2-Valued and 3-Valued Abstraction-Refinement Frameworks for Model Checking

Orna Grumberg
Technion
Haifa, Israel

MLQA workshop at FLOC 2010

Outline

- 2-valued Abstraction
  - CounterExample-Guided Abstraction-Refinement (CEGAR)
- 3-Valued Abstraction
  - Three-Valued abstraction-Refinement (TVAR)
  - Application

Main limitation of Model Checking

The state explosion problem:

Model checking is efficient in time but suffers from high space requirements:

The number of states in the system model grows exponentially with

- the number of variables
- the number of components in the system

Solutions to the state explosion problem

Small models replace the full, concrete model:

- Abstraction
- Compositional verification
- Partial order reduction
- Symmetry

Branching-time Temporal Logics

CTL, CTL*, μ-calculus

Can characterize properties referring to

- All behaviors
- Some behavior
- Their combination

ACTL / ACTL* / Aμ-calculus (also LTL)

The **universal** fragments of the logics, with can characterize only all behaviors

2-valued CounterExample-Guided Abstraction Refinement (CEGAR)

for Universal temporal logics

[CGJLV00]

Abstraction preserving Aμ-calculus

Existential Abstraction:

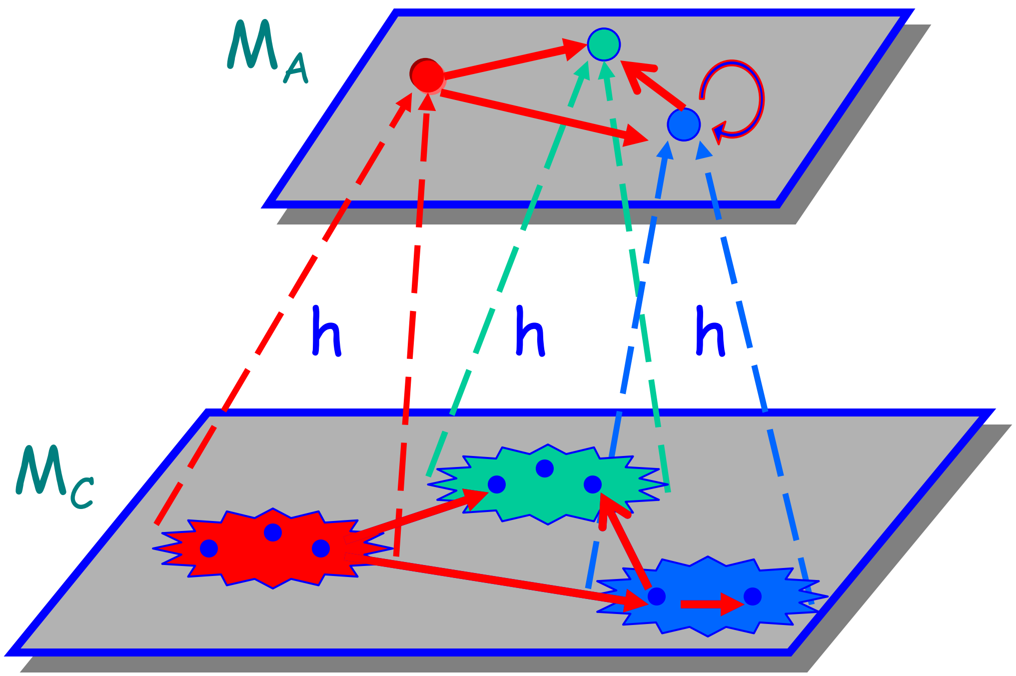
The abstract model is an **over-approximation** of the concrete model:

- The abstract model has **more behaviors**
- But no concrete behavior is lost

- Every ACTL/ACTL*/Aμ-calculus property true in the abstract model is also true in the concrete model

Existential Abstraction

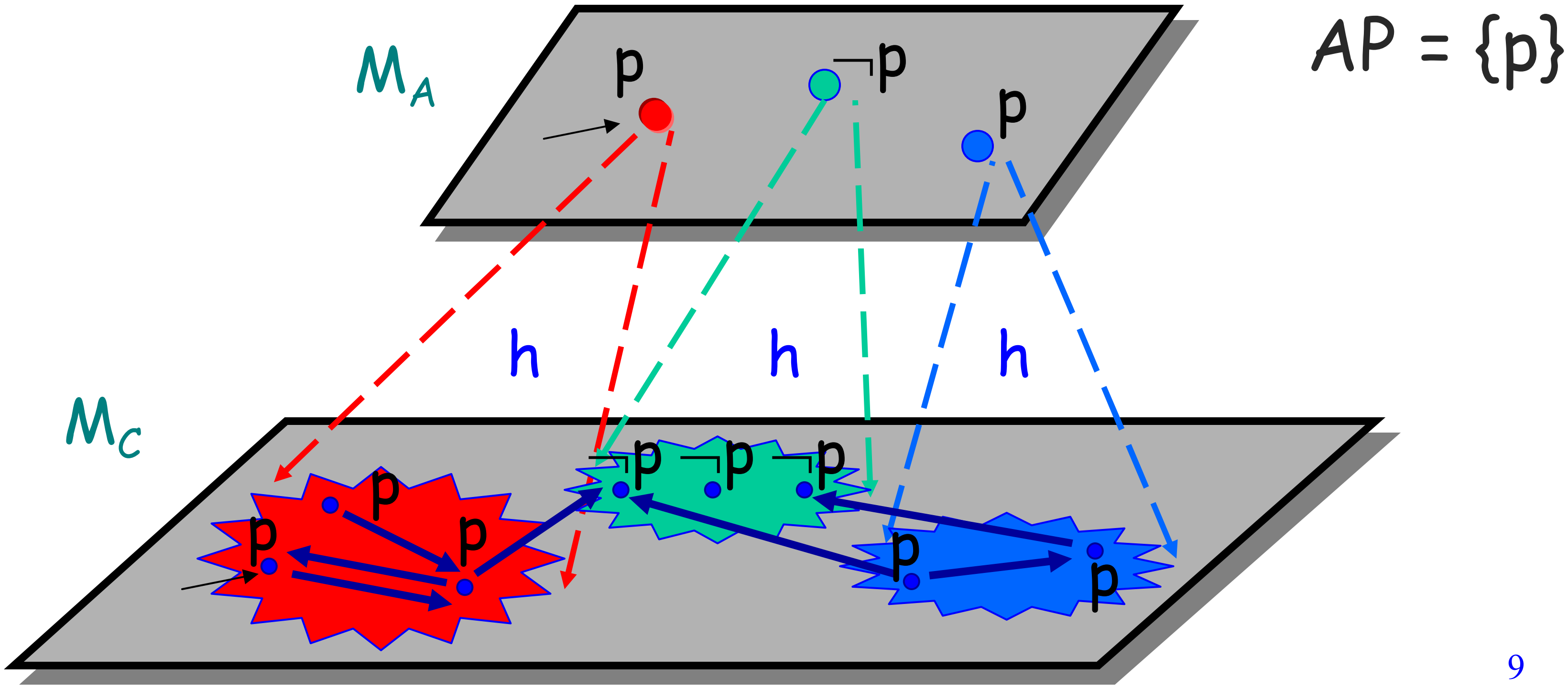
Given an abstraction function $h : S \rightarrow S_A$, the concrete states are grouped and mapped into abstract states :



$M_C \leq M_A$

Existential Abstraction (cont.)

Given an abstraction function $h : S \rightarrow S_A$, the concrete states are grouped and mapped into abstract states :



Labeling of abstract states

The abstraction function $h : S \rightarrow S_A$ is chosen so that:

If $h(s) = h(t) = s_A$ then $L(s) = L(t)$

- $L_A(s_A) = L(s)$

Widely used Abstractions (S_A, h)

- For Hardware:
  Localization reduction: each variable either keeps its concrete behavior or is fully abstracted (has free behavior) **[Kurshan94]**

- For Software:
  Predicate abstraction: concrete states are grouped together according to the set of predicates they satisfy **[GS97,SS99]**

They are determined based on the program's control flow and the checked property

Logic Preservation Theorem

- **Theorem** $M_C \leq M_A$, therefore for every Aμ-calculus formula φ,

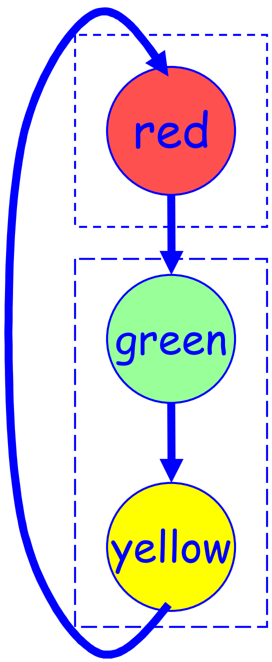
$$M_A \models \varphi \Rightarrow M_C \models \varphi$$

- However, the reverse may not be valid.

Traffic Light Example

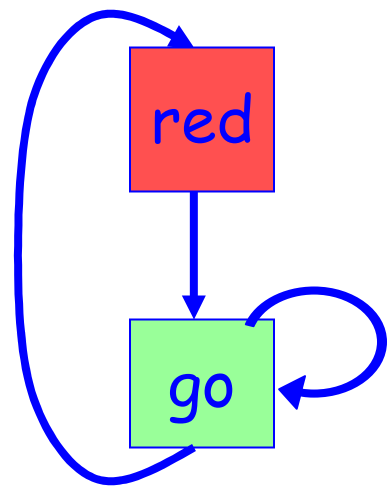
Property:
φ = **AG AF** ¬ (state=red)

Abstraction function h maps green, yellow to go.



M_C

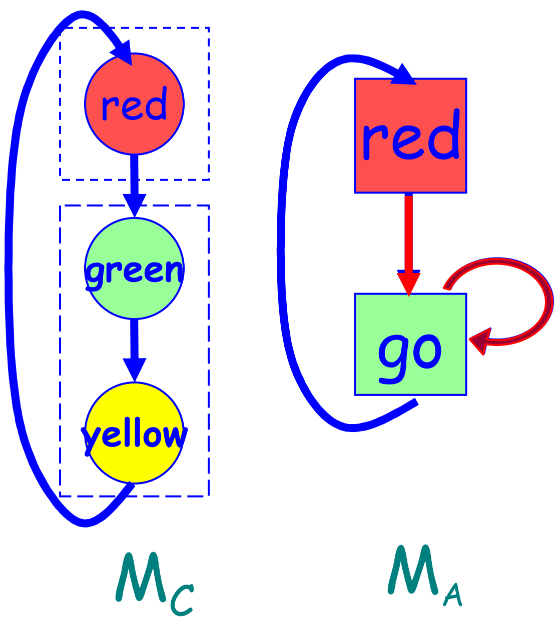
$M_C \models \varphi \Leftarrow \mathbf{M_A} \models \varphi$



M_A

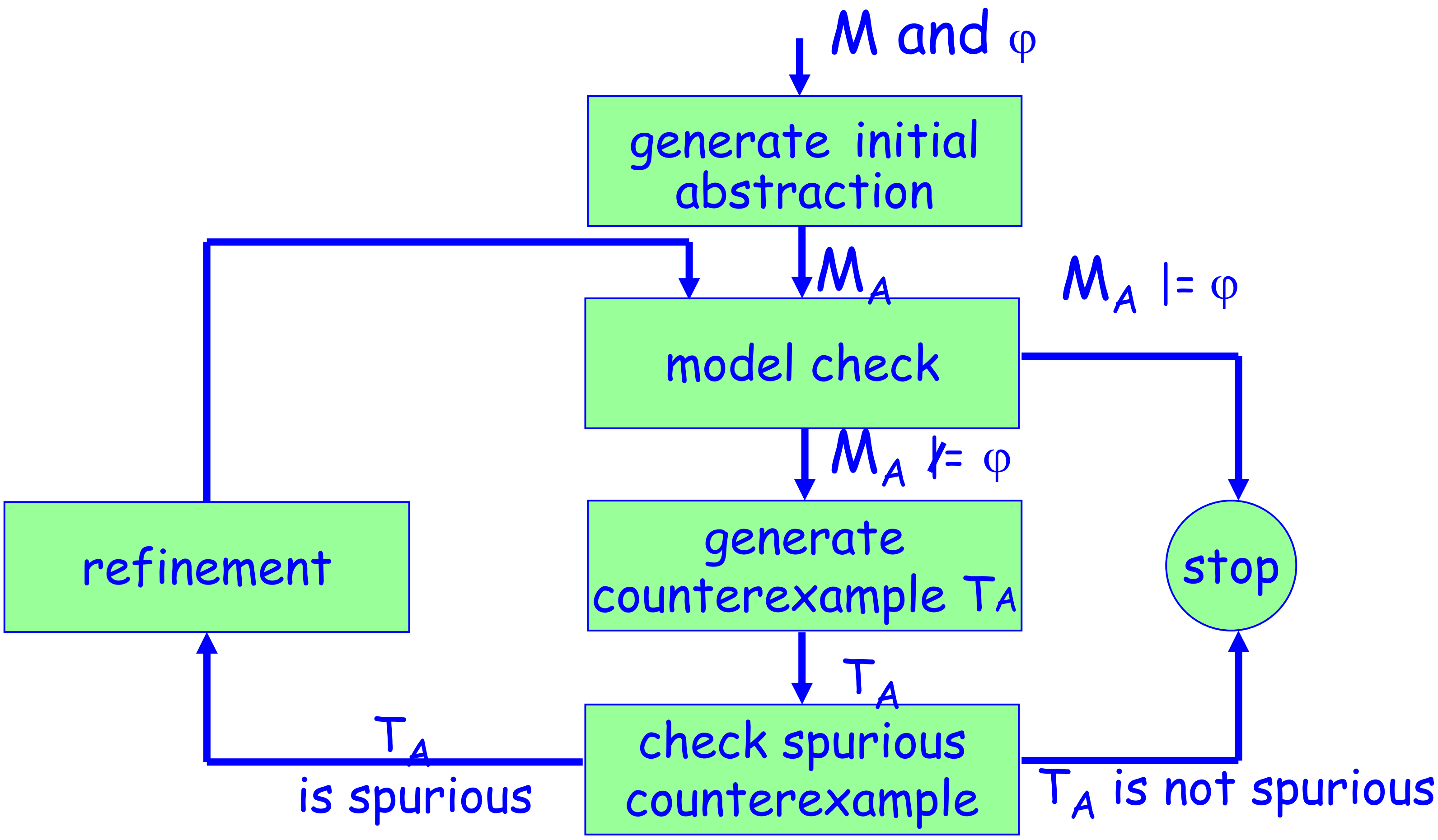
Traffic Light Example (Cont)

If the abstract model invalidates a specification,
the actual model may still satisfy the specification.



- Property:
  φ = **AG AF** (state=red)
- $M_C \models \varphi$ but $M_A \not\models \varphi$
- Spurious Counterexample:
  ⟨red,go,go, ...⟩

The CEGAR Methodology



Generating the Initial Abstraction

- If we use predicate abstraction then predicates are extracted from the program's control flow and the checked property
- If we use localization reduction then the unabstracted variables are those appearing in the predicates above

Counterexamples

- For AGp it is a finite path to a state satisfying $\neg p$
- For AFp it is an infinite path represented by a lasso (finite path+loop), where all states satisfy $\neg p$

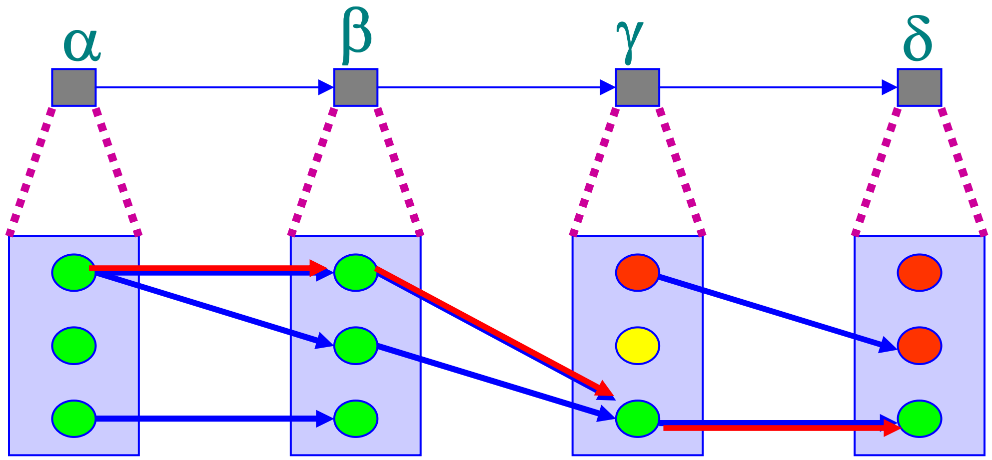
Path Counterexample

Assume that we have four abstract states

$\{1,2,3\} \leftrightarrow \alpha$ $\{4,5,6\} \leftrightarrow \beta$

$\{7,8,9\} \leftrightarrow \gamma$ $\{10,11,12\} \leftrightarrow \delta$

Abstract counterexample $T_A = \langle \alpha, \beta, \gamma, \delta \rangle$



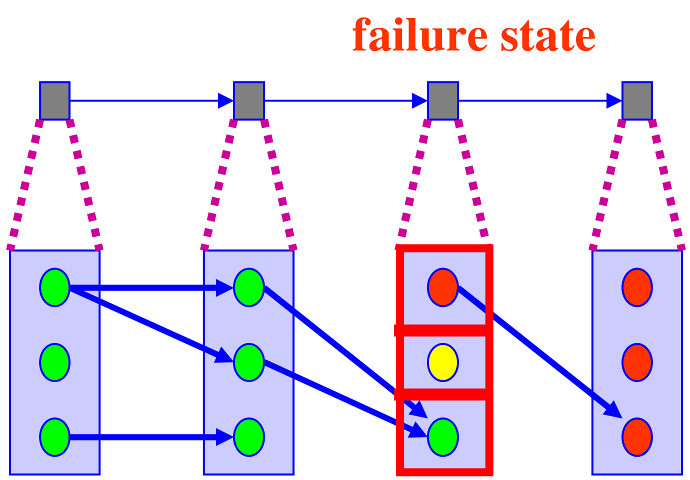
T_A is not spurious, therefore, $M \not\models \varphi$

Remark:

- δ and {10, 11, 12} are labeled the same
  - If δ satisfies $\neg p$ then 10, 11, 12 also satisfy $\neg p$

Therefore, (1, 4, 9, 12) is a concrete path counterexample

Spurious Path Counterexample



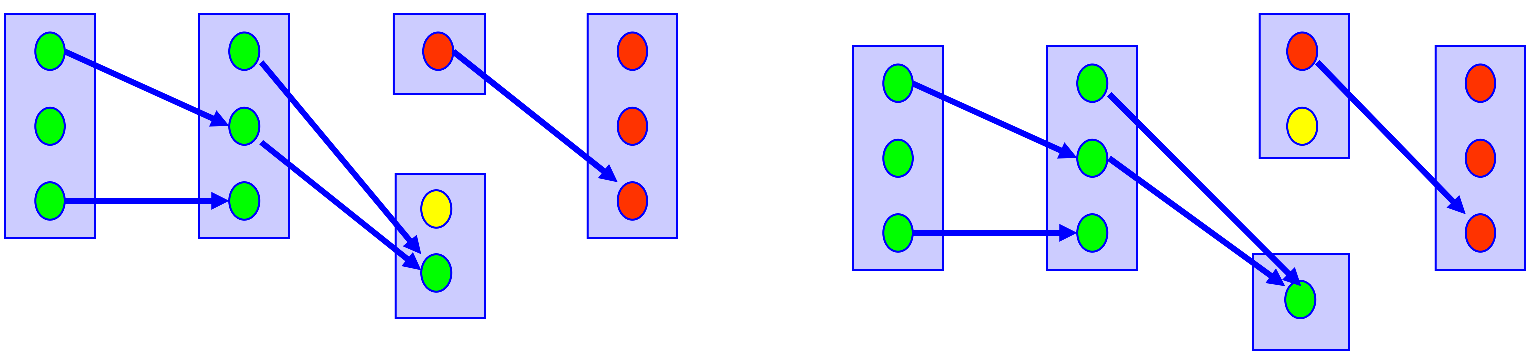
The concrete states mapped to the failure state are partitioned into **3** sets

states	dead-end	bad	irrelevant
reachable	yes	no	no
out edges	no	yes	no

T_A is spurious

Refining The Abstraction

- Goal : refine h so that the dead-end states and bad states do not belong to the same abstract state.
- For this example, two possible solutions.



Automatic Refinement

If the counterexample is spurious

- Find a **splitting criterion** that separates the **bad states** from the **dead-end states** in the failure state
- Apply the **splitting criterion** to splitting either only **the failure state** or **all states**
  - Faster convergence of the CEGAR loop
  - Faster growing abstract models

Checking for Spurious Path Counterexample

- $T = (a_1,\ldots a_n)$ - a path abstract counterexample

$h^{-1}(a) = \{ s \mid h(s) = a \}$

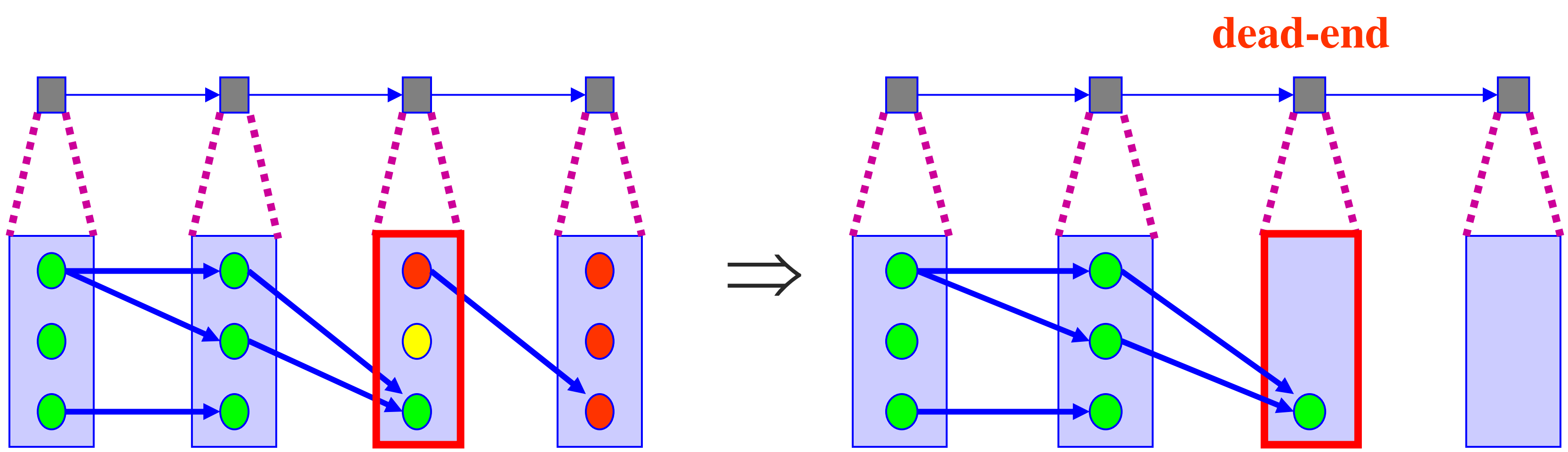
Checking for Spurious Path Counterexample (cont.)

The set of concrete counterexamples corresponding to $T = (a_1,\ldots a_n)$:

$$h^{-1}(T) = \{ (s_1,\ldots s_n) \mid \Lambda_i\, h(s_i)=a_i \wedge I(s_1) \wedge \Lambda_i R(s_i,s_{i+1}) \}$$

Is $h^{-1}(T)$ empty?

Checking for Spurious Path Counterexample



T_h is spurious

Refining the abstraction

- Refinement separates dead-end states from bad states, thus, eliminates the spurious transition from a_{i-1} to a_i

BDD-based computation of $h^{-1}(a_1), \ldots, h^{-1}(a_n)$

$S_1 = h^{-1}(a_1) \cap I$

For i = 2,…,n do

  S_i = successors(S_{i-1}) $\cap\ h^{-1}(a_i)$

  if $S_i = \varnothing$ then

    dead-end := S_{i-1}

    return(**i-1**, **dead-end**)

print ("counterexample exists")

Return $(S_1, \ldots, S_n)$

Computing a concrete counterexample from $S_1, \ldots, S_n$

t_n = choose (S_n)

For i = n-1 to 1

t_i = choose (predecessors(t_{i+1}) $\cap$ S_i)

Return (($t_1, \ldots, t_n$))

Implementing CEGAR

With **BDDs**:

- The concrete model M is finite but too big to directly apply model checking on
- R and I can be held as BDDs in memory, **R** possibly **partitioned**:
  $R(V, V') = \Lambda_i \, R_i \, (V, \, v_i')$
- **h** is held as BDD over concrete and abstract states

Can also be implemented with **SAT** or **Theorem Prover**

Three-Valued Abstraction Refinement (TVAR)

for Full μ-calculus

[SG03,GLLS05]

Goal: Logic preservation for full μ-calculus

Theorem

If M_A is an abstraction of M_C then for every μ-**calculus** formula φ,

$$\mathbf{M_A} \models \varphi \Rightarrow \mathbf{M_C} \models \varphi$$

$$\mathbf{M_A} \not\models \varphi \Rightarrow \mathbf{M_C} \not\models \varphi$$

- But sometimes $[\mathbf{M_A} \models \varphi] =$ **don't know**

Abstract Models for μ-calculus

- Two transition relations [LT88]

- Kripke Modal Transition System (KMTS)
- $M = (S, S_0, Rmust, Rmay, L)$
  - Rmust: an under-approximation
  - Rmay: an over-approximation
  - Rmust $\subseteq$ Rmay

Abstract Models for CTL* (cont.)

Labeling function :

- $L: S \rightarrow 2^{Literals}$
- $Literals = AP \cup \{\neg p \mid p \in AP\}$
- At most one of p and ¬p is in L(s).
  - Concrete: exactly one of p and ¬p is in L(s).
  - KMTS: possibly none of them is in L(s).

Abstract Models for CTL (cont.)

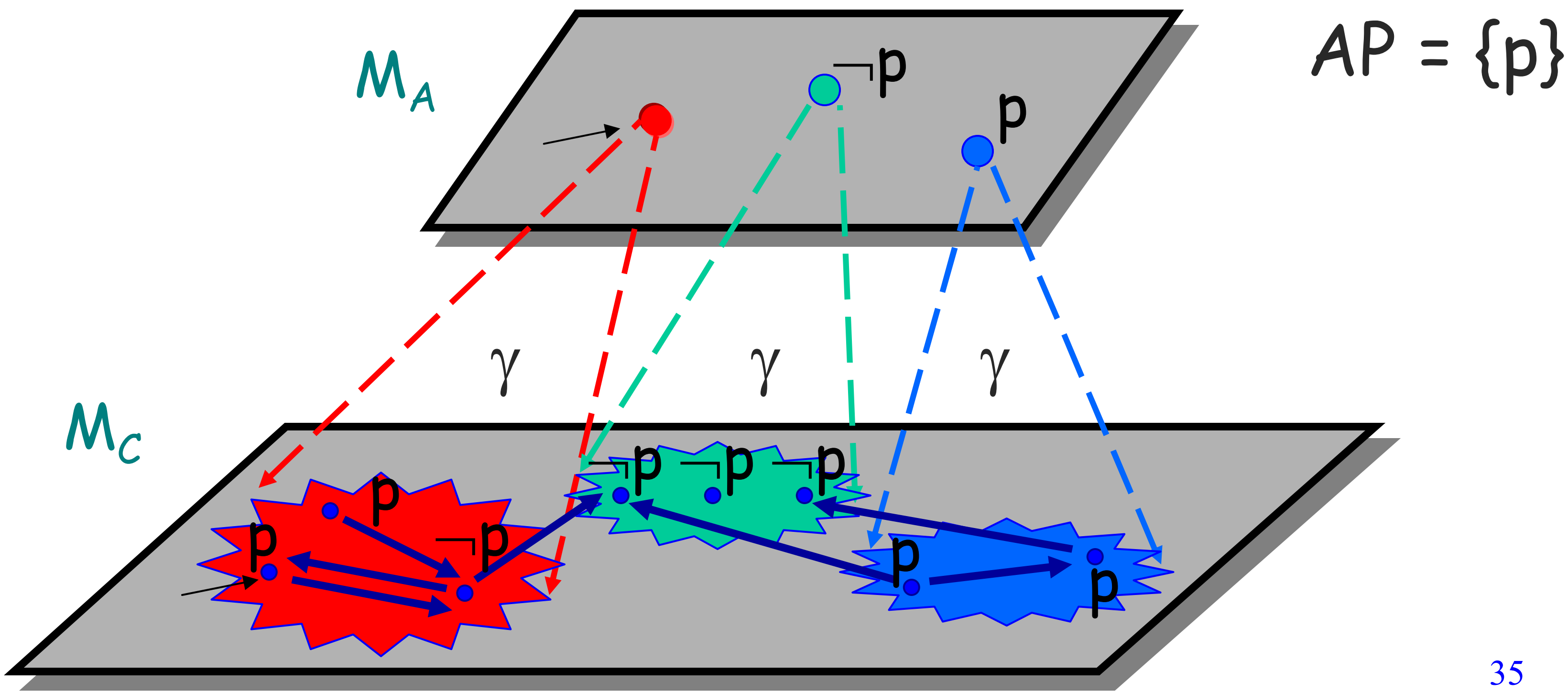
- Concrete Kripke structure
  $M_C = (S_C, S_{0C}, R_C, L_C)$
- Set of abstract states S_A
- Concretization function $\gamma : S_A \rightarrow 2^{S_C}$

⇩

- Abstract KMTS
  $M_A = (S_A, S_{0A}, Rmust, Rmay, L_A)$

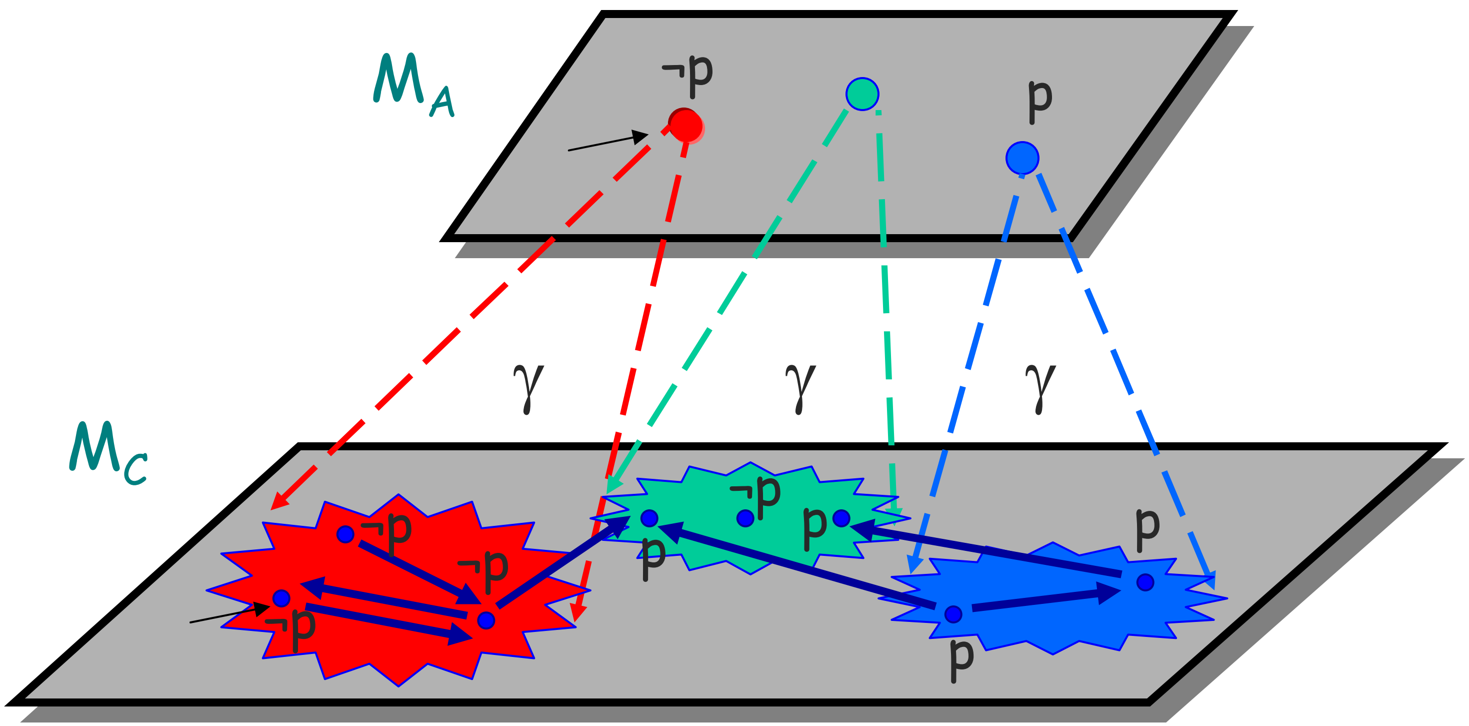
Abstract Models for CTL (cont.)

Given a concretization function $\gamma : S_A \rightarrow 2^{Sc}$, the concrete states are grouped and mapped into abstract states :



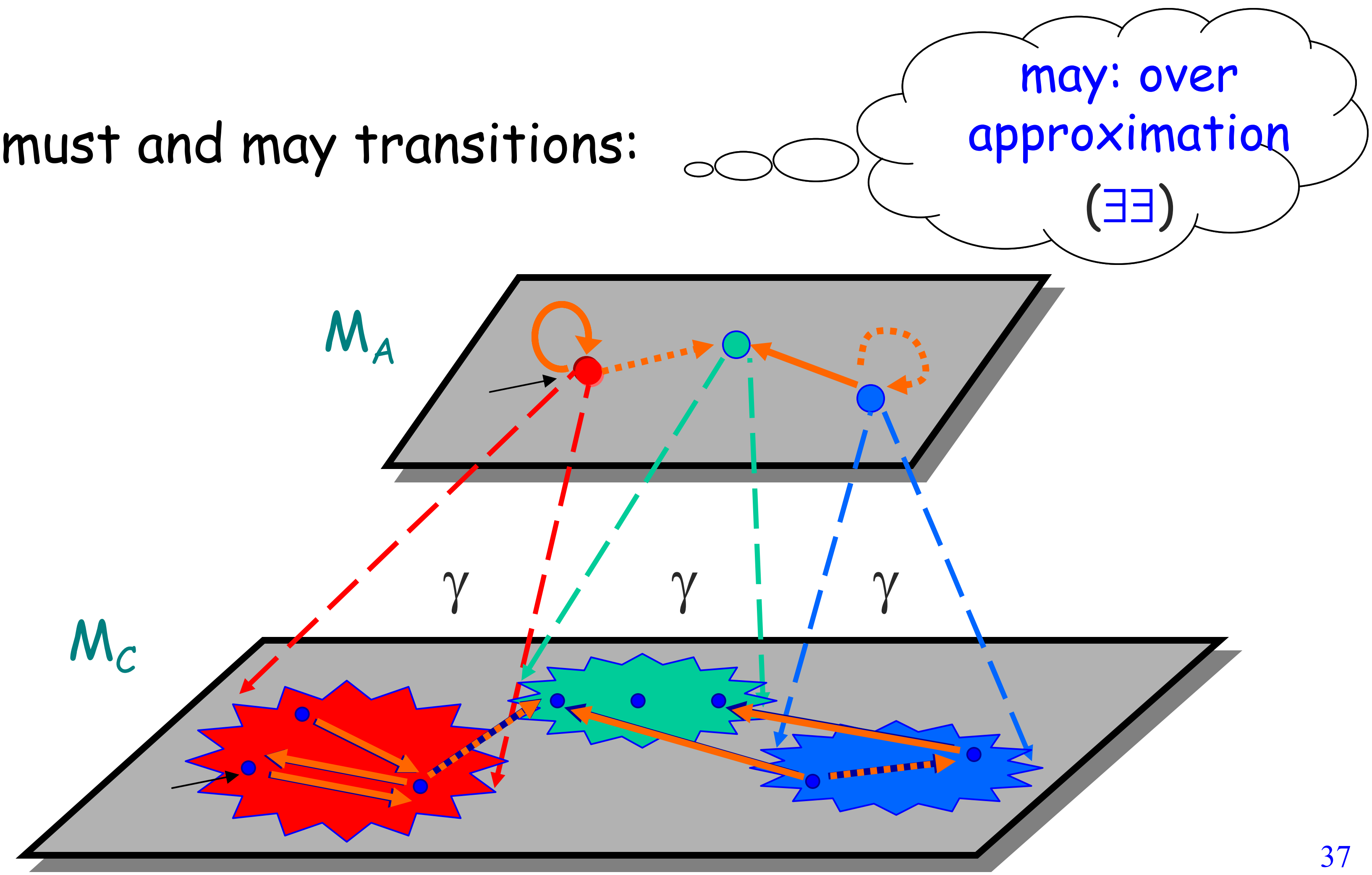
Abstract Models for μ-calculus (cont.)

Labeling of abstract states



Abstract Models for μ-calculus (cont.)

must and may transitions:



3-Valued Semantics

- Universal properties (Aψ) :
  - **Truth** is examined along *all* **may**-successors
  - **Falsity** is shown by a *single* **must**-successor

- Existential properties (Eψ) :
  - **Truth** is shown by a *single* **must**-successor
  - **Falsity** is examined along *all* **may**-successors

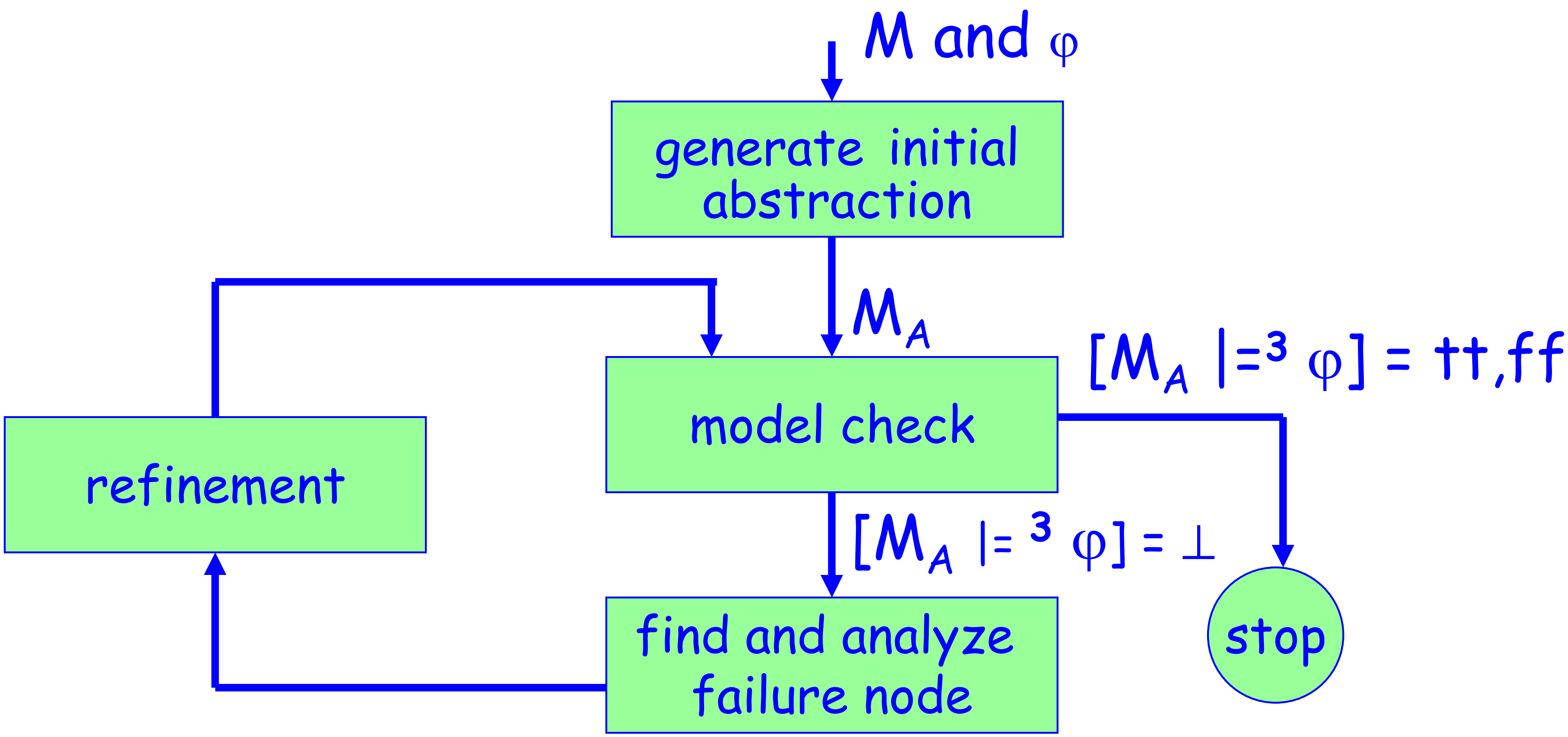
3-Valued Framework

tt, ff are definite

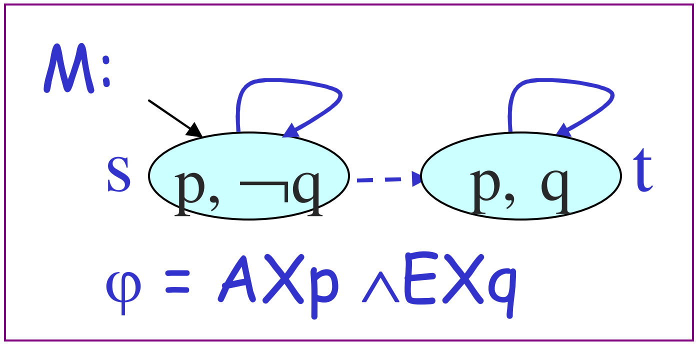
- Additional truth value: $\perp$ (indefinite)
- Abstraction preserves both **truth** and **falsity**
- (abstract) s_a represents (concrete) s_c:
  - φ is true in $s_a \Rightarrow \varphi$ is true in s_c
  - φ is false in $s_a \Rightarrow \varphi$ is false in s_c
  - φ is $\perp$ in $s_a \Rightarrow$ the value of φ in s_c is **unknown**

[BG99]

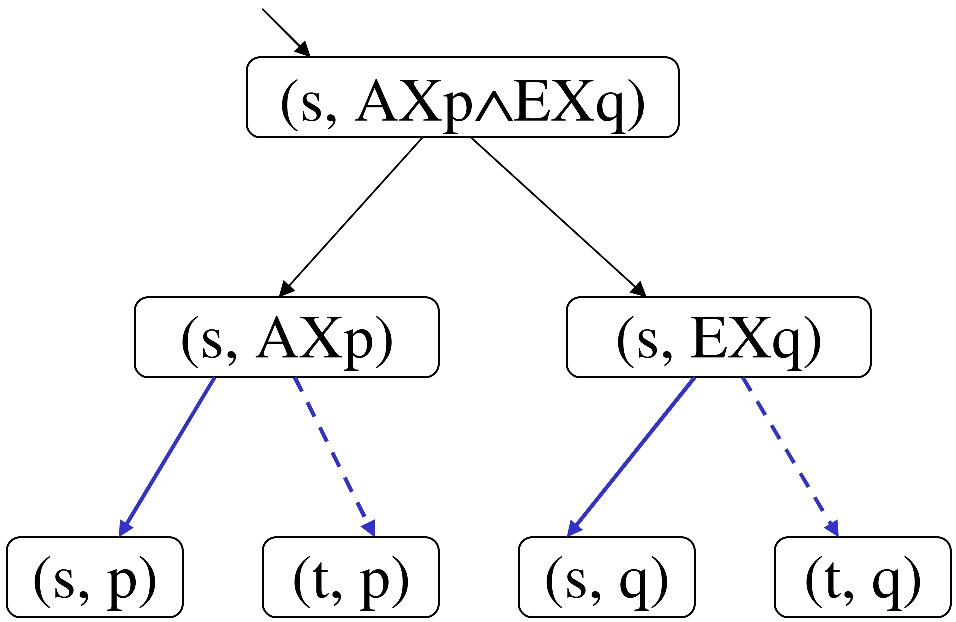
The TVAR Methodology

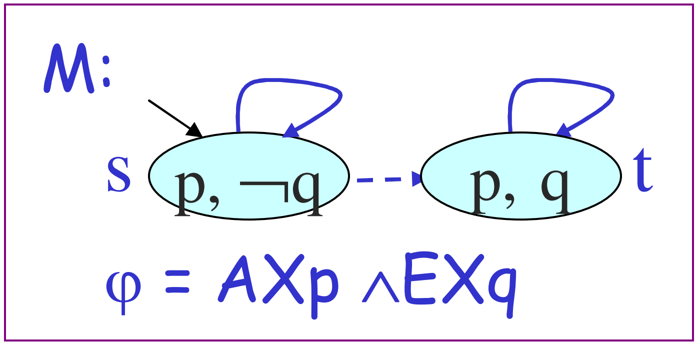


3-Valued Model Checking: Example



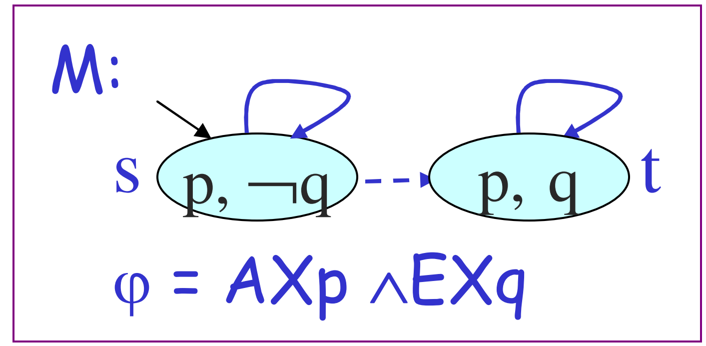
MC graph

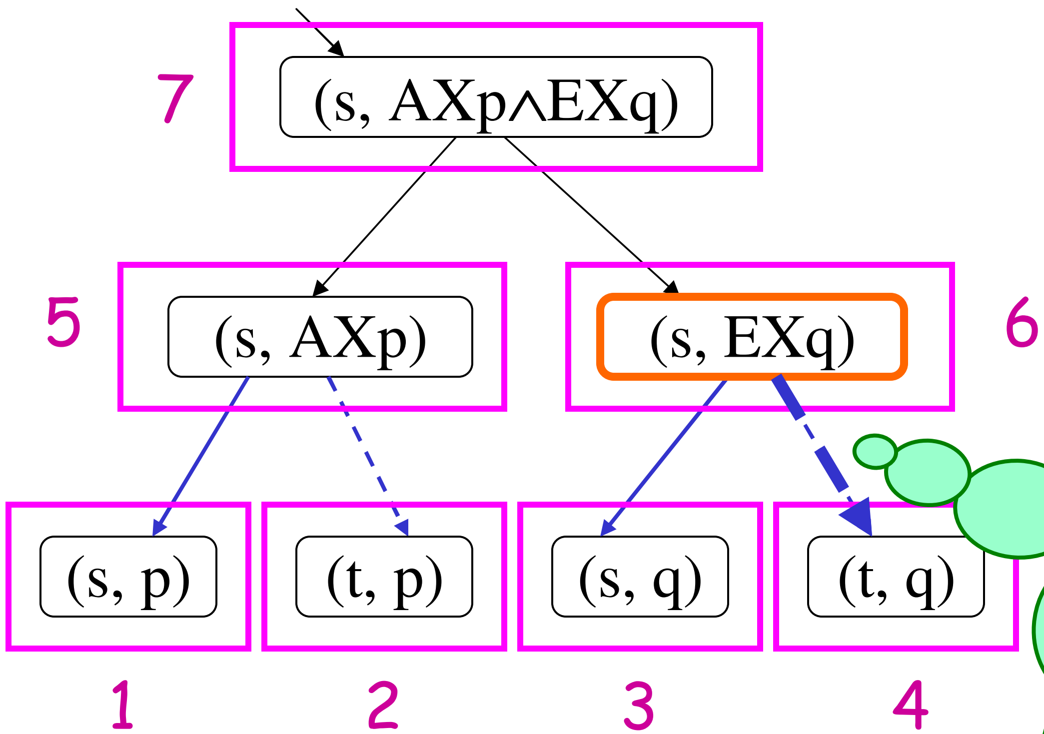




tt
ff
⊥

Coloring the MC graph





reason for unknown:
may-son
- not enough to verify
- prevents refutation

Abstraction-Refinement

- Traditional abstraction-refinement is designed for 2-valued abstractions:
  - True holds in the concrete model.
  - False may be a false alarm.

⇒ Refinement is needed when the result is false and is based on a counterexample analysis.

3-Valued Model Checking Results

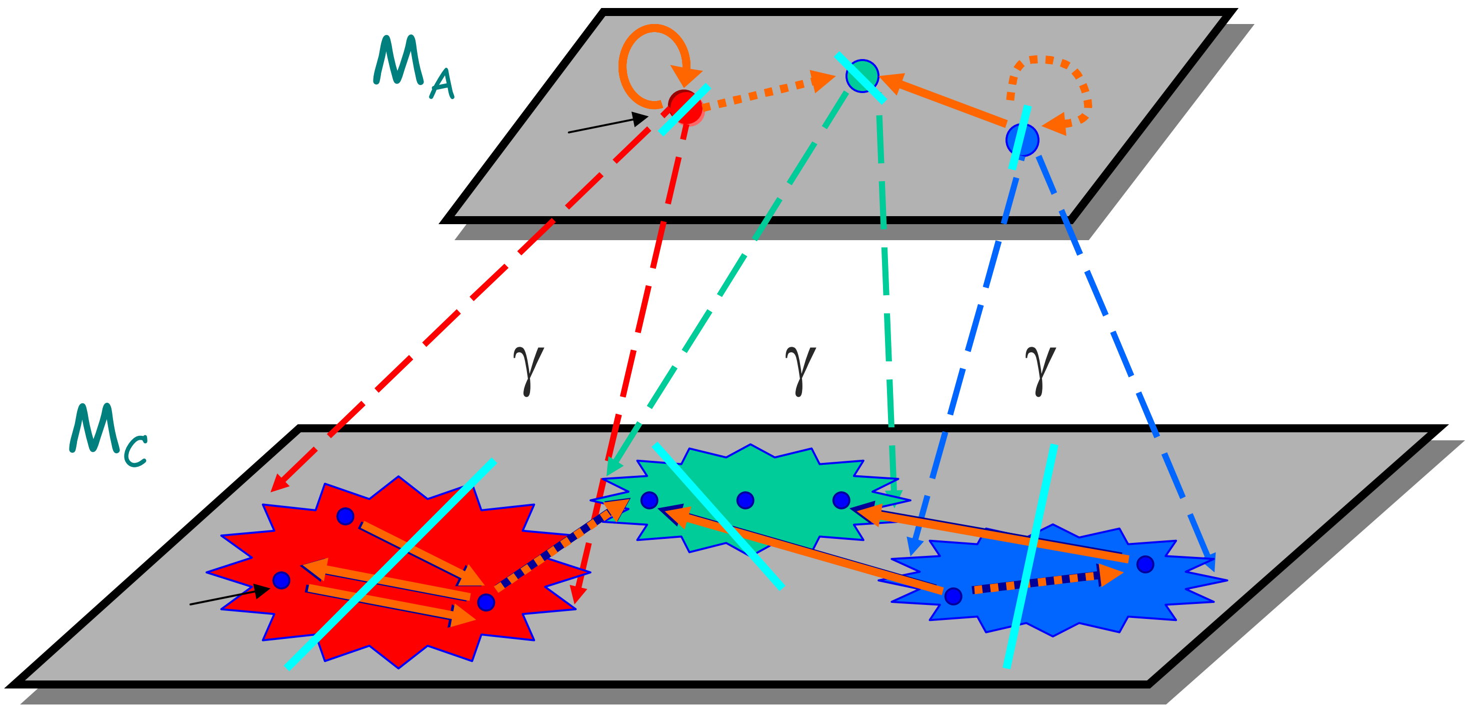
- tt and ff are definite: hold in the concrete model as well.

- ⊥ is indefinite

  ⇒ Refinement is needed.

Refinement

- As for the case of 2-values, done by splitting abstract states



Refinement

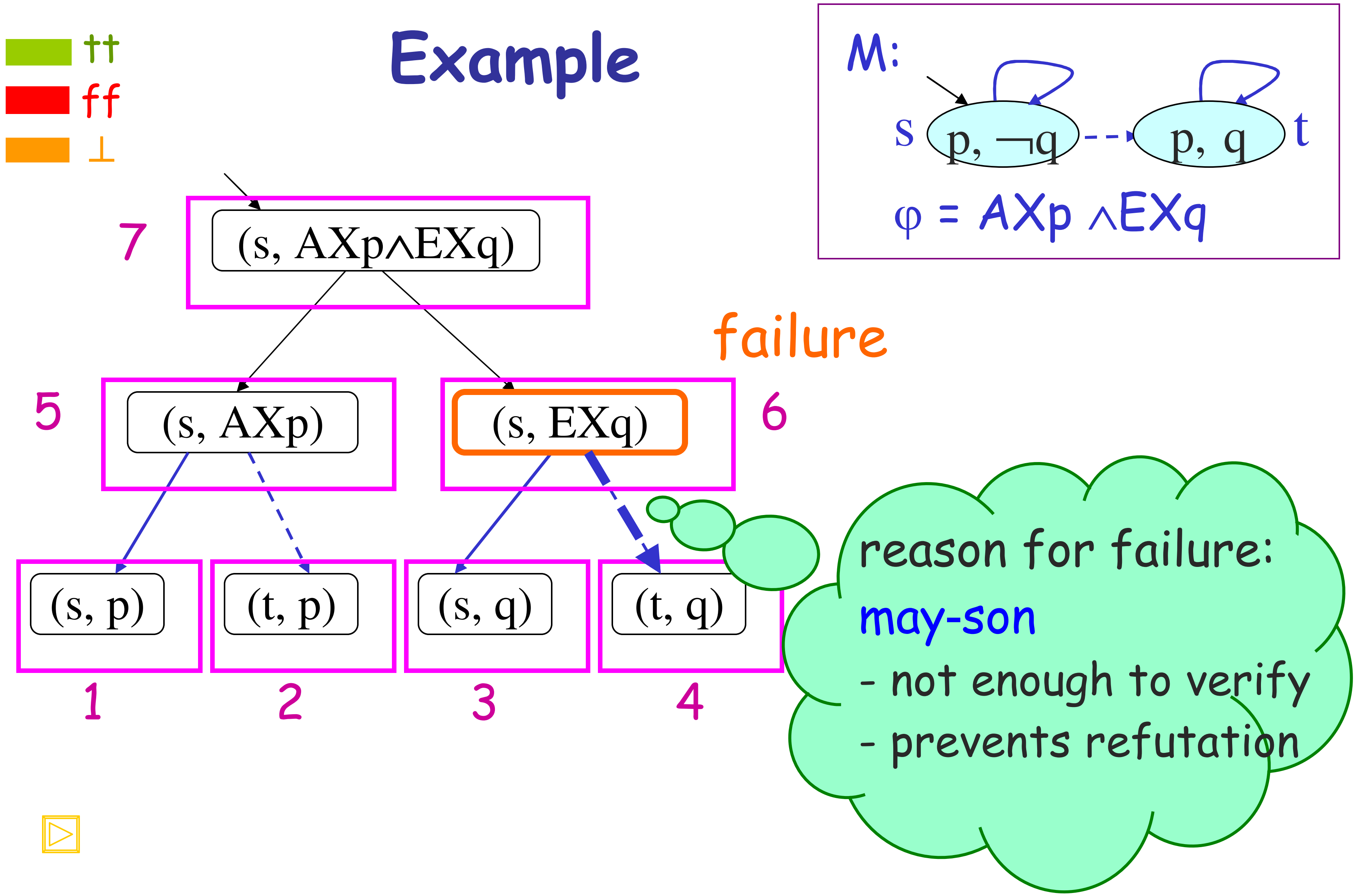
- Identify a failure state: a state s_a for which some subformula φ is $\perp$ in s_a
  - Done during model checking
- Split s_a so that
  - an indefinite atomic proposition becomes definite (true or false), or
  - A may transition becomes a must transition or disappears

Refinement (cont.)

- Uses the colored MC graph
- Find a failure node n_f:
  - a node colored $\perp$ whereas none of its sons was colored $\perp$ at the time it got colored.
  - the point where certainty was lost
- purpose: change the $\perp$ color of n_f .

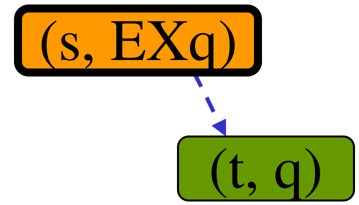
Refinement is reduced to separating subsets of the concrete states represented by n_f.

Example

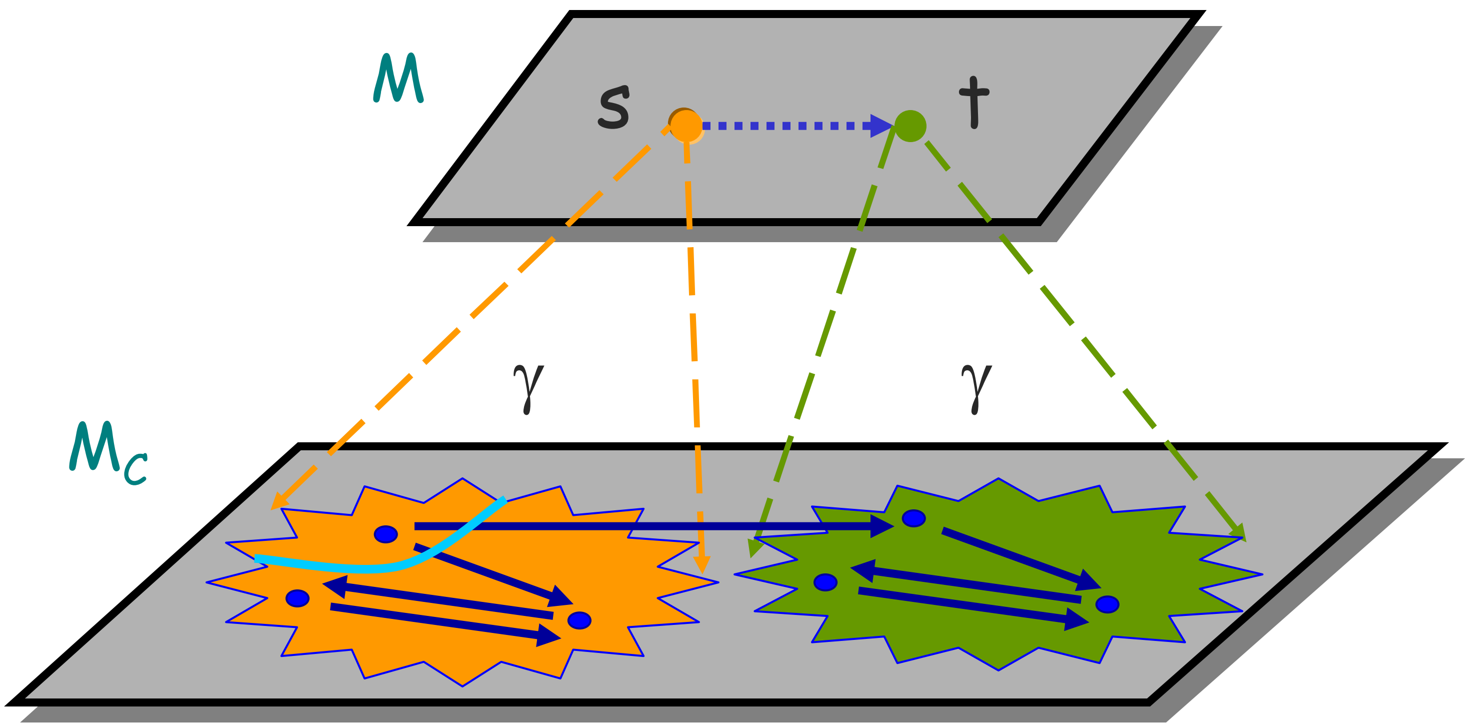


tt
ff
⊥
7
(s, AXp∧EXq)
5
(s, AXp)
(s, EXq)
6
failure
(s, p)
(t, p)
(s, q)
(t, q)
1
2
3
4
M:
s
p, ¬q
p, q
t
φ = AXp ∧EXq
reason for failure:
may-son
- not enough to verify
- prevents refutation

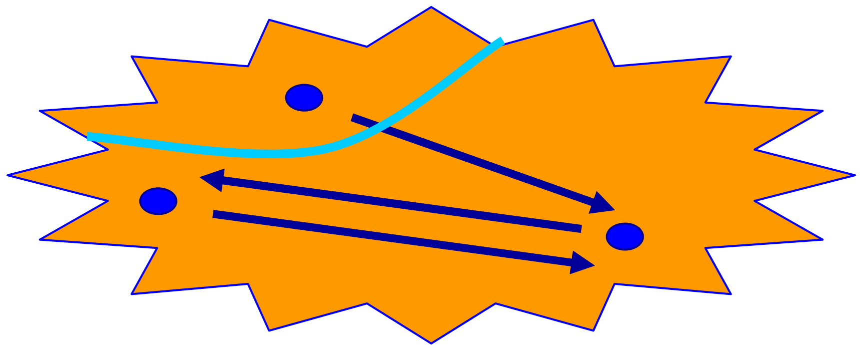
Example (cont.)



concrete states that have a son corresponding to the may-edge are separated from the rest



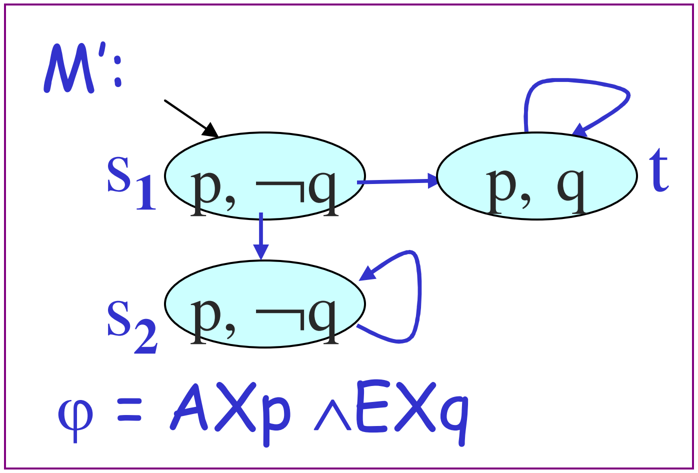
Example (cont.)

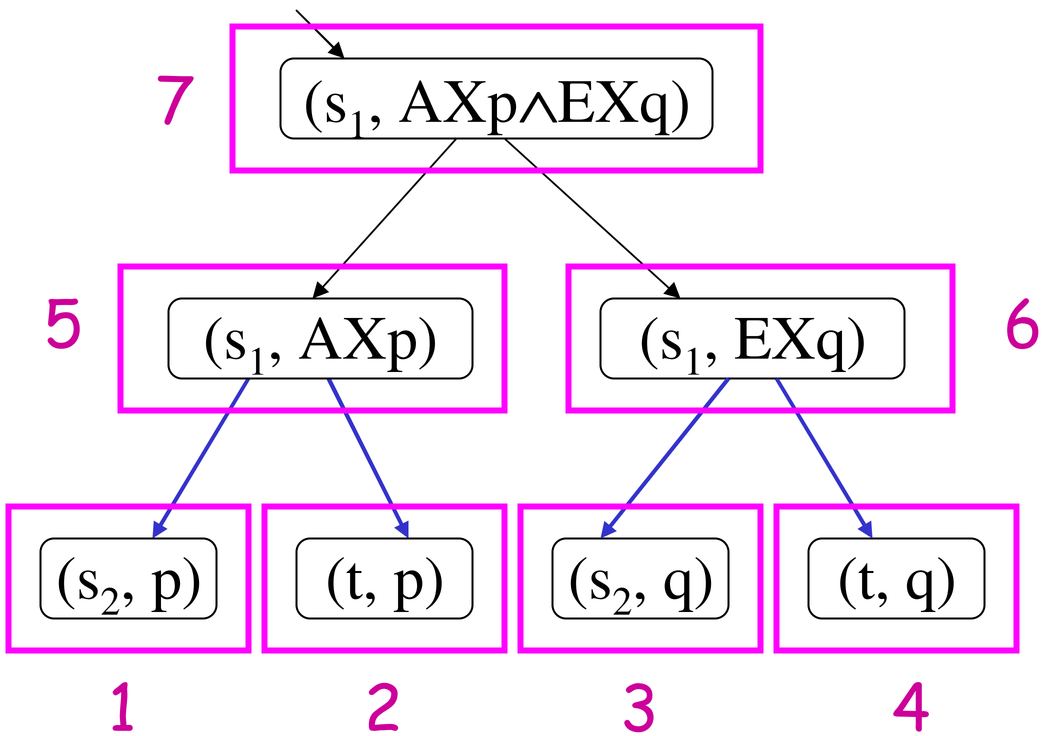


- Find a criterion that separates the two sets of concrete states.
  - Can be done using known techniques. [CGJLV00,CGKS02]

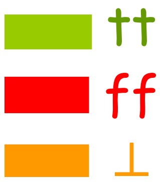
$\Rightarrow$ build a refined model accordingly

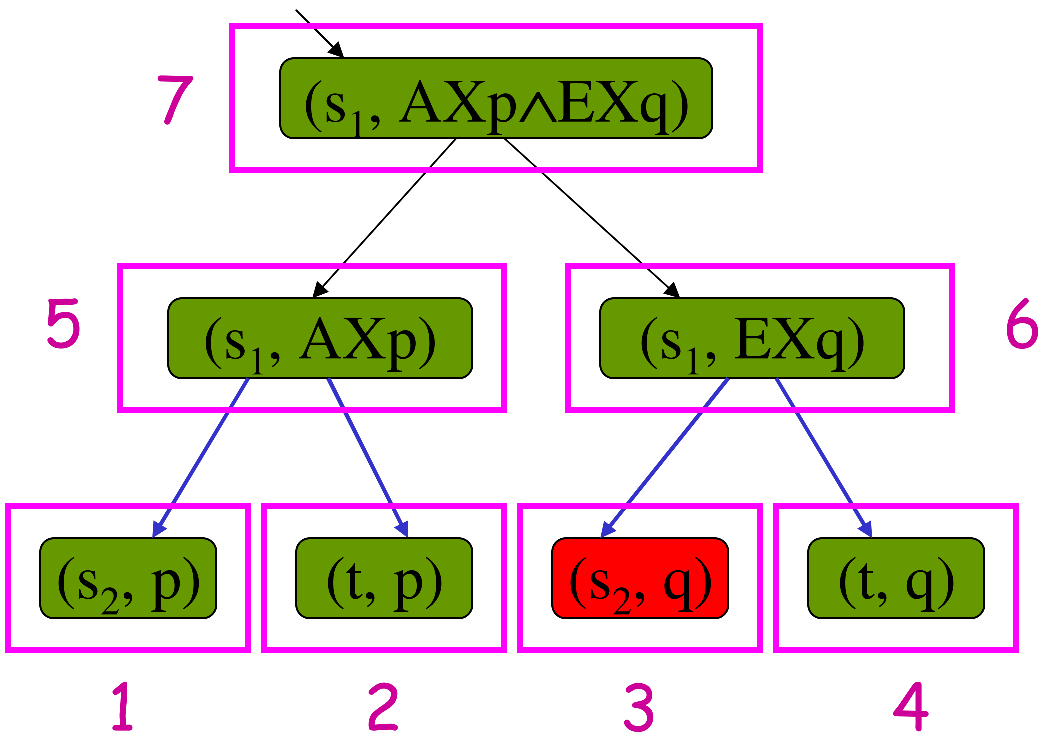
Example (cont.)

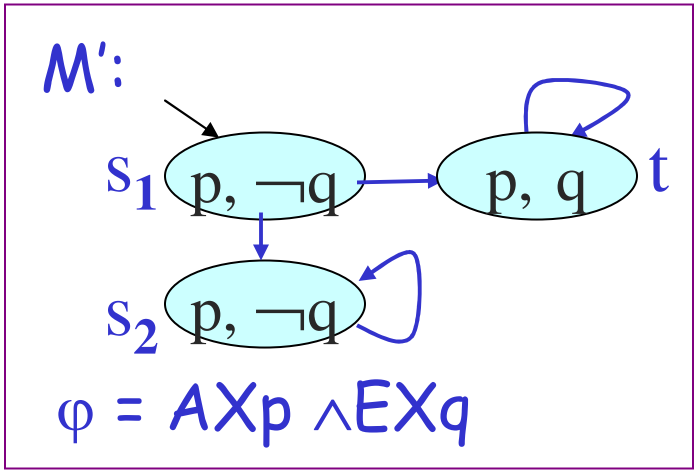




Example (cont.)







Completeness

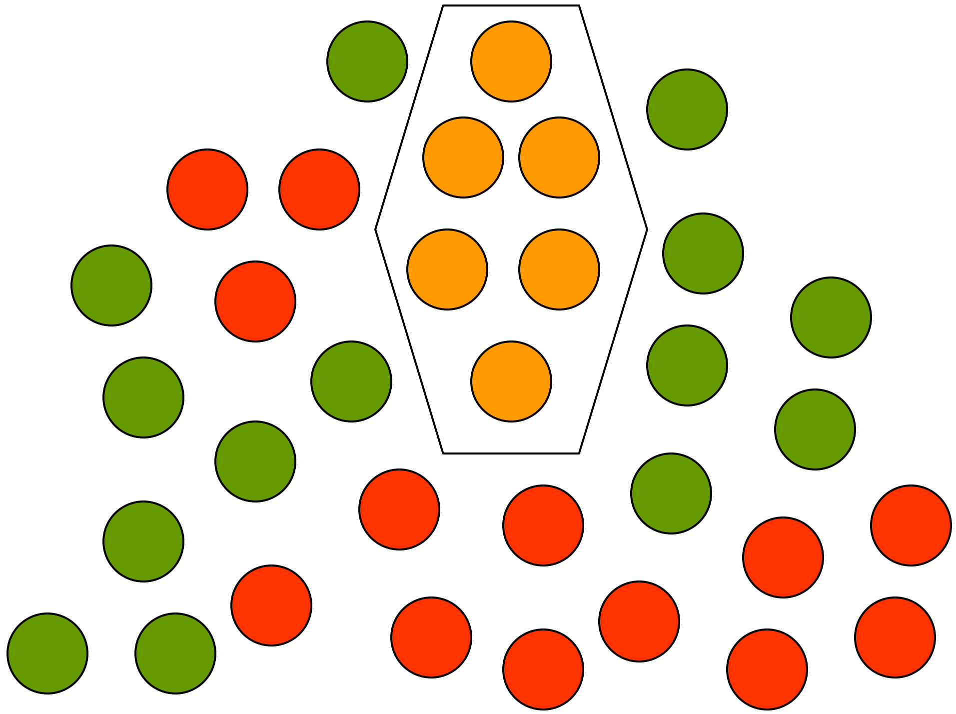
- Our methodology refines the abstraction until a definite result is received.
- For finite concrete models iterating the abstraction-refinement process is guaranteed to terminate, given any **CTL / CTL* / μ-calculus** formula.

Incremental Abstraction-Refinement

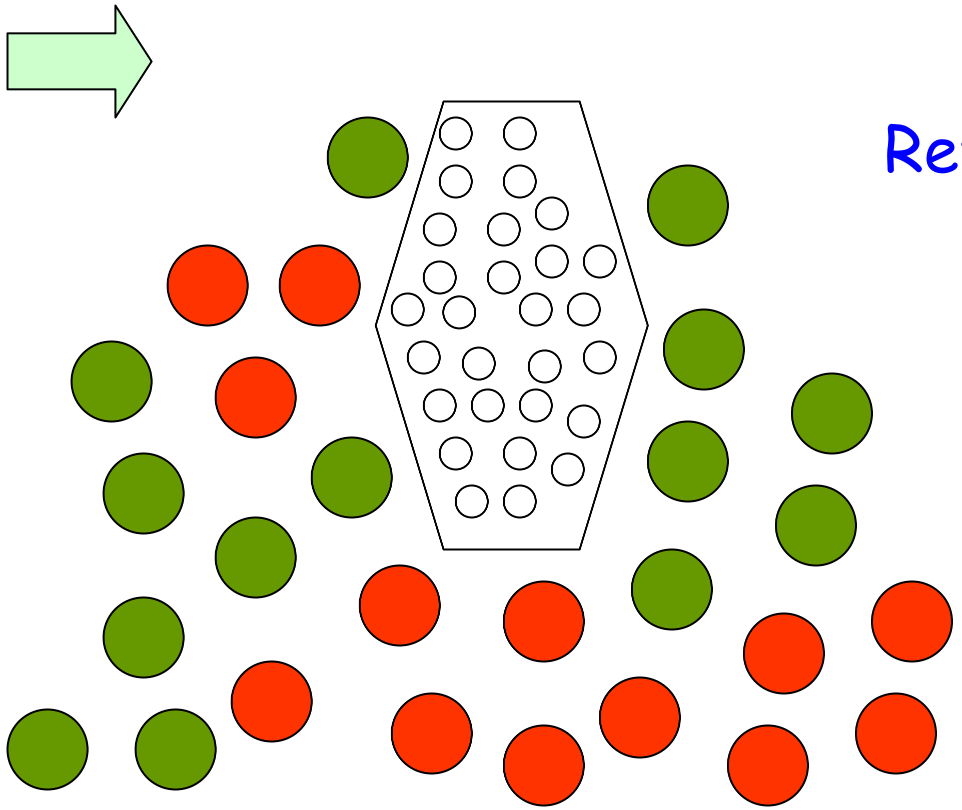
No reason to split states for which MC results are definite during refinement.

- After each iteration remember the nodes colored by definite colors.
- Prune the refined MC graph in sub-nodes of remembered nodes.
  [(s_a, φ) is a sub-node of (s_a', φ') if $\varphi=\varphi'$ and $\gamma(s_a)\subseteq\gamma'(s_a')$]
- Color such nodes by their previous colors.

Example

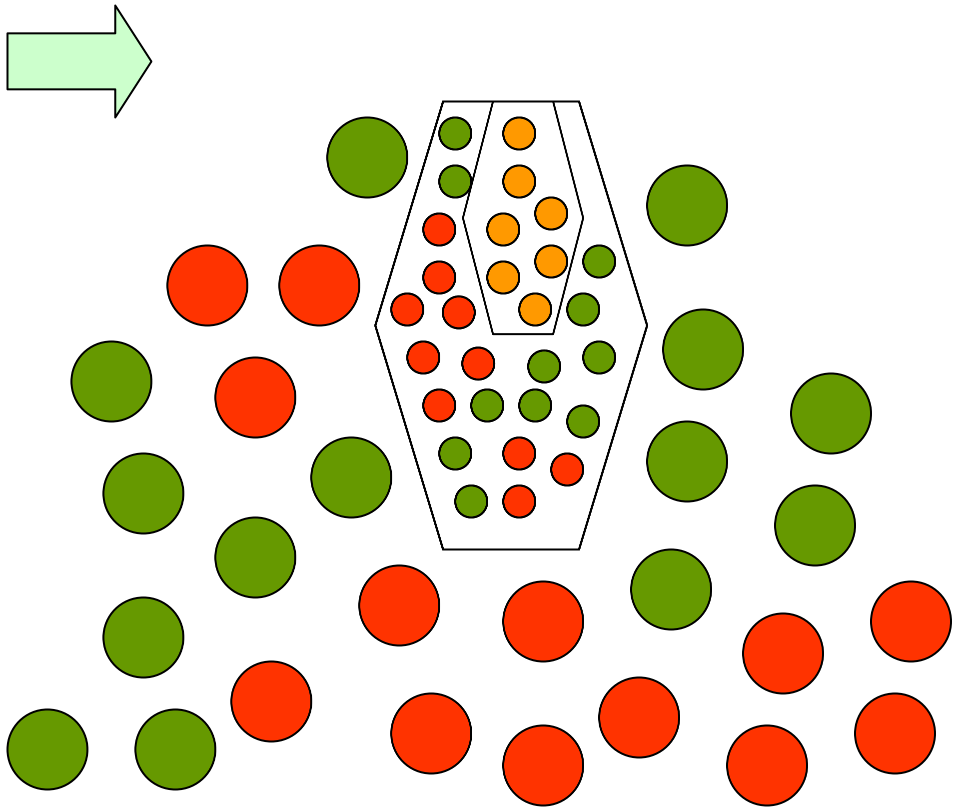


Example (cont.)

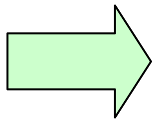


Refined MC-graph

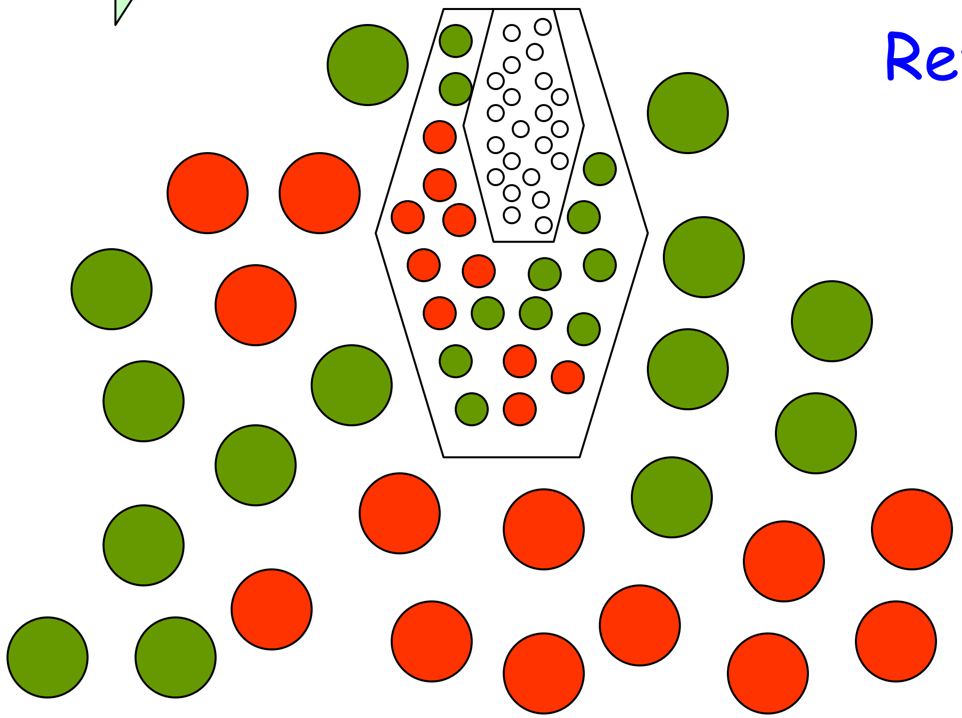
Example (cont.)



Example (cont.)



Refined MC-graph



…

Conclusion

We presented two frameworks for abstraction-refinement in model checking:

- Model Checking for abstract models
  - For 2-valued semantics: as for concrete models
  - For 3-valued semantics: using MC-graph
- Refinement eliminating
  - Counterexamples, in the 2-valued case
  - indefinite results, in the 3-valued case
- Incremental abstraction-refinement
  - Called lazy abstraction in the 2-valued case

Summary

We presented two frameworks, CEGAR and TVAR, for abstraction-refinement in model checking:

- Properties preserved:
  - CEGAR: Aμ-**calculus** (ACTL)
  - TVAR: Full μ-**calculus**

- Refinement eliminates
  - CEGAR: Counterexamples
  - TVAR: indefinite results ($\perp$)

Summary (cont.)

The TVAR framework requires

- Different abstract models (Rmust, Rmay)
  - Rmust is harder to compute
- Adapted model checking algorithm

Successful applications in:

- Compositional model checking
- 3-valued Bounded Model Checking (BMC)

Its usefulness worth the extra effort

Conclusion

3-valued abstract models are useful:

- More precise
- Enable verification and falsification
- Avoid false negative results

Thank You