is a request that is never granted"

# Model checking (cont.)

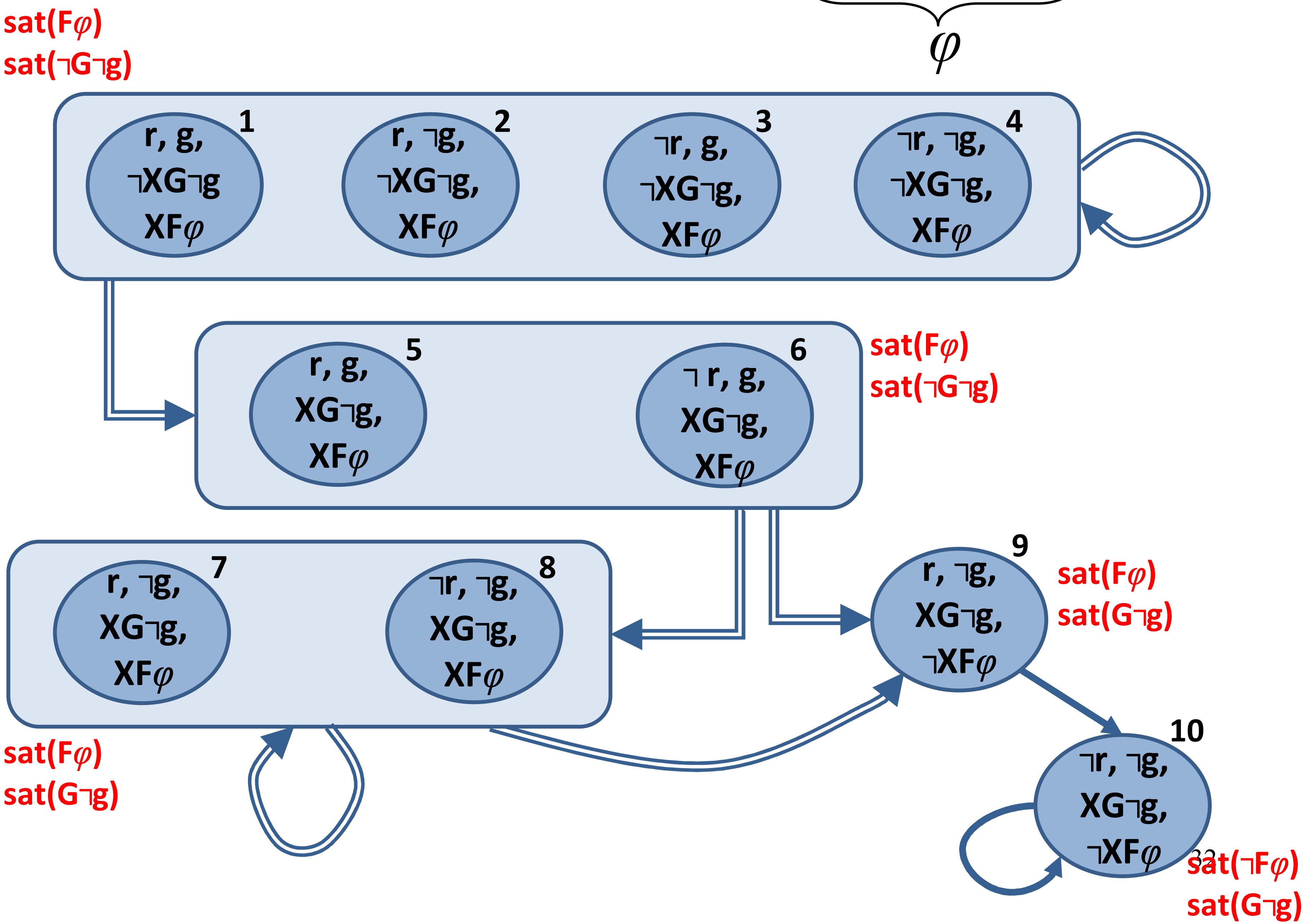
- Construct the product $M \times \mathbf{T}_{\neg\varphi}$
  - Contains all paths in M that do not satisfy $\varphi$: **Counterexamples!**

- Check if $M \times \mathbf{T}_{\neg\varphi}$ contains a path
- Yes – $M \not\models \varphi$
  - The path is a counterexample
- No – $M \models \varphi$

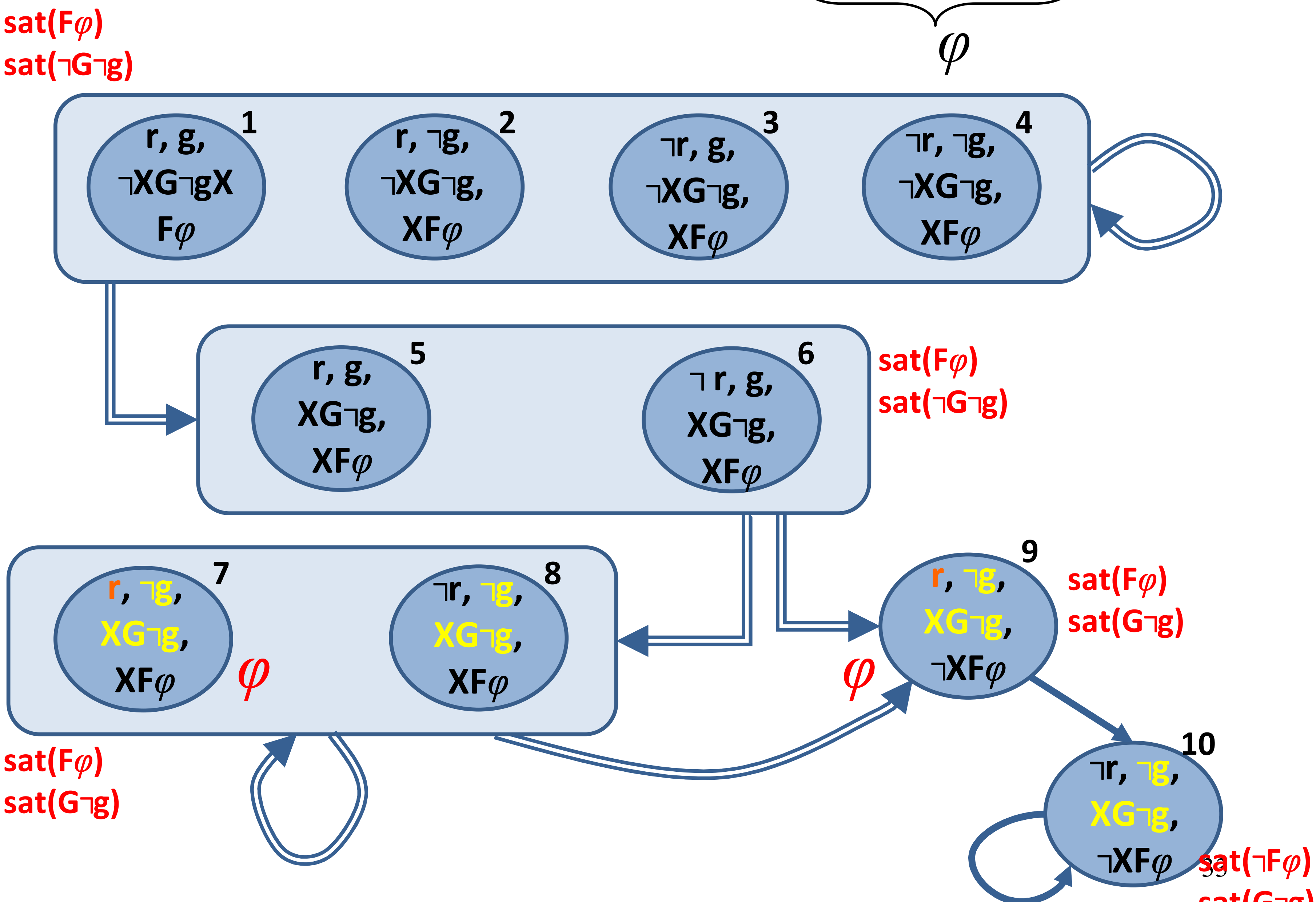
# Tableau for $\psi = F(\underbrace{r \wedge G\neg g}_{\varphi})$



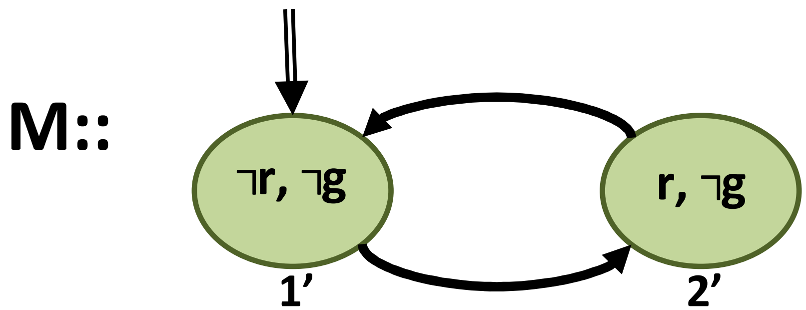
# Tableau for $\psi = F(\underbrace{r \wedge G\neg g}_{\varphi})$

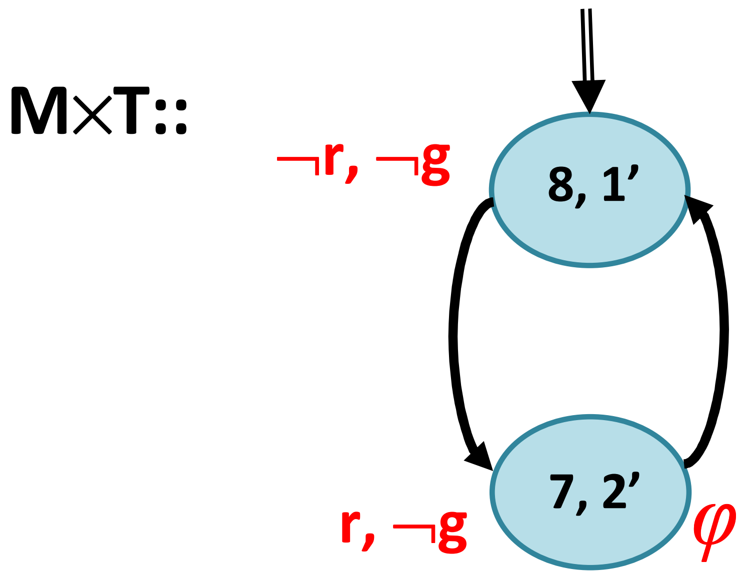


# Tableau for $\psi = F(\underbrace{r \wedge G\neg g}_{\varphi})$



# Tableau for $\psi = F\underbrace{(r \wedge G\neg g)}_{\varphi}$



We found a (fair) path in MxT

- Conclusion: $M \not\models G(req \rightarrow F\ grant)$

# Other contributions in this context

- Using different notions of fairness for modeling systems:
  impartiality, justice, fairness
  - deductive rules for fair systems
  - model checking of fair models

- $CTL^*$ compositional model checking

- And many more

Thank you!

# References


- The Temporal Logic of Programs
  Amir Pnueli, FOCS 1977
- On the Temporal Analysis of Fairness
  D. Gabbay, A. Pnueli, S. Shelah, J. Stavi
  POPL 1980
- The Temporal logic of Branching-time
  M. Ben-Ari, Z. Manna, A. Pnueli
  POPL 1981, Acta Informatica, 1983
- Temporal Verification of Reactive Systems: Safety
  Zohar Manna and Amir Pnueli
  Springer, 1995

- Impartiality, Justice and Fairness: The Ethics of Concurrent programs
  D. Lehman, A. Pnueli, J. Stavi
  ICALP 1981

- Algorithmic and Deductive Verification Methods for CTL*
  Amir Pnueli, Yonit Kesten
  Models, Algebras and Logic of Eng. Software 2003