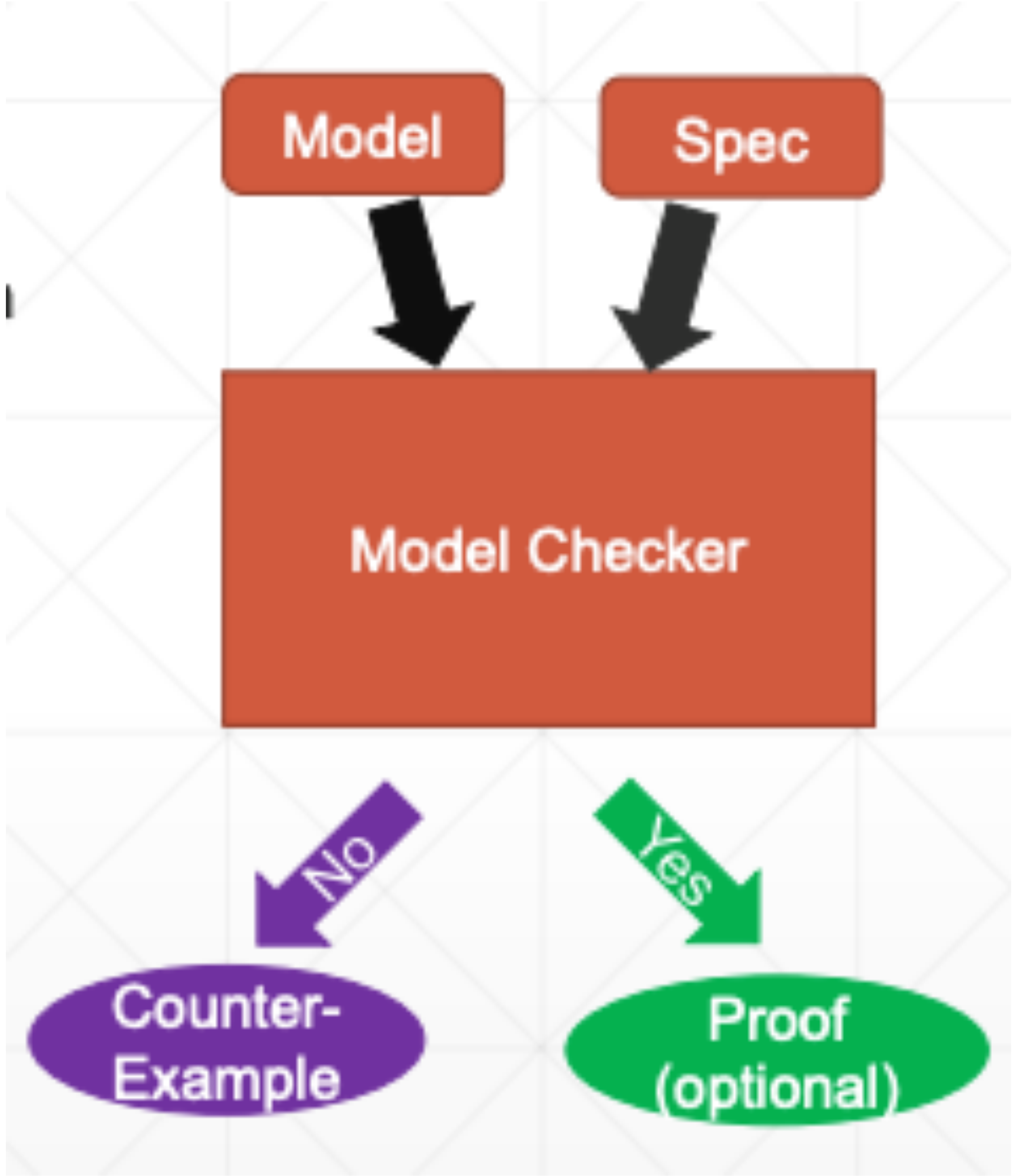
Model Checking : A brief view with CounterExample-Guided Abstraction / Refinement

Konkuk University
Department of Computer Science & Engineering
Kunha Kim

Model Checking

- Model Checking is an approach for verifying the temporal behavior of a system.
- We need Model, and Specification to verify. Model Checker takes them for input, and gives “Yes” or “No” with counterexample.
- Then how can we model a complex system? And how can we ask a correct question formally to Model Checker?
- The answer is : Kripke Model and CTL.


Model
Spec
Model Checker
No
Yes
Counter-
Example
Proof
(optional)

Formal Definition of Kripke Model

- Formally, a Kripke model is a 4-tuple $M = < S, I, R, L >$, where
    - S is the set of states,
    - $I \subseteq S$ is the set of initial states,
    - $R \subseteq S \times S$ is the transition relation, and
    - $L: S \rightarrow P(AP)$ gives the set of atomic propositions true in each state.
- We assume that R is total, and a path in M is an infinite sequence of states $\pi = s_0, s_1, \cdots$ such that for $i \geq 0, (s_i, s_{i+1}) \in R$.
- Also, the model is finite.

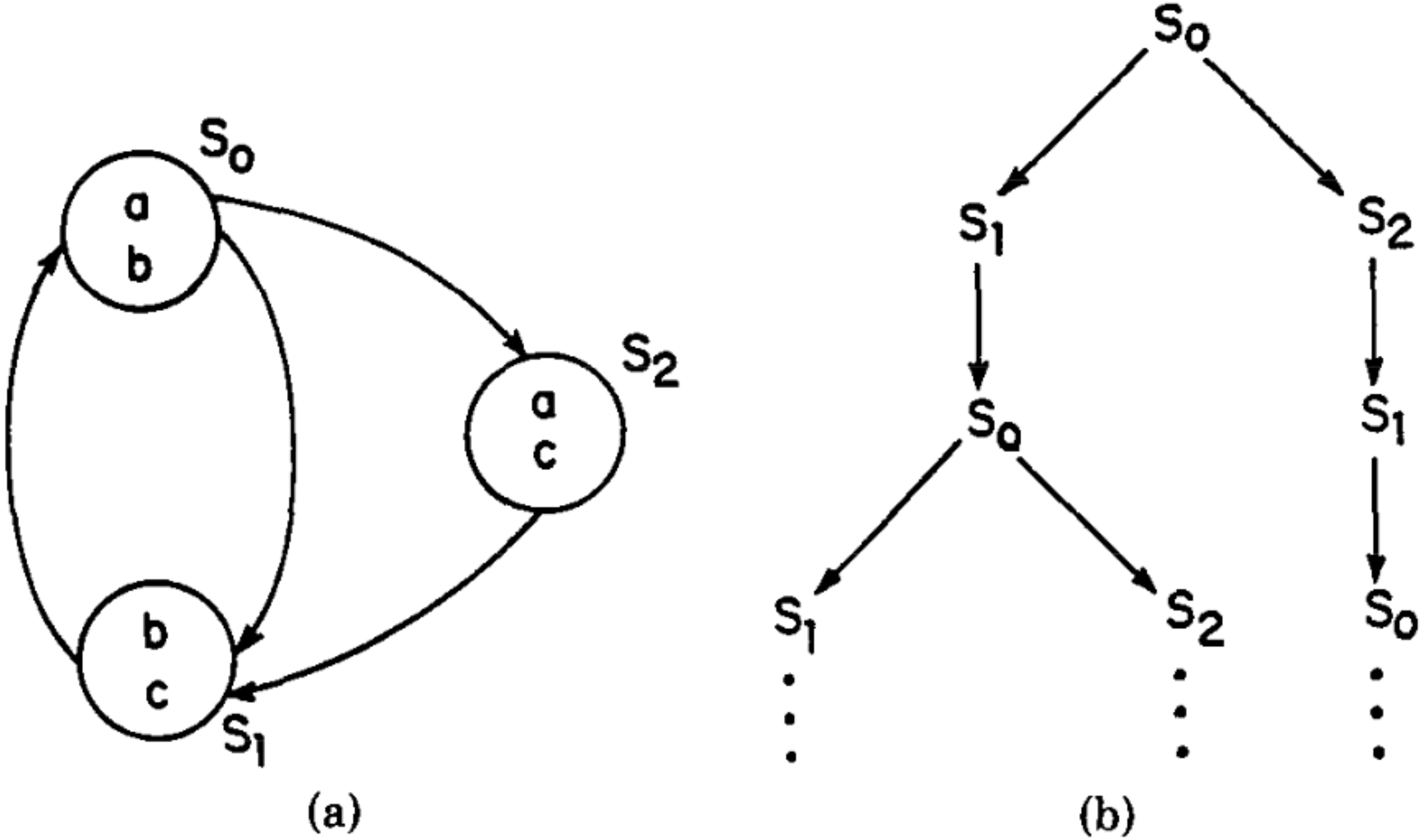
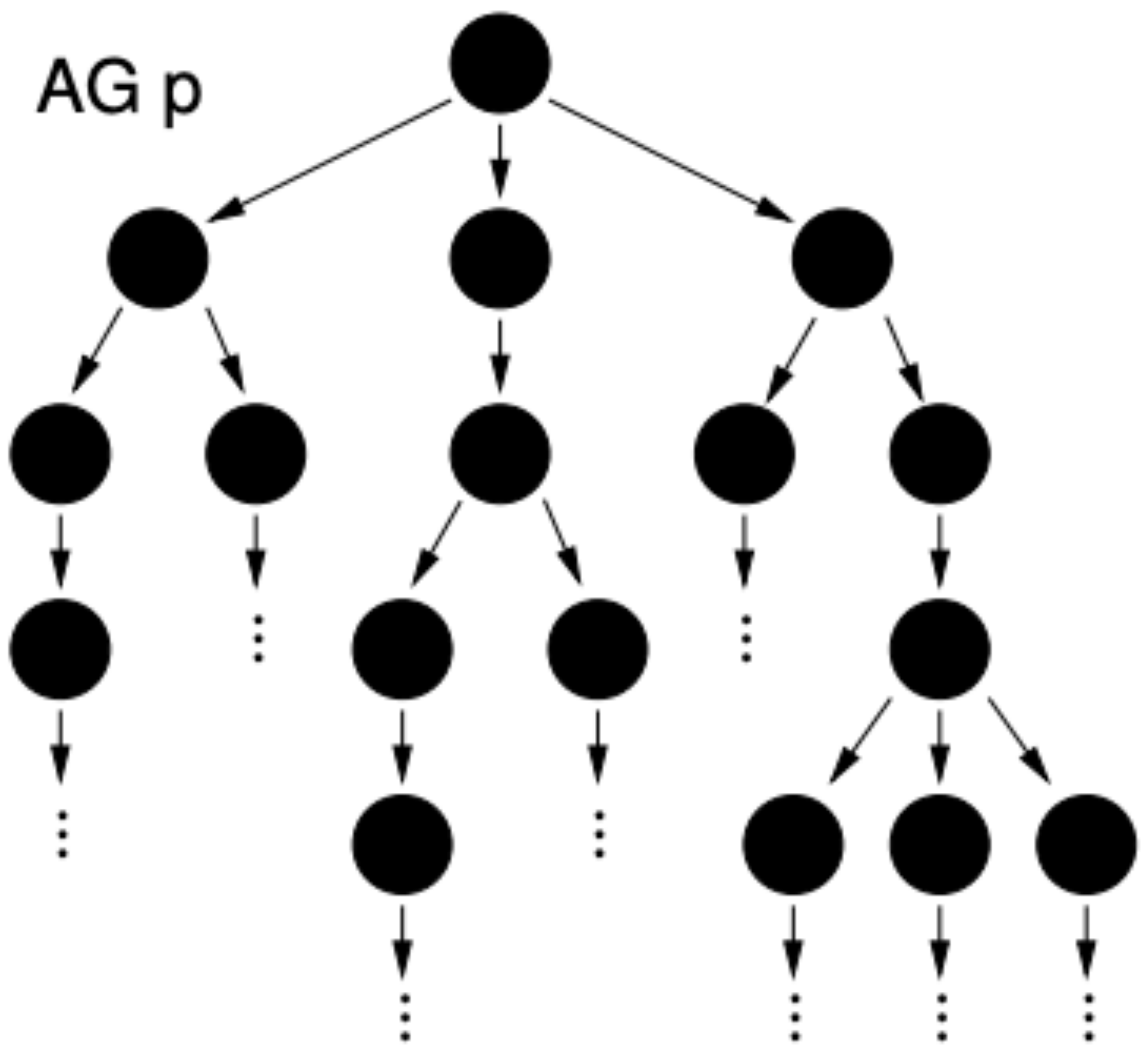
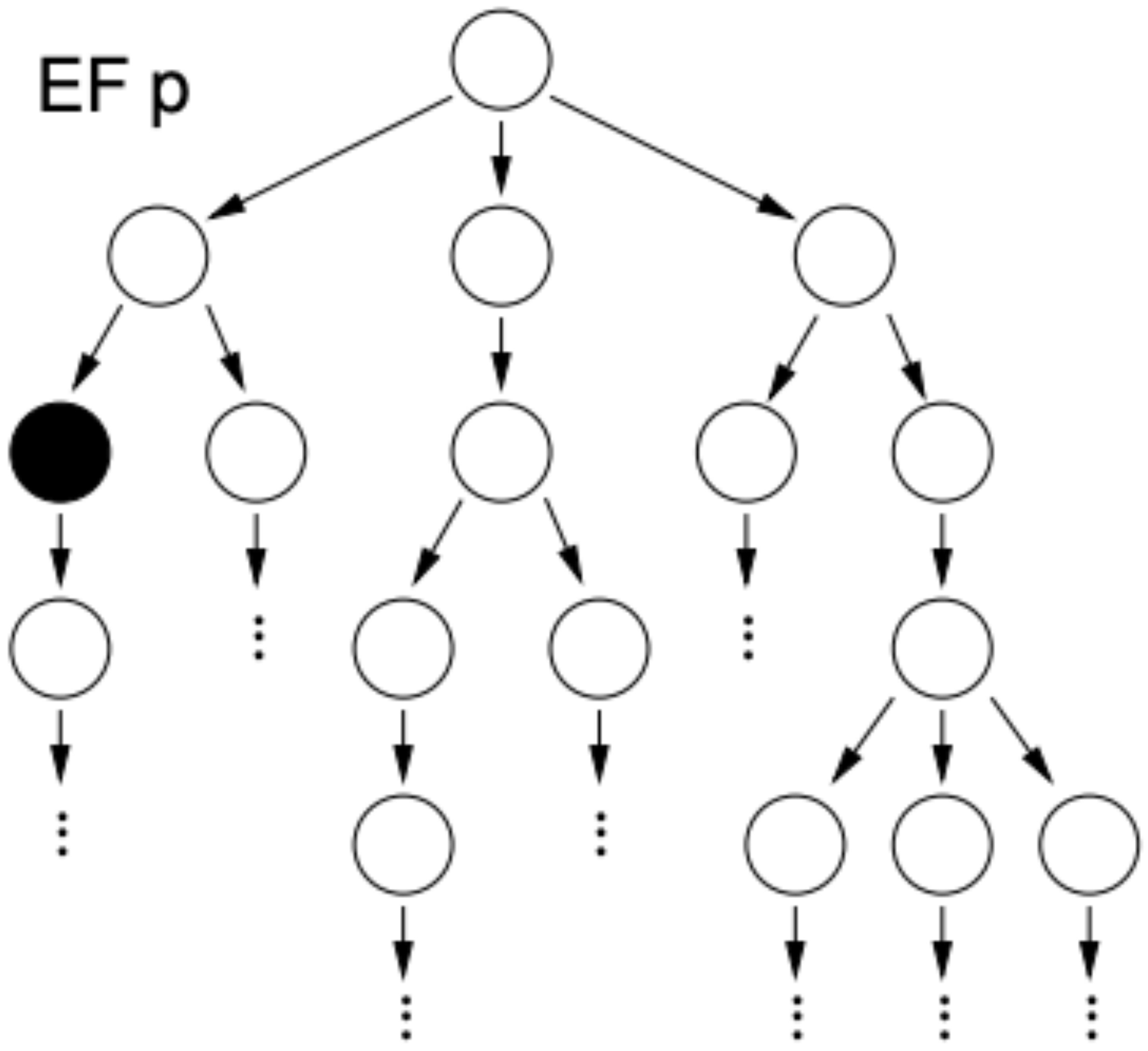


Fig. 1. (a) A structure. (b) The corresponding tree for start state S_0.

Computation Tree Logic

- Because R is total, there is an $infinite computation tree$ for any model M.
- CTL is a branching time logic, consists of path quantifier and linear-time operators.
- We don't have much time, so we're gonna focus on AG and EF.
- AGp means for every path, p holds globally in the future.
- EFp means there exists a path p eventually holds.


AG p


EF p

Over-approximation

- There are many complex systems such as CPU, autonomous driving system, software, etc.

- Due to original system's complexity, model's states can extremely increase. This is called state - explosion.

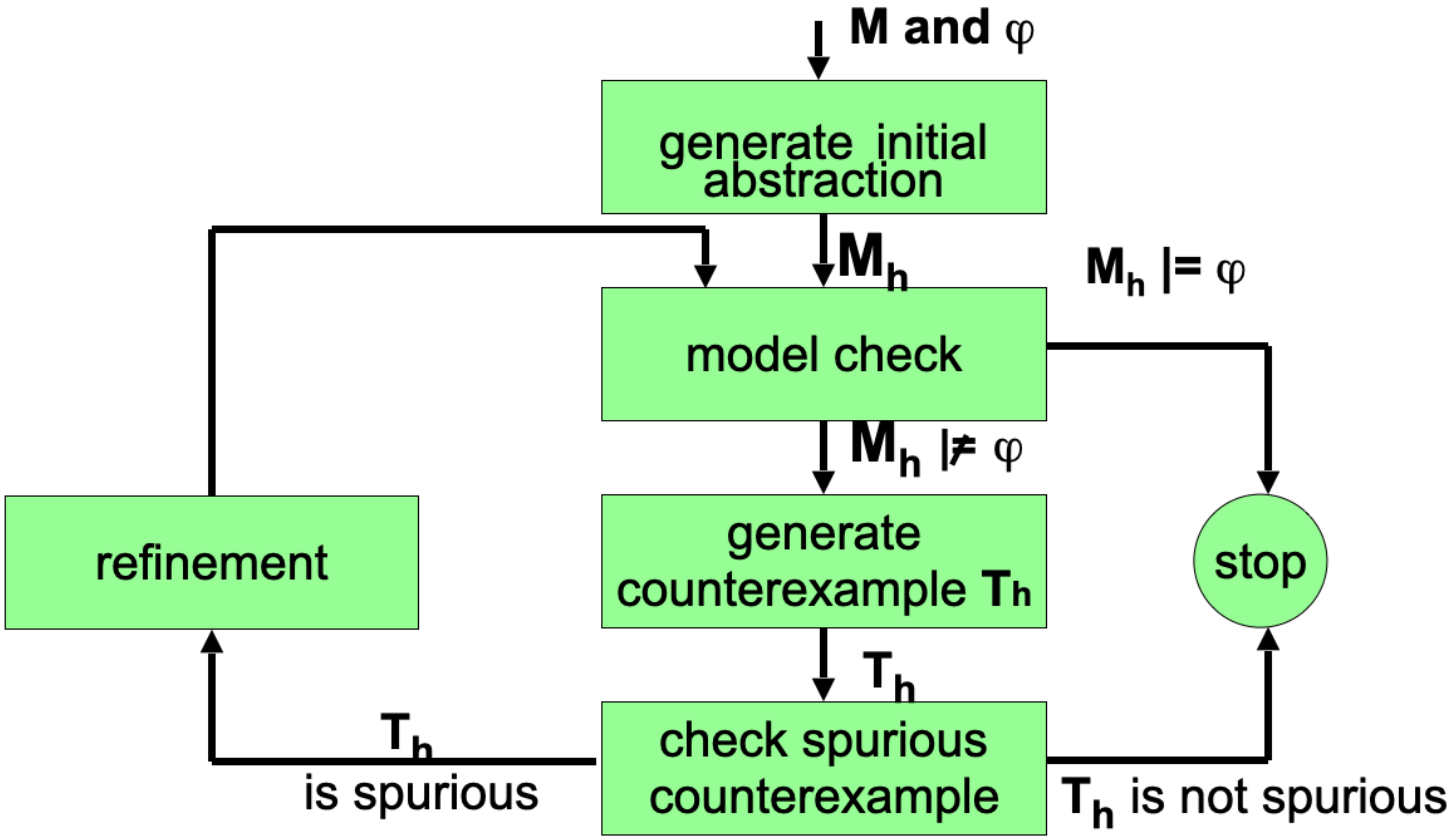
- There are many approaches to handle this problem, but now, we focus on Abstraction / Refinement technique.

- We over-approximate original model's behavior, and check whether specification is true or false.

- This admits only false negative(spurious counterexample).

- When model checker outputs counterexample, we check this is spurious or not. If so, we refine the abstract model by this spurious counterexample.

Our Abstraction Methodology (CAV'2000)



Abstract function h

- Now, we will abstract the Kripke Model by abstraction function h.

- Intuitively speaking, existential abstraction amounts to **partitioning the states** of a Kripke structure **into clusters**, and treating the clusters as new abstract states.

- Formally, an abstraction function h is described by a surjection $h: S \to \hat{S}$ where $\hat{S}$ is the set of abstract states.

- Let d, e be states in S. We say d and e are **logically equivalent on relation h** iff $h(d) = h(e)$ and denoted by $d \equiv_h e$

Formal Definition of Abstraction

- The *abstract Kripke structure* $\widehat{M} = (\widehat{S}, \widehat{I}, \widehat{R}, \widehat{L})$ generated by the abstraction function h is defined as

  - $\widehat{I}(\widehat{d})$ iff $\exists d(h(d) = \widehat{d} \wedge I(d))$

  - $\widehat{R}(\widehat{d}_1, \widehat{d}_2)$ iff $\exists d_1 \exists d_2(h(d_1) = \widehat{d}_1 \wedge h(d_2) = \widehat{d}_2 \wedge R(d_1, d_2))$

  - $\widehat{L}(\widehat{d}) = \cup_{h(d)=\widehat{d}} L(d)$

- An abstraction function h is *appropriate* for a specification ρ if for all atomic sub-formulas f of ρ and for all states d and e in the domain S such that $d \equiv_h e$ it holds $d \models f \Leftrightarrow e \models f$.

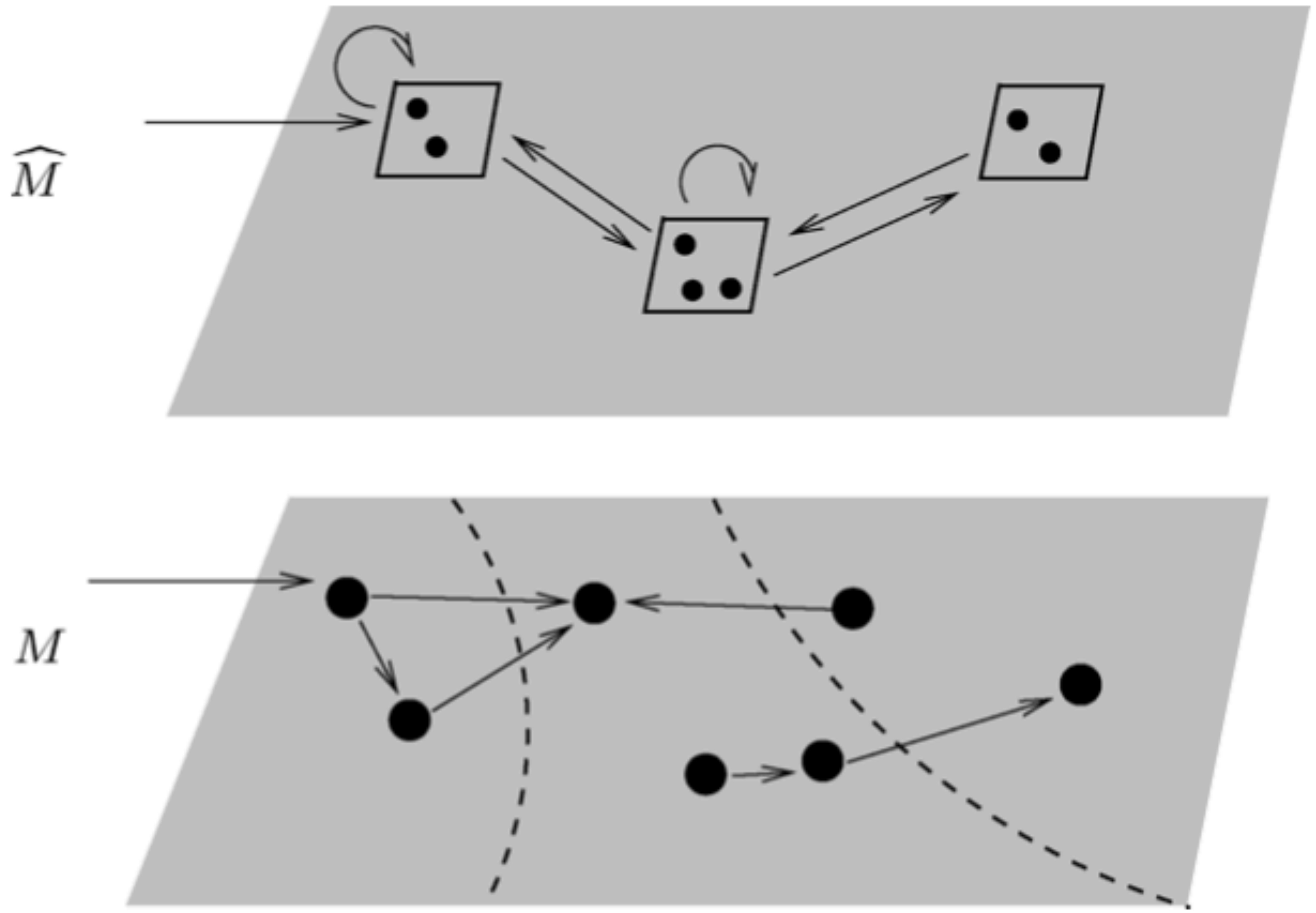


FIG. 1. Existential Abstraction. M is the original Kripke structure, and $\widehat{M}$ the abstracted one. The dotted lines in M indicate how the states of M are clustered into abstract states.

Identification of Spurious Counterexamples: Path

- First we will tackle the case $\widehat{T}$ is a path $< \widehat{s_1}, \cdots, \widehat{s_n} >$. Given an abstract state $\widehat{s}$, the set of concrete states s s.t. $h(s) = \widehat{s}$ is denoted by $h^{-1}(\widehat{s})$.

- We extend h^{-1} to sequences in the following way: $h^{-1}(\widehat{T})$ is the set of concrete paths given by the following expression: $\{ < s_1, \cdots, s_n > \mid \wedge_{i=1}^{n} h(s_i) = \widehat{s_i} \wedge I(s_1) \wedge \wedge_{i=1}^{n-1} R(s_i, s_{i+1})\}$

- Let $S_1 = h^{-1}(\widehat{s_1}) \cap I$. For $1 < i \leq n$, we define S_i in the following manner:
$S_i \doteqdot Img(S_{i-1}, R) \cap h^{-1}(\widehat{s_i})$. This can be computed by using OBDD and standard image computation algorithm.

- The following lemma establishes the correctness of this procedure.

- $Lemma 1$. If the path $\widehat{T}$ corresponds to a concrete counterexample, the set of concrete paths $h^{-1}(\widehat{T})$ is non-empty and $S_i \neq \emptyset$ for all $1 \leq i \leq n$.

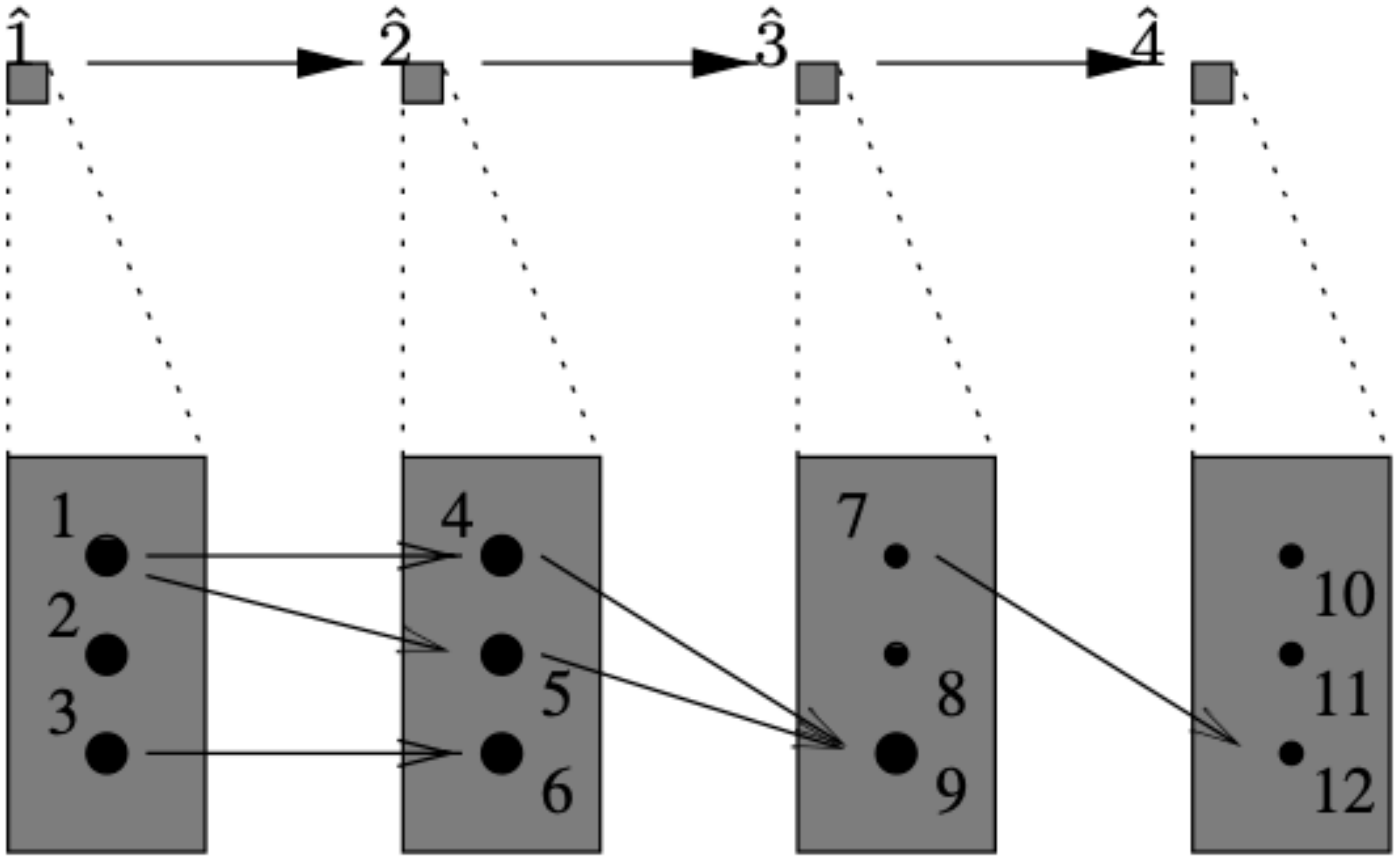


Fig. 3. An abstract counterexample

Example

- Consider a program with only one variable with domain $D = \{1, \cdots, 12\}$.
  Assume that h maps $x \in D$ to $\lfloor (x-1)/3 \rfloor + 1$.

- There are four abstract states $\widehat{1}, \widehat{2}, \widehat{3}$ and $\widehat{4}$ correspond to $\{1,2,3\}, \{4,5,6\}, \{7,8,9\}, \{10,11,12\}$ respectively.

- Suppose we obtain an abstract counterexample $\widehat{T} = <\widehat{1}, \widehat{2}, \widehat{3}, \widehat{4}>$. It is easy to see $\widehat{T}$ is spurious.

- Also, we can check $S_1 = \{1,2,3\}, S_2 = \{4,5,6\}, S_3 = \{9\}, S_4 = \emptyset$. Notice that S_4 and therefore $Img(S_3, R)$ are both empty.

- The algorithm **SplitPATH** will output 3 and S_3.

Algorithm SplitPATH

$S := h^{-1}(\widehat{s_1}) \cap I$
$j := 1$
while $(S \neq \emptyset$ and $j < n)$ $\{$
  $j := j + 1$
  $S_{\text{prev}} := S$
  $S := Img(S, R) \cap h^{-1}(\widehat{s_j})$ $\}$
if $S \neq \emptyset$ **then** output counterexample
else output j, S_{prev}

Fig. 4. SplitPATH checks spurious path.

Clue of Refinement

- We see that the abstract path does not have a corresponding concrete path.
- Whichever concrete path we follow, we will end up in state D, from which we cannot go any further. Therefore, D is called a deadend state.
- On the other hand, the bad state is state B, because it made us believe that there is an outgoing transition.
- In addition, there is the set of irrelevant states I that are neither deadend nor bad, and it is immaterial whether states in I are combined with D or B.
- To eliminate the spurious path, the abstraction has to be refined as indicated by the thick line.

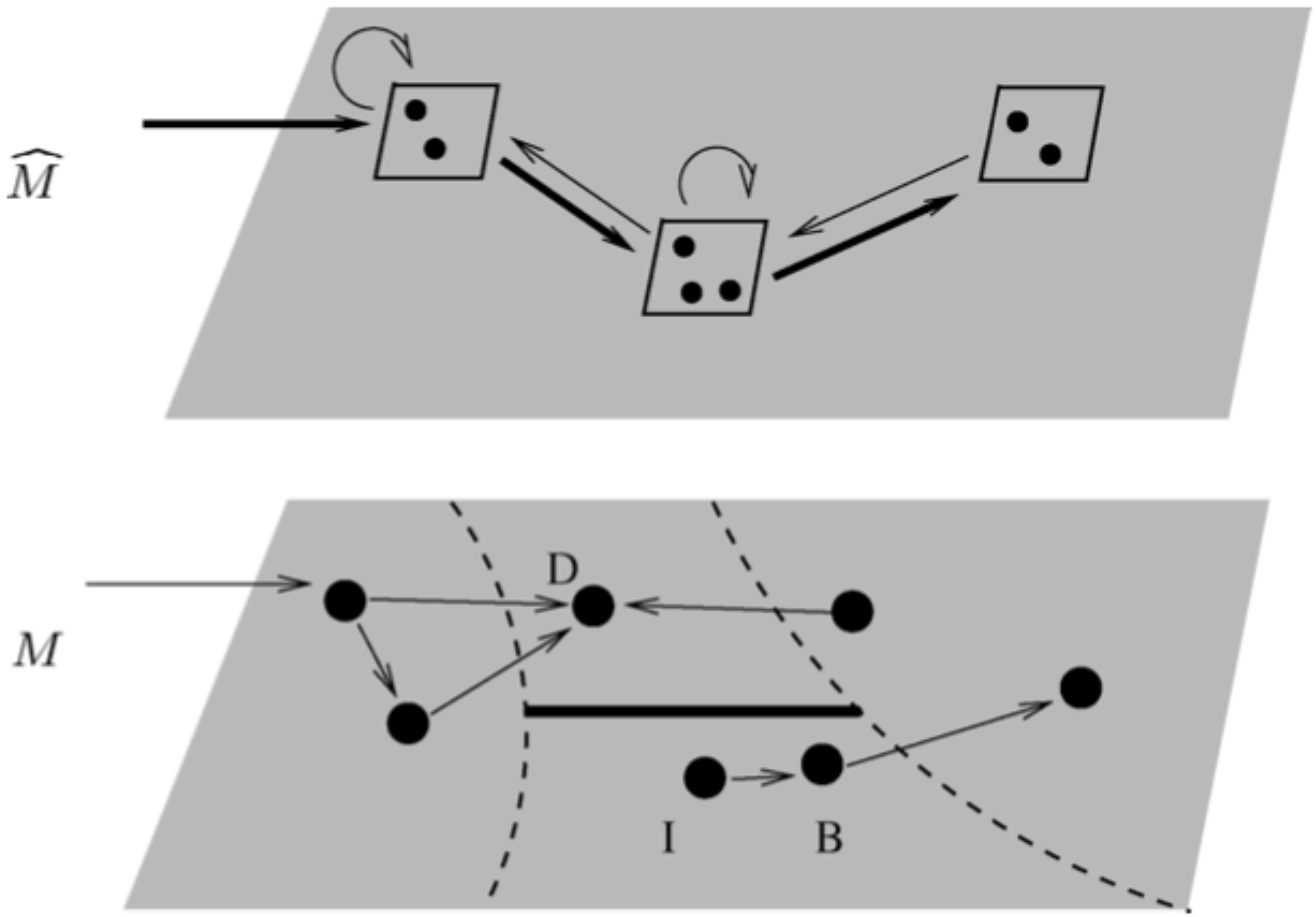


FIG. 3. The abstract path in $\widehat{M}$ (indicated by the thick arrows) is spurious. To eliminate the spurious path, the abstraction has to be refined as indicated by the thick line in M.

Recall : Fig3

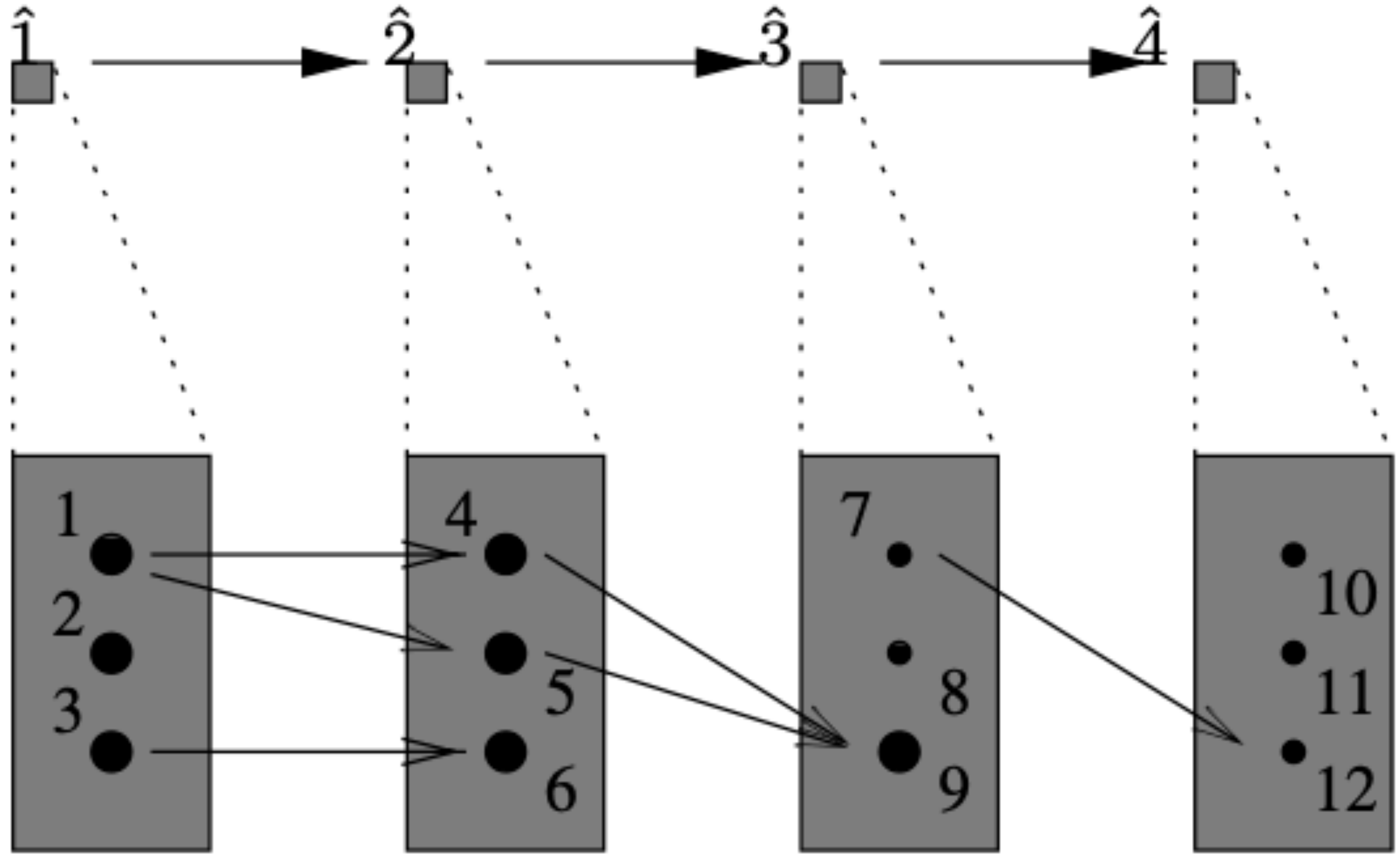


Fig. 3. An abstract counterexample

State Splitting

- For a set $h^{-1}(\widehat{s_i})$, a set of concrete states correspond to $\widehat{s_i}$ occurring in spurious counterexample, split it into $h^{-1}(\widehat{s_i}) \cap Image(h^{-1}(\widehat{s}_{i-1}))$ and $h^{-1}(\widehat{s_i}) \setminus Image(h^{-1}(\widehat{s}_{i-1}))$, provided both non-empty.

- In the case $i = 1$, split it into $h^{-1}(\widehat{s_1}) \cap I$ and $h^{-1}(\widehat{s_1}) \setminus I$.

- In other words, replace $\widehat{s_i}$ by two states $\widehat{s_i}^+$ and $\widehat{s_i}^-$ representing $h^{-1}(\widehat{s_i}) \cap Image(h^{-1}(\widehat{s}_{i-1}))$ and $h^{-1}(\widehat{s_i}) \setminus Image(h^{-1}(\widehat{s}_{i-1}))$, respectively.

- Therefore, $\widehat{S}$ is turned into $\widehat{S}' = \widehat{S} \setminus \{\widehat{s_i}\} \cup \{\widehat{s_i}^+, \widehat{s_i}^-\}$.

State Splitting(Cont.)

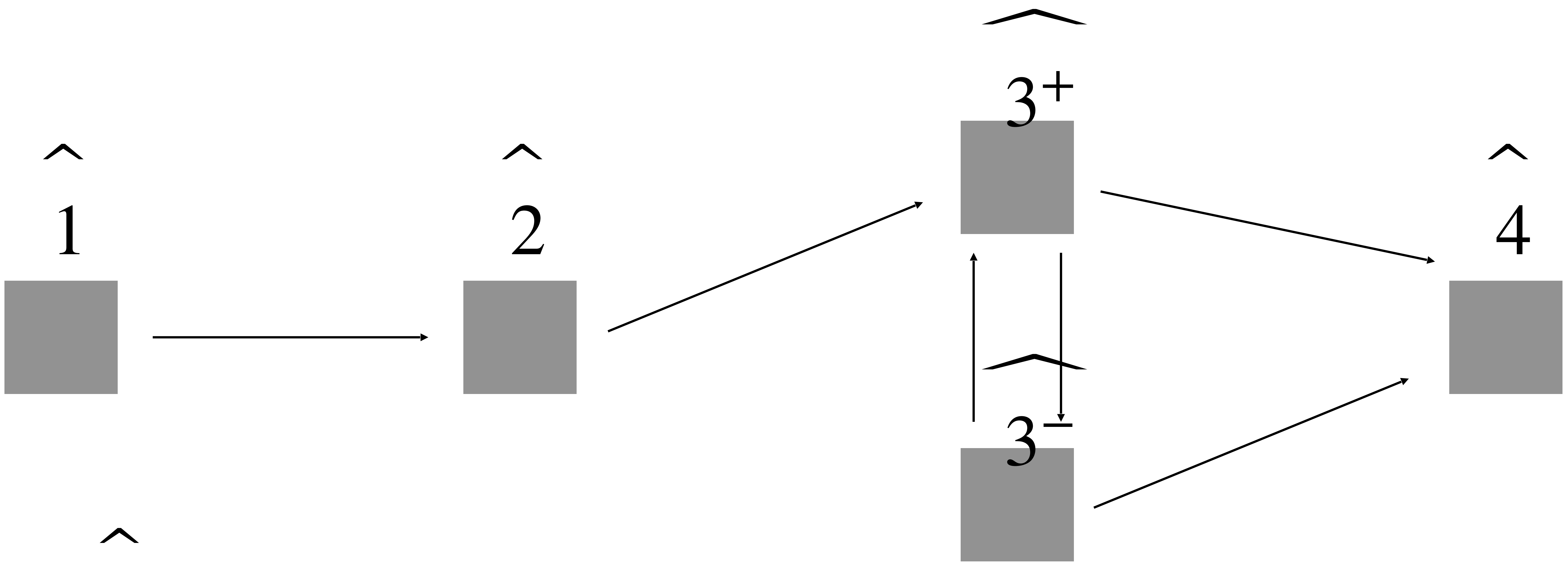
- So, replace $\widehat{M} = (\hat{S}, \hat{I}, \widehat{R}, \hat{L})$ to $\widehat{M}' = (\hat{S}', \hat{I}', \widehat{R}', \hat{L}')$ denoted as

$$\hat{S}' = \hat{S} \setminus \{\hat{s}_i\} \cup \{\hat{s}_i^+, \hat{s}_i^-\}$$

$$\hat{I}' = \hat{I} \text{ if } h^{-1}(\hat{s}_i) \cap I = \emptyset, \text{ otherwise } I \setminus \{\hat{s}_i\} \cup \{\hat{s}_i^+\}$$

$$\widehat{R}' = \widehat{R} \cap (\hat{S}' \times \hat{S}') \cup \{(\hat{s}_i^+, \hat{s}_i^-), (\hat{s}_i^-, \hat{s}_i^+)\} \cup \{(\hat{s}, \hat{s}_i^+) \mid (\hat{s}, \hat{s}_i) \in \widehat{R}\} \cup$$
$$\{(\hat{s}, \hat{s}_i^-) \mid (\hat{s}, \hat{s}_i) \in \widehat{R}, \hat{s} \neq \hat{s}_{i-1}\} \cup \{(\hat{s}_i^+, \hat{s}), (\hat{s}_i^-, \hat{s}) \mid (\hat{s}_i, \hat{s}) \in \widehat{R}\}$$

$$\hat{L}'(\hat{s}) = L(\hat{s}) \text{ if } \hat{s} \in \hat{S}, L(\hat{s}_i) \text{ if } \hat{s} \in \{\hat{s}_i^+, \hat{s}_i^-\}$$



$$h^{-1}(\hat{1}) = \{1,2,3\}$$

$$h^{-1}(\hat{2}) = \{4,5,6\}$$

$$h^{-1}(\widehat{3^+}) = \{9\}$$

$$h^{-1}(\widehat{3^-}) = \{7,8\}$$

$$h^{-1}(\hat{4}) = \{10,11,12\}$$

Abstraction and Refining transition

- So, resulting abstraction is $h'(s) = \widehat{s_i}^{+}$ if $s \in h^{-1}(\widehat{s_i}) \cap Image(\widehat{s}_{i-1})$, $\widehat{s_i}^{-}$ if $s \in h^{-1}(\widehat{s_i}) \setminus Image(\widehat{s}_{i-1})$, otherwise $h(s)$.

- Pre- and post-images of $h'^{-1}(\widehat{s_i}^{+})$ or $h'^{-1}(\widehat{s_i}^{-})$ may well have empty intersections with sets that the pre- or post-set of $h'^{-1}(\widehat{s_i})$ did intersect with. In such cases, $\widehat{R}'$ contains spurious edges.

- Thus, remove such edges by pruning $\widehat{R}'$ to $\widehat{E}'' = \{(s,t) \in \widehat{R}' \mid Image(h'^{-1}(s)) \cap h'^{-1}(t) \neq \emptyset\}$

Experimental Results

TABLE II. RUNNING RESULTS FOR THE BENCHMARK DESIGNS

Design	NuSMV+COI				NuSMV+ABS			
	#COI	Time	$\lvert TR \rvert$	$\lvert MC \rvert$	#ABS	Time	$\lvert TR \rvert$	$\lvert MC \rvert$
gigamax	0	0.3	8346	1822	9	0.2	13151	816
guidance	30	35	140409	30467	34–39	30	147823	10670
waterpress	0–1	273	34838	129595	4	170	38715	3335
PCI bus	4	2343	121803	926443	12–13	546	160129	350226
ind1	0	99	241723	860399	50	9	302442	212922
ind2	0	486	416597	2164025	84	33	362738	624600
ind3	0	617	584815	2386682	173	15	426162	364802