Learning Abstractions for Model Checking

Anubhav Gupta
Cadence Berkeley Labs

Overview

- Abstraction for Model Checking ≡ Inductive Learning
  - Learning and Abstraction-Refinement
  - Learning Abstractions without Refinement

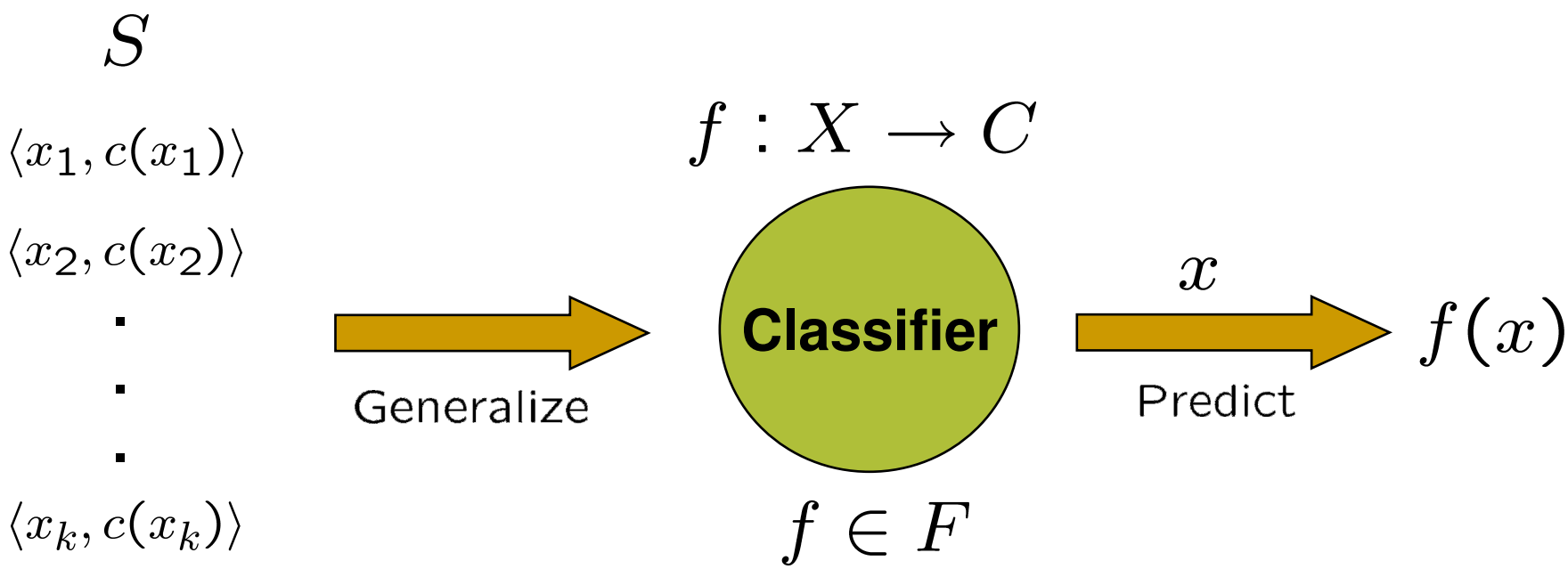
Outline

- Machine Learning
- Abstraction
- Learning and Abstraction-Refinement
- Learning Abstractions without Refinement

Machine Learning

- Process that causes a system to improve its performance at a particular task with experience [Mitchell]

Inductive Learning



Inductive Learning: Generalizing from Samples

Inductive Bias

- Generalization requires bias towards certain target functions
  - Completely Unbiased Learner: Learning boolean functions by memorization
- Inductive bias captures the domain-specific assumptions that help in classifying unseen instances
- Two forms on inductive biases:
  - Restriction Bias: Set of candidate functions is restricted
  - Preference Bias: Certain functions preferred over others

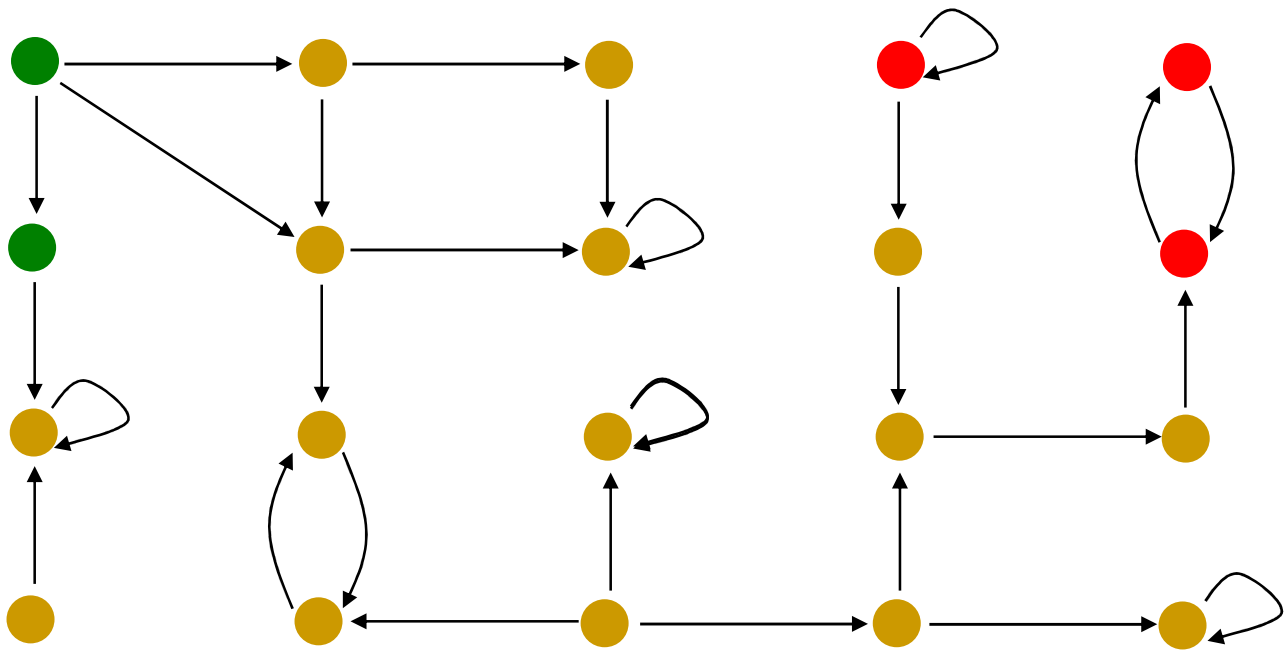
Generating Samples

- Random Sampling: Training set provided to learner
- Queries: Learner asks teacher specific questions about the target function to generate samples
  - Membership queries
    - Input: Object
    - Output: Classification
  - Equivalence queries
    - Input: Target function
    - Output: Done <u>or</u> Misclassified object with classification

Outline

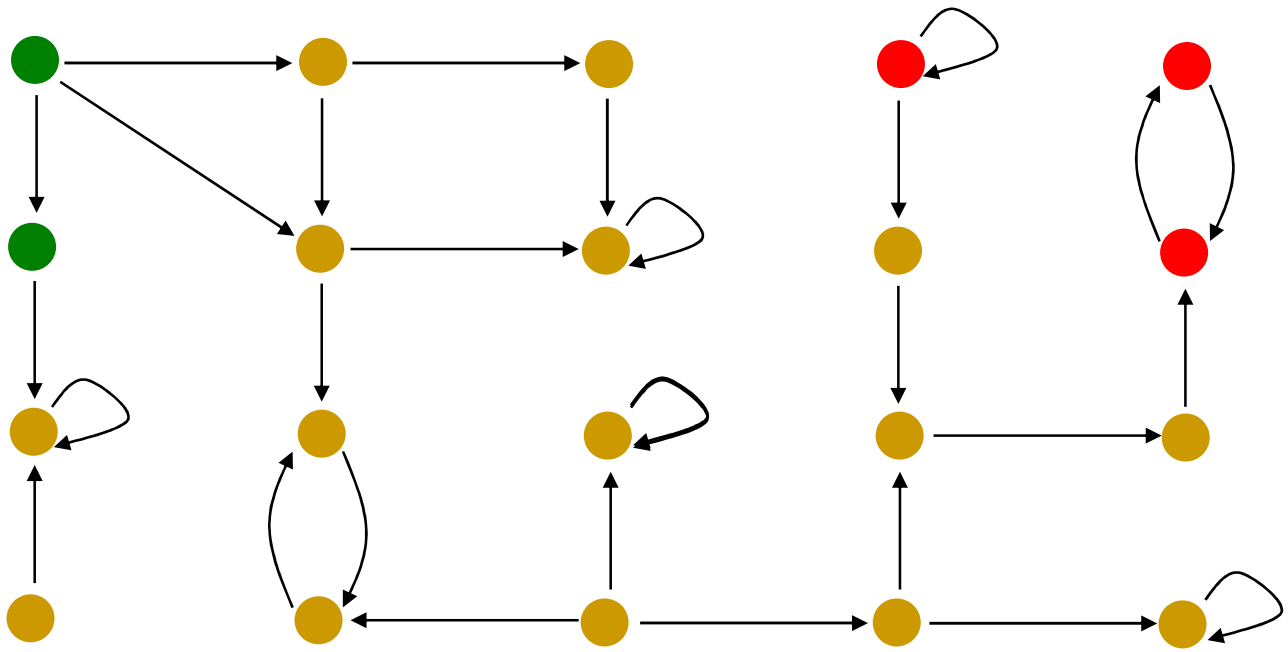
- Machine Learning
- Abstraction
- Learning and Abstraction-Refinement
- Learning Abstractions without Refinement

Model Checking



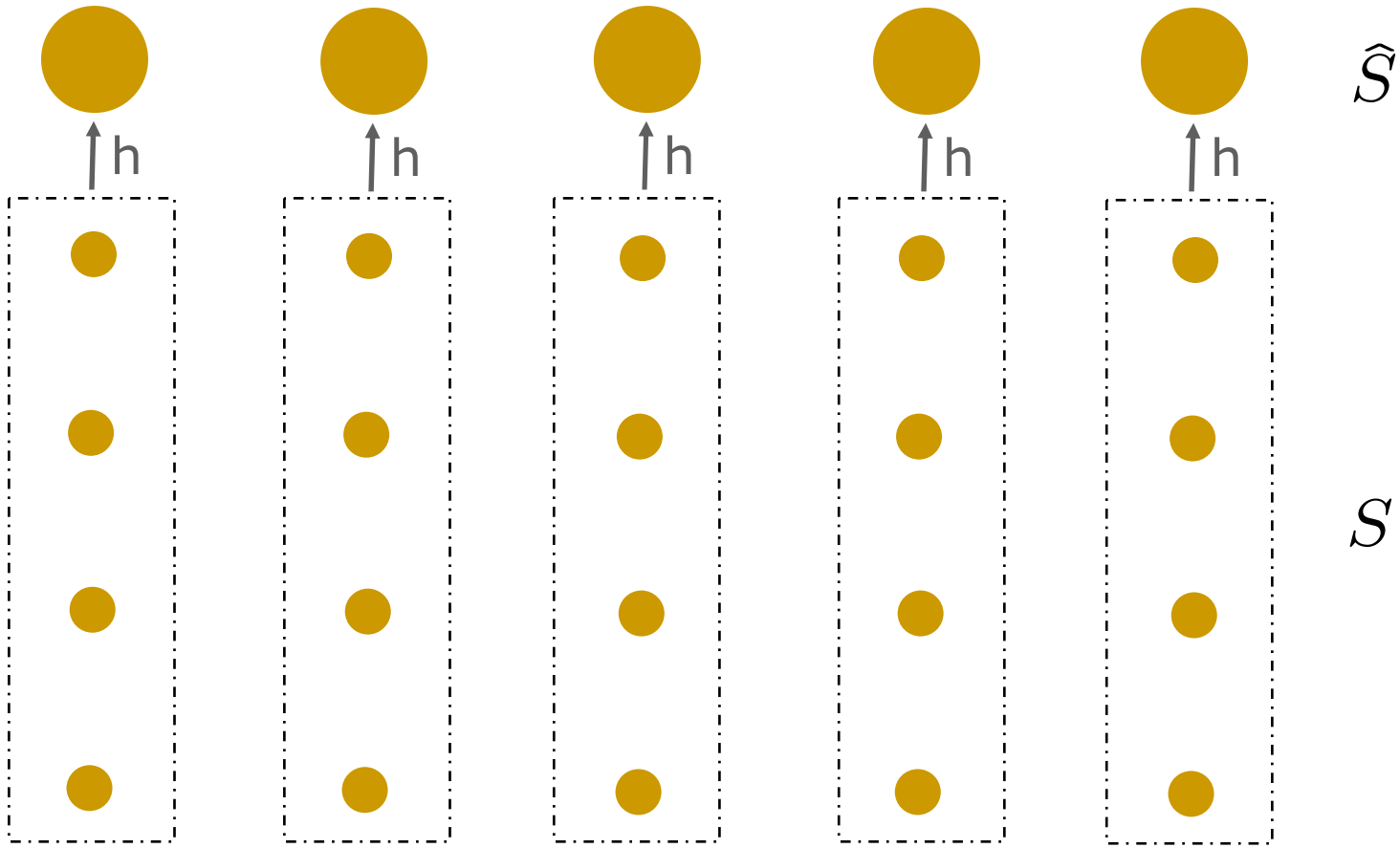
Model Checking for Safety Properties

State-Explosion Problem



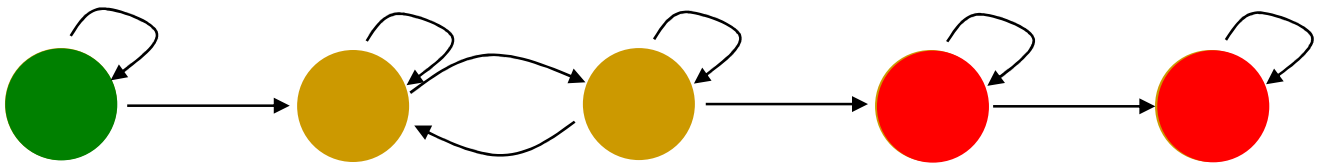
Too many states to handle

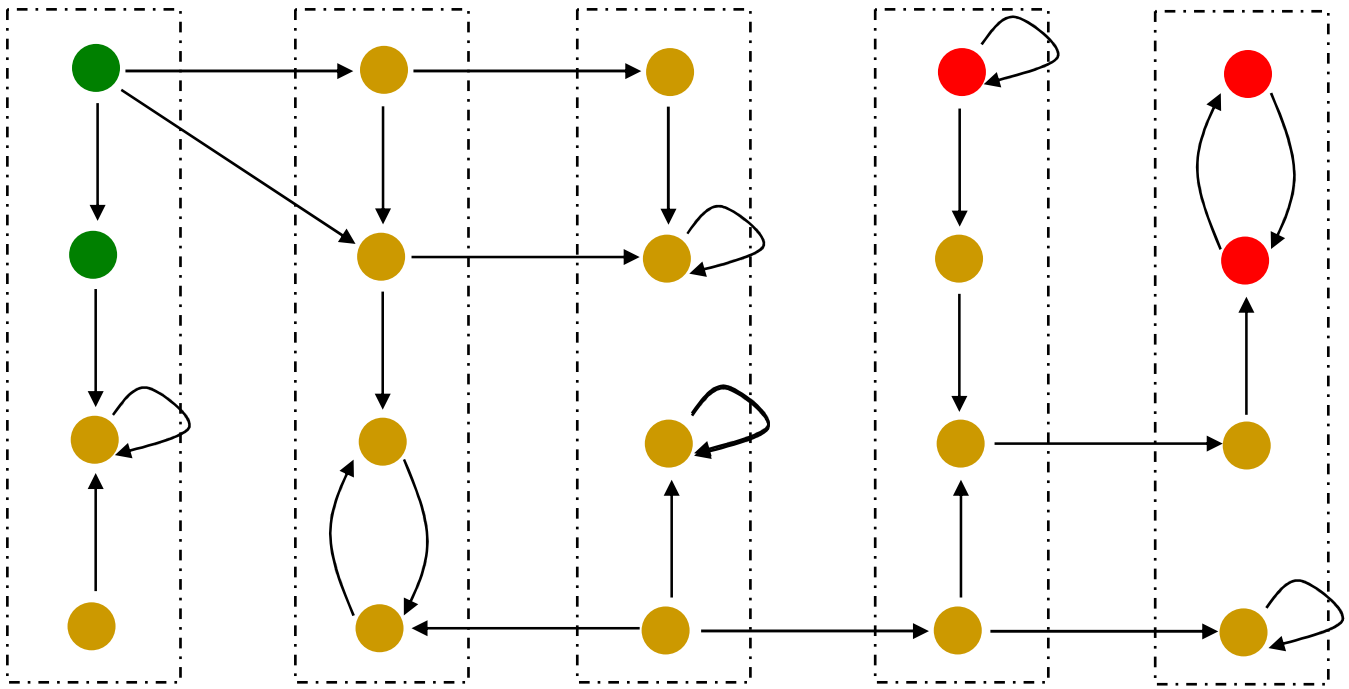
Abstraction



Abstraction Function $h : S \longrightarrow \widehat{S}$

Abstraction



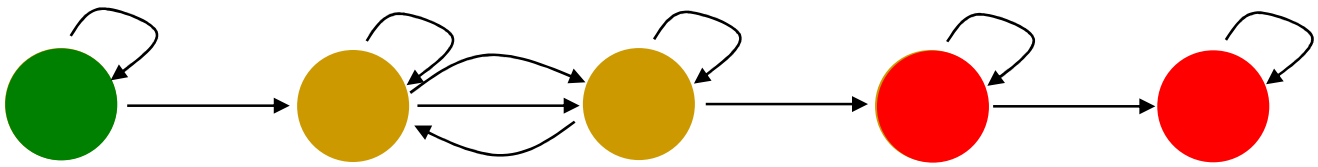


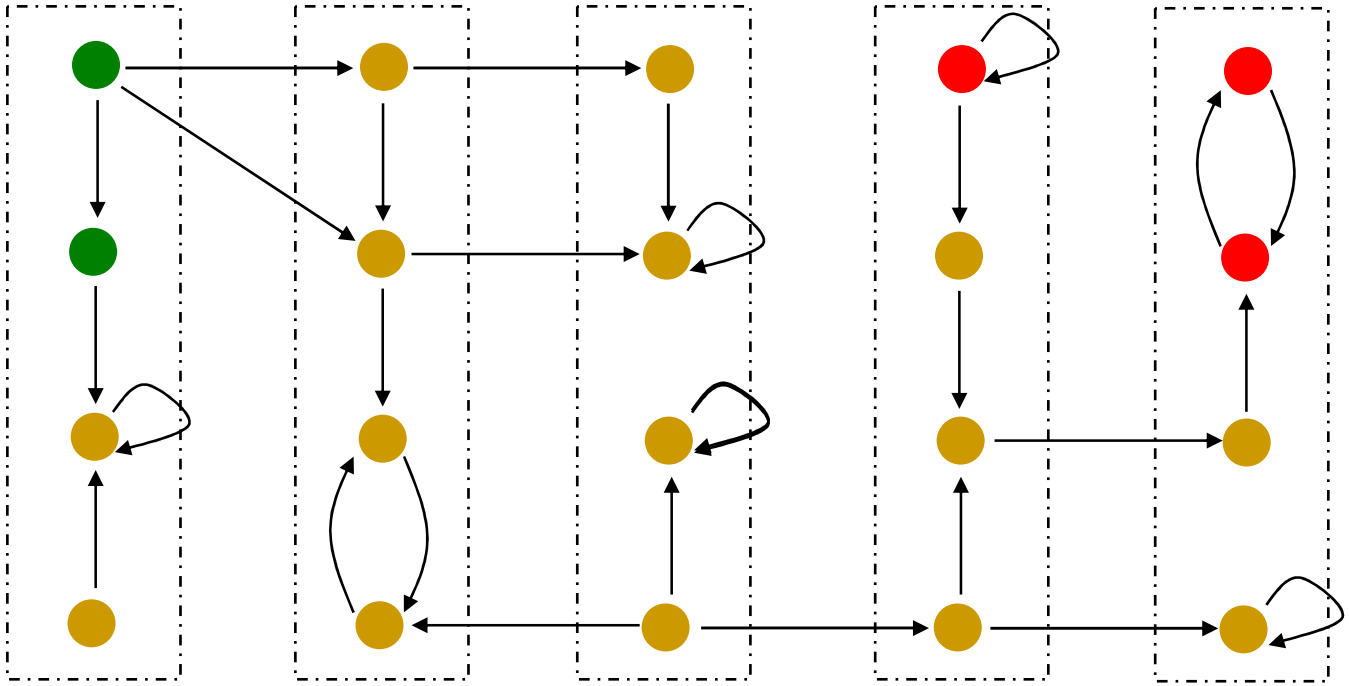
Preserves all the behaviors of the concrete model

Abstraction

- Preservation Theorem: If property holds on abstract model then property holds on concrete model
- Abstraction For Model Checking: Find a small abstract model on which the property holds

Abstraction





Abstract model

Try and find an abstraction function

Check if the abstraction is spurious

Check if the abstraction preserves behavior

Abstraction Functions

- Candidate abstraction functions are implicitly defined by the technique used for constructing abstract models
- Two popular techniques
  - Predicate Abstraction
  - Localization Abstraction

Localization Abstraction

- Partition state variables into visible ($\mathcal{V}$) and invisible ($\mathcal{I}$) variables
  - Intuitively, visible variables are the important variables
- Abstract model consists of only the visible variables
- Abstraction function maps a concrete state to its projection onto the visible variables

Abstraction Functions for Localization

$V = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$

$\mathcal{V} = \{x_1, x_2, x_4, x_6\}$

$\mathcal{I} = \{x_3, x_5, x_7\}$

x_1	x_2	x_4	x_6	x_3	x_5	x_7
0	1	1	0	0	0	0
0	1	1	0	0	0	1
0	1	1	0	0	1	0
0	1	1	0	0	1	1
0	1	1	0	1	0	0
0	1	1	0	1	0	1
0	1	1	0	1	1	0
0	1	1	0	1	1	1

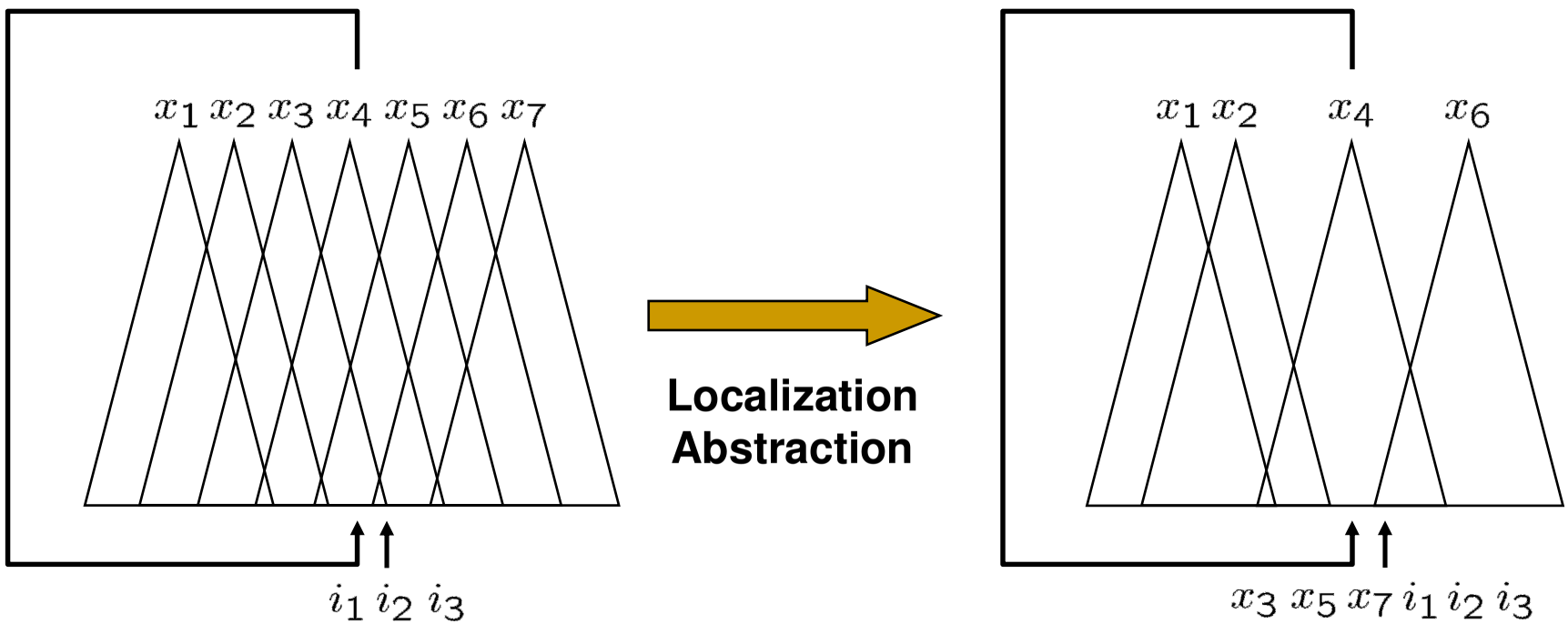
$\xrightarrow{h}$

x_1	x_2	x_4	x_6
0	1	1	0

Concrete states having the same value for visible variables are mapped to same abstract state

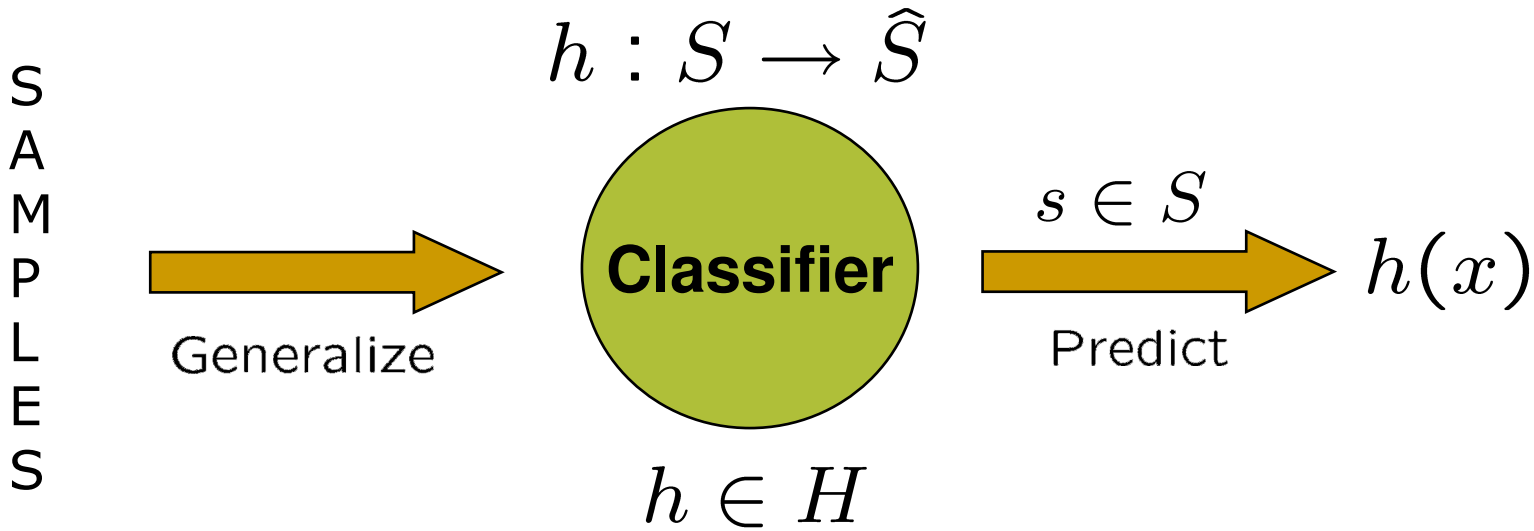
Localization Abstraction for Circuits

$V = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$
$\mathcal{V} = \{x_1, x_2, x_4, x_6\}$
$\mathcal{I} = \{x_3, x_5, x_7\}$



Hence the name localization

Abstraction ≡ Inductive Learning



Goal of abstraction is to learn an abstraction function that classifies the concrete states into abstract states while preserving the property

Inductive Bias of Abstraction

- Restriction Bias
  - Number of possible abstraction functions is huge
    - Circuit with n boolean variables
    - Number of ways to partition 2^n states into disjoint subsets
    - Bell Number

$$B_{2^n} \gg 2^{2^n}$$

  - Number of candidate functions is usually much smaller
    - Localization Abstraction: 2^n abstraction functions
  - Captures domain knowledge: Property is localizable

- Preference Bias
  - Smaller abstract models are better

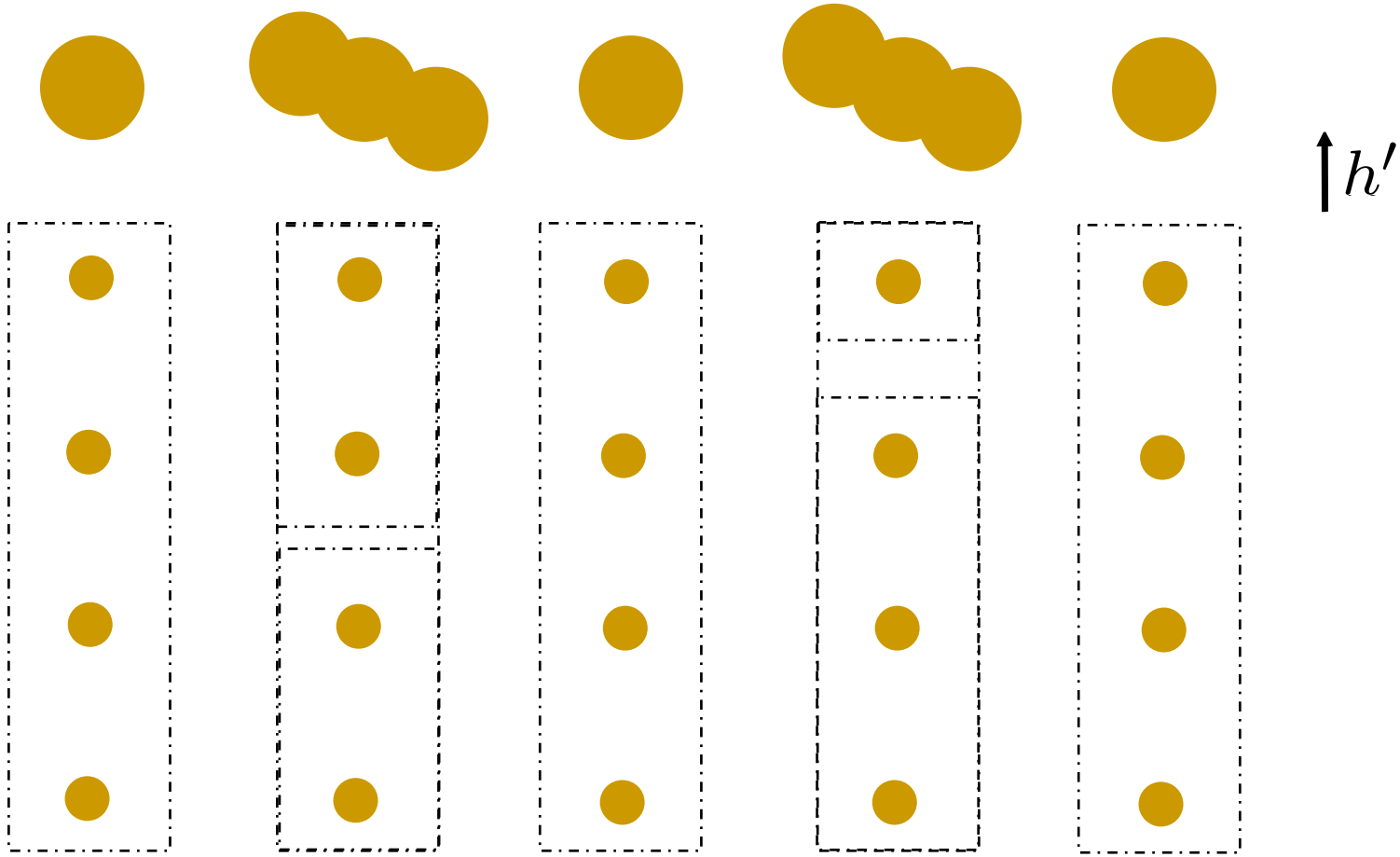
Samples

- What are the samples ?
- How are the samples generated ?
- How is the abstraction function computed from the samples ?

Outline

- Machine Learning
- Abstraction
- Learning and Abstraction-Refinement
- Learning Abstractions without Refinement

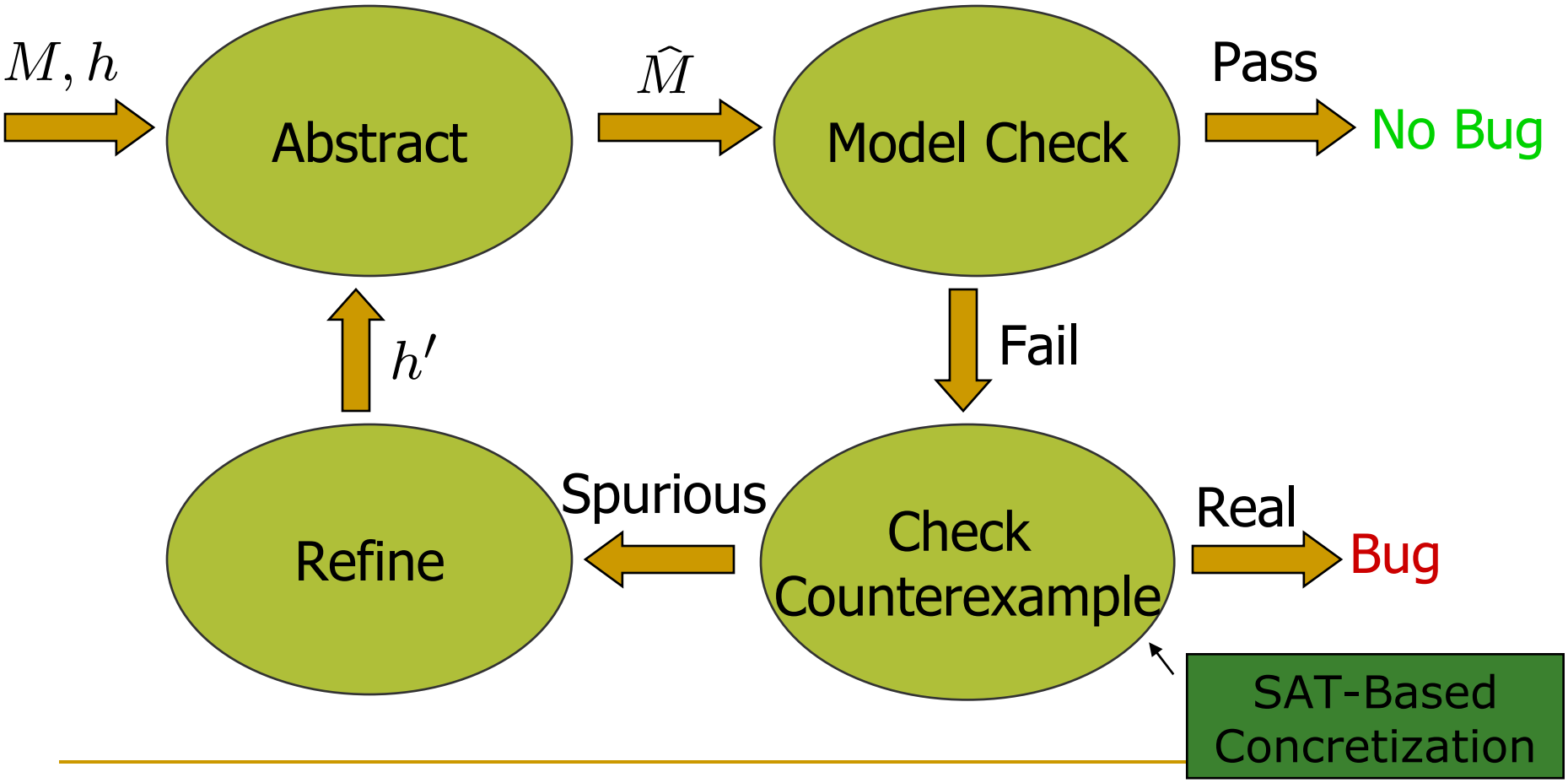
Refinement



For localization, refinement corresponds to making more variables visible

h' is a refinement of h

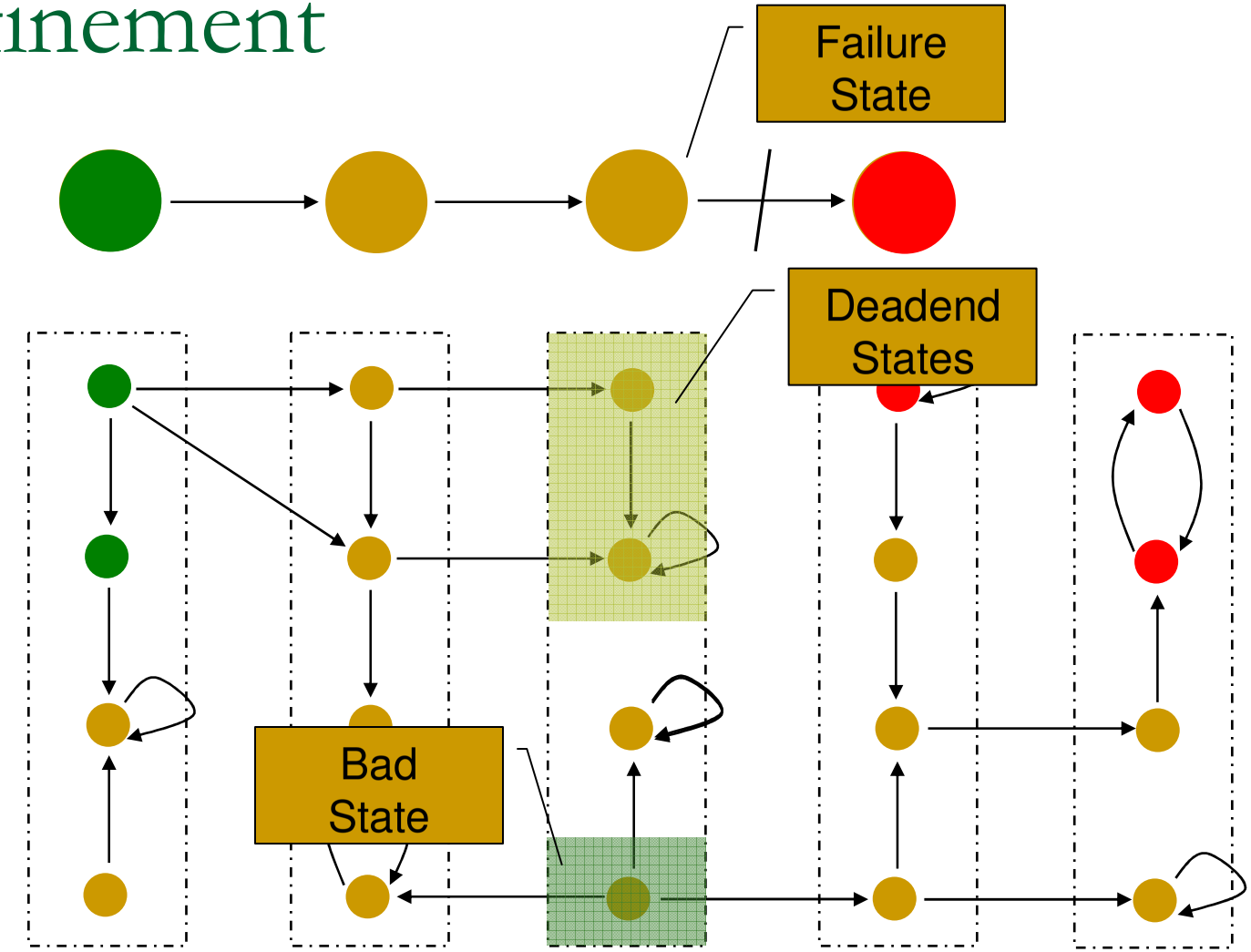
Abstraction-Refinement Loop



Many Refinement Heuristics

- Identifying conflicting latches with 3-valued simulation of counterexample [Wang et. al.]
- Identify common variable assignments across multiple counterexamples [Glusman et. al.]
- SAT-Proof based refinement [Chauhan et. al.]
- Refinement by failure-state splitting [Yuan Lu et. al.]

Refinement

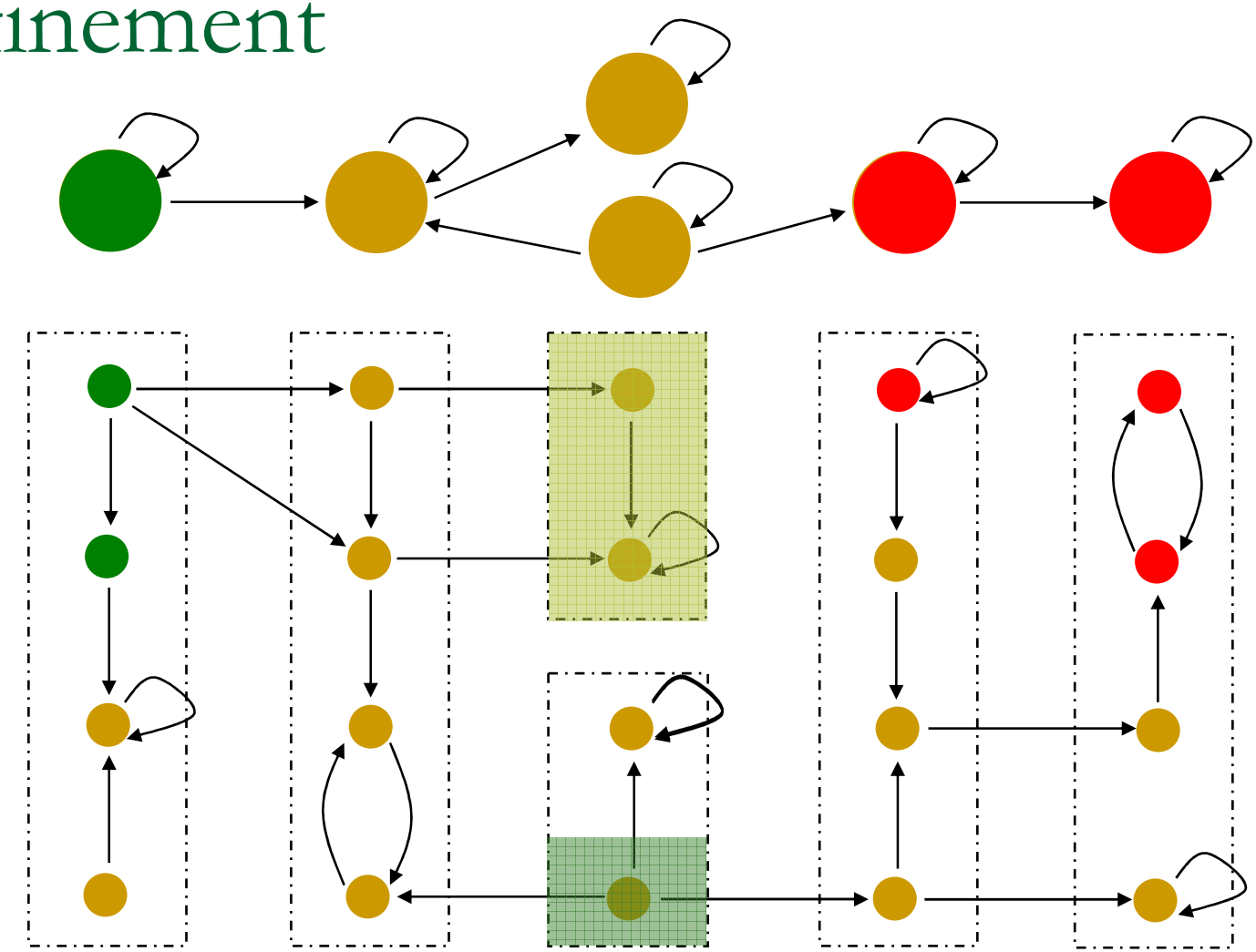


Put $\mathcal{B}(s_f) = \{R(s_f, s_{f+1}) \wedge \mathcal{C}_f(s_f) \wedge \mathcal{C}_{f+1}(s_{f+1})\}$:ates

$f-1$ f

$i=1$ $i=1$

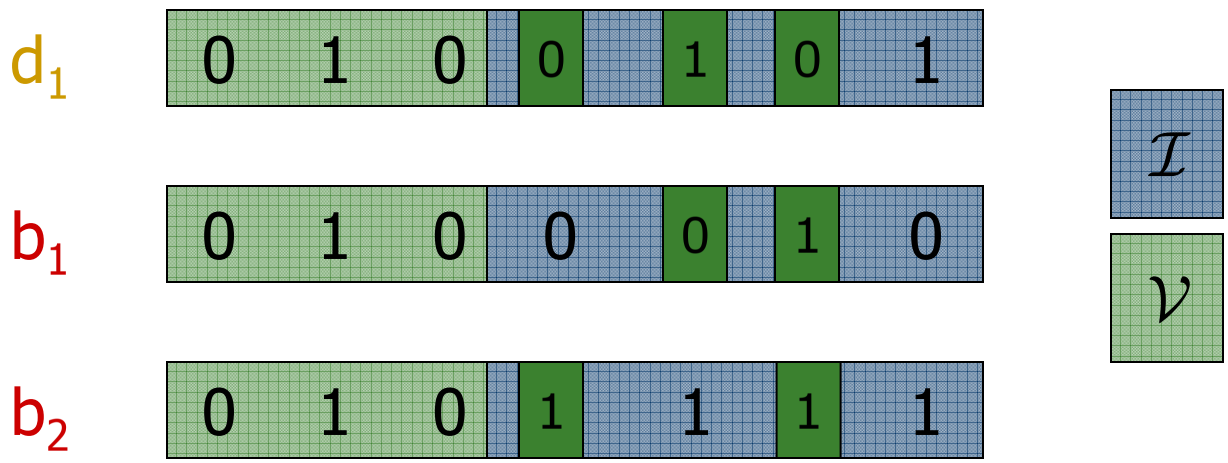
Refinement



Put deadend and bad states in separate abstract states

Propagate splits

State-Separation Problem



Refinement: Find subset $\mathcal{U}$ of $\mathcal{I}$ that separates all pairs of deadend states separate the state visible

Find the smallest set

A Simple Approach

- Generate all the deadend and bad states
  - Explicitly
  - Symbolically
- Compute the separating set from these
- Previous work [Yuan Lu et. al.] generated BDDs for deadend and bad states

- Infeasible for large systems

Sampling

- Learn the separating set from samples of deadend and bad states
- Use SAT-solvers to generate multiple samples efficiently

Learning and Abstraction-Refinement

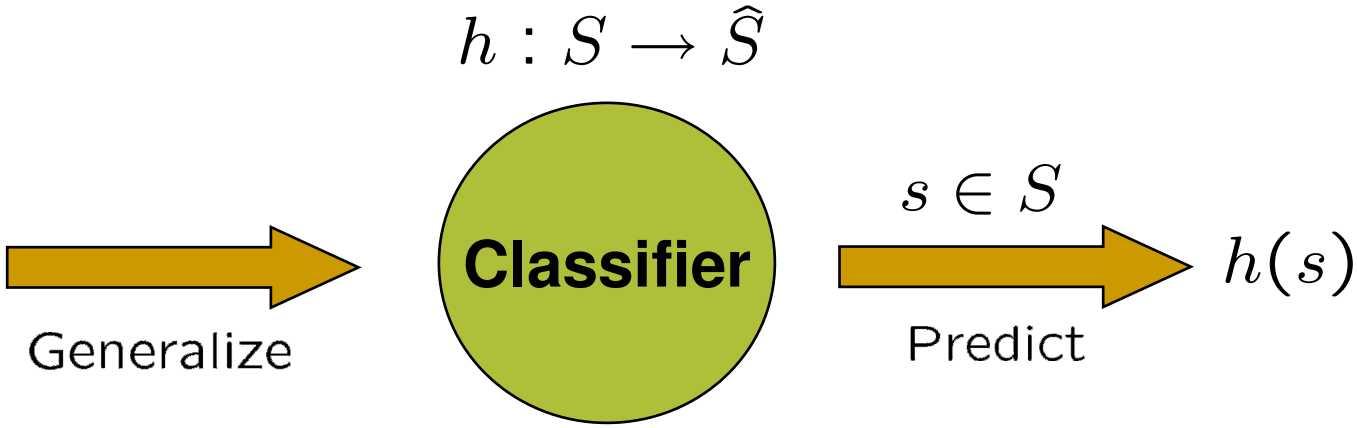
$S_D \cup S_B$

d_1 b_1

d_2 b_2

$\vdots$ $\vdots$

d_p b_q



$\forall d \in D. b \in B.\ h(d) \neq h(b)$

Computing the Separating Set

- Integer Linear Programming (ILP)
  - Smallest separating set
  - Computationally expensive
- Decision Tree Learning
  - Computationally efficient
  - Non-optimal

Computing Separating Set using ILP

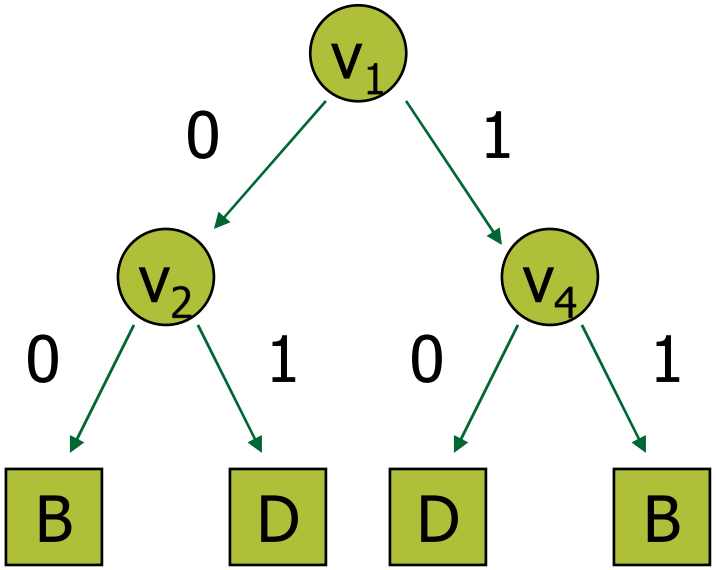
$$\text{Min } \sum_{i=1}^{|\mathcal{I}|} v_i$$

$$\text{subject to:} \quad (\forall d \in S_D)\ (\forall b \in S_B) \quad \sum_{\substack{1 \le i \le |\mathcal{I}|, \\ d,b \text{ differ at } v_i}} v_i \ge 1$$

$v_i = 1$ means that v_i is in the separating set

Computing Separating Set using Decision Tree Learning

- Decision Tree Learning constructs a decision tree that classifies a set of samples using a set of attributes
- Samples: $S_D \cup S_B$
- Attributes: $\mathcal{I}$
- ID3 algorithm
  - Construct small tree
- Separating set consists of variables on the nodes of the decision tree



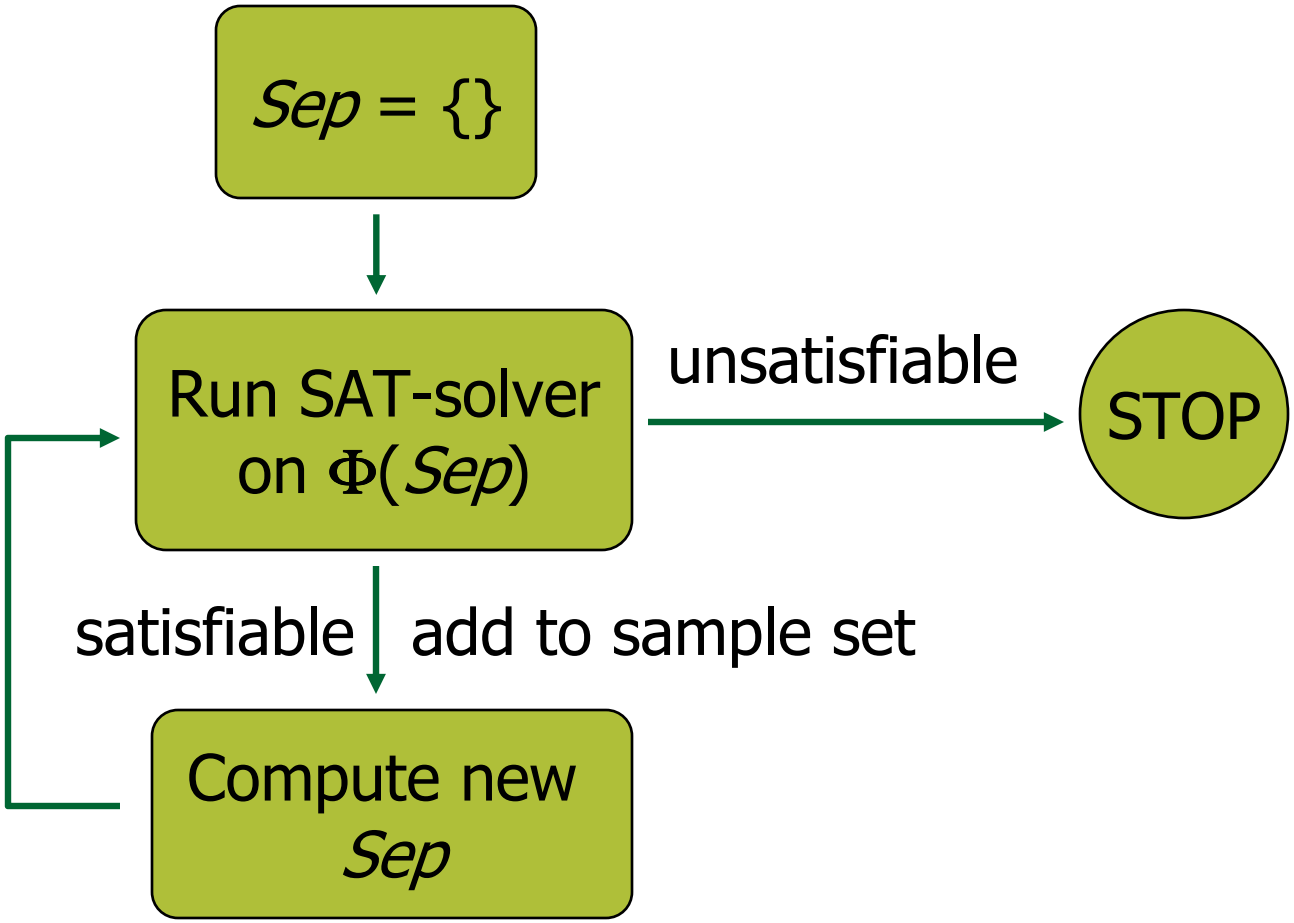
Separating Set
$\{v_1, v_2, v_4\}$

Generating Samples

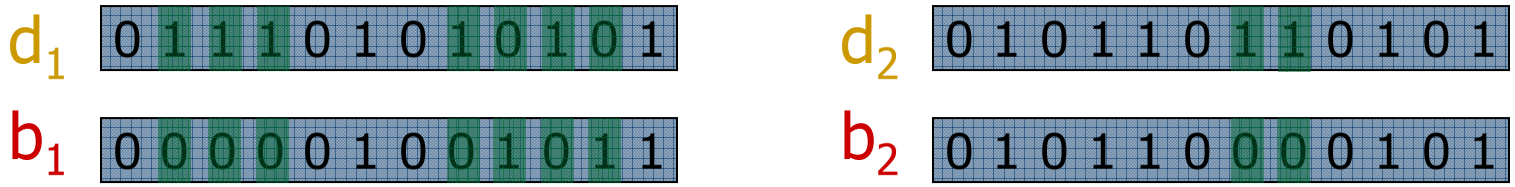
- Random Sampling
  - Generate multiple satisfying assignments using SAT-solver on $\mathcal{D}$ and $\mathcal{B}$
- Equivalence Queries
  - Query the teacher for samples that are not separated by the current separating set
  - Teacher:

$$\Phi(Sep) \doteq \mathcal{D}(v_i) \wedge \mathcal{B}(v_i') \wedge \bigwedge_{v_i \in Sep} v_i = v_i'$$

Sampling with Equivalence Queries



Generating Good Samples



- Deadend and bad state pairs that differ in small number of variables are good
  - Eliminate a larger portion of the search space
  - Faster convergence to the separating set
- Can be formulated as optimization problem with Pseudo-Boolean Constraints
  - Solved with Pseudo-Boolean Solver (PBS)

Metrics for Quality of Abstract Models

- Number of State Variables
- Number of Gates
- Number of Inputs

Experimental Evaluation

- AbsRef Tool
  - Implemented inside NuSMV
  - SAT-solver: zChaff
  - ILP-solver: lpsolve
  - Model Checker: Cadence SMV
- Compared with
  - BDD-based Model Checking (Cadence SMV)
  - SAT-Proof based refinement [Chauhan et. al.]

Results

Circuit	SMV	Rand, ILP			Rand, DTL			EqvQ, DTL			Chauhan			EqvQ, Inp		
	Time	Time	S	L	Time	S	L	Time	S	L	time	S	L	time	S	L
$IU30$	7.3	8.0	3	20	7.5	3	20	6.5	3	20	1.9	4	20	5.56	3	20
$IU35$	19.1	11.8	4	21	12.7	4	21	11.0	4	21	10.4	5	21	22.58	4	21
$IU40$	53.6	25.9	6	23	19.0	5	22	16.1	5	22	13.3	6	22	33.78	5	22
$IU45$	226.1	28.3	5	22	25.3	5	22	22.1	5	22	25	6	22	38.9	5	22
$IU50$	1754	160.4	13	32	85.1	10	27	15120	7	31	32.8	6	22	57.39	5	22
$IU55$	-	-	-	-	-	-	-	-	-	-	61.9	4	20	58.94	3	20
$IU60$	-	-	-	-	-	-	-	-	-	-	65.5	4	20	76.74	3	20
$IU65$	-	-	-	-	-	-	-	-	-	-	67.5	4	20	79.99	3	20
$IU70$	-	-	-	-	-	-	-	-	-	-	71.4	4	20	69.39	3	20
$IU75$	-	1080	21	38	586.7	16	33	130.5	5	26	15.7	5	21	22.59	4	21
$IU80$	-	1136	21	38	552.5	16	33	153.4	5	26	21.1	5	21	25.61	4	21
$IU85$	-	1162	21	38	581.2	16	33	167.7	5	26	24.6	5	21	27.5	4	21
$IU90$	-	965	20	37	583.3	16	33	167.1	5	26	24.3	5	21	27.96	4	21

		Chauhan			EqvQ, Inp		
Design	Length	Time	S	L	Time	S	L
M9	TRUE	10.2	2	38	**2.9**	1	38
M6	TRUE	44.3	4	50	**18.8**	4	50
M16	TRUE	1162	61	35	**44.7**	3	34
M17	TRUE	-			**733**	8	39
D6	20	**917**	46	89	1773	43	92
IUp1	TRUE	**3350**	13	19	-	9	41

Outline

- Machine Learning
- Abstraction
- Learning and Abstraction-Refinement
- Learning Abstractions without Refinement

Many Refinement Heuristics

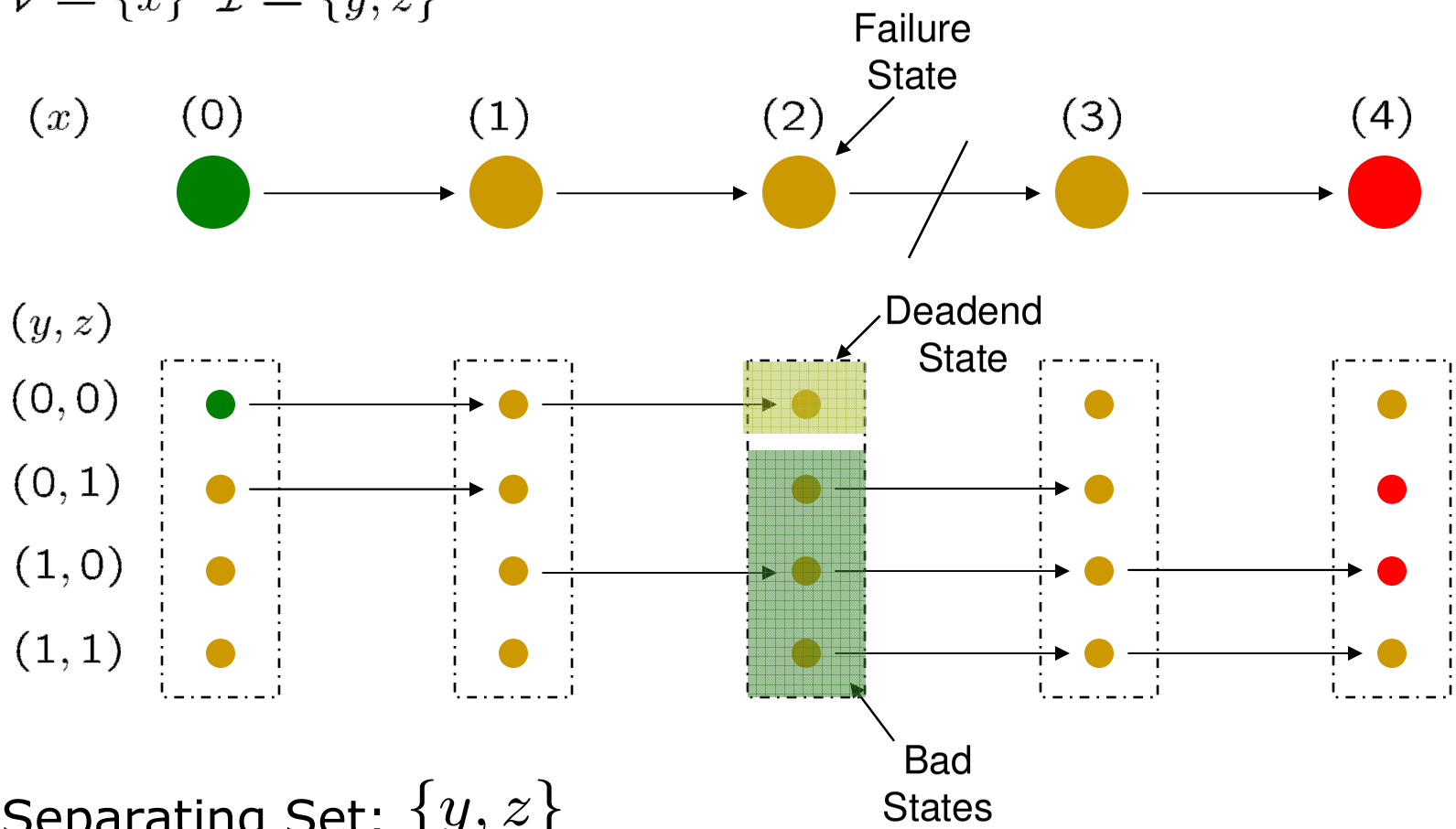
- Identifying conflicting latches with 3-valued simulation of counterexample [Dong Wang et. al.]
- Ide All of these are Heuristics ! cross multiple counterexamples [Glusman et. al.]
- SAT-Proof based refinement [Chauhan et. al.]
- Refinement by failure-state analysis [Yuan Lu et. al.]

Many Refinement Heuristics

- Identifying conflicting latches with 3-valued simulation of counterexample [Dong Wang et. al.]
- Ide All of these are Heuristics ! cross multiple counterexamples [Glusman et. al.]
- SAT-Proof based refinement [Chauhan et. al.]
- Refinement by failure-state analysis [Yuan Lu et. al.]

Drawbacks of Failure-State Splitting

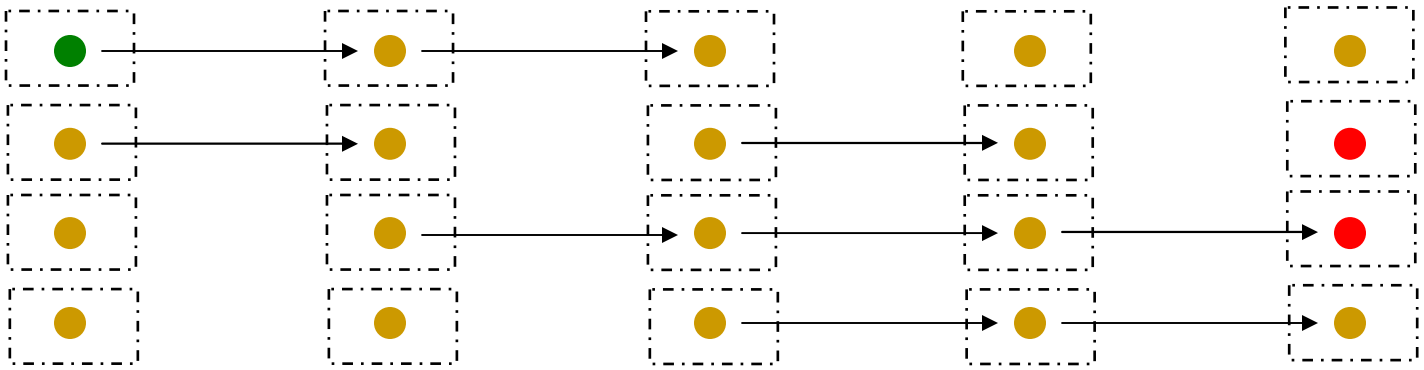
$\mathcal{V} = \{x\}$ $\mathcal{I} = \{y, z\}$



Separating Set: $\{y, z\}$

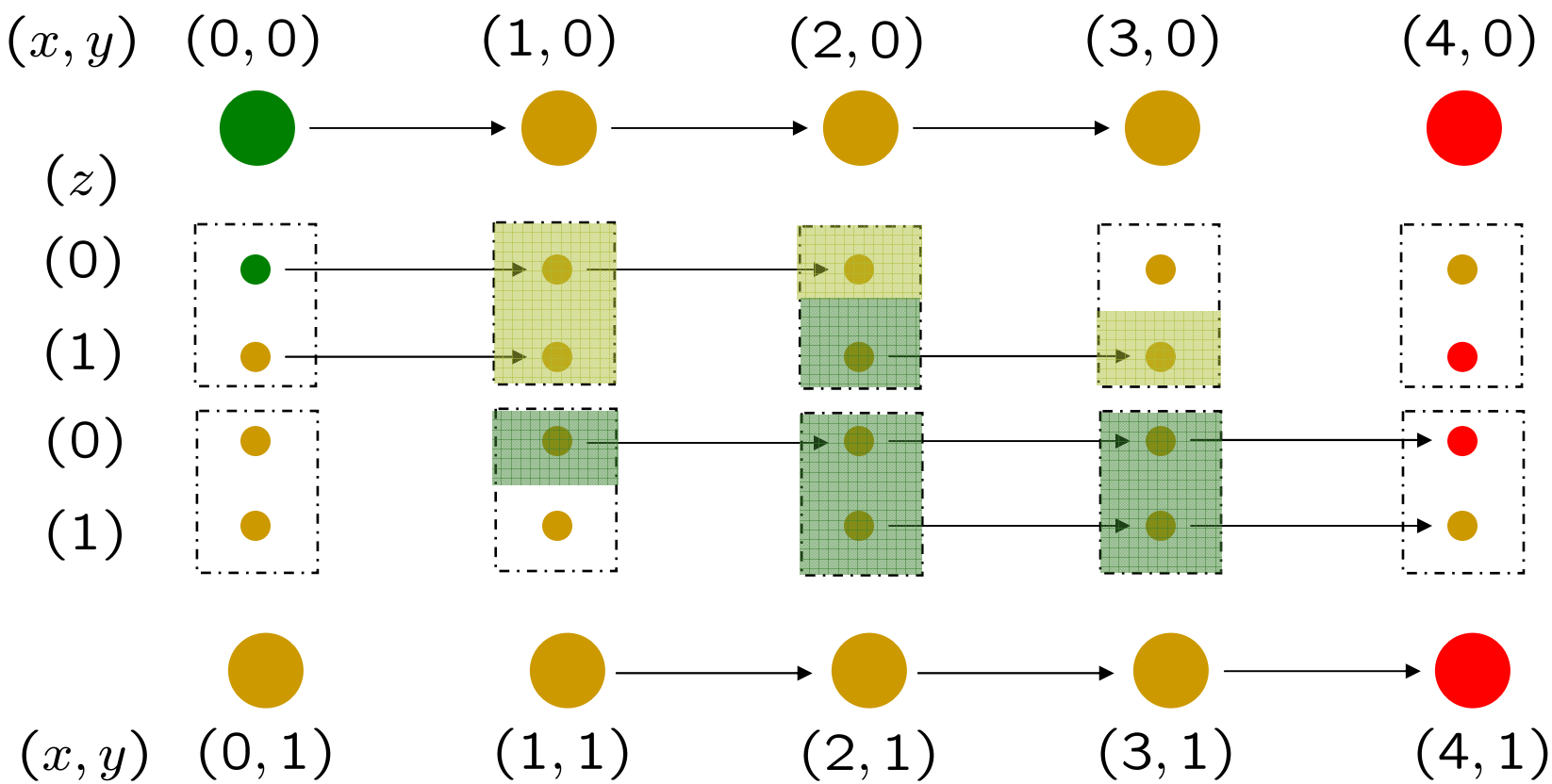
Drawbacks of Failure-State Splitting

$\mathcal{V} = \{x, y, z\}$



Drawbacks of Failure-State Splitting

$\mathcal{V} = \{x, y\}$



Drawbacks of Abstraction-Refinement

- Adds details to abstract model; never removes anything
  - Information added to eliminate counterexample might also eliminate previously seen counterexample
- Does not look at many counterexamples of different lengths simultaneously
  - Abstract model depends on what counterexamples are considered and in what order
- Abstraction-Refinement cannot find the smallest abstract model
- This drawback is present no matter what heuristic is used to compute the refinement

What is needed ?

- We need a strategy of eliminating spurious behavior that is not heuristic
- We need a strategy that is not based on refinement
- We need a strategy that analyzes all the counterexamples simultaneously

Broken Traces

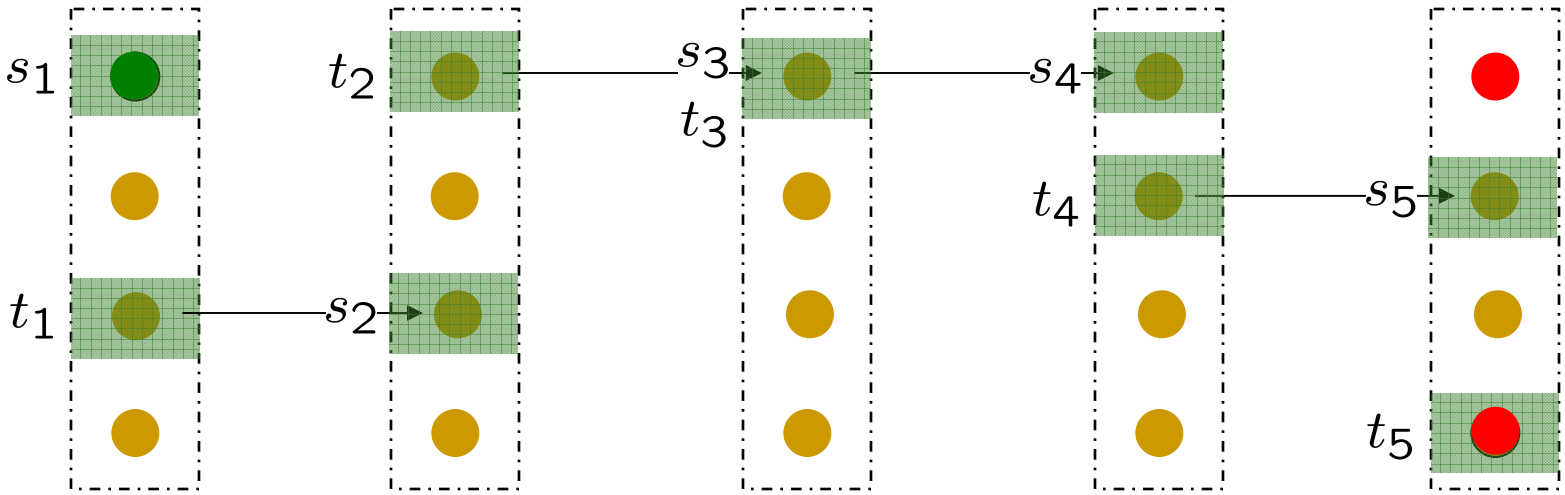
- Broken Traces on concrete model corresponding to an abstraction function
- Sequence of *k* pairs of concrete states

$$\langle (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k) \rangle$$

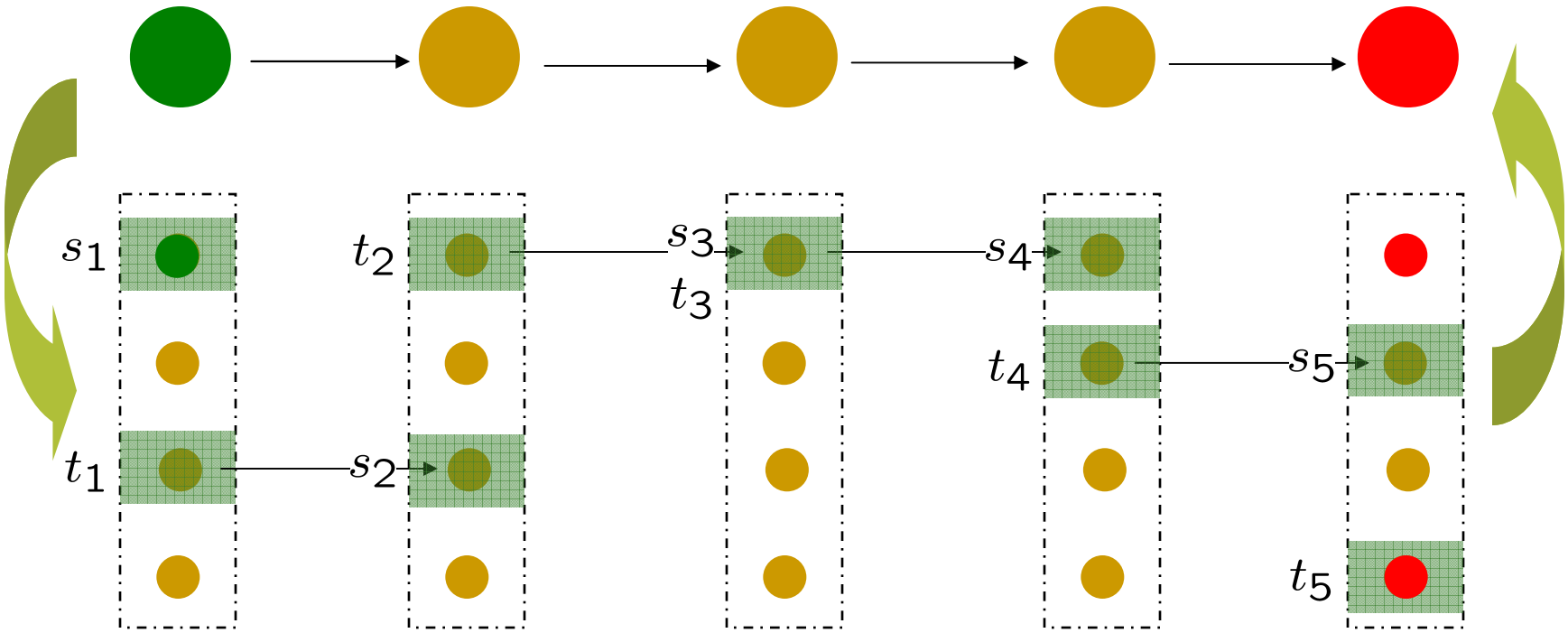
- Each s_i and t_i map to same abstract state
- s_1 is an initial state
- t_k is an error state
- Each $t_i \rightarrow s_{i+1}$ is a concrete transition
- Break at *i* if $s_i \neq t_i$. No breaks = Real bug

Broken Traces

$$\langle (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4), (s_5, t_5) \rangle$$



Broken Traces And Abstract Counterexamples

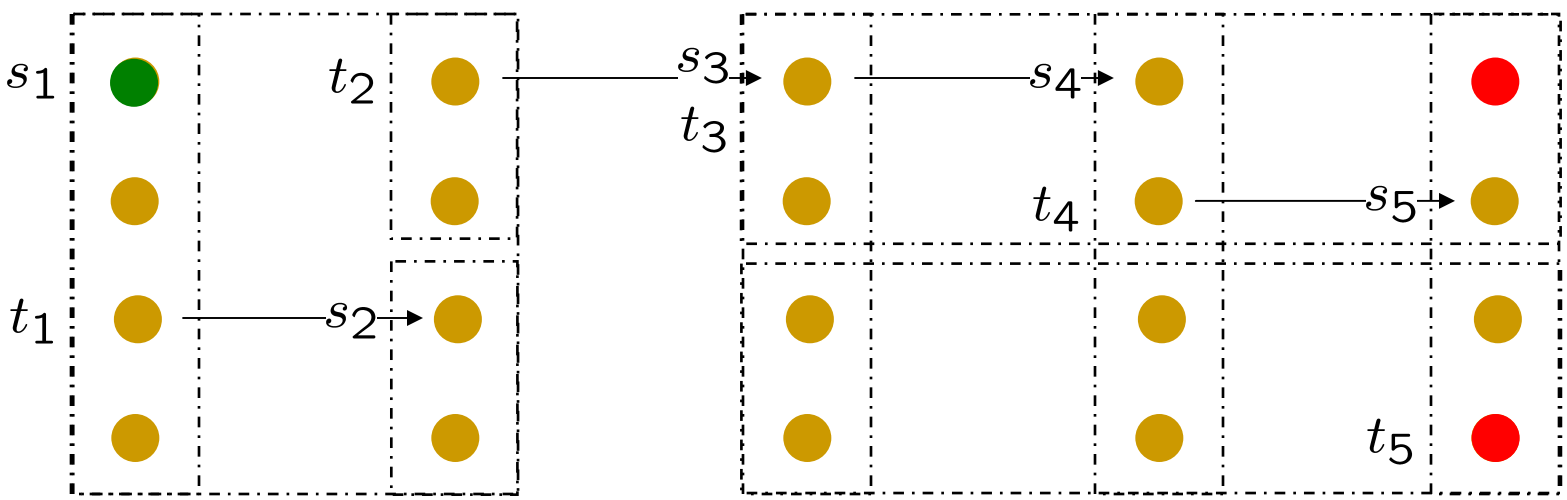


Broken Traces and Abstract Counterexamples

- Broken Trace Theorem: There is a counterexample on the abstract model if and only if there is a corresponding broken trace on the concrete model

Eliminating Broken Traces

- Abstraction function eliminates a broken trace if it maps some s_i and t_i into separate abstract states



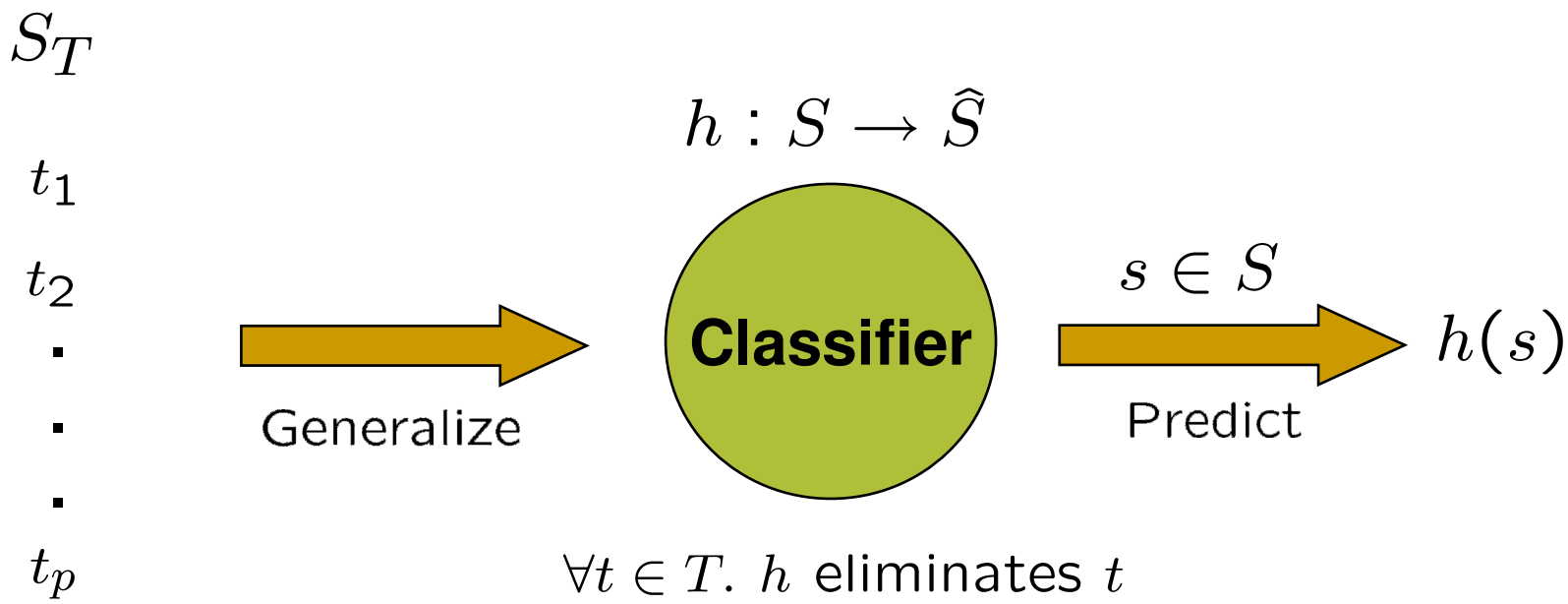
Our Abstraction Strategy

- Find an abstraction function that eliminates all broken traces
- The smallest abstract model that eliminates all broken traces is the smallest abstract model that can prove the property

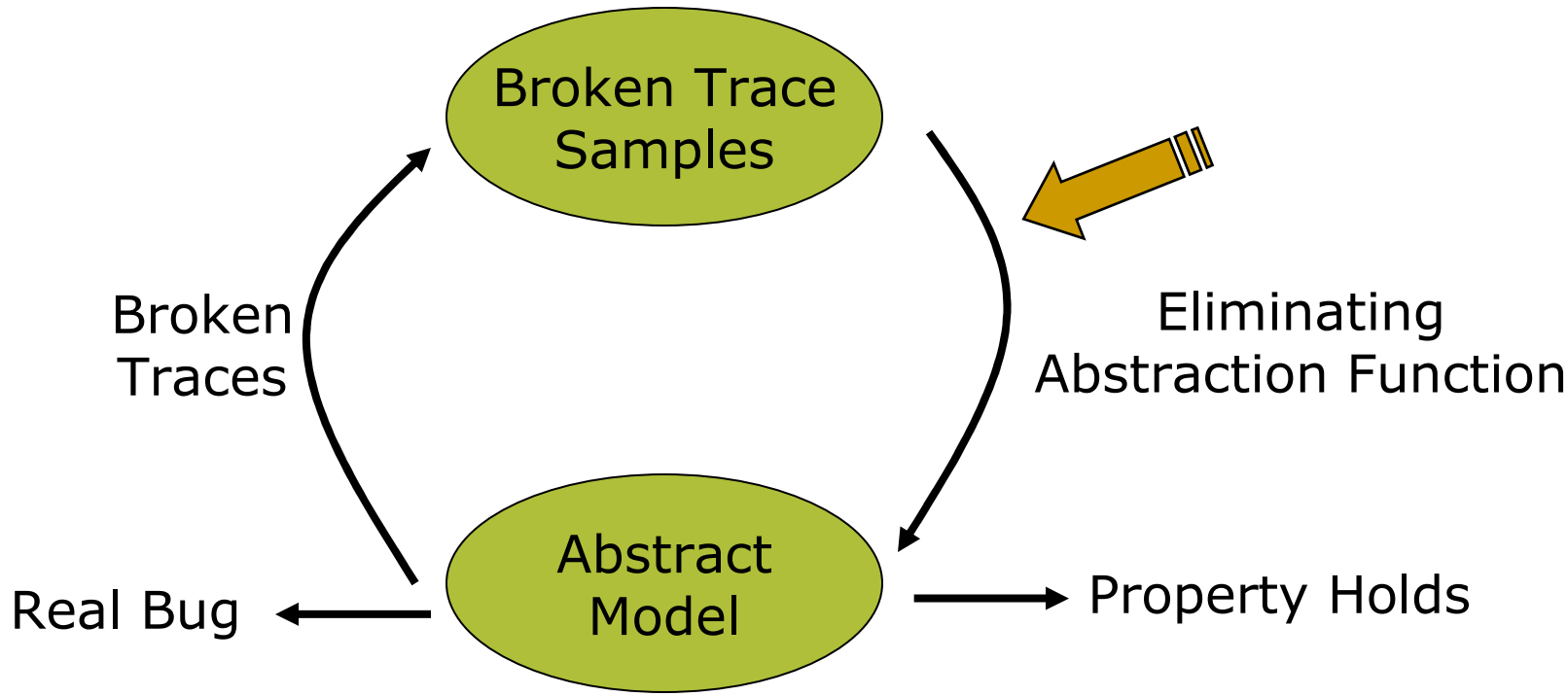
Sampling

- Computationally infeasible to generate all broken traces and eliminate them
- Learn the abstraction function from samples of broken traces
- Use abstract counterexamples to guide the search for broken trace samples

Learning Abstractions without Refinement

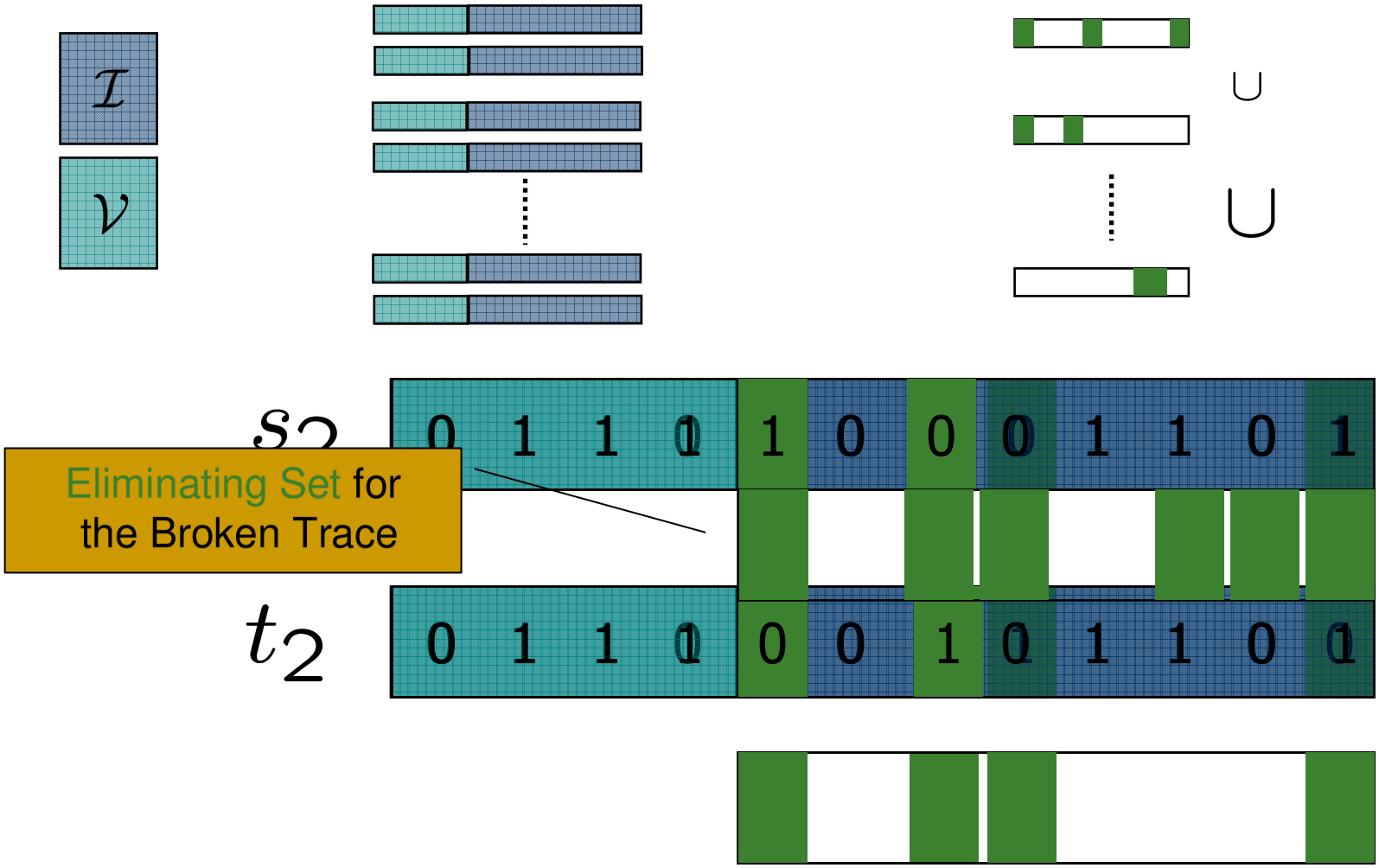


Learning Abstractions (LearnAbs)

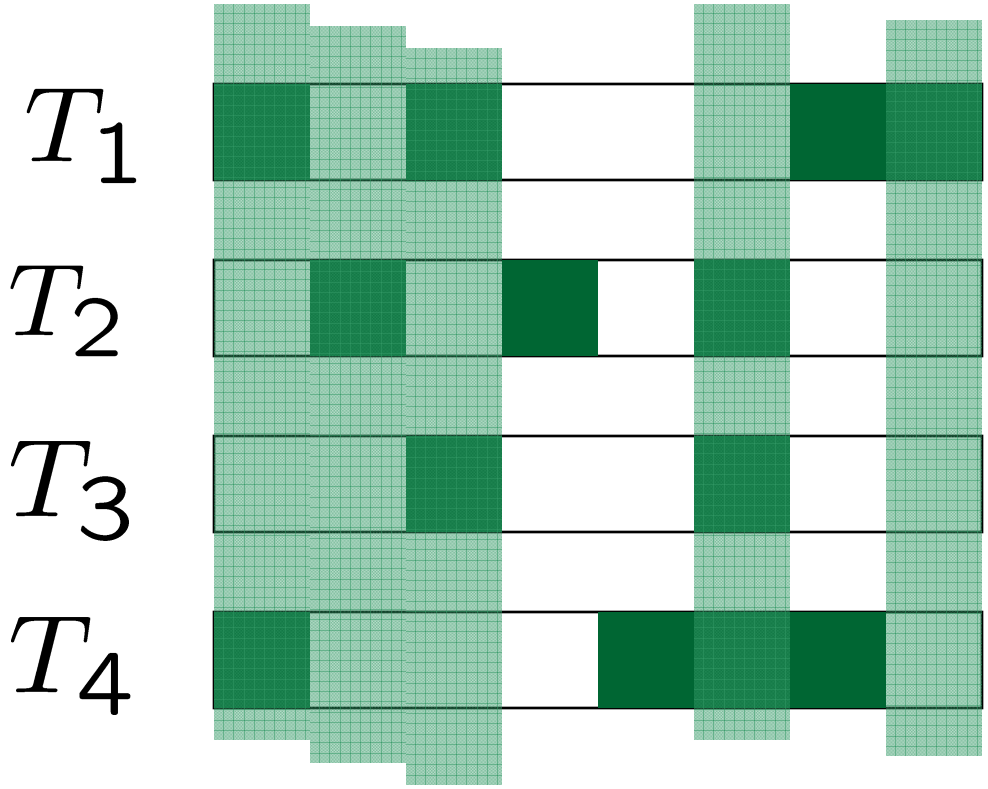


Computing the Eliminating Model

$\langle (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k) \rangle$



Computing Eliminating Model



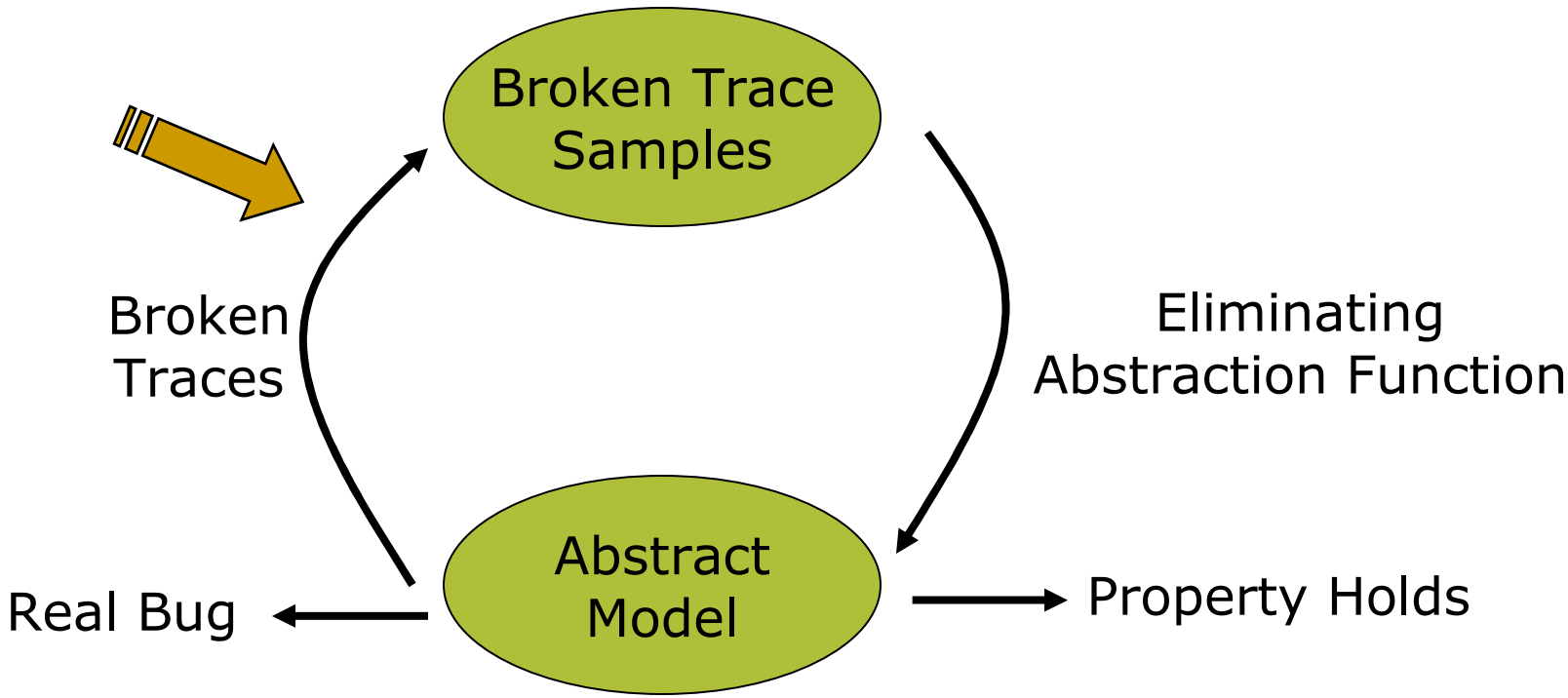
Find subset $\mathcal{V}$ of variables that hits the eliminating set of all broken trace samples

Find smallest $\mathcal{V}$

Computing Eliminating Model

- Minimum Hitting Set
  - Can be formulated as an Integer Linear Program
  - Smallest Eliminating Model
- Approximate algorithms
  - Faster but non-optimal

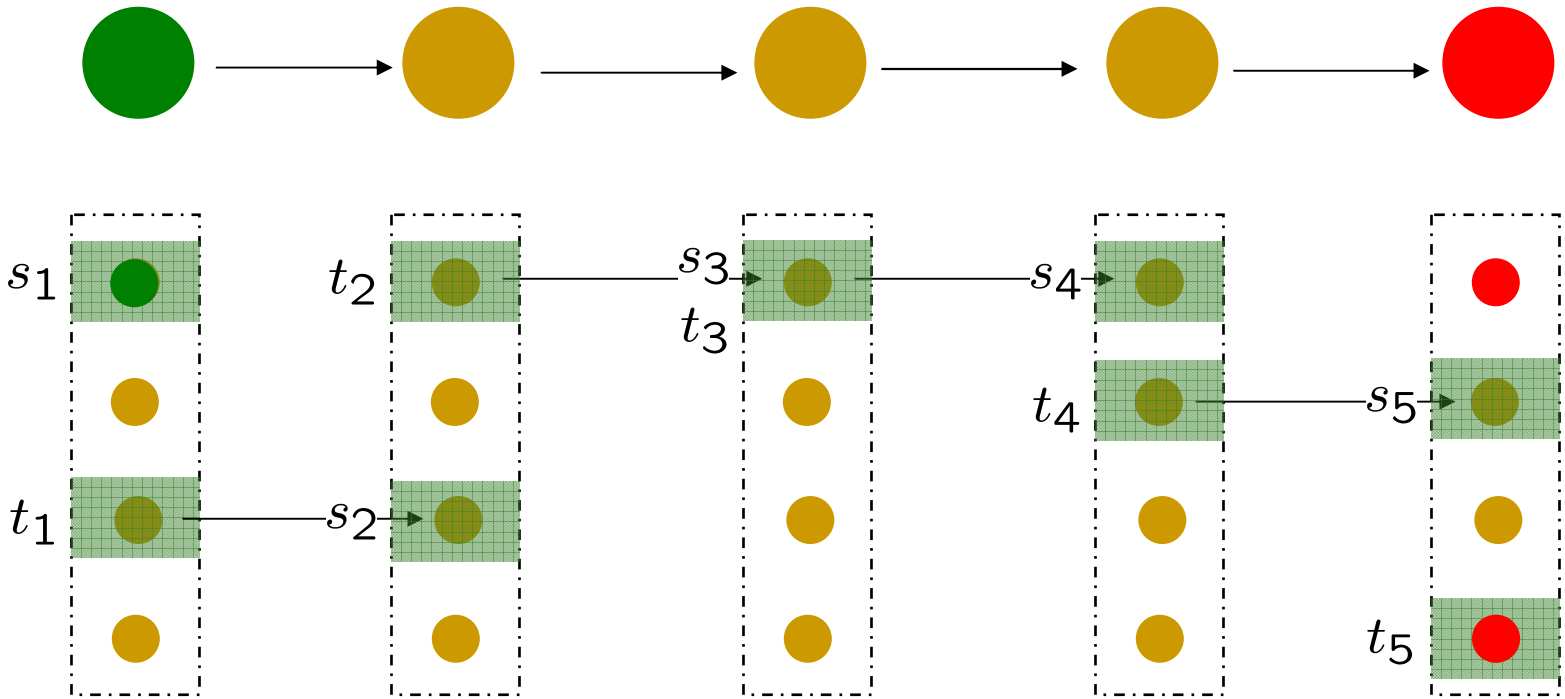
Learning Abstractions (LearnAbs)



SAT with Hints

- SAT-solver modified to produce a satisfying assignment that is close to a given partial assignment (hint)
  - SAT-solver is forced to first make decisions corresponding to the hint

Generating Broken Traces



- Use SAT with hints
  - Hints from previous state
- Break if necessary
- No expensive BMC unfolding

Experimental Evaluation

- LEARNABS Tool
  - Input: Bit-level SMV net-lists
  - SAT-solver: zChaff
  - ILP-solver: CPLEX
  - Model Checker: Cadence SMV
- Compared with
  - SAT-Proof based abstraction [Chauhan et. al., McMillan et. al.]
    - Single Counterexample (S) mode: Model Checker called after abstract counterexample is eliminated
    - All Counterexamples (A) mode: Model Checker called after all counterexamples of current length are eliminated

Results

circuit	reg	cex	SATProof (S)			LearnAbs(S)			SATProof (A)			LearnAbs(A)		
			time	itr	abs	time	itr	abs	time	itr	abs	time	itr	abs
PJ00	348	T	7	3	9	6	3	4	7	3	9	6	3	4
PJ01	321	T	6	3	3	5	2	0	6	3	3	5	2	0
PJ02	306	T	7	3	4	6	3	4	7	3	4	6	3	4
PJ03	306	T	6	3	4	5	3	4	6	3	4	6	3	4
PJ04	305	T	7	3	3	5	2	0	6	3	3	5	2	0
PJ05	105	T	9	7	34	35	8	28	9	7	34	34	8	28
PJ06	328	T	209	7	56	23	7	41	210	7	56	24	7	41
PJ07	94	T	18	11	36	13	7	32	20	10	36	14	7	32
PJ08	116	T	58	6	41	75	7	41	58	6	41	76	7	41
PJ09	71	T	8	6	31	4	6	25	8	6	31	4	6	25
PJ10	85	T	3	3	5	2	3	4	3	3	5	2	3	4
PJ11	294	T	11	5	12	5	2	0	12	5	12	5	2	0
PJ12	312	T	4	1	0	4	1	0	4	1	0	4	1	0
PJ13	420	T	10	5	13	8	3	8	10	5	13	8	3	8
PJ14	127	T	25	6	32	9	4	13	35	5	32	9	4	13
PJ15	355	T	184	5	42	14	5	23	185	5	42	15	5	23
PJ16	290	T	248	6	44	64	7	32	246	6	44	65	7	32
PJ17	212	T	2126	14	43	1869	20	29	5037	11	43	3685	14	31
PJ18	145	T	993	22	49	390	8	32	161	7	45	542	7	36
PJ19	52	T	>2hr	-	-	18	3	12	>2hr	-	-	19	3	12

Results

circuit	reg	cex	SATProof (S)			LearnAbs(S)			SATProof (A)			LearnAbs(A)		
			time	itr	abs	time	itr	abs	time	itr	abs	time	itr	abs
RB05_1	313	31	11	10	24	141	17	13	17	6	12	137	11	13
RB09_1	168	T	1	4	9	1	3	4	1	4	9	1	3	4
RB10_1	236	T	3	5	9	1	4	3	3	5	9	1	4	3
RB10_2	236	T	3	6	11	2	4	3	3	6	11	2	4	3
RB10_3	236	T	5	7	34	2	5	5	5	7	34	1	5	5
RB10_4	236	T	8	10	23	1	4	5	6	8	22	2	4	5
RB10_5	236	T	1	4	7	2	4	4	2	4	7	2	4	4
RB10_6	236	T	3	5	7	2	4	4	2	5	7	2	4	4
RB11_2	242	T	>1hr	-	-	128	24	26	>1hr	-	-	219	14	30
RB14_1	180	T	37	7	47	3	5	15	37	7	47	3	5	15
RB14_2	180	T	>1hr	-	-	1258	60	37	334	11	87	179	13	38
RB15_1	270	9	2	7	8	17	9	4	2	7	8	13	9	4
RB16_1_1	1117	T	8	7	92	324	8	80	8	7	92	260	8	80
RB16_2_4	1113	5	4	5	40	61	5	32	4	5	40	48	5	32
RB26_1	608	T	1	1	0	1	1	0	1	1	0	1	1	0
RB31_2_1	111	T	>1hr	-	-	38	27	29	>1hr	-	-	17	12	26
IUP1	4494	T	*mem*	-	-	1295	18	8	*mem*	-	-	151	13	5

Conclusion

- This work has shown the viability of using machine learning techniques to improve abstraction-based model checking
  - Machine learning techniques help the model checker to efficiently identify the relevant information in the model