Deciding Full Branching Time Logic by Program Transformation

Alberto Pettorossi (Univ. Tor Vergata, Rome, Italy),

Maurizio Proietti (IASI-CNR, Rome, Italy),

Valerio Senni (Univ. Tor Vergata, Rome, Italy)

Pescara, Italy ***April 16, 2010***

Our Goal

... is to establish by program transformation

the correctness of

finite state concurrent systems with infinite behaviour

(reactive systems) such as:

- communication protocols
- security protocols
- hardware controllers
- …

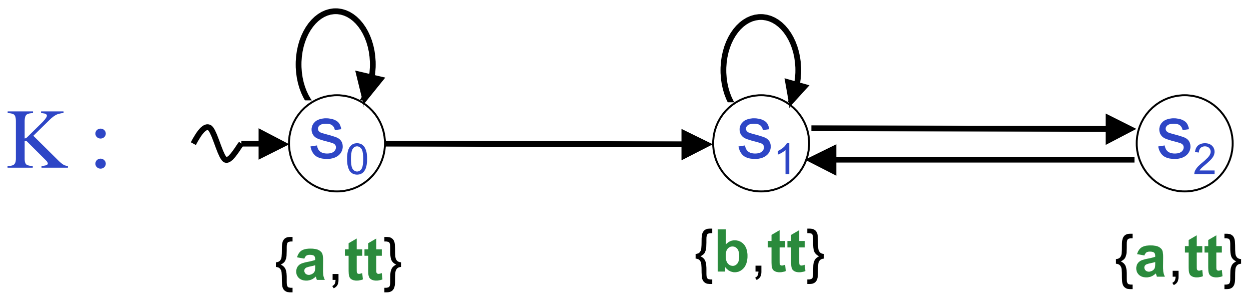
Related Work

- Proving first order formulas via unfold/fold rules
  [Kott 1982, P.P. 1999, Roychoudhury et al. 1999, P.P. 2000, ...]

- Verifying temporal properties of infinite state systems
  via specialization and unfold/fold rules
  [Leuschel 1999, 2000, Roychoudhury et al. 2000,
  Fioravanti et al. 2001, ...]

Concurrent Systems and Properties

Concurrent Systems as Kripke Structures



*Properties as Formulas in CTL**

Elem = {a,b,tt}

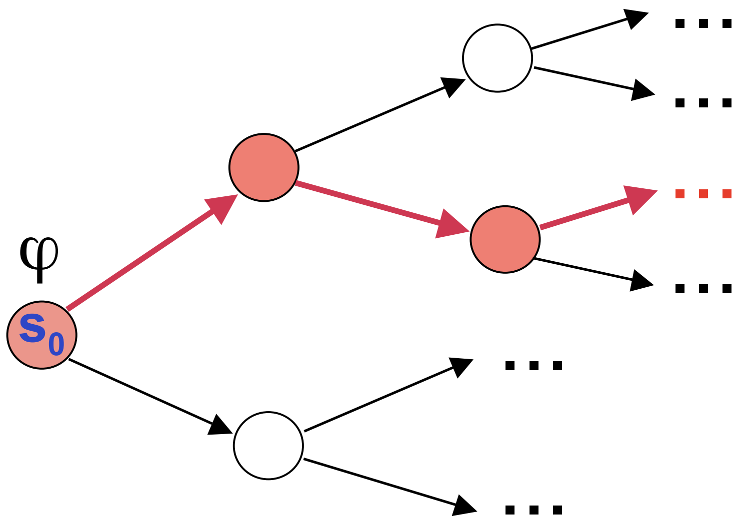
$$\varphi = E\,(a\;U\;\neg E\,(tt\;U\,\neg(tt\;U\;\neg b)))$$

LTL (linear-time temporal logic) $\subset$ CTL*

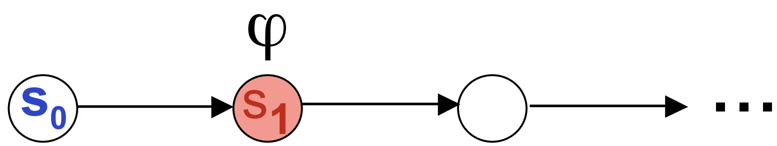
CTL (computational tree logic) $\subset$ CTL*

exists a path - next - until

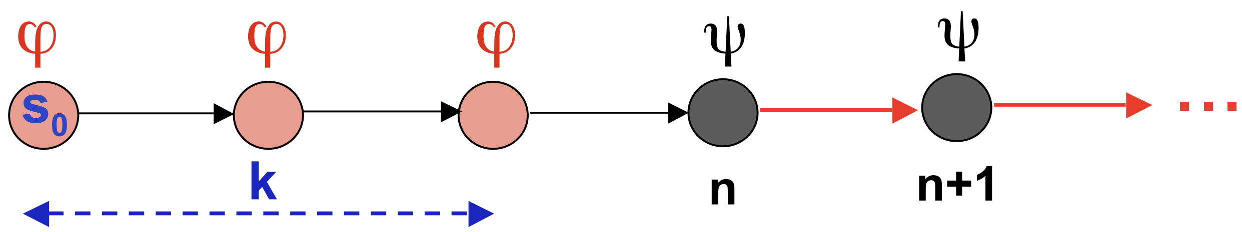
E φ :
(**e**xists a path)



X φ :
(ne**x**t)



φ **U** ψ :
(**u**ntil)



A Kripke Structure

$$K = \langle \Sigma, s_{in}, \rho, \lambda \rangle$$

Σ = finite set of states

s_{in} = initial state

ρ = **total** transition relation $\subseteq \Sigma \times \Sigma$

λ = labelling function: $\Sigma \rightarrow 2^{Elem}$

A **computation path** $\pi = [s_0, s_1, \ldots]$ is an **infinite list** of states.

*Formulas of CTL**

elementary formulas :

$$\mathbf{d} \in \text{Elem}$$

state formulas :

$$\varphi = \mathbf{d} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\,\psi$$

(**e**xists a path)

path formulas :

$$\psi = \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X}\,\psi \mid \psi\,\mathbf{U}\,\psi$$

(ne**x**t) (**u**ntil)

Semantics of CTL*

Let K be a Kripke structure.

Let π be the infinite list $[s_0, s_1, \ldots, s_k, \ldots, s_n, \ldots]$ of states.

Let **d**, φ, ψ be formulas of CTL*.

$K, \pi \models \mathbf{d}$ iff $\mathbf{d} \in \lambda(s_0)$

$K, \pi \models \neg\varphi$ iff $K, \pi \models \varphi$ does not hold

$K, \pi \models \varphi \wedge \psi$ iff $K, \pi \models \varphi$ and $K, \pi \models \psi$

$K, \pi \models \mathbf{E}\, \varphi$ iff $\exists\, \pi' = [s_0, \ldots],\ K, \pi' \models \varphi$ (**e**xists a path)

$K, \pi \models \mathbf{X}\, \varphi$ iff $K, [s_1, \ldots] \models \varphi$ (ne**x**t)

$K, \pi \models \varphi\, \mathbf{U}\, \psi$ iff $\exists n \geq 0\ ((\forall k, 0 \leq k < n,\ K, [s_k, \ldots] \models \varphi)$ and $K, [s_n, \ldots] \models \psi)$ (**u**ntil)

CTL* *Model Checking*

Definition.

A **state formula** φ holds in the **Kripke structure** $\mathcal{K}$ with **initial state** s_{in}:

$$\mathcal{K} \models \varphi \quad \text{iff} \quad \exists\, \pi = [s_{in}, \ldots],\ \mathcal{K}, \pi \models \varphi$$

CTL* Model Checking:

given $\mathcal{K}$ and φ, verify whether or not $\mathcal{K} \models \varphi$ holds.

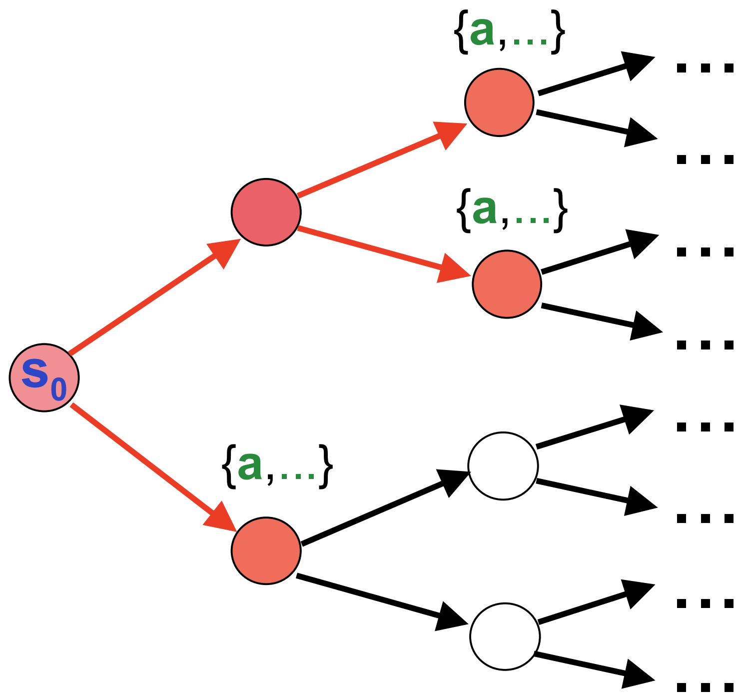
More Syntax

in the **F**uture : $\mathbf{F}\,\varphi = \mathbf{tt}\ \mathbf{U}\ \varphi$

Globally : $\mathbf{G}\,\varphi = \neg\mathbf{F}\neg\varphi$

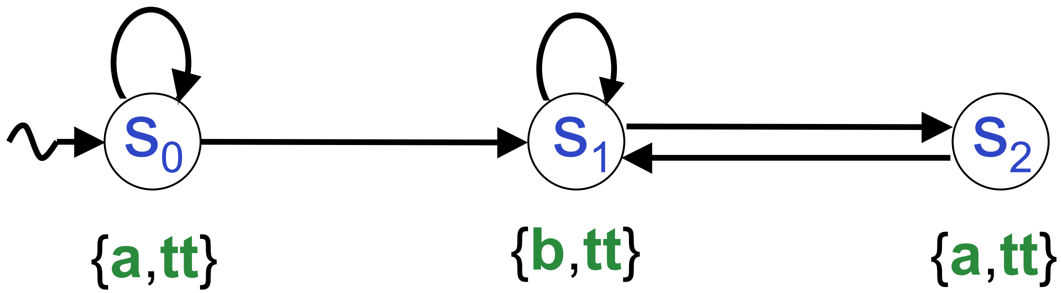
for**A**ll : $\mathbf{A}\,\varphi = \neg\mathbf{E}\neg\varphi$

$\mathcal{K}, [s_0, \ldots] \models$ **AFa** holds if



An Example

K :



φ = E (a U ¬E (tt U ¬(tt U b)))

= E (a U A G F b)

path witnesses: $s_0^+ (s_1^+ s_2)^\omega$

Thus, K ⊨ φ

Overview

- ✓ Modeling reactive systems and their properties via Kripke structures and CTL* formulas. $K \models \varphi$.

- Encoding CTL* formulas as ω-programs (programs on infinite lists)

- Transforming ω-programs into monadic ω-programs

- Proof system for monadic ω-programs

ω-programs

- ω-programs are a **typed**, **locally stratified logic programs** where [_|_] is interpreted as the constructor of **infinite lists**.

- Semantics of ω-programs = **perfect model** constructed over **infinite lists** (= least Herbrand model for definite programs).

$\Sigma = \{s_0, s_1\}$. L ranges over $(s_0+s_1)^\omega$.

- **P:** $p([s_0|L]) \leftarrow q(L)$ $p(L) \in M(P)$ iff $L \in s_0(s_0+s_1)^\omega$
  $q(L) \leftarrow$

- **P:** $p(L) \leftarrow \neg\, q(L)$ $p(L) \in M(P)$ iff $L \in s_0^\omega$
  $q([s_0|L]) \leftarrow q(L)$ $q(L) \in M(P)$ iff $L \in s_0^* s_1(s_0+s_1)^\omega$
  $q([s_1|L]) \leftarrow$

Negation gives extra expressivity. For **P:** $p([s_0|L]) \leftarrow p(L)$, $M(P) = \varnothing$.

Encoding the Satisfaction Relation $\models$ (1)

$P_{K,\varphi}$: prop $\leftarrow$ sat([s_0|X], $\lceil\varphi\rceil$)

sat([S|X], D) $\leftarrow$ elem(D,S)

sat(X, not F) $\leftarrow$ $\neg$ sat(F,S)

sat(X, and(F_1,F_2)) $\leftarrow$ sat(X,F_1) $\wedge$ sat(X,F_2)

sat([S|X], e(F)) $\leftarrow$ exists-sat(S,F)

exists-sat(S,F) $\leftarrow$ path([S|Y]) $\wedge$ sat([S|Y], F)

sat([S|X], x(F)) $\leftarrow$ sat(X,F)

sat(X, u(F_1,F_2)) $\leftarrow$ sat(X,F_2)

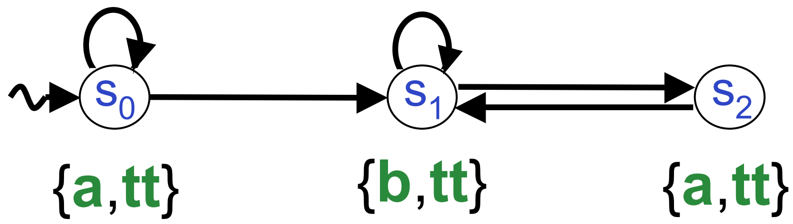
sat([S|X], u(F_1,F_2)) $\leftarrow$ sat([S|X], F_1) $\wedge$ sat(X, u(F_1,F_2))

path(X) $\leftarrow$ $\neg$ notpath(X)

notpath([S_1,S_2|X]) $\leftarrow$ $\neg$ transition(S_1,S_2)

notpath([S|X]) $\leftarrow$ notpath (X)

Encoding the Satisfaction Relation $\models$ (2)



elem(a, s_0) $\leftarrow$
elem(b, s_1) $\leftarrow$
elem(a, s_2) $\leftarrow$

elem(tt, X) $\leftarrow$

transition(s_0, s_0) $\leftarrow$
transition(s_0, s_1) $\leftarrow$
transition(s_1, s_1) $\leftarrow$
transition(s_1, s_2) $\leftarrow$
transition(s_2, s_1) $\leftarrow$

φ = E (a U ¬E (tt U¬(tt U b)))

$\lceil\varphi\rceil$ = e(u(a,not(e(u(tt,not(u(tt,b)))))))

Correctness of the Encoding

> *Theorem* 1. $K \models \varphi$ iff $\mathbf{M(P_{K,\varphi}) \models prop}$

How to check whether or not $\mathbf{M(P_{K,\varphi}) \models prop}$?

- *Top-down* evaluation of the query **prop** does not terminate.
- *Bottom-up* construction of $\mathbf{M(P_{K,\varphi})}$ does not terminate, because of infinite lists.

Overview

✓• Modeling reactive systems and their properties
via Kripke structures and CTL* formulas. $\mathcal{K} \models \varphi$.

✓• Encoding CTL* formulas as ω-programs (programs on infinite lists)

- 1. Transforming ω-programs into monadic ω-programs
- 2. Decision algorithm for monadic ω-programs

Monadic ω-programs

A monadic ω-program is a stratified set of monadic ω-clauses of the form:

$$\mathbf{p_0([s|X_0])} \leftarrow \boxed{\mathbf{p_1(X_1)} \wedge \ldots \wedge \mathbf{p_k(X_k)}} \wedge \boxed{\neg\, \mathbf{p_{k+1}(X_{k+1})} \wedge \ldots \wedge \neg\, \mathbf{p_m(X_m)}}$$

where:

s is a constant of type **`state`**,

$\mathbf{X_0}, \mathbf{X_1}, \ldots, \mathbf{X_k}, \mathbf{X_{k+1}}, \ldots, \mathbf{X_m}$ are variables of type **`infinite-list`**, and

let a clause be $A_0 \leftarrow \ldots \wedge L_i \wedge \ldots$

there exists a *level mapping* h such that for i=**1**,...,**m**,

if vars(L_i) $\not\subseteq$ vars(A_0) **and** $1 \leq i \leq k$ **then** $h(L_i) < h(A_0)$ **else** $h(L_i) \leq h(A_0)$

(Recall that: If L_i is $\mathbf{p_i(X_i)}$ then $h(L_i) = h(\mathbf{p_i})$. If L_i is $\neg\, \mathbf{p_i(X_i)}$ then $h(L_i) = h(\mathbf{p_i}) + 1$.)

Some of the predicates $\mathbf{p_i}$'s may be nullary, and they may be the same.

Some of the variables may be the same.

From $\mathbf{P}_{K,\varphi}$ *to a Monadic* ω*-program* **T**

By applying the strategy:

$$\Big(\textbf{instantiate} \,;\ \boxed{\textbf{unfold}^{*}_{pos\,\&\,neg}} \,;\ \textbf{subsume} \,;\ \boxed{\textbf{define-fold}^{*}_{pos\,\&\,neg}}\Big)^{*}$$

from $\mathbf{P}_{K,\varphi}$ we get a monadic ω-program **T**.

Theorem 2. $\mathbf{M}(\mathbf{P}_{K,\varphi}) \models \mathbf{prop}$ iff $\mathbf{M}(\mathbf{T}) \models \mathbf{prop}$

Proof. Extension of the rules in [Seki 91].

Theorems

$\mathbf{prop} \leftarrow \mathbf{sat}([s_0|X], \lceil\varphi\rceil)$

Theorem 1. $\mathrm{K} \models \varphi$ iff $\mathbf{M}(\mathbf{P}_{K,\varphi}) \models \mathbf{prop}$

Theorem 2. $\mathbf{M}(\mathbf{P}_{K,\varphi}) \models \mathbf{prop}$ iff $\mathbf{M}(\mathbf{T}) \models \mathbf{prop}$

The Monadic ω-program T

T :

$prop \leftarrow \neg p_1(X) \wedge p_2(X)$
$prop \leftarrow \neg p_1(X) \wedge \neg p_3$

$p_1([s_0|X]) \leftarrow p_1(X)$
$p_1([s_1|X]) \leftarrow p_4(X)$
$p_1([s_2|X]) \leftarrow$

$p_2([s_0|X]) \leftarrow \neg p_3$
$p_2([s_0|X]) \leftarrow p_2(X)$
$p_2([s_1|X]) \leftarrow \neg p_5$
$p_2([s_2|X]) \leftarrow \neg p_6$
$p_2([s_2|X]) \leftarrow p_2(X)$

$p_3 \leftarrow \neg p_1(X) \wedge \neg p_7(X)$
$p_3 \leftarrow \neg p_1(X) \wedge p_8(X)$

$p_4([s_0|X]) \leftarrow$
$p_4([s_1|X]) \leftarrow p_4(X)$
$p_4([s_2|X]) \leftarrow p_9(X)$

$p_5 \leftarrow \neg p_4(X) \wedge p_8(X)$

$p_6 \leftarrow \neg p_9(X) \wedge \neg p_7(X)$
$p_6 \leftarrow \neg p_9(X) \wedge p_8(X)$

$p_7([s_0|X]) \leftarrow p_7(X)$
$p_7([s_1|X]) \leftarrow$
$p_7([s_2|X]) \leftarrow p_7(X)$

$p_8([s_0|X]) \leftarrow \neg p_7(X)$
$p_8([s_0|X]) \leftarrow p_8(X)$
$p_8([s_1|X]) \leftarrow p_8(X)$
$p_8([s_2|X]) \leftarrow \neg p_7(X)$
$p_8([s_2|X]) \leftarrow p_8(X)$

$p_9([s_0|X]) \leftarrow$
$p_9([s_1|X]) \leftarrow p_4(X)$
$p_9([s_2|X]) \leftarrow$

Completeness of the Algorithm

Let **T** be a monadic ω-program.

Let **F** be a formula in $\mathcal{F}$.

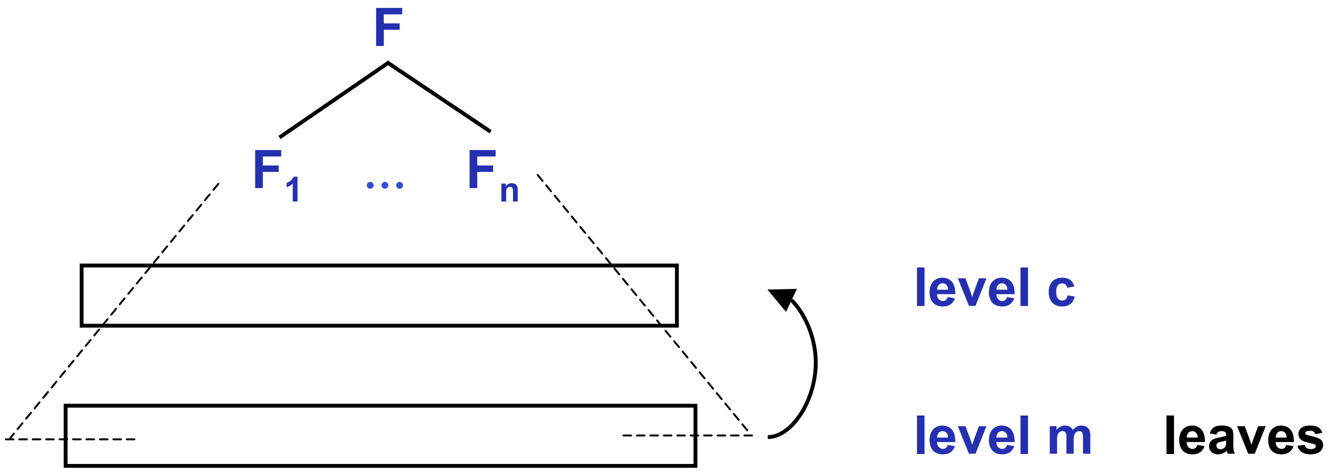
Theorem 3. **M(T)**$\models$**F** iff **F** has a **proof**.

- **Derivation Tree: - w.r.t. a program T**
  - an AND-tree
  - constructed level-by-level.

- **Proof and Refutation.**

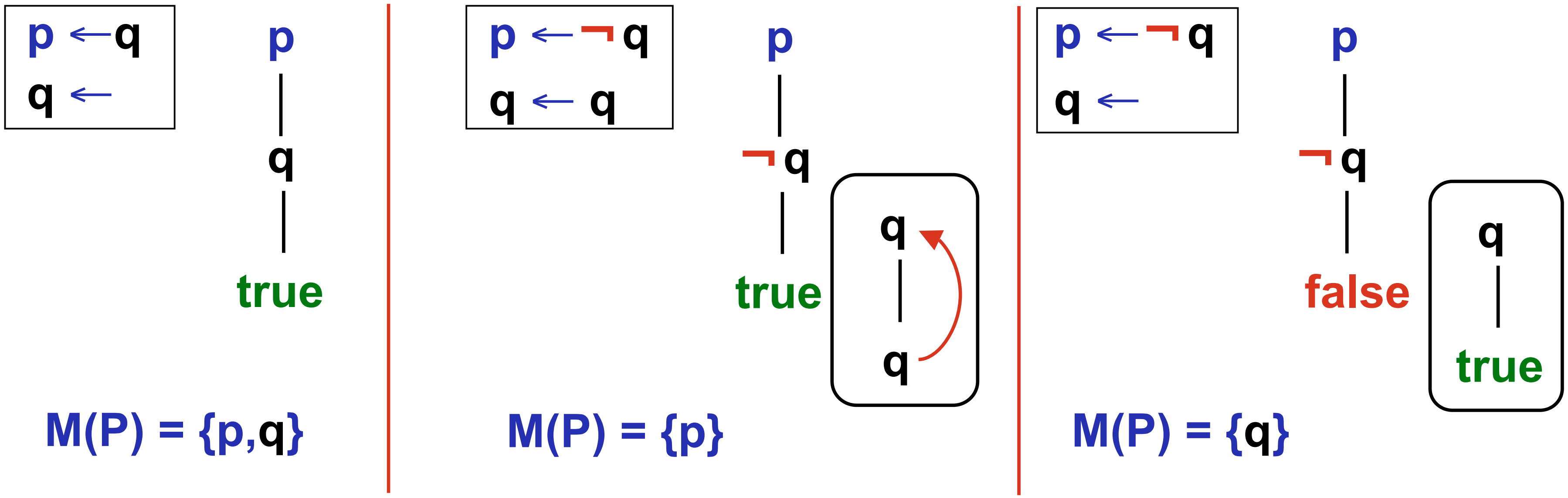
Derivation Tree: Basic Ideas

- **Finiteness.**



Every literal at level m also occurs at level c.

- **Positive loops.**



Derivation Tree (1)

true | false

monadic literals: $M = q(X) \mid \neg q(X)$

literals: $M \mid p \mid \neg p$

$F \in \mathcal{F}$ $\quad \mathcal{F} = p \mid \exists X\,(M_1 \wedge \ldots \wedge M_n)$

$L \in \mathcal{L}$ $\quad \mathcal{L} = M \mid \mathcal{F} \mid \neg \mathcal{F}$

complement: $\overline{F} = \neg F$ (with cancellation of $\neg\neg$)

Derivation Tree of $\mathbf{F} \in \mathcal{F}$ (2)

1. Explicit existential quantifiers

$$q([s|X]) \leftarrow \ldots \wedge \quad q_1(Y) \wedge q_2(Y) \wedge \ldots$$

$$\downarrow$$

$$q([s|X]) \leftarrow \ldots \wedge \; \exists Y(q_1(Y) \wedge q_2(Y)) \wedge \ldots$$

Derivation Tree of F $\in \mathcal{F}$ (3)

2. AND-tree, constructed level-by level.

- **The root is F.**
  If the root F is $\exists X\ (M_1 \wedge \ldots \wedge M_n)$ expand it:

  F

  M_1 ... M_n

- **Stop if: true, false, $\exists X\ (M_1 \wedge \ldots \wedge M_n)$, $\neg \exists X\ (M_1 \wedge \ldots \wedge M_n)$,**
  literals at level d $\subseteq$ literals at level c, with c<d

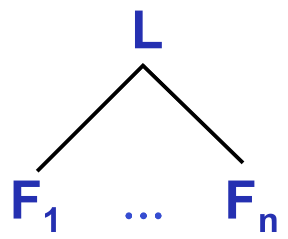
- **Nondeterministically expand every literal L at lowest level.**
  Choose a state s.

Derivation Tree of F $\in \mathcal{F}$ (4)

- **positive literal L :**

Choose a clause for L :

$$q([s|X]) \leftarrow F_1 \wedge \ldots \wedge F_n$$



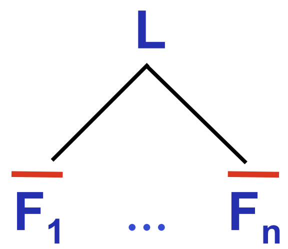
If n=0 : L — **true**

- **negative literal L :**

All clauses for L :

$$\begin{cases} q([s|X]) \leftarrow B_1 \\ \ldots \\ q([s|X]) \leftarrow B_n \end{cases}$$

Choose $F_1 \in B_1, \ldots, F_n \in B_n$ **(all** F_n**'s in** $\mathcal{L}$**) :**



If n=0 : L — **true**

If $\exists i\ B_i$**=true :** L — **false**

Proof and Refutation of $F \in \mathcal{F}$ *w.r.t.* T (5)

- A *proof* of $F \in \mathcal{F}$ is a derivation tree

  - with root F

  - every leaf is: true, p, $\neg$p, q(X), $\neg$q(X),

    $\exists X\ (M_1 \wedge \ldots \wedge M_n)$ which has a *proof* w.r.t. T

    $\neg\exists X\ (M_1 \wedge \ldots \wedge M_n)$ which has a *refutation* w.r.t. T

  - for every leaf at m with a positive literal L, $r^+(L, L)$ does *not* hold,

    where $r(L_c, L_m)$ holds iff

    a node N_c at c has label L_c, a node N_m at m has label L_m,

    and N_c is ancestor of N_m in T.

- $F \in \mathcal{F}$ has a *refutation* w.r.t. T iff F has no proof w.r.t. T.

Theorems

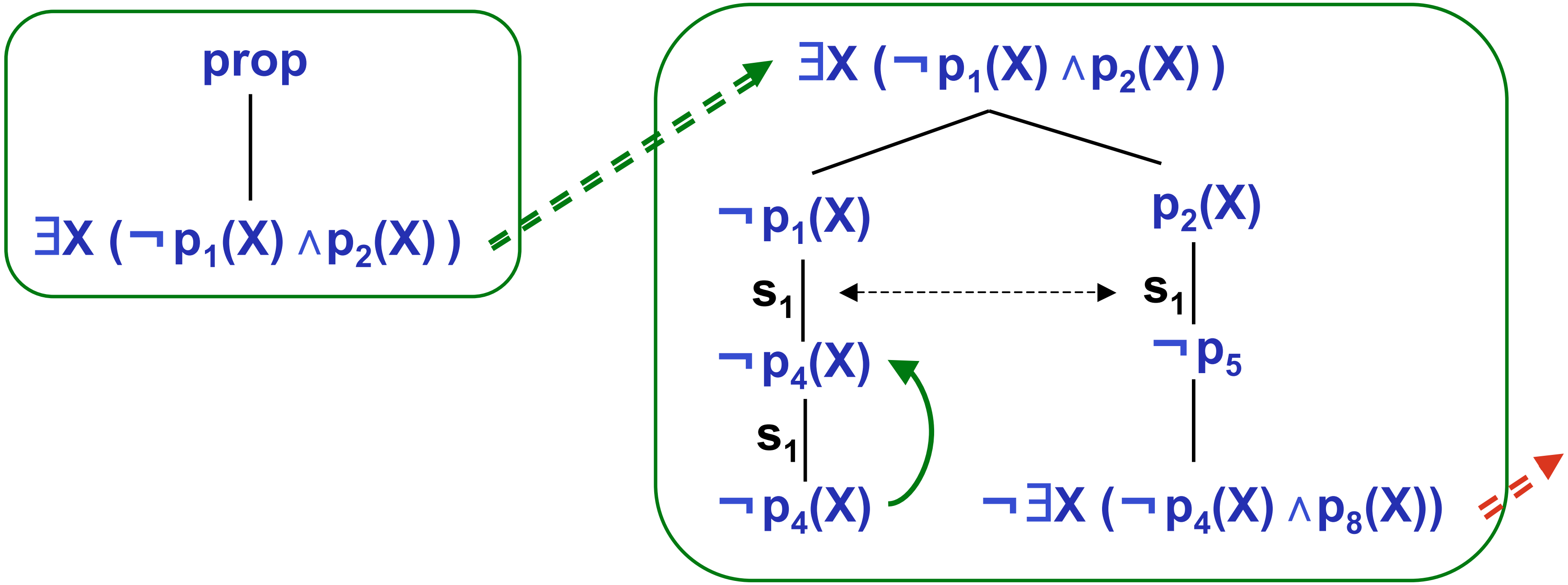
$\mathbf{prop} \leftarrow \mathbf{sat}([s_0|X], \lceil\varphi\rceil)$

Theorem 1. $K \models \varphi$ iff $\mathbf{M}(\mathbf{P}_{K,\varphi}) \models \mathbf{prop}$

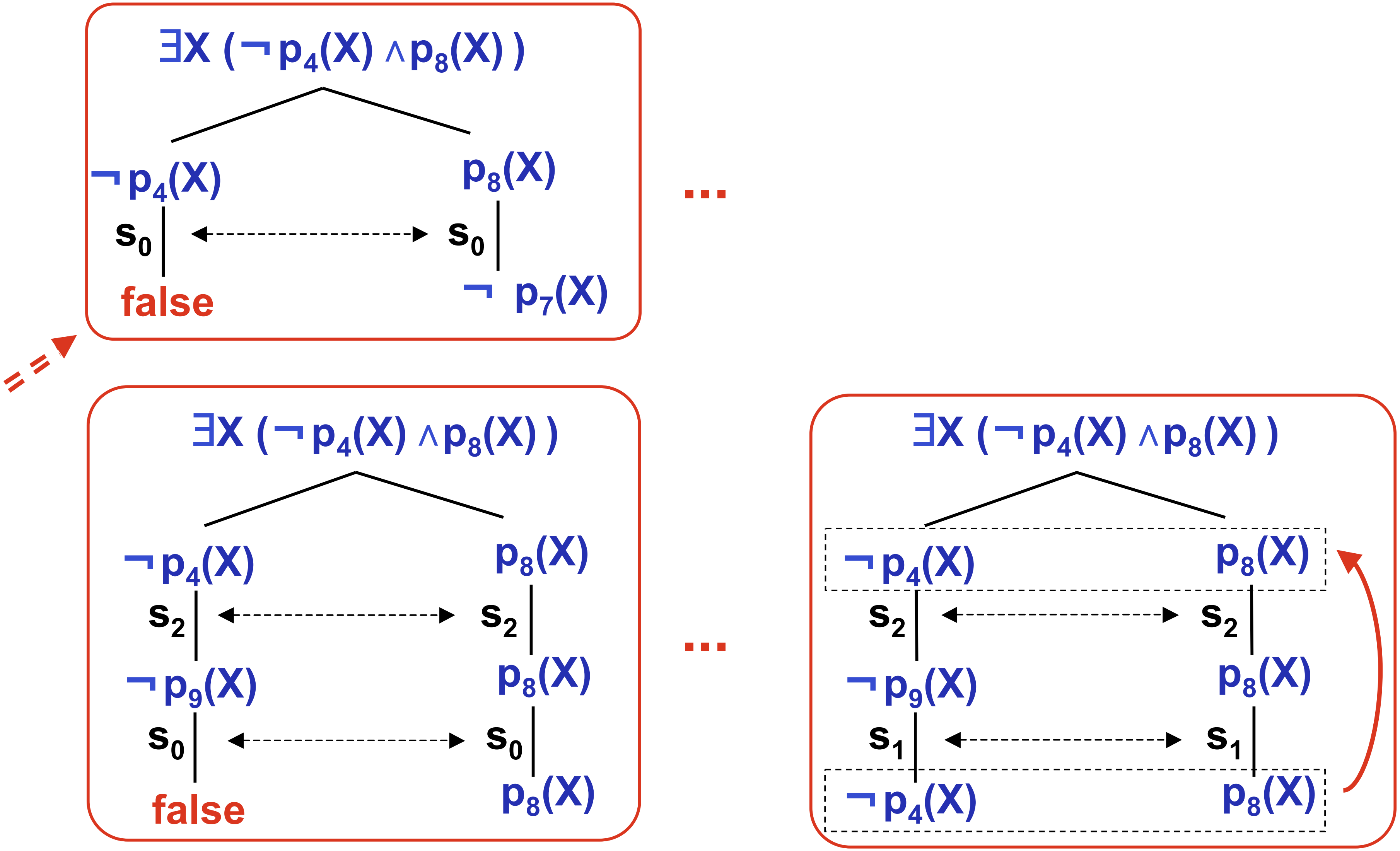
Theorem 2. $\mathbf{M}(\mathbf{P}_{K,\varphi}) \models \mathbf{prop}$ iff $\mathbf{M}(\mathbf{T}) \models \mathbf{prop}$

Theorem 3. $\mathbf{M}(\mathbf{T}) \models$ **prop** iff **F** has a **proof**.

Proof of prop *w.r.t.* T



Refutation of $\neg\exists X\,(\neg p_4(X) \wedge p_8(X))$ *w.r.t.* **T**



Future Work

- Use of constraints to avoid explicit state representation
- Infinite state model checking