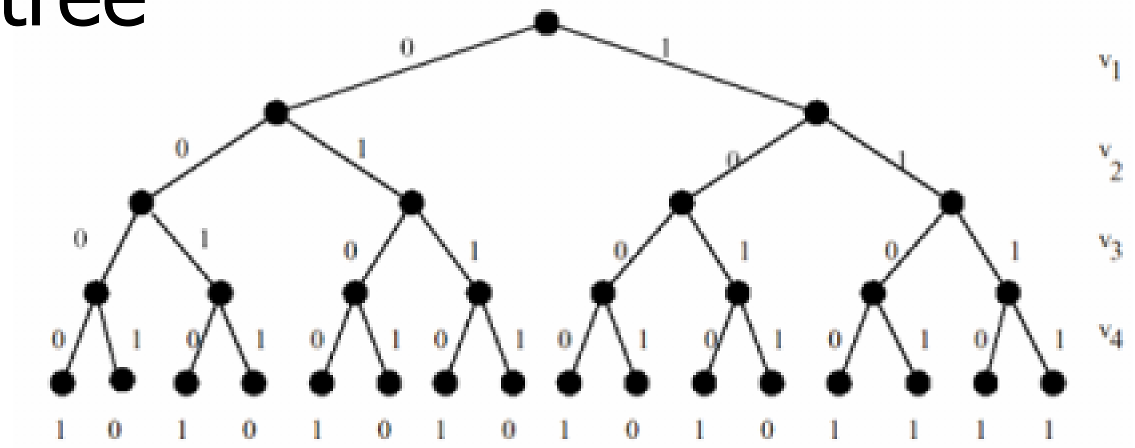
Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

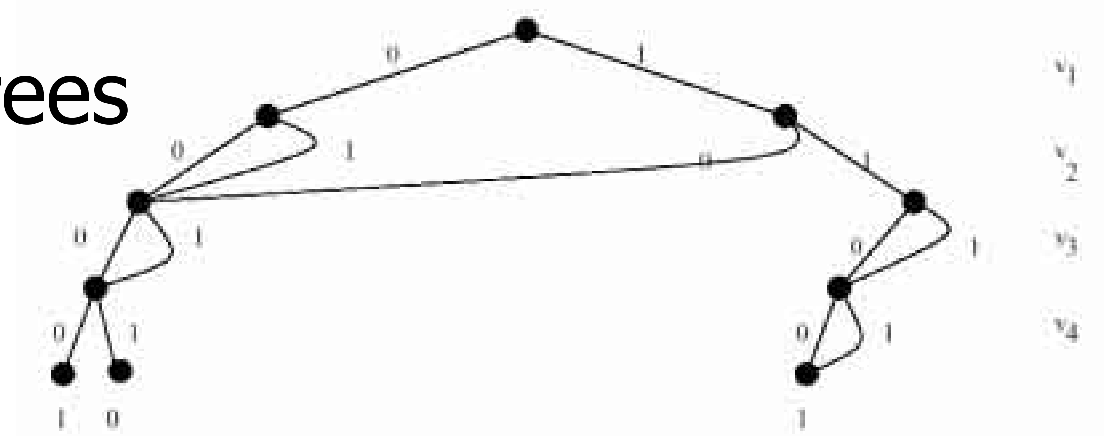
Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Binary Decision Trees (BDTs)

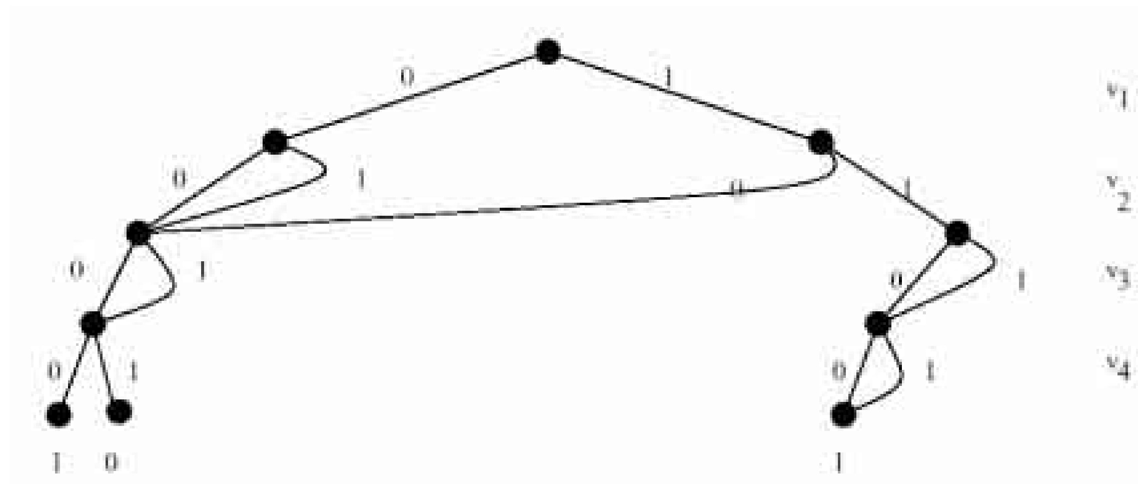
- Binary decision tree



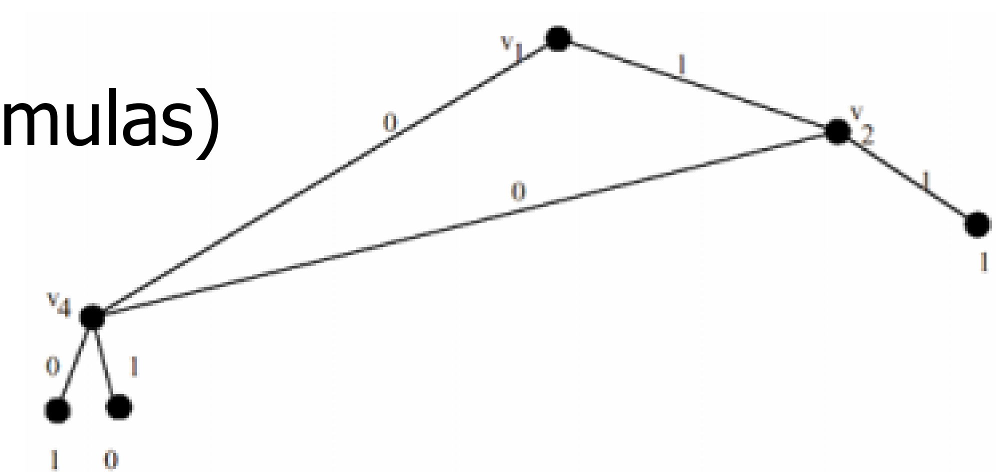
- Elimination of isomorphic subtrees (abbreviations)



Binary Decision Diagrams (BDDs)



- Elimination of redundant nodes (redundant subformulas)
  Ite (v,ψ,ψ) by ψ



Calculation of BDDs

```
function PL2BDD (Formula φ) : (Nodeset, Int)
    /* Calculates the BDD of φ
        as a set of nodes and a pointer to the topmost node */
    Nodeset table := {}; /* Table of BDD nodes (δ, i, δ1, δ2) */
    Int max := 1; /* Index of maximal table entry */
    Int result := BDD(φ,1); /* Index of topmost BDD node */
    return (table, result);

function BDD (Formula φ, Int i) : Int
    /* φ is the current subformula, i is the current BDD variable */
    /* Return value is a pointer to the maximal BDD node */
    if i > n then return eval(φ) /* φ is a boolean constant */
    else δ1 := BDD(φ{v_i := ⊥}, i + 1); δ2 := BDD(φ{v_i := ⊤}, i + 1);
        if δ1 = δ2 then return δ1
        elsif ∃δ : (δ, i, δ1, δ2) ∈ table then return δ
        else max := max + 1; table := table ∪ {(max, i, δ1, δ2)}; return max;
```

Boolean operations on BDDs

function BDD_imp (Int φ, ψ) : Int
/* Calculates the BDD of $(\varphi \rightarrow \psi)$ from the BDDs of φ and ψ */
if $\varphi = 0$ **or** $\psi = 1$ **then return** 1
elsif $\varphi = 1$ **and** $\psi = 0$ **then return** 0
elsif $\varphi = 1$ **and** $(\psi, j, \psi_1, \psi_2) \in table_\psi$
then return new_node(j, BDD_imp($1, \psi_1$), BDD_imp($1, \psi_2$))
elsif $\psi = 0$ **and** $(\varphi, i, \varphi_1, \varphi_2) \in table_\varphi$
then return new_node(i, BDD_imp($\varphi_1, 0$), BDD_imp($\varphi_2, 0$))
else $(\varphi, i, \varphi_1, \varphi_2) \in table_\varphi$ **and** $(\psi, j, \psi_1, \psi_2) \in table_\psi$
if $i = j$ **then return** new_node(i, BDD_imp(φ_1, ψ_1), BDD_imp(φ_2, ψ_2))
elsif $i < j$ **then return** new_node(i, BDD_imp(φ_1, ψ), BDD_imp(φ_2, ψ))
elsif $i > j$ **then return** new_node(j, BDD_imp(φ, ψ_1), BDD_imp(φ, ψ_2));

Generally

- This procedure can be applied for arbitrary boolean connectives (or, and, not)
  - this amounts to set union, intersection, and complement with respect to the base set
- Substitution by constants is trivial
- Boolean quantification:

$$\exists q(\varphi) \quad \leftrightarrow \quad (\varphi\{q := \top\} \vee \varphi\{q := \top\})$$

Binary Encoding of Relations

- A relation is a subset of the product of two sets
  - Thus, a relation is nothing but a set
- Example: var v: {0..3}, w:{0..7};

  var v0, v1, w0, w1, w2: boolean;

  “divides”-Relation:

  v divides w iff
  v=1, or v=2 and w even,
  or v=3 and w in {0,3,6}

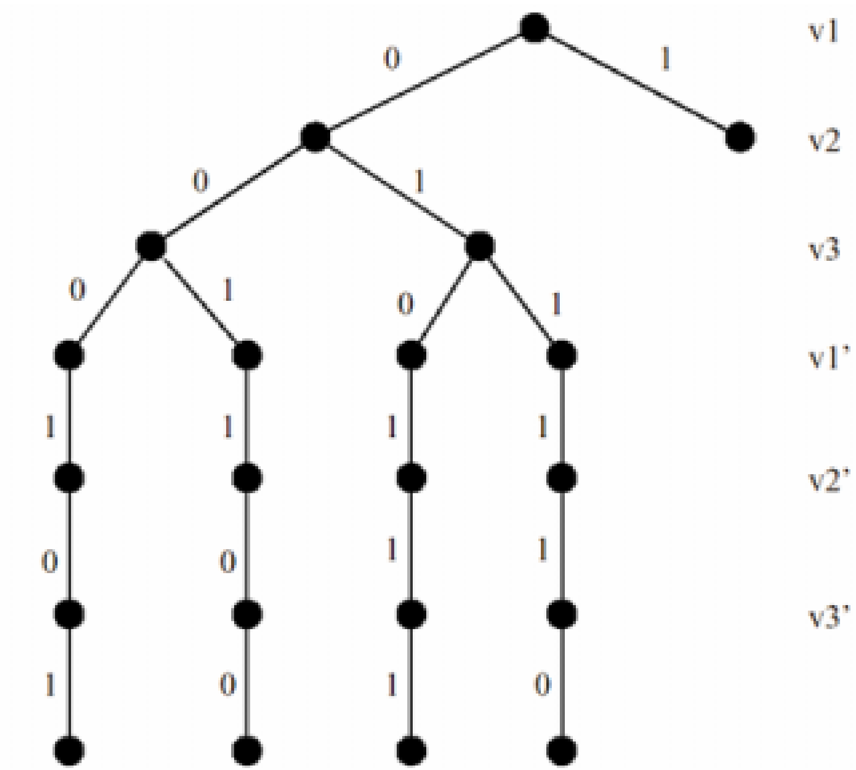
	0	1	2	3	4	5	6	7
0	-	-	-	-	-	-	-	-
1	+	+	+	+	+	+	+	+
2	+	-	+	-	+	-	+	-
3	+	-	-	+	-	-	+	-

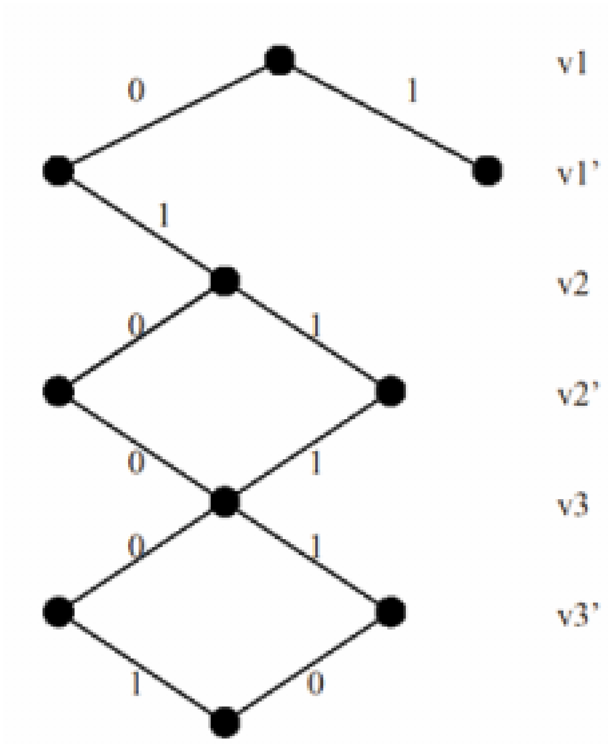
boolean formula:

$$
\begin{aligned}
v \| w \leftrightarrow & ((\neg v_0 \wedge v_1) \vee (v_0 \wedge \neg v_1 \wedge \neg w_2) \vee \\
& (v_0 \wedge v_1 \wedge ((\neg w_0 \wedge \neg w_1 \wedge \neg w_2) \vee \\
& (\neg w_0 \wedge w_1 \wedge w_2) \vee (w_0 \wedge w_1 \wedge \neg w_2)))
\end{aligned}
$$

The Influence of Variable Ordering

$$v_1 = 0 \rightarrow ((v_1' = 1) \wedge (v_2' = v_2) \wedge (v_3' \neq v_3))$$





Transitive Closure

- Each finite (transition) relation can be represented as a BDD
- The transitive closure of a relation R is defined recursively by

$$R^*(x,y) \leftrightarrow R(x,y) \vee \exists z(R(x,z) \wedge R^*(z,y))$$

- Thus, transitive closure be calculated by an iteration on BDDs

$$R^0(x,y) \triangleq R(x,y)$$
$$R^{i+1}(x,y) \leftrightarrow R^i(x,y) \vee \exists z(R(x,z) \wedge R^i(z,y))$$

Reachability

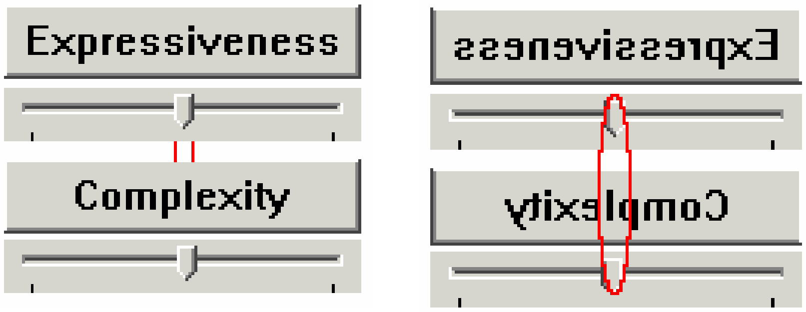
- State s is reachable iff $s_0 R^* s$, where $s_0 \in S_0$ is an initial state and R is the transition relation
- Reachability is one of the most important properties in verification
  - most safety properties can be reduced to it
  - in a search algorithm, is the goal reachable?
- Can be arbitrarily hard
  - for infinite state systems undecidable
- Can be efficiently calculated with BDDs

Pause!

(ein Screenshot)

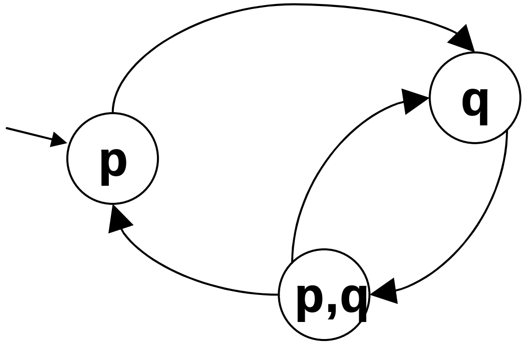
Logical Languages, Expressiveness

- So far, we've been doing validation of systems without using anything but propositional logic!
- It is more interesting (though not necessarily more practical) to consider more expressive logics
- Dilemma:



Models

- A model M is a graph consisting of
  - a set of nodes U (universe)
  - a transition relation R between nodes
    - transitive closure of R denoted by <
  - an initial node w_0
  - an assignment I of propositions to nodes

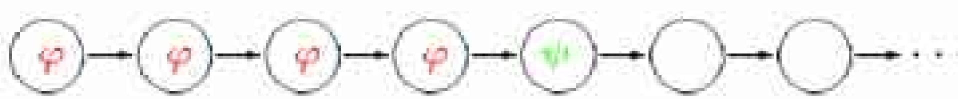


Temporal logic

- “Modal logic with ‘until’”

- $w \not\models \bot$, and $w \models (\varphi \to \psi)$ iff $w \models \varphi$ implies $w \models \psi$;
- $w \models \mathrm{p}$ iff $w \in \mathcal{I}(w, \mathrm{p})$;
- $w \models (\varphi\ \mathbf{U}^{+}\ \psi)$ iff $v \models \psi$ for some $v > w$, and $u \models \varphi$ for $w < u < v$.

- $w \models (\varphi\ \mathbf{U}^{*}\ \psi)$ iff $v \models \psi$ for some $v \geq w$, and $u \models \varphi$ for $w \leq u < v$.



Examples

(3) $\forall t_1((t_0 \leq t_1 \wedge \mathtt{req}(t_1)) \rightarrow$
$\exists t_2((t_1 < t_2 \wedge \mathtt{ack}(t_2)) \wedge$
$\forall t_3((t_1 < t_3 \wedge t_3 < t_2) \rightarrow \mathtt{req}(t_3))))$
No request is withdrawn before it is acknowledged.

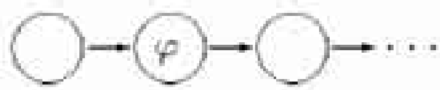
$$\mathbf{G}^* \; (\mathtt{req} \rightarrow (\mathtt{req}\; \mathbf{U}^+ \; \mathtt{ack}))$$

```
NEVER OUTPUT MTea INSIDE EACH INTERVAL
      STARTING AT EVENT WCof ENDING AT STATE Idle
```

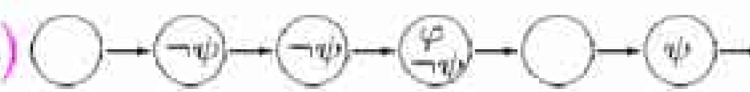
Other connectives

- *sometime*: $\mathbf{F}^+ \varphi \triangleq (\top \, \mathbf{U}^+ \varphi)$ 

- *always*: $\mathbf{G}^+ \varphi \triangleq \neg \mathbf{F}^+ \neg\varphi$ 

- *next-time*: $\mathbf{X} \varphi \triangleq (\bot \, \mathbf{U}^+ \varphi)$ 

- *atnext*: $(\varphi \, \mathbf{A}^+ \psi) \triangleq (\neg\psi \, \mathbf{U}^+ (\varphi \wedge \psi))$ 

- *before*: $(\varphi \, \mathbf{B}^+ \psi) \triangleq (\neg\psi \, \mathbf{U}^+ (\varphi \wedge \neg\psi))$ 

Linear and Branching Time Logics

Executions of a program =

- set of execution sequences, or
- single execution tree

LTL (linear temporal logic) is interpreted on **sequences**.
(Syntactically, there is no difference between **TL** and **LTL**!)

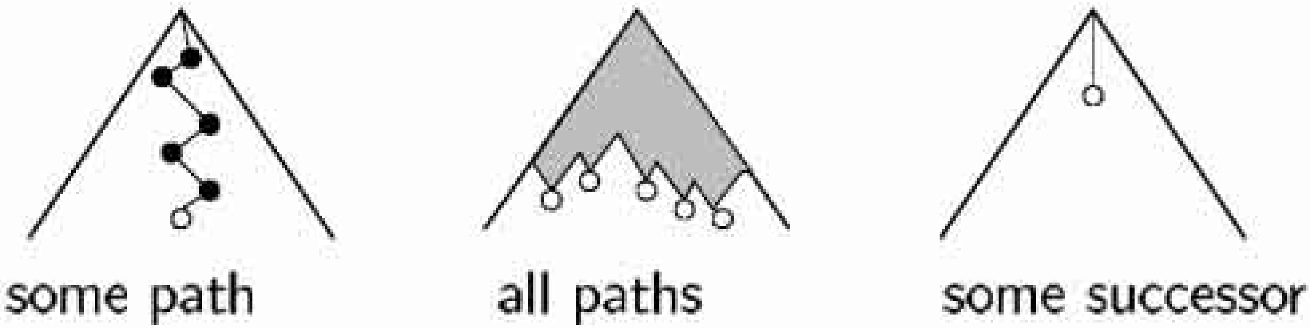
Syntax of **CTL** (computation tree logic):

$$\mathbf{CTL} ::= \mathcal{P} \mid \bot \mid (\mathbf{CTL} \to \mathbf{CTL}) \mid \mathbf{E}(\mathbf{CTL}\ \mathcal{U}^{+}\ \mathbf{CTL}) \mid \mathbf{A}(\mathbf{CTL}\ \mathcal{U}^{+}\ \mathbf{CTL})$$

Semantics of CTL

CTL is interpreted on **trees**, where $<$ is the usual tree-order.

- $w_0 \models \mathbf{E}(\psi \, \mathbf{U}^+ \, \varphi)$ iff $\exists \, w_1 > w_0, \, w_1 \models \varphi, \, \forall \, w_0 < w_2 < w_1, \, w_2 \models \psi$
- $w_0 \models \mathbf{A}(\psi \, \mathbf{U}^+ \, \varphi)$ iff *for all paths p* from w_0,
  $\exists \, w_1 > w_0$ *on path p* s.t. $w_1 \models \varphi$, and $\forall \, w_0 < w_2 < w_1, \, w_2 \models \psi$



CTL Examples

— $\mathbf{E\,F}^{+}$ ($\mathtt{started} \wedge \neg\mathtt{ready}$): it is possible to get to a state where `started` holds but `ready` does not hold.

— $\mathbf{A\,G}^{*}$ ($\mathtt{req} \rightarrow \mathbf{A\,F}^{+}\ \mathtt{ack}$): if a request occurs, then it will be eventually acknowledged

— $\mathbf{A\,G}^{*}\ \ \mathbf{A\,F}^{*}$ `stack_is_empty`: the proposition `stack_is_empty` holds infinitely often on every computation path

— $\mathbf{A\,G}^{*}\ \ \mathbf{E\,F}^{*}$ `restart`: from any state it is possible to get to a `restart` state.

Model Checking

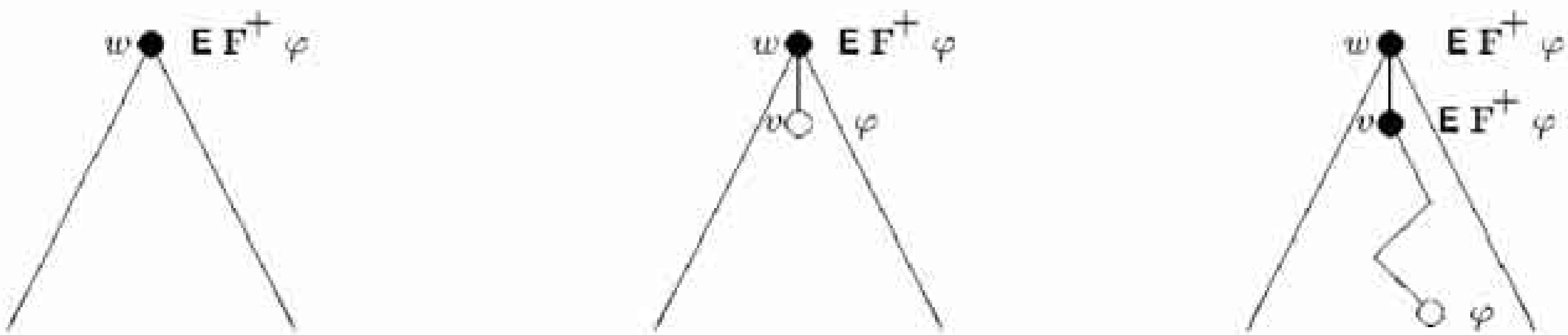
Model checking problem:

Given finite Kripke model $\mathcal{M} = (U, \longrightarrow, \mathcal{I}, w_0)$ and formula φ, check if

$$\mathcal{M} \models \varphi$$

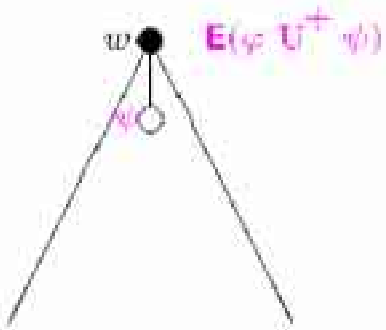
For **CTL**: Recursive descent on subformulas.

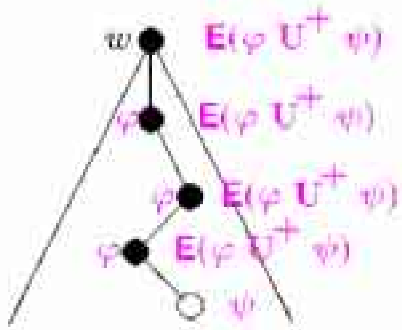
$w \models \mathbf{E}\,\mathbf{F}^{+}\,\varphi$ iff $\exists w \longrightarrow v$ s.t. $v \models \varphi$ or $\exists w \longrightarrow v$ s.t. $v \models \mathbf{E}\,\mathbf{F}^{+}\,\varphi$

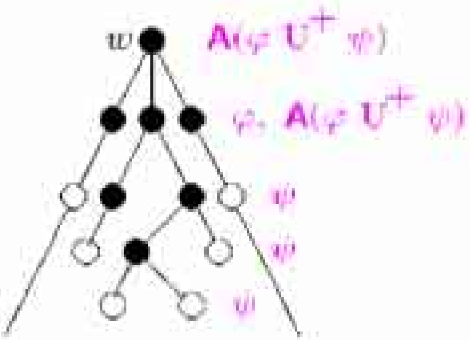


Similarly,

- $w \models \mathbf{E}(\varphi\,\mathbf{U}^{+}\,\psi)$ iff
  for some $w \longrightarrow v$ it holds that $v \models \psi$ or $v \models \varphi$ and $v \models \mathbf{E}(\varphi\,\mathbf{U}^{+}\,\psi)$
- $w \models \mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi)$ iff
  for all $w \longrightarrow v$ it holds that $v \models \psi$ or $v \models \varphi$ and $v \models \mathbf{A}(\varphi\,\mathbf{U}^{+}\,\psi)$





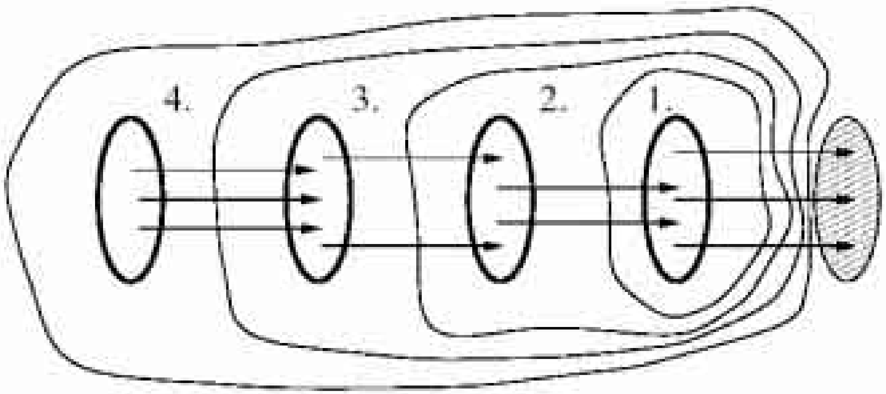


Let $\varphi^{\mathcal{M}} = \{w \mid w \models \varphi\}$.

$(\mathbf{E}\,\mathbf{F}^{+}\,\varphi)^{\mathcal{M}}$ is the set of points from which some point in $\varphi^{\mathcal{M}}$ is reachable.
How to determine $(\mathbf{E}\,\mathbf{F}^{+}\,\varphi)^{\mathcal{M}}$ from $\varphi^{\mathcal{M}}$? (*Inverse reachability problem*)

Backward iteration marks all points in $(\mathbf{E}\,\mathbf{F}^{+}\,\varphi)^{\mathcal{M}}$:

- Initially mark all points for which some direct successor is in $\varphi^{\mathcal{M}}$.
- Repeatedly add all points which have some marked successor.



Comparison

- CTL model checking
  - uses sets, breath-first search
  - can be directly implemented with BDDs
  - systems: e.g. nuSMV
- LTL model checking
  - depth-first search, enumerates states
  - implementation allows state-space hashing, partial order reduction etc.
  - systems: e.g. SPIN