3. Temporal Logics and Model Checking

Temporal Logics

Temporal Logics

- Temporal logic is a type of modal logic that was originally developed by philosophers to study different *modes* of "truth"
- Temporal logic provides a formal system for qualitatively describing and reasoning about how the truth values of assertions change *over time*
- It is appropriate for describing the time-varying behavior of systems (or programs)

Classification of Temporal Logics

- The underlying nature of time:

  Linear: at any time there is only one possible future moment, linear behavioral trace

  Branching: at any time, there are different possible futures, tree-like trace structure

- Other considerations:

  Propositional vs. first-order

  Point vs. intervals

  Discrete vs. continuous time

  Past vs. **future**

Linear Temporal Logic

- **Time lines**

  Underlying structure of time is a totally ordered set (S,<), isomorphic to (N,<):

  Discrete, an initial moment without predecessors, infinite into the future.

- Let AP be set of atomic propositions, a *linear time structure* M=(S, x, L)

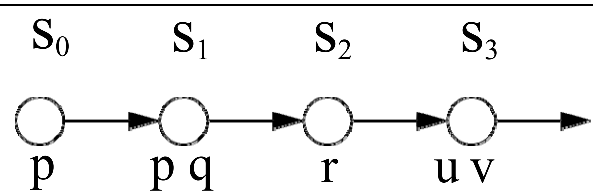
  S: a set of states

  x: $N \rightarrow S$ an infinite sequence of states, ($x=s_0,s_1,\ldots$)

  L: $S \rightarrow 2^{AP}$ labeling each state with the set of atomic propositions in AP true at the state.

- Example:

x:



AP = {p, q, r, u, v}

$L(s_0) = \{p\}$, $L(s_1) = \{p, q\}$, $L(s_2) = \{r\}$, $L(s_3) = \{u, v\}$,.......

Propositional Linear Temporal Logic (PLTL)

- Classical propositional logic + temporal operators

Basic temporal operators

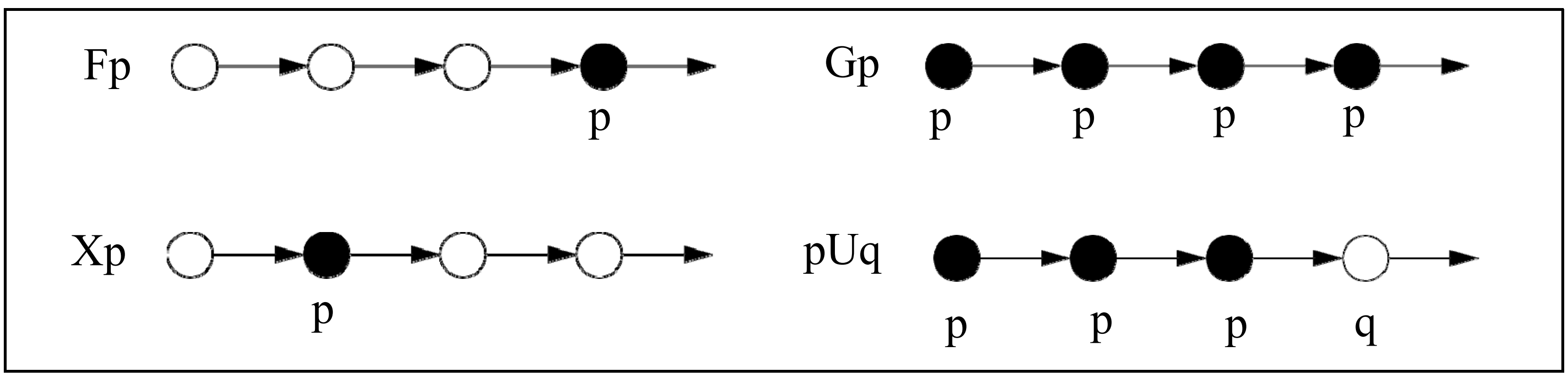
F*p* ("eventually *p*", "sometime *p*")

G*p* ("always *p*", "henceforth *p*")

X*p* ("next time *p*")

*p*U*q* ("*p* until *q*")

- Other common notation: G = □ F = ◊ X = O
- Examples:



Propositional Linear Temporal Logic (cont'd)

Syntax

- The set of formulas of PLTL is the least set of formulas generated by the following rules:
  (1) Atomic propositions are formulas,
  (2) p and q formulas: $p \wedge q$, $\neg p$, p U q, and Xp are formulas.

- The other formulas can then be introduced as abbreviations:

$p \vee q$	abbreviates	$\neg(\neg p \wedge \neg q)$,
$p \Rightarrow q$	abbreviates	$\neg p \vee q$,
$p \equiv q$	abbreviates	$(p \Rightarrow q) \wedge (q \Rightarrow p)$,
true	abbreviates	$p \vee \neg p$,
false	abbreviates	$\neg$*true*,
Fp	abbreviates	(*true* U p),
Gp	abbreviates	$\neg$F$\neg p$.

Examples: $p \Rightarrow Fq$: *"if p is true now then at some future moment q will be true."*
$G(p \Rightarrow Fq)$: *"whenever p is true, q will be true at some subsequent moment."*

Propositional Linear Temporal Logic (cont'd)

Semantics of a formula p of PLTL with respect to a linear-time structure M=(S, x, L)

- $(M, x) \models p$ means that "in structure M, formula p is true of timeline x."
- x^i : suffix of x starting at s_i, $x^i = s_i, s_{i+1}, \ldots$
- Semantics

  $(M, x) \models p$ iff $p \in L(s_0)$, for atomic proposition p

  $(M, x) \models p \wedge q$ iff $(M, x) \models p$ and $(M, x) \models q$

  $(M, x) \models \neg p$ iff it is not the case that $(M, x) \models p$

  $(M, x) \models Xp$ iff $x^1 \models p$

  $(M, x) \models Fp$ iff $\exists j.(x^j \models p)$

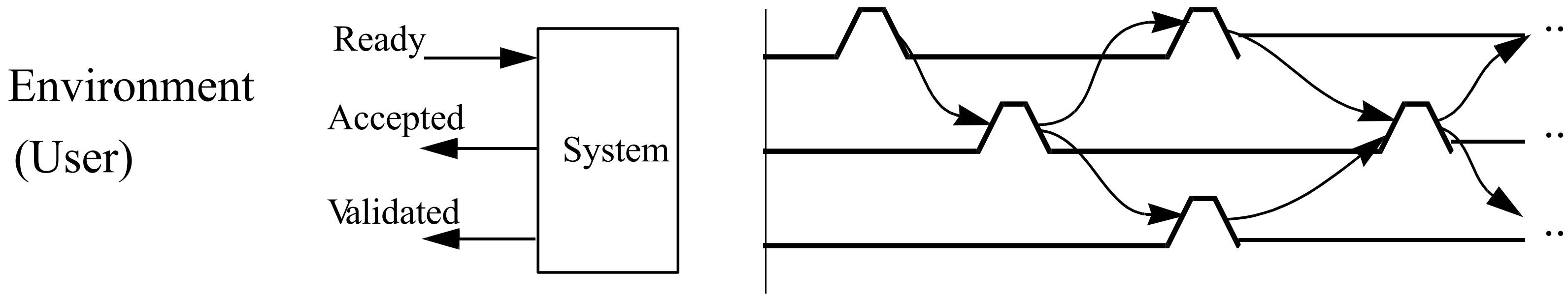
  $(M, x) \models Gp$ iff $\forall j.(x^j \models p)$

  $(M, x) \models p \, U \, q$ iff $\exists j.(x^j \models q$ and $\forall k, 0 \leq k < j \; (x^k \models p))$

- Duality between linear temporal operators $\models G\neg p \equiv \neg Fp$, $\models F\neg p \equiv \neg Gp$, $\models X\neg p \equiv \neg Xp$
- PLTL formula p is *satisfiable* iff there exists M=(S, x, L) such that $(M, x) \models p$ (any such structure defines a ***model*** of p).

Propositional Linear Temporal Logic (cont'd)

Example: A simple interface protocol, pulses one clock period wide



Safety property — nothing bad will ever happen:

$$\forall t.(Validated(t) \rightarrow \neg Validated(t+1))$$

$$\Box(Validated \rightarrow \mathbf{O}\, \neg Validated)$$

$$\mathbf{G}\,(Validated \rightarrow \mathbf{X} \neg Validated)$$

Liveness property — something good will eventually happen:

$$\forall t.(Ready(t) \rightarrow \exists (t' \geq t+1).Accepted(t'))$$

$$\Box(Ready \rightarrow \Diamond\, Accepted)$$

$$\mathbf{G}(Ready \rightarrow \mathbf{F} Accepted)$$

- Fairness constraint: $\mathbf{G}(\ Accepted \Rightarrow \mathbf{F}\ Ready\ )$ (it models a live environment for System)
- Behavior of environment (constraint): $\mathbf{G}\ (\ Ready \Rightarrow \mathbf{X}(\neg Ready\ \mathbf{U}\ Accepted\ ))$
- What about other properties of *Accepted* (initial state, periodic behavior), etc.?

$\Rightarrow$ Prove the system property under the assumption of valid environment constraints

Branching Time Temporal Logic (BTTL)

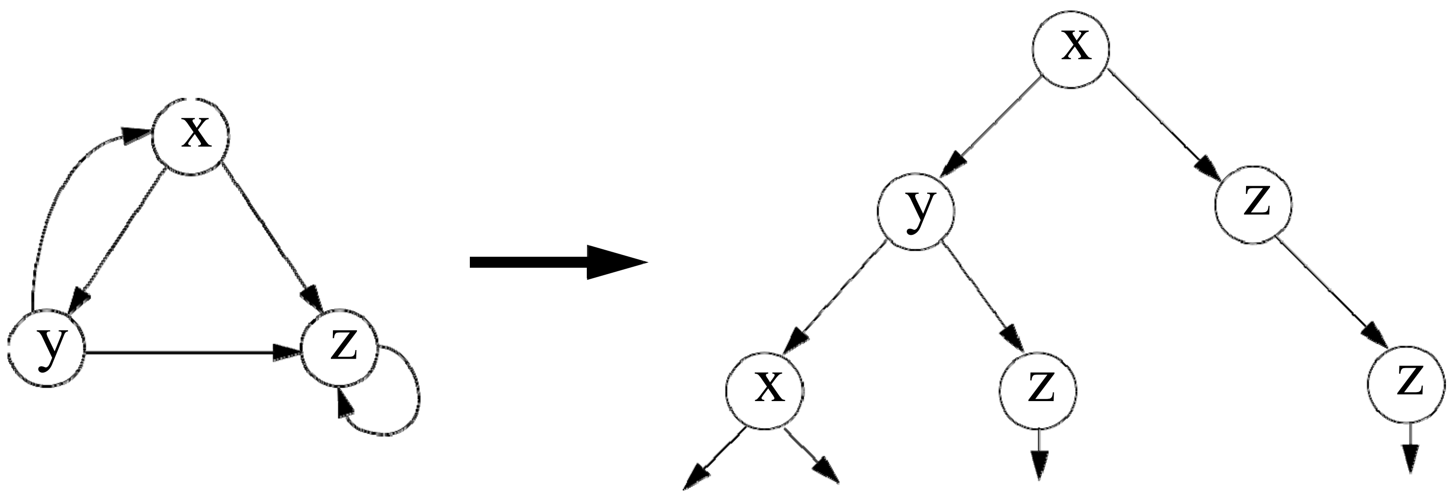
- **Structure** of time: **an infinite tree**, each instant may have many successor instants
  Along each path in the tree, the corresponding timeline is isomorphic to **N**
- **State quantifiers**: Xp, Fp, Gp, pUq (like in linear temporal logic)
- **Path quantifiers:** *for All paths* (A) and *there Exists a path* (E) from a given state

  Other frequent notation: G = $\square$ F = $\lozenge$ X = O
  A = $\forall$ E = $\exists$

- In linear time logic, temporal operators are provided for describing events along a single future, however, when a linear formula is used for specification, there is usually an *implicit universal quantification* over all possible futures (linear traces)
- In contrast, in branching time logic the operators usually reflect the branching nature of time by allowing *explicit quantification* over possible futures in any state
- One supporting argument for branching time logic is that it offers the ability to reason about *existential* properties in addition to *universal* properties
- But, it requires some knowledge of internal state for branching, closer to implementation than LTL that describes properties of observable traces and has simpler fairness assumptions

CTL: a BTTL

- CTL = Computation Tree Logic
- Example of Computation Tree



State Transition graph (Kripke Model) **Infinite Computation Tree**

- Paths in the tree = possible computations or behaviors of the system

CTL (cont'd)

Syntax

1. Every atomic proposition is a CTL formula
2. If f and g are CTL formulas, then so are $\neg f$, $f \wedge g$, AXf, EXf, A(f U g), E(f U g)

- Other operators:

  AFg = A(true U g) EFg = E(true U g) AGf = $\neg$E(true U $\neg$f)

  EGf = $\neg$A(true U $\neg$f)

- EX, E(... U ...), EG are sufficient to characterize the entire logic:

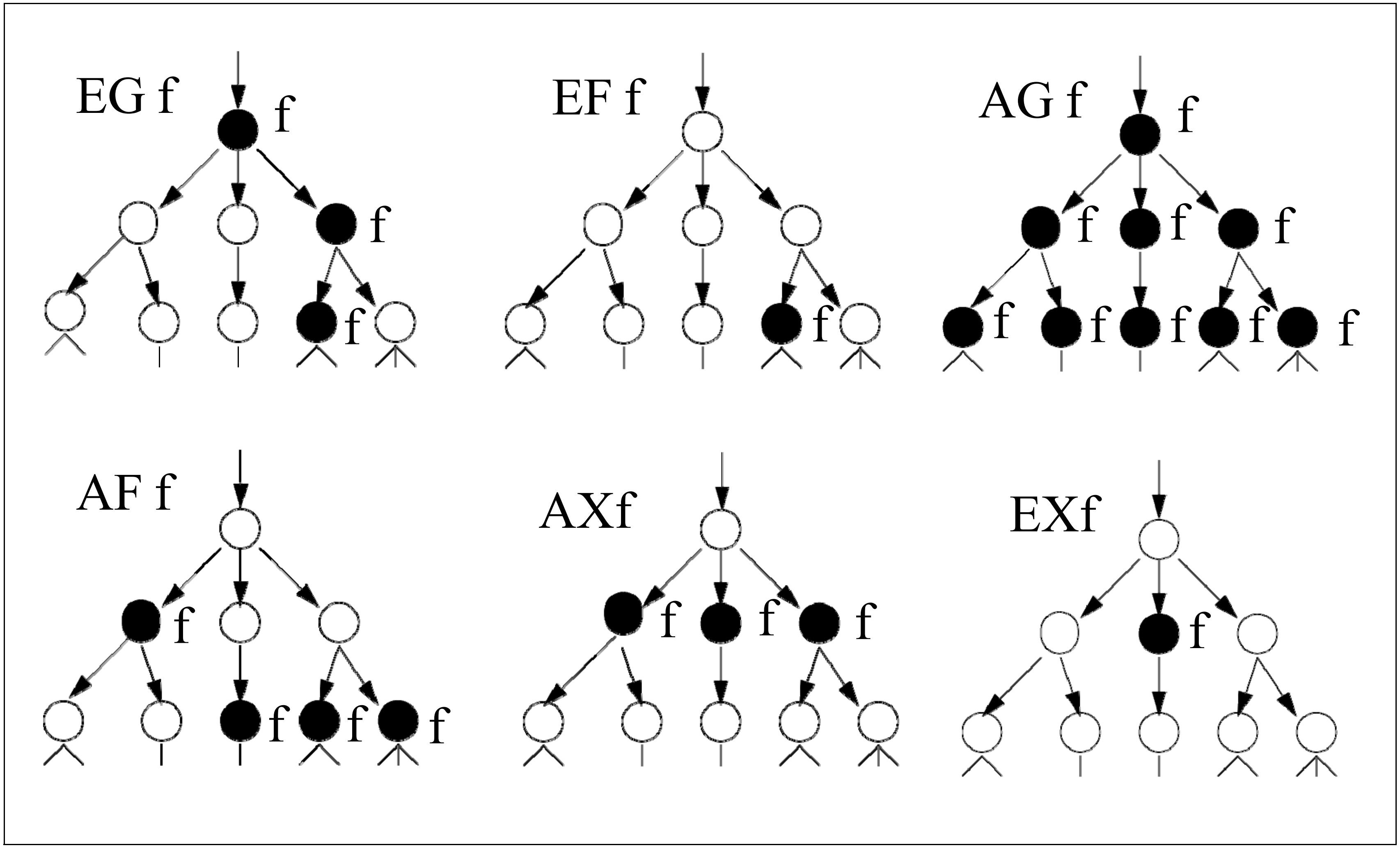
  EFp = E(true U p)

  AXp = $\neg$EX$\neg$p AGp = $\neg$EF$\neg$p

  A(qUp) = $\neg$(E(($\neg$p U $\neg$q) $\wedge$ $\neg$p) $\vee$ EG$\neg$p)

CTL (cont'd)

Intuitive Semantics of Temporal Operators



CTL (cont'd)

Semantics

- A *Kripke structure*: triple $M = <S, R, L>$
  S: set of states $R \subseteq S \times S$: transition relation
  $L: S \rightarrow 2^{AP}$: (Truth valuation) set of atomic propositions true in each state
- R is *total*: $\forall s \in S$ there exists a state $s' \in S$ such that $(s, s') \in R$
- *Path* in M: infinite sequence of states, $x = s_0, s_1, \ldots, i \geq 0, (s_i, s_{i+1}) \in R$.
- x_i denotes the suffix of x starting at s_i: $x_i = s_i, s_{i+1}, \ldots$
- Truth of a CTL formula is defined inductively:

  $(M, s_0) \models p$ iff $p \in L(s_0)$, where p is an atomic proposition

  $(M, s_0) \models \neg f$ iff $(M, s_0) \not\models f$

  $(M, s_0) \models f \wedge g$ iff $(M, s_0) \models f$ and $(M, s_0) \models g$

  $(M, s_0) \models AX\ f$ iff $\forall$ states t, $(s_0, t) \in R$, $(M, t) \models f$

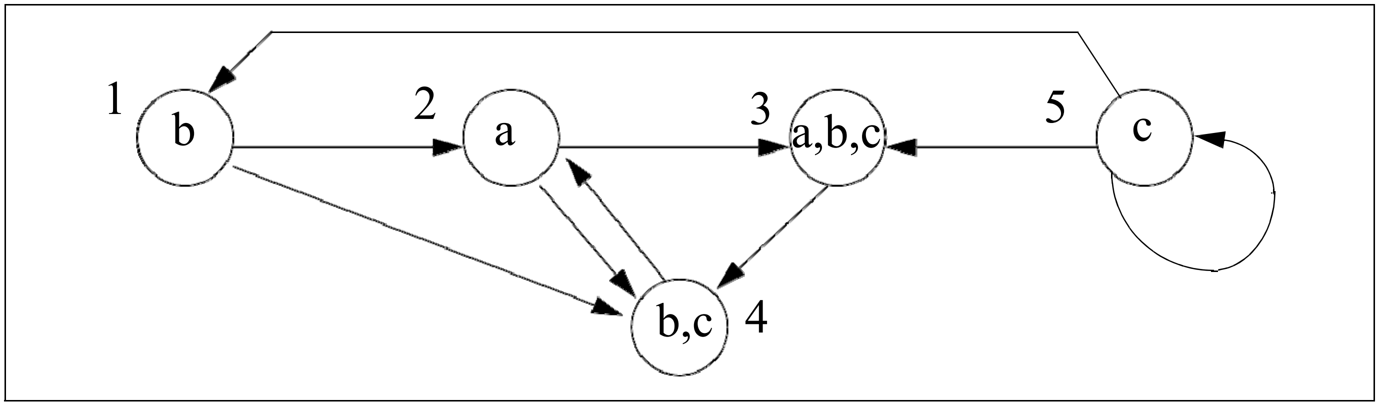
  $(M, s_0) \models EX\ f$ iff $\exists$ state t, $(s_0, t) \in R$, $(M, t) \models f$

  $(M, s_0) \models A(f\ U\ g)$ iff $\forall\ x = s_0, s_1, s_2, \ldots, \exists j \geq 0, (M, s_j) \models g$ and $\forall k, 0 \leq k < j, (M, s_k) \models f$

  $(M, s_0) \models E(f\ U\ g)$ iff $\exists\ x = s_0, s_1, s_2, \ldots, \exists j \geq 0, (M, s_j) \models g$, and $\forall k, 0 \leq k < j, (M, s_k) \models f$

CTL (cont'd)

Example Structure M <S,R,L>



S = {1,2,3,4,5}, AP = {a,b,c},

R = {(1,2), (2,3), (5,3), (5,5), (5,1), (2,4), (4,2), (1,4), (3,4)}

L(1) = {b}, L(2) = {a}, L(3) = {a,b,c}, L(4) = {b,c}, L(5) = {c}

CTL (cont'd)

Example CTL formulas

EF(*started* $\wedge$ $\neg$*ready*): possible to get to a state where *started* holds but *ready* does not

AG(*req* $\rightarrow$ AF *ack*): if a *request* occurs, then there is eventually an *acknowledgment*
(does not ensure that the number of *req* is the same as that of *ack* !)

AG(AF *enabled*): *enabled* holds infinitely often on every computation path

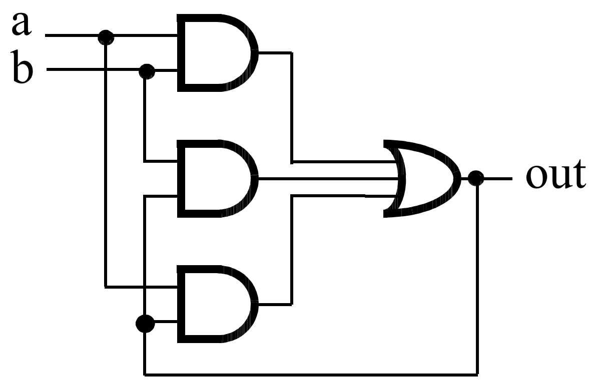
AG(EF *restart*): from any state it is possible to get to the *restart* state

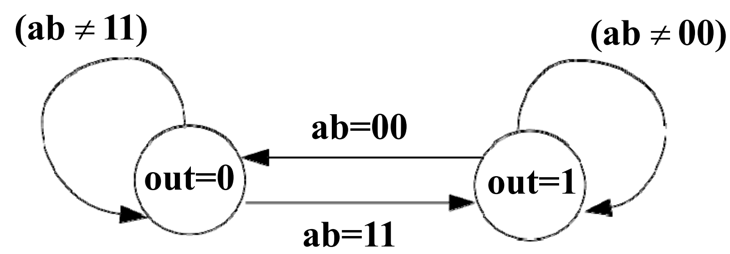
CTL*

- Computational tree logic CTL* combines branching-time and linear-time operators
- CTL* is sometimes referred to as **full branching-time logic**
- In CTL each linear-time operators G, F, X, and U must be immediately preceded by a path quantifier
- In CTL* a path quantifier can prefix an assertion composed of **arbitrary combinations of the usual linear-time operators (**F, G, X and U**)**
- Example: **EFp** is a basic modality of **CTL**; **E(Fp $\wedge$ Fq)** is a basic modality of **CTL***

CTL (cont'd)

Example: Two input Muller C-element (assuming finite discrete delays):





Specification in CTL:

- **Liveness**: *If inputs remain equal, then eventually the output will change to this value*. $AG(A((a=0 \wedge b=0) U (out=0 \vee a=1 \vee b=1)))$

  $AG(A((a=1 \wedge b=1) U (out=1 \vee a=0 \vee b=0)))$

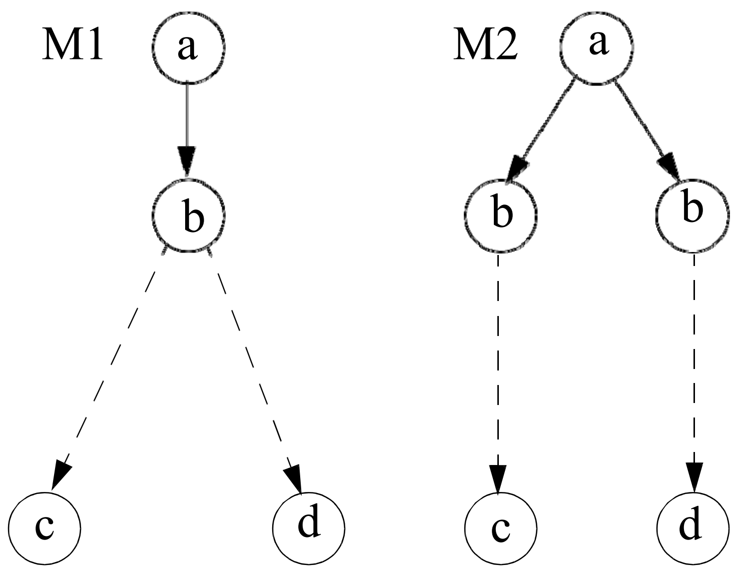
- **Safety**: *If all inputs and the output have the same value then the output should not change until all inputs change their values.*

  $AG((a=0 \wedge b=0 \wedge out=0) \Rightarrow A(out=0 \; U \; (a=1 \wedge b=1)))$

  $AG((a=1 \wedge b=1 \wedge out=1) \Rightarrow A(out=1 \; U \; (a=0 \wedge b=0)))$

- What about the environment? It may have to be constrained to satisfy some **fairness**!

Linear vs. Branching Time TL



Trace set is the same in both M1 and M2:
{ ab... c, ab... d }

Characterization by LTL:

$[a \wedge X (b \wedge F c)] \vee [a \wedge X (b \wedge F d)] =$

$a \wedge X (b \wedge (F (c \vee d))) =$

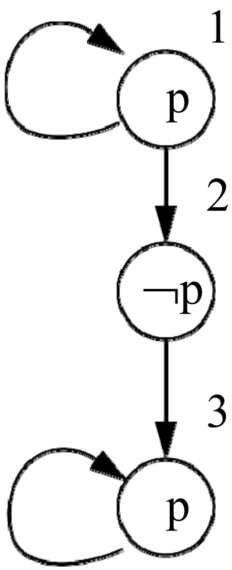
$a \wedge X (b \wedge (F (c) \vee F (d)))$

Characterization by CTL:

M1 and M2: $a \wedge AX (b \wedge (AF (c \vee d)))$

M2 only: $a \wedge AX (b \wedge (AF (c) \vee AF (d)))$

Linear vs. Branching Time TL (cont'd)



- In LTL the property F(G p) holds ((on all paths) eventually always p), but
- In CTL this cannot be expressed: AF(AG p) does not hold as there is no time instant where AG p holds,

  i.e., in state 1 the next state is either 1 or 2, the selfloop satisfies G p, but the transition to 2 (and then to 3) does not satisfy G p, hence AG p does not hold

- LTL: - easier inclusion of fairness constraints as preconditions in the same LTL language
  - AG EF p cannot be expressed
  - complexity of model checking: *exponential* in the length of the formula
- CTL: - fairness properties GF p $\Rightarrow$ GF q not expressible
  - fairness constraints often specified using exception conditions H_i
  - complexity of model checking: *deterministic polynomial*

Model Checking Problem for Temporal Logic

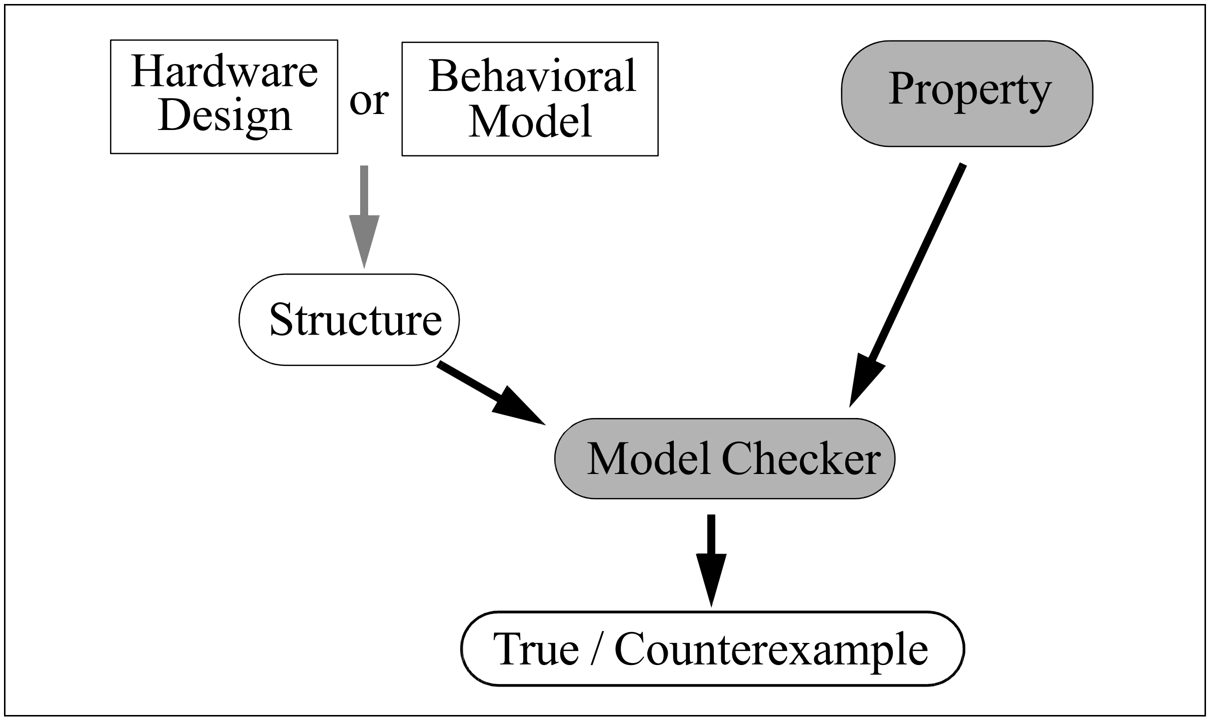
- Given an FSM M (equivalent Kripke structure) and a temporal logic formula p, does M define a model of p?
  - Determine the truth of a formula with respect to a given (initial) state in M
  - Find all states s of M such that **(M, s)** $\models$ **p**
- For any **propositional** temporal logic, the model checking problem is **decidable:** exhaustive search of all paths through the finite input structure

Some Theoretical Results

- Theorem [Wolper, 1986]: *The model checking for CTL is in deterministic polynomial time*
- Theorem [Sistla & Clark, 1985]: *The model checking problem for PLTL is PSPACE-complete*
- Theorem [Emerson & Lei, 1987]: *Given any model-checking algorithm for a linear logic LTL, there is a model checking algorithm for the corresponding branching logic BTL, whose basic modalities are defined by the LTL, of the same order of complexity*
- Theorem [Clark, Emerson & Sistla, 1986]: *The model checking problem for CTL* is PSPACE-complete*

Structure of Model Checker

Basic Idea:



- Specification Language: CTL
- Model of Computation: Finite-state systems modeled by labeled state-transition graphs (***Finite Kripke Structures)***
- If a state is designated as the *initial state*, the structure can be unfolded into an infinite tree with that state as the root: *Computation Tree*

Fixpoints

Model Checking Algorithms

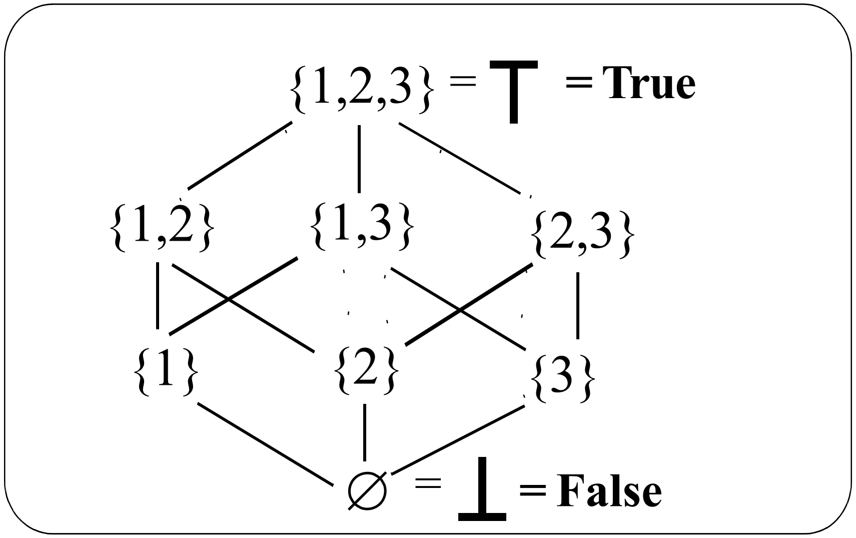
- Original algorithm described in terms of *labeling* the CTL structure (Clark83)
  Required explicit representation of the whole state space
- Better algorithm based on *fixed point* calculations
- Algorithm amenable to *symbolic* formulation
  Symbolic evaluation allows implicit enumeration of states
  Significant improvement in maximum size of systems that can be verified

Some Notions on Fixpoint

- (*Poset*) $<P, \leq>$ is a partially ordered set: P is a set and $\leq$ is a binary relation on P which is *reflexive*, *anti-symmetric* and *transitive*
- Let $<P, \leq>$ be a Poset and $S \subseteq P$
- (*lub*) $y \in P$ is a *least upper bound* of S in P means y is an upper bound of S and $\forall z \in P$ which is an upper bound of S, $y \leq z$
- (*glb*) $y \in P$ is a *greatest lower bound* of S in P means y is a lower bound of S and $\forall z \in P$ which is a lower bound of S, $z \leq y$
- If *lub*(S) (or *glb*(S)) exists, it is unique

Fixpoints (cont'd)

- A poset $\langle P, \leq \rangle$ has a universal lower bound $\bot \in P$ iff for all $y \in P$, $\bot \leq y$
- A poset $\langle P, \leq \rangle$ has a universal upper bound $\top \in P$ iff for all $y \in P$, $y \leq \top$
- A poset $\langle P, \leq \rangle$ is a *complete lattice* if *lub*(S) and *glb*(S) exist for every subset $S \subseteq P$
- Let 2^S be the power set of S (the set of all subsets of S)
- Poset $(2^S, \subseteq)$ is a complete lattice
- Example: S={1, 2, 3}



Fixpoints (cont'd)

- Let $<2^S, \subseteq>$ be complete lattice on S. Let f be a function: $2^S \rightarrow 2^S$
- f is *monotonic* $\Leftrightarrow \forall\, x, y \in 2^S .\ x \subseteq y \Rightarrow f(x) \subseteq f(y)$
- f is $\cup$–*continuous* if $P_1 \subseteq P_2 \subseteq P_3 \subseteq \ldots \Rightarrow f(\cup_i P_i) = \cup_i f(P_i)$, $\quad P_i \subseteq S$
- f is $\cap$–*continuous* if $P_1 \supseteq P_2 \supseteq P_3 \supseteq \ldots \Rightarrow f(\cap_i P_i) = \cap_i f(P_i)$, $\quad P_i \subseteq S$

Lemma: If S is finite, then any monotonic f is necessarily $\cup$–continuous and $\cap$–continuous (Monotonicity + Finiteness $\Rightarrow$ Continuity)

Proof. Any sequence of subsets $P_1 \subseteq P_2 \subseteq P_3 \subseteq \ldots$ of a finite set S must have a maximum element, say P_{max}, where $P_{max}=\cup_i P_i$. Since f is monotonic, we have $f(P_1) \subseteq f(P_2) \subseteq f(P_3) \subseteq \ldots \subseteq f(P_{max})$ such that $f(P_{max})=\cup_i f(P_i)$. On the other hand, $f(P_{max})=f(\cup_i P_i)$, thus $\cup_i f(P_i)=f(\cup_i P_i)$. $\cap$-continuous can be proven similarly.

- x is a fixpoint of f means $f(x) = x$
- *x is a least fixpoint* of f means $f(x) = x$ and $\forall y$ a fixpoint of f, $x \subseteq y$
- *x is a greatest fixpoint* of f means $f(x) = x$ and $\forall y$ a fixpoint of f, $y \subseteq x$

Fixpoints (cont'd)

Basic Fixpoint Theorems

Theorem 1. (Tarski & Knaster, 1955)

If f *is monotonic, then it has a least fixpoint,* $\mu Z.[f(Z)] = \cap\{Z \mid f(Z)=Z\}$, *and a greatest fixpoint,* $\upsilon Z.[f(Z)] = \cup\{Z \mid f(Z)=Z\}$.

- If f is monotonic, f has the least (greatest) fixpoint which is the intersection (union) of all the fixpoints.

Theorem 2. (Tarski & Knaster, 1955)

If f is $\cup$-continuous, $\mu Z.[f(Z)] = \bigcup_{i=1}^{\infty} f^i(\mathrm{False})$, and

if f is $\cap$-continuous, $\upsilon Z.[f(Z)] = \bigcap_{i=1}^{\infty} f^i(\mathrm{True})$

- Each fixpoint can be characterized as the limit of a series of approximations

Fixpoint Algorithm

- For a monotonic f and finite S:
  1. f is $\cup$-continuous and $\cap$-continuous
  2. $\forall i$, $f^i(False) \subseteq f^{i+1}(False)$ and $f^i(True) \supseteq f^{i+1}(True)$
  3. $\exists i_0$ such that $f^i(False) = f^{i_0}(False)$ for $i \geq i_0$
  4. $\exists j_0$ such that $f^j(True) = f^{j_0}(True)$ for $j \geq j_0$
  5. $\exists i_0$ such that $\mu Z.[f(Z)] = f^{i_0}(False)$
  6. $\exists j_0$ such that $\upsilon Z.[f(Z)] = f^{j_0}(True)$
- Standard **Least** (**Greatest**) Fixpoint Algorithm

  Y := ∅; {or **Y := S**}
  repeat
  Y' := Y; Y := f(Y)
  until Y' = Y;
  return Y;

- Terminates in at most |S| + 1 iterations with the least (greatest) fixpoint of f(Y).

Fixpoint Characterization of CTL

- M=(S,R,L) : a finite Kripke structure.
  - Identify each CTL formula f with a set of states $S_f = \{s \mid f \text{ is } \textbf{true} \text{ on } s \in S\}$.

Any formula f $\Leftrightarrow$ a set S_f of states

False $\Leftrightarrow$ the empty set $\varnothing$ True $\Leftrightarrow$ the complete set of states S

- 2^S forms a lattice under union and intersection, ordered by set inclusion $\subseteq$
- A functional $\tau: 2^S \rightarrow 2^S$ can be seen as *predicate transformer* on M
  e.g., $\tau(Z) = p \vee EX\, Z$

Theorem (Clark&Emerson, 1981): Given a finite structure M=(S,R,L)

$AFp = p \vee AX\, AFp = \mu Z.[p \vee AX\, Z]$ $EFp = p \vee EX\, EFp = \mu Z.[p \vee EX\, Z]$

$AGp = p \wedge AX\, AGp = \upsilon Z.[p \wedge AX\, Z]$ $EGp = p \wedge EX\, EGp = \upsilon Z.[p \wedge EX\, Z]$

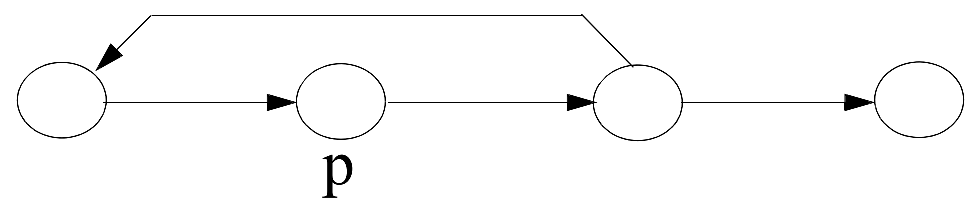
$A(pUq) = q \vee (p \wedge AX\, A(pUq)) = \mu Z.[q \vee (p \wedge AX\, Z)]$

$E(pUq) = q \vee (p \wedge EX\, E(pUq)) = \mu Z.[q \vee (p \wedge EX\, Z)]$

Fixpoint Characterization of CTL (cont'd)

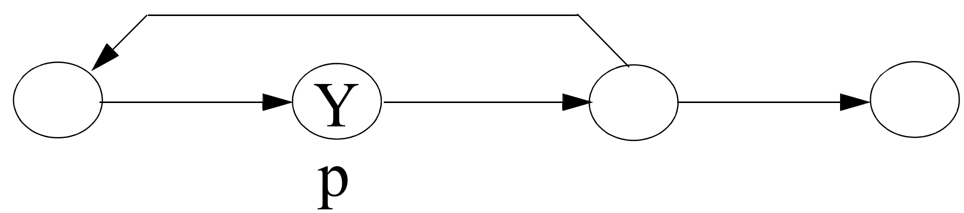
Example for EFp

- EFp in the following model: $|S| = 4$ and $\tau(Y) = p \vee EX(Y)$



- False does not hold in any states, since False represents the empty set of states ($\emptyset$)

Iteration 1:
$\tau^1(\text{False}) = p \vee EX(\text{False}) = p$

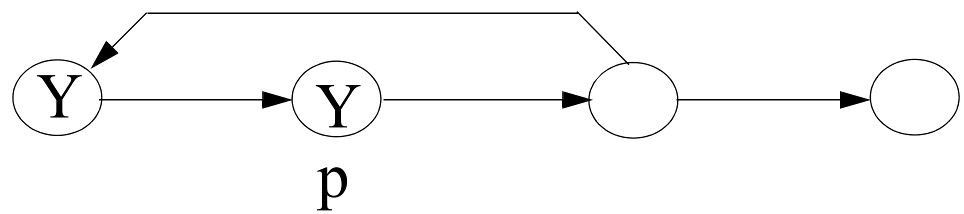


- EX(False): set of states such that False holds in at least one of their next states
- Use Y to mark the states where the current τ^1(False) holds

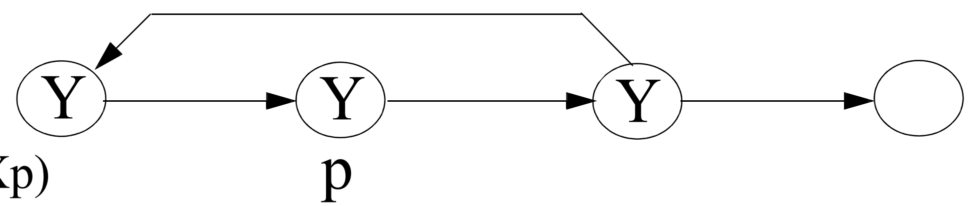
Fixpoint Characterization of CTL (cont’d)

Example for EFp (cont’d)

Iteration 2:
$\tau^2(\text{False}) = \tau(\tau(\text{False}))$
$= \tau(p) = p \vee EXp$



Iteration 3:
$\tau^3(\text{False}) = \tau(\tau(\tau(\text{False})))$
$= \tau(p \vee EXp) = p \vee EX(p \vee EXp)$



Iteration 4: $\tau^4(\text{False}) = \tau^3(\text{False})$

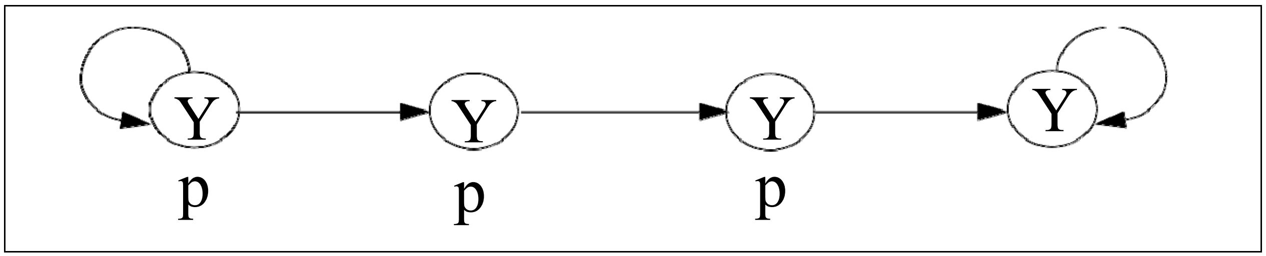
- Each iteration propagates the formula EFp **backward** in the graph by **one step**
- When fixpoint reached, Y labels exactly the set of states on a path to a state labeled with p
- To check if EFp holds in a certain state s, check if $s \in EFp$

Properties characterized as least fixpoints correspond to Eventualities

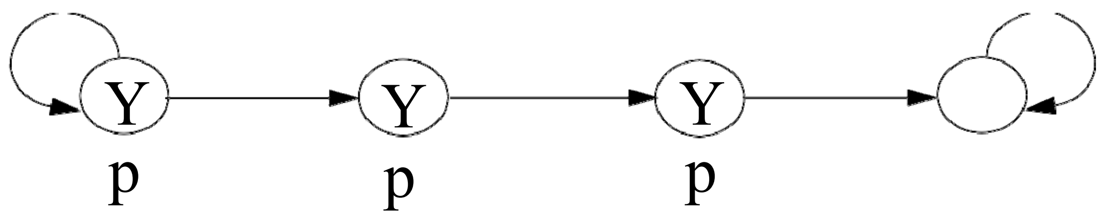
Fixpoint Characterization of CTL (cont'd)

Example for EGp

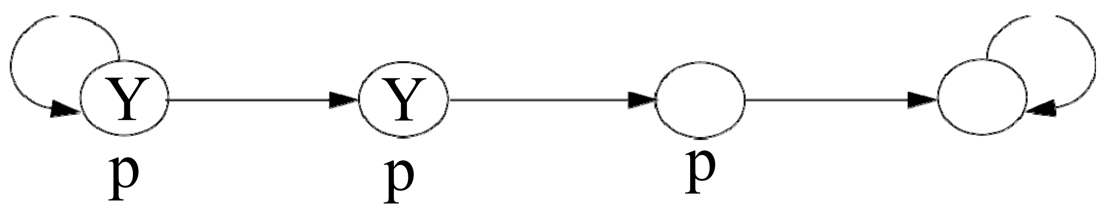
- EGp in the following model: $|S| = 4$ and $\tau(Y) = p \wedge EX(Y)$
- True holds in all states (True represents the set of all states), marked by Y



Iteration-1:
$\tau^1(\text{True}) = p \wedge EX(\text{True}) = p$



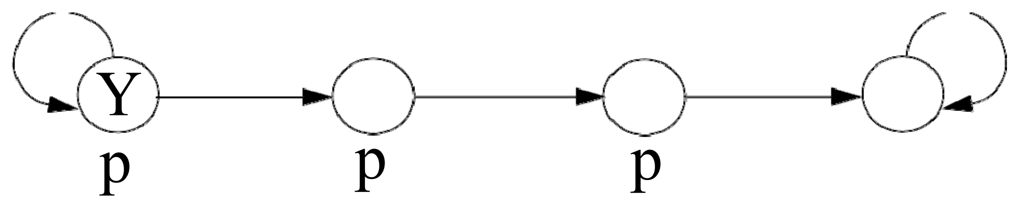
Iteration-2:
$\tau^2(\text{True}) = \tau(\tau(\text{True}))$
$= p \wedge EXp$



Fixpoint Characterization of CTL (cont'd)

Example for EGp (cont'd)

Iteration-3:
$\tau^3(\text{True}) = \tau(\tau(\tau(\text{True})))$
$= \tau(p \wedge EXp) = p \wedge EX(p \wedge EXp)$



- Iteration-4: $\tau^4(\text{True}) = \tau^3(\text{True})$
- At iteration i, Y labels the set of states such that there is a path of length i where every state satisfies p
- In fixpoint, every state in the set has a successor in the set satisfying p
- For any state in the set, there exists an infinite path where p is always true
- To verify if EGp holds in a certain state s, check if $s \in EGp$

Properties characterized as greatest fixpoints correspond to Invariants

CTL Model Checking Algorithm

- Given a Kripke Structure M = <S,R,L> and a CTL formula f, the following recursive algorithm computes the set of states $H(f) \subseteq S$ that satisfies f:

  $H(a) = \{s \mid s \text{ is labeled with } a\}$ for atomic formula a

  $H(\neg f) = S - H(f)$

  $H(f \wedge g) = H(f) \cap H(g)$

  $H(AXf) = \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in H(f)\}$

  $H(EXf) = \{s \mid \exists t.\ (s,t) \in R \Rightarrow t \in H(f)\}$

  $H(AGf) = \upsilon Z.[f \wedge AXZ] = \upsilon Z.\ (\ H(f) \cap \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in Z\}\ )$

  $H(EGf) = \upsilon Z.[f \wedge EXZ] = \upsilon Z.\ (\ H(f) \cap \{s \mid \exists t.\ (s,t) \in R \Rightarrow t \in Z\}\ )$

  $H(AFf) = \mu Z.[f \vee AXZ] = \mu Z.\ (\ H(f) \cup \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in Z\}\ )$

  $H(EFf) = \mu Z.[f \vee EXZ] = \mu Z.\ (\ H(f) \cup \{s \mid \exists t.\ (s,t) \in R \Rightarrow t \in Z\}\ )$

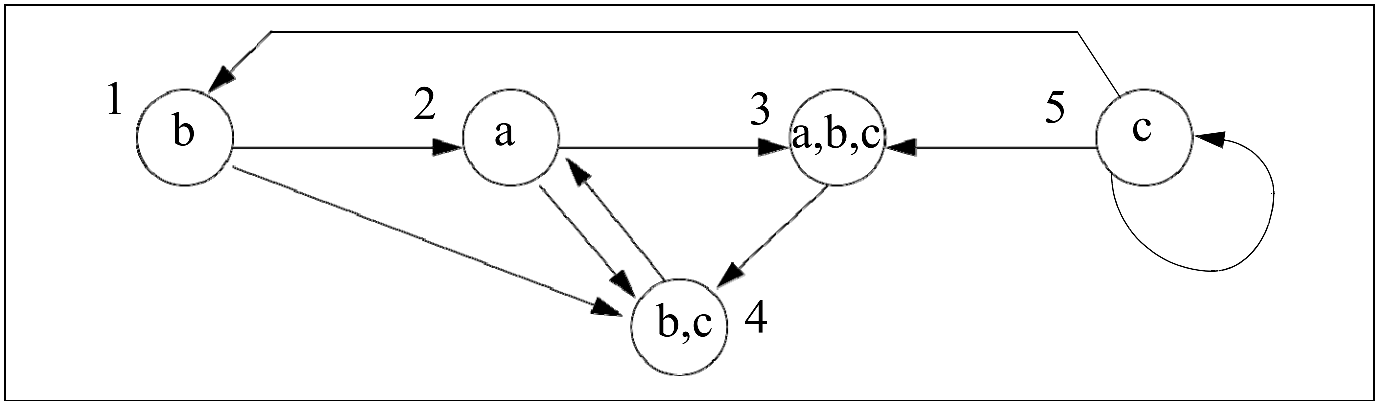
  $H(A(fUg)) = \mu Z.[g \vee (f \wedge AXZ)] = \mu Z.\ (\ H(g) \cup (\ H(f) \cap \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in Z\}\ )\ )$

  $H(E(fUg)) = \mu Z.[g \vee (f \wedge EXZ)] = \mu Z.\ (\ H(g) \cup (\ H(f) \cap \{s \mid \exists t.\ (s,t) \in R \Rightarrow t \in Z\}\ )\ )$

CTL Model Checking Algorithm (cont’d)

Example

Structure M <S,R,L>:



$S = \{1,2,3,4,5\}$, $AP = \{a,b,c\}$,

$R = \{(1,2), (2,3), (5,3), (5,5), (5,1), (2,4), (4,2), (1,4), (3,4)\}$

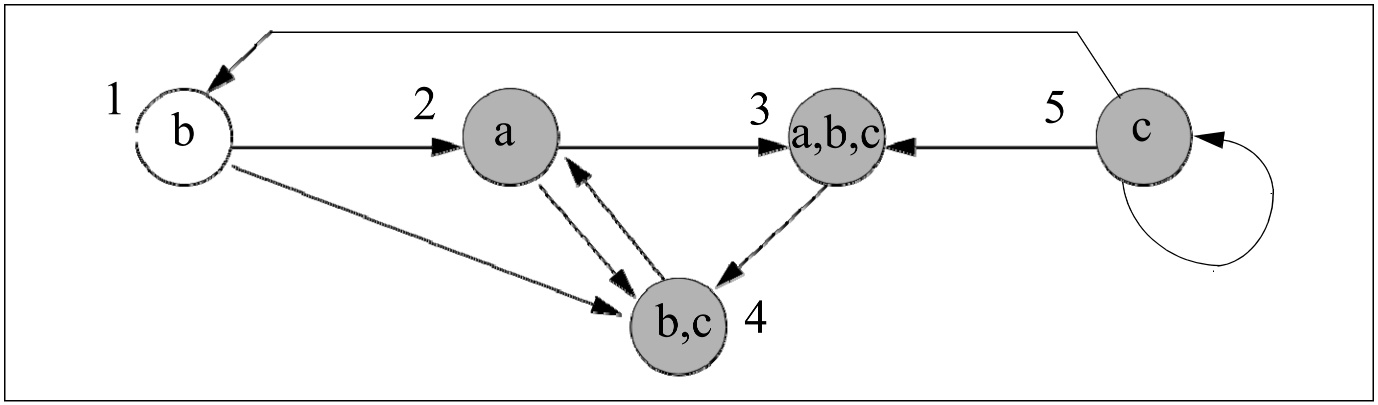
$L(1) = \{b\}$, $L(2) = \{a\}$, $L(3) = \{a,b,c\}$, $L(4) = \{b,c\}$, $L(5) = \{c\}$

Property: $AG(a \vee c)$

CTL Model Checking Algorithm (cont'd)

Example AG($a \vee c$) (cont'd)

- $H(a \vee c) = H(a) \cup H(c) = \{2,3\} \cup \{3,4,5\} = \{2,3,4,5\}$



- $H(AG(a \vee c)) = \upsilon Z.\{2,3,4,5\} \cap \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in Z\}$
- The greatest fixpoint calculation:

  $Z_0 = S = \{1,2,3,4,5\}$

  $Z_1 = \{2,3,4,5\} \cap \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in Z_0\} = \{2,3,4,5\} \cap \{1,2,3,4,5\} = \{2,3,4,5\}$

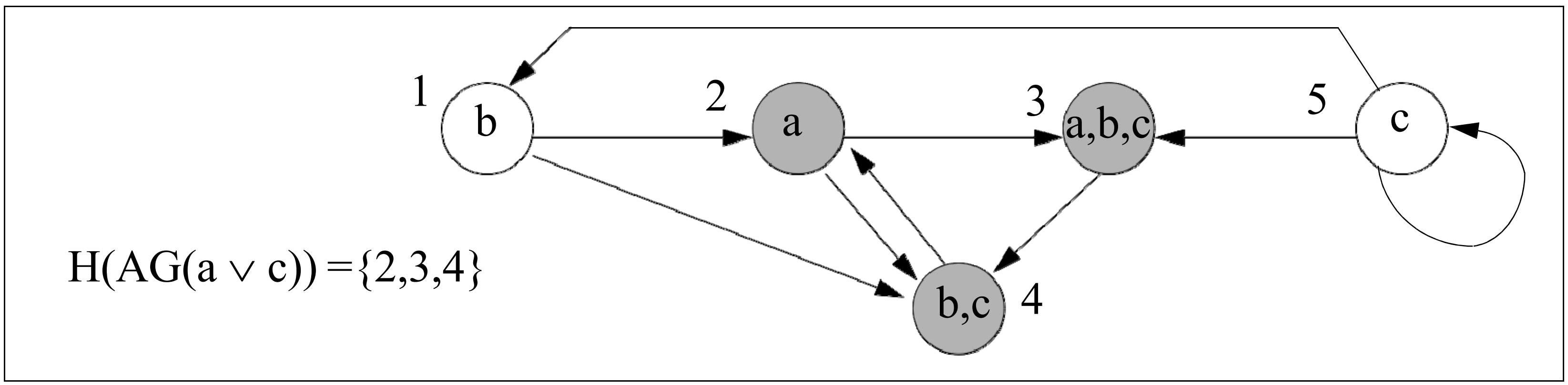
  $Z_2 = \{2,3,4,5\} \cap \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in Z_1\} = \{2,3,4,5\} \cap \{1,2,3,4\} = \{2,3,4\}$

  $Z_3 = \{2,3,4,5\} \cap \{s \mid \forall t.\ (s,t) \in R \Rightarrow t \in Z_2\} = \{2,3,4,5\} \cap \{1,2,3,4\} = \{2,3,4\}$

  $Z_3 = Z_2$

CTL Model Checking Algorithm (cont'd)

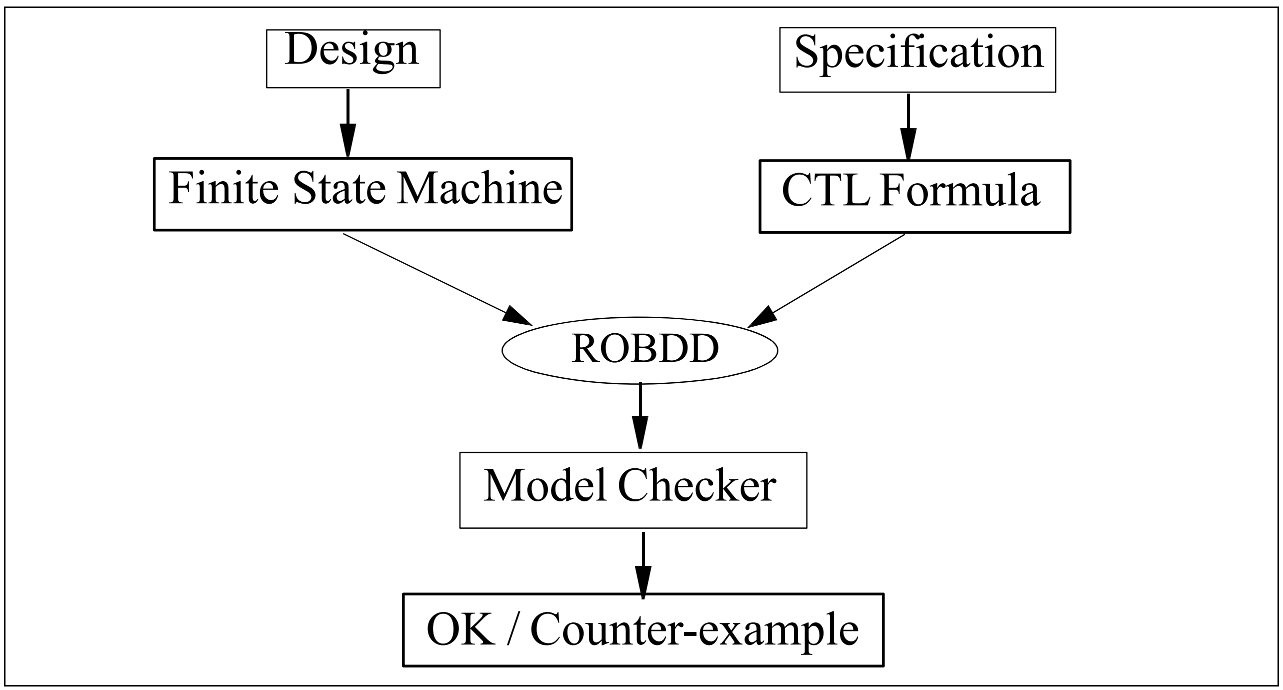
Example AG($a \vee c$) (cont'd)



- To verify that f holds in state s, check if $s \in H(f)$

Symbolic Model Checking

- Explicit State Representation $\Longrightarrow$ State Explosion Problem (about 10^8 states maximum)
- Breakthrough: Implicit State Representation using ROBDD (about 10^{20} states).
- Use Boolean characteristic functions represented by ROBDDs to encode sets of states and transition relations.



Symbolic Model Checking (cont'd)

- Let p be a set of states and **p** its Boolean encoding (ROBDD), then

$$p=\lambda(v_1, v_2,..., v_n)\ \mathbf{p}$$

- For a relation R on states, there is a unique representation **R** such that

$$R=\lambda(v_1, v_2, ..., v_n, v_1' ,v_2' , ..., v_n').\ \mathbf{R}$$

Computing EXp

- $EXp=\lambda v.\ \exists v'(R(v, v') \wedge p(v'))$, where $v=(v_1, v_2, ..., v_n)$, $v'=(v_1', v_2', ..., v_n')$

  $R(v, v')$ (relation) = **R**

  $p(v')$ (logic expression) = **p'**, where $\mathbf{p'} = \mathbf{p}[v_i \leftarrow v_i']$

  $\Longrightarrow EXp = \lambda v.\ \exists v'(\mathbf{R} \wedge \mathbf{p'})$

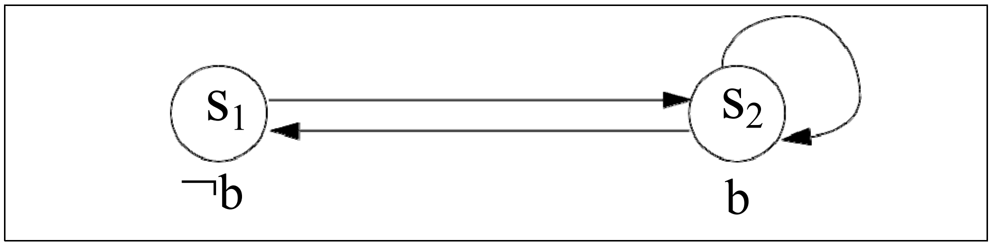
- Algorithm: Given **p** for p;

  1. $\mathbf{p'} := \mathbf{p}[v_i \leftarrow v_i']$;
  2. $S(v) := \exists v' (\mathbf{R} \wedge \mathbf{p'})$;
  3. Check if initial state $s_0 \in S(v)$.

Symbolic Model Checking (cont'd)

Example 1: EX¬b

- EX¬b in a model with v = (b), v' = (b'), R = b ∨ b' and **R** = R(b, b') = b ∨ b':



- EX¬b

$$= \exists b'(\mathbf{R} \wedge \mathbf{p'})$$

$$= \exists b' ((b \vee b') \wedge ((\neg b)[b \leftarrow b']))$$

$$= \exists b' ((b \vee b') \wedge \neg b')$$

$$= \exists b' (b \wedge \neg b')$$

$$= (b \wedge \neg 0) \vee (b \wedge \neg 1)$$

= b (state s_2 makes EX¬b true)

Symbolic Model Checking (cont'd)

Example 1: EFb

EFb = μy. (b $\vee$ EXy) on R = b $\vee$ b' as before.

- Use least fixed point algorithm:

  $\tau^1[0] = b \vee EX[0]=b$

  $\tau^2[0] = b \vee EXb$

  $= b \vee \exists b'. ((b \vee b') \wedge b')$ {go backward along transitions}

  $= b \vee (b \vee 1)$ {existentially quantify away b'}

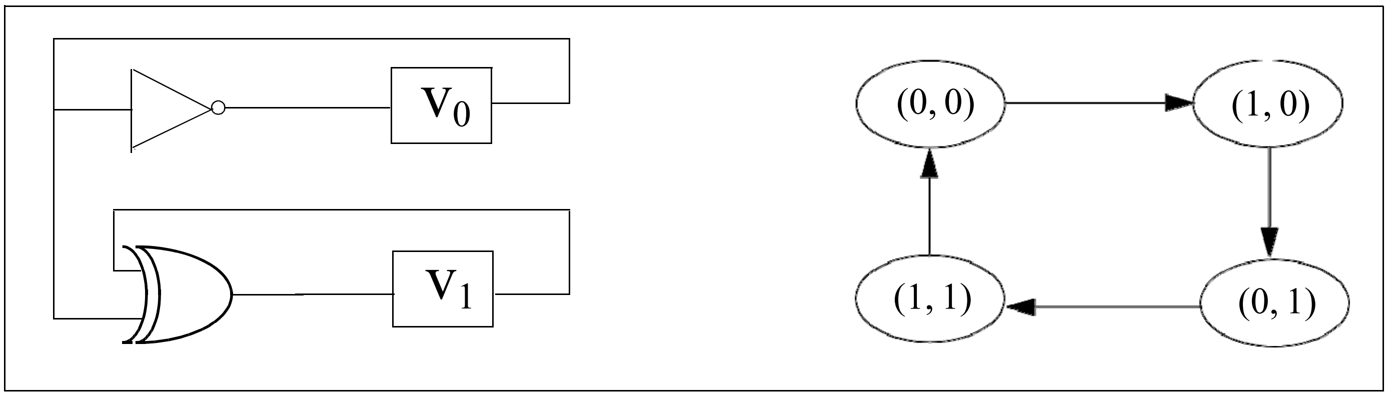
  $= 1$

  $\tau^3[0] = b \vee EX1 = 1$

- EFb = $\{s_1, s_2\}$: for any state of the model, there is a state in the future in which b is true.

Symbolic Model Checking (cont'd)

Example 2: Counter



- State variables: v_0, v_1, $\{v = (v_0, v_1)\}$
- Next state variables: v_0', v_1', $\{v' = (v_0', v_1')\}$
- Transition relation: $\mathbf{R} = (v_0' \Leftrightarrow \neg v_0) \wedge (v_1' \Leftrightarrow (v_0 \oplus v_1))$

Symbolic Model Checking (cont'd)

Example 2: Counter (cont'd)

- $EX(v_0 \wedge v_1)$

  $= \exists v'. (\mathbf{R} \wedge \mathbf{p'})$

  $= \exists(v_0', v_1'). (\mathbf{R} \wedge (v_0' \wedge v_1'))$

  $= \exists(v_0', v_1'). ([(v_0' \Leftrightarrow \neg v_0) \wedge (v_1' \Leftrightarrow (v_0 \oplus v_1))] \wedge (v_0' \wedge v_1'))$

  $= \exists v_0' . ((v_0' \Leftrightarrow \neg v_0) \wedge (v_0 \oplus v_1) \wedge v_0')$

  $= \neg v_0 \wedge (v_0 \oplus v_1)$

  $= \neg v_0 \wedge ((\neg v_0 \wedge v_1) \vee (v_0 \wedge \neg v_1))$

  $= \neg v_0 \wedge v_1$

- Meaning: state (0, 1) satisfies $EX(v_0 \wedge v_1)$

Symbolic Model Checking (cont'd)

Example 2: Counter (cont'd) **$EF(v_0 \wedge v_1) = \mu y. ((v_0 \wedge v_1) \vee EXy)$**

$\tau^1[0] = (v_0 \wedge v_1) \vee EX0 = (v_0 \wedge v_1)$

$\tau^2[0] = (v_0 \wedge v_1) \vee EX(v_0 \wedge v_1)$

$= (v_0 \wedge v_1) \vee (\neg v_0 \wedge v_1)$ {from the result of $EX(v_0 \wedge v_1)$}

$= v_1$

$\tau^3[0] = (v_0 \wedge v_1) \vee EX(v_1)$

$= (v_0 \wedge v_1) \vee [\exists(v_0', v_1'). (\mathbf{R} \wedge v_1')]$

$= (v_0 \wedge v_1) \vee [\exists(v_0', v_1'). ((v_0' \Leftrightarrow \neg v_0) \wedge (v_1' \Leftrightarrow (v_0 \oplus v_1)) \wedge v_1')]$

$= (v_0 \wedge v_1) \vee [\exists v_0'. ((v_0' \Leftrightarrow \neg v_0) \wedge (v_0 \oplus v_1))]$

$= (v_0 \wedge v_1) \vee [(v_0 \oplus v_1)]$

$= (v_0 \wedge v_1) \vee (\neg v_0 \wedge v_1) \vee (v_0 \wedge \neg v_1) = v_0 \vee v_1$

$\tau^4[0] = (v_0 \wedge v_1) \vee EX(v_0 \vee v_1)$

$= (v_0 \wedge v_1) \vee [\exists(v_0', v_1'). (\mathbf{R} \wedge (v_0' \vee v_1'))]$

$= (v_0 \wedge v_1) \vee \neg v_0 \vee (v_0 \oplus v_1)$

$= (v_0 \wedge v_1) \vee \neg v_0 \vee (\neg v_0 \wedge v_1) \vee (v_0 \wedge \neg v_1) = 1$

- $EF(v_0 \wedge v_1) = \{(0, 0), (0, 1), (1, 0), (1, 1)\} \rightarrow$ All states satisfy $EF(v_0 \wedge v_1)$

Symbolic Model Checking Algorithm

- ***eval*** takes a CTL formula as its argument and returns the ROBDD for the set of states that satisfy the formula

- **`function`** eval(f)

```
   case
     f an atomic proposition: return f;
     f = ¬p:        return ¬eval(p);
     f = p∨q:       return eval(p) ∨ eval(q);
     f = EXp:       return evalEX(eval(p));
     f = E(pUq):    return evalEU(eval(p), eval(q), False);
     f = EGp:       return evalEG(eval(p), True)
   end case
 end function;
```

- **`function`** evalEX(**p**) = $\exists$v'(**R**$\wedge$**p'**)

- **`function`** evalEU(**p**, **q**, **y**)

```
  y' = q ∨ (p ∧ evalEX(y))
  if y' = y
    then return y
    else return evalEU(p, q, y')
  end function
```

- **`function`** evalEG(**p**, **y**)

```
  y' = p ∧ evalEX(y)
  if y' = y
    then return y
    else return evalEG(p, y')
  end function
```

Model Checking Tools

SMV (Symbolic Model Verifier)

- A tool for checking finite state systems against specifications in the temporal logic CTL.
- Developed at Carnegie Mellon University by E. Clarke, K. McMillan et. al.
- Supports a simple input language: SMV
- For more information: http://www.cs.cmu.edu/~modelcheck/smv.html

Cadence SMV

- Updated version of SMV by K. McMillan at Berkeley Cadence Labs
- Input languages: extended SMV and synchronous Verilog
- Supports temporal logics CTL and LTL, finite automata, embedded assertions, and refinement specifications.
- Features compositional reasoning, link with a simple theorem prover, an easy-to-use graphical user interface and source level debugging capabilities
- For more information: http://www.kenmcmil.com/smv.html

Model Checking Tools (cont'd)

VIS (Verification Interacting with Synthesis)

- A system for formal verification, synthesis, and simulation of finite state systems.
- Developed jointly at the University of California at Berkeley and the University of Colorado at Boulder.
- VIS provides the following features:
  - Fast simulation of logic circuits
  - Formal "implementation" verification (equivalence checking) of combinational and sequential circuits
  - Formal "design" verification using fair CTL model checking and language emptiness
- For more information: https://embedded.eecs.berkeley.edu/research/vis

Model Choking Tools (cont'd)

CheckOff-M

- Commercial product by Abstract Hardware Ltd. (UK) and Siemens AG (Germany)
- Performs verification of properties stated in a temporal logic on an FSM
- Input EDIF netlist + library or superset of synthesizable synchronous VHDL and Verilog
- Converts to *Macro FSM* by merging transition (represented by ROBDDs)
- Temporal logic: subset of Computation Tree Logic (CTL) + Intervals = CIL
  - VHDL-like syntax for predicates, temporal operators *always*, *possibly*, *within*, *during*, ...
  - Property = theorem = assumption on valid sequences + consequence
- Tool does not exist anymore

Model Checking Tools (cont’d)

FormalCheck

- Developed at Bell Labs. Now commercial product of Cadence
- Performs model checking of properties stated in temporal logic
- Supports the synthesizable subsets of Verilog and VHDL hardware design languages.
- User supplies FormalCheck with a set of queries (properties and constraints)
- Each property is defined using semantics of the class of omega automata.
- Tool provides powerful *model reduction* options.
- Tool replaced by JasperGold® Formal Verification Platform

References

Temporal Logics:

1. E. A. Emerson. Temporal Logics in *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B.V., 1990
2. Z. Manna, A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1991.

CTL:

3. E.M. Clarke, E. A. Emerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM transactions on Programming Languages and Systems*, 8(2):244-263 (April 1986).
4. E. A. Emerson, C.L. Lei. Modalities for model checking: Branching time strikes back. In *Proc. ACM Symposium on Principles of Programming Language*, ACM, New York, 1985, pp. 84-96.
5. A. P. Sistla, E. M. Clarke. The complexity of propositional linear temporal logics.*JACM*, 32(3), 1985, pp. 733-749.
6. E. A. Emerson, E.M. Clarke. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *JACM*, 33(1), 1986, pp. 151-178.

Model Checking:

7. C. Baier, J.-P. Katoen: *Principles of Model Checking*, MIT Press, 2008.
8. K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

References (cont'd)

9. E.M. Clarke, E. A. Emerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM transactions on Programming Languages and Systems*, 8(2), 1986, pp. 244-263.
10. E. A. Emerson and C.L. Lei. Modalities for model checking: Branching time strikes back. In Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Language, ACM, New York, 1985, pp. 84-96.
11. M.C. Browne, E. M. Clarke, D.L. Dill. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12), 1986, pp. 1035-1044.
12. E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction", *Proc. ACM Symp. on Principles of Programming Languages*, January 1992.
13. J. R. Burch, E. M. Clarke, D. Long, K. L. McMillan, D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on CAD*, 13(4), 1994, pp. 401-424.
14. O. Coudert, I.C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their date diagrams. In *Proc. Computer-Aided Verification*, Springer-Verlag, New York, NY, 1991.
15. H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDDs. *Proc. International Conference on Computer-Aided Design*, 1990.
16. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2), 1992.