Specification Mining:

New Formalisms, Algorithms and Applications

Wenchao Li

Dissertation Talk
EECS Department, UC Berkeley

Thesis Committee: Sanjit A. Seshia (Advisor), Andreas Kuehlmann, Francesco Borrelli

Acknowledgement

- Prof. Sanjit A. Seshia
- Prof. Andreas Kuehlmann, Prof. Francesco Borrelli
- Dr. Alessandro Forin (MSR, Redmond)
- Dr. Natarajan Shankar, Dr. Shalini Ghosh (SRI International, Menlo Park)
- GigaScale Research Center
- TerraSwarm Research Center
- Daniel Holcomb, Bryan Brady, Susmit Jha, Alexandre Donze, Ruediger Ehlers, Indranil Saha, Jonathan Kotker, Rohit Sinha, Dorsa Sadigh, Zach Wasson, Wei Yang Tan, Garvit Juniwal, Ankush Desai, Nishant Totla, Daniel Fremont
- Colleagues in the DOP center
- Friends
- Nuo Zhang

Tiny bugs can have catastrophic consequences

Ubiquitous computing results in ubiquitous bugs

Formalization of requirement helps finding bugs, but is hard

Cost of Bugs

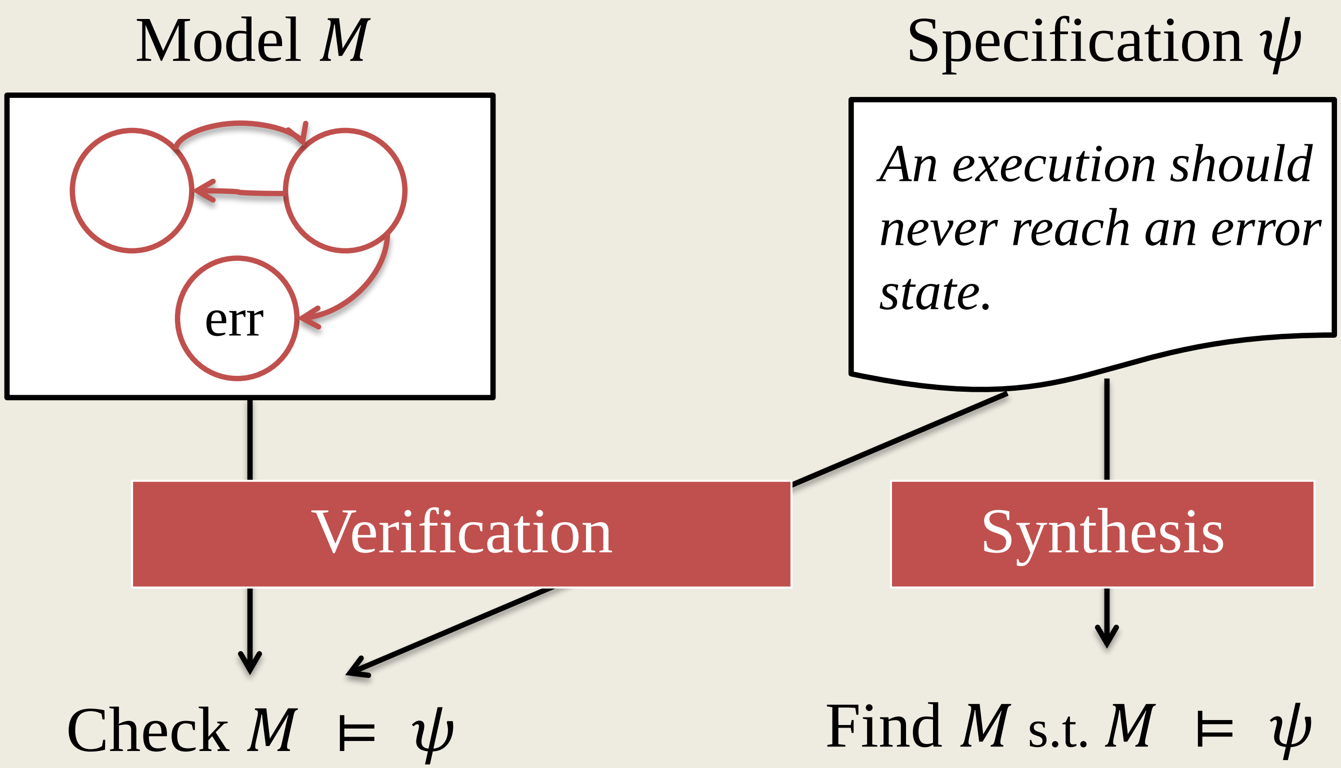
- ***Human loss***: Pacemakers, Aircraft, Nuclear reactor controllers, Car engine management system, etc.
- ***Financial Loss***: 1994 Pentium FDIV costs $475 million, Mars Rover, North America Blackout, etc.

Much of the challenge in bug finding lies in finding the specification that mechanized tools can use to find bugs

Reality Check:

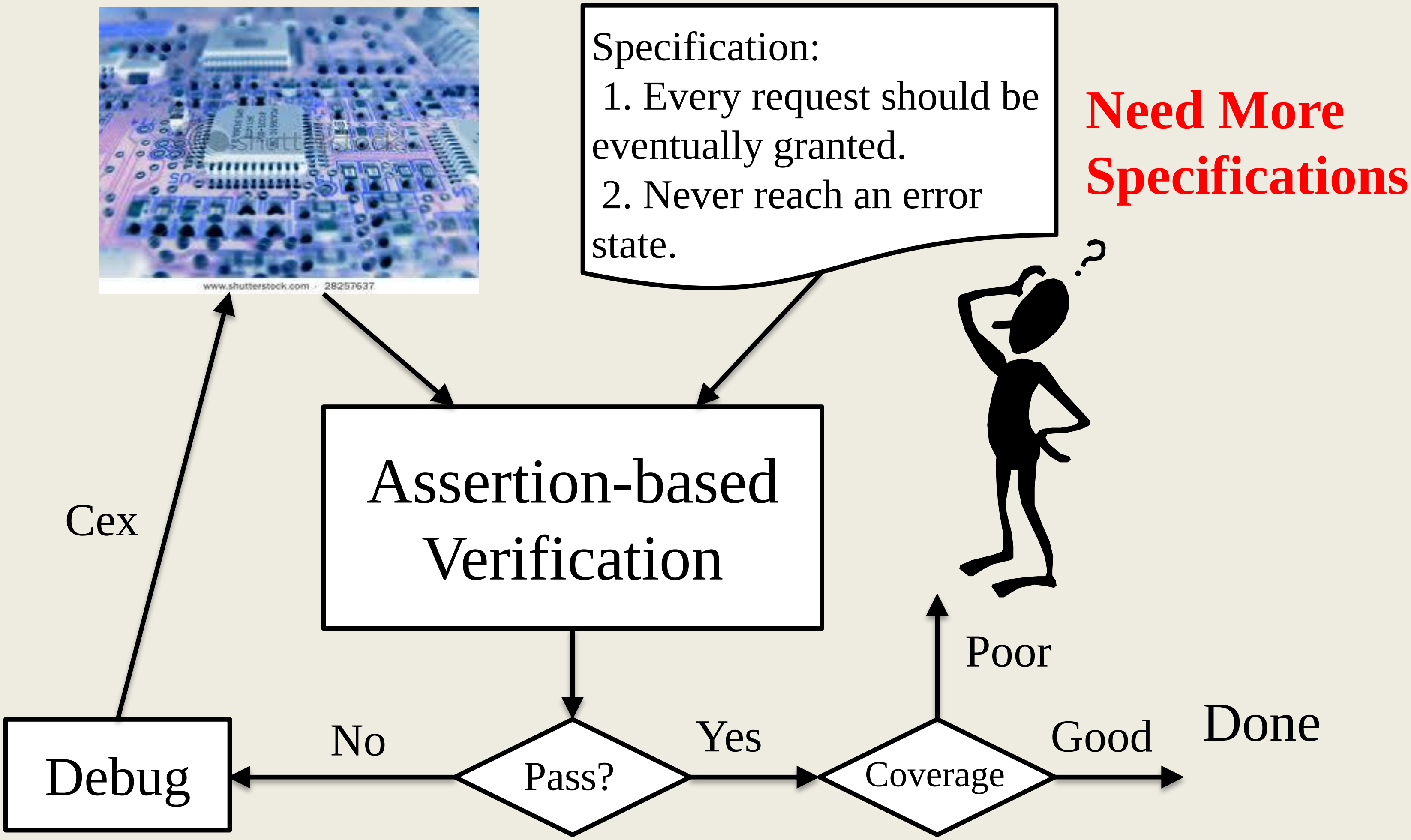
- Writing assertion is a time-consuming manual process and is perceived as "difficult".
- "During the first formal verification runs of a new hardware design, typically **20%** of the formulas are found to be trivially valid." [IBM Haifa]

Verification and Synthesis

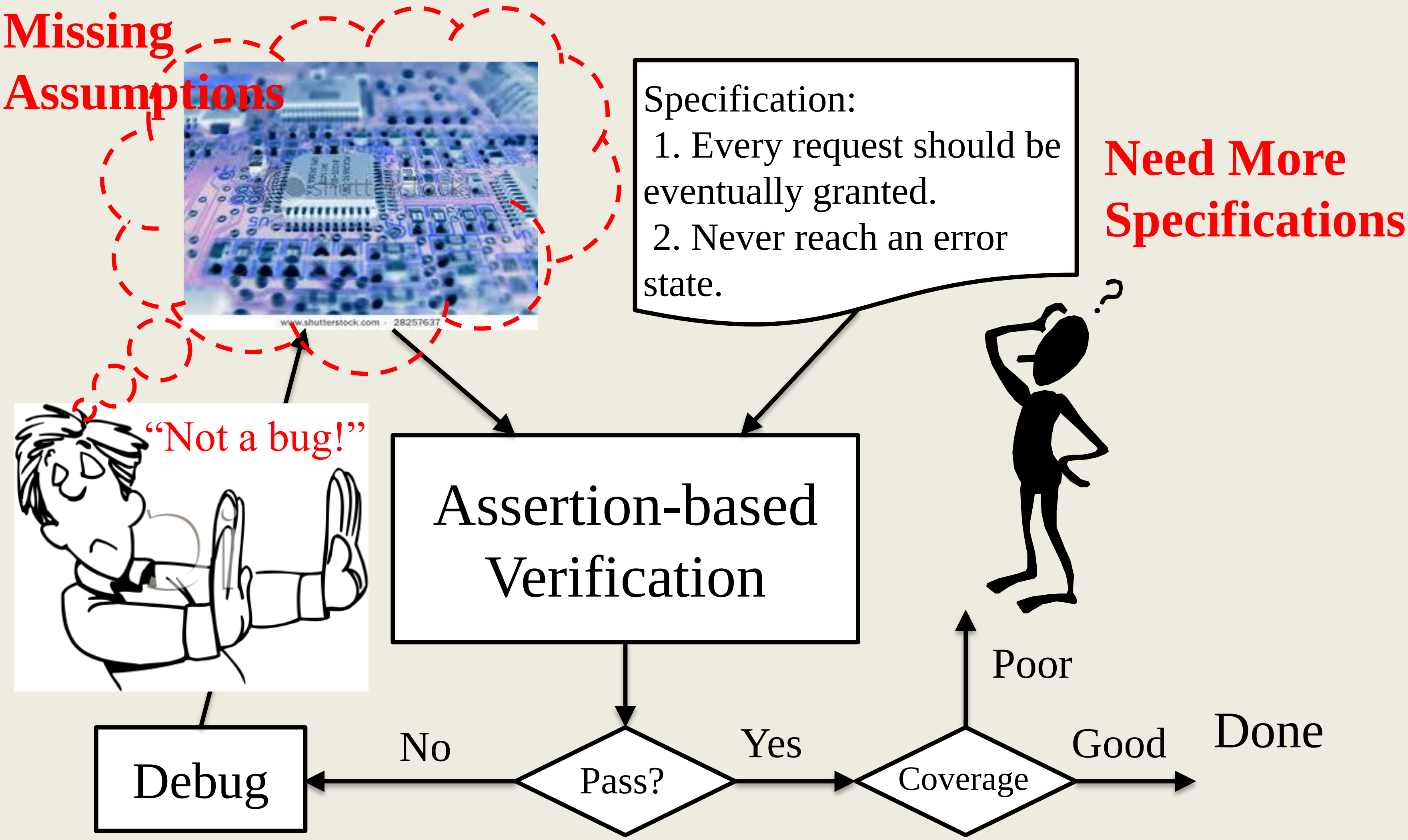


Specification is arguably the most important step for formal verification and correct-by-construction synthesis

Verification is as *Good* as Specification



Verification is as *Bad* as Specification



Temporal specifications can be mined systematically both from observed and counteracting behaviors, and are useful for automating difficult tasks in verification and synthesis such as localizing bugs and finding missing assumptions.

Formalisms

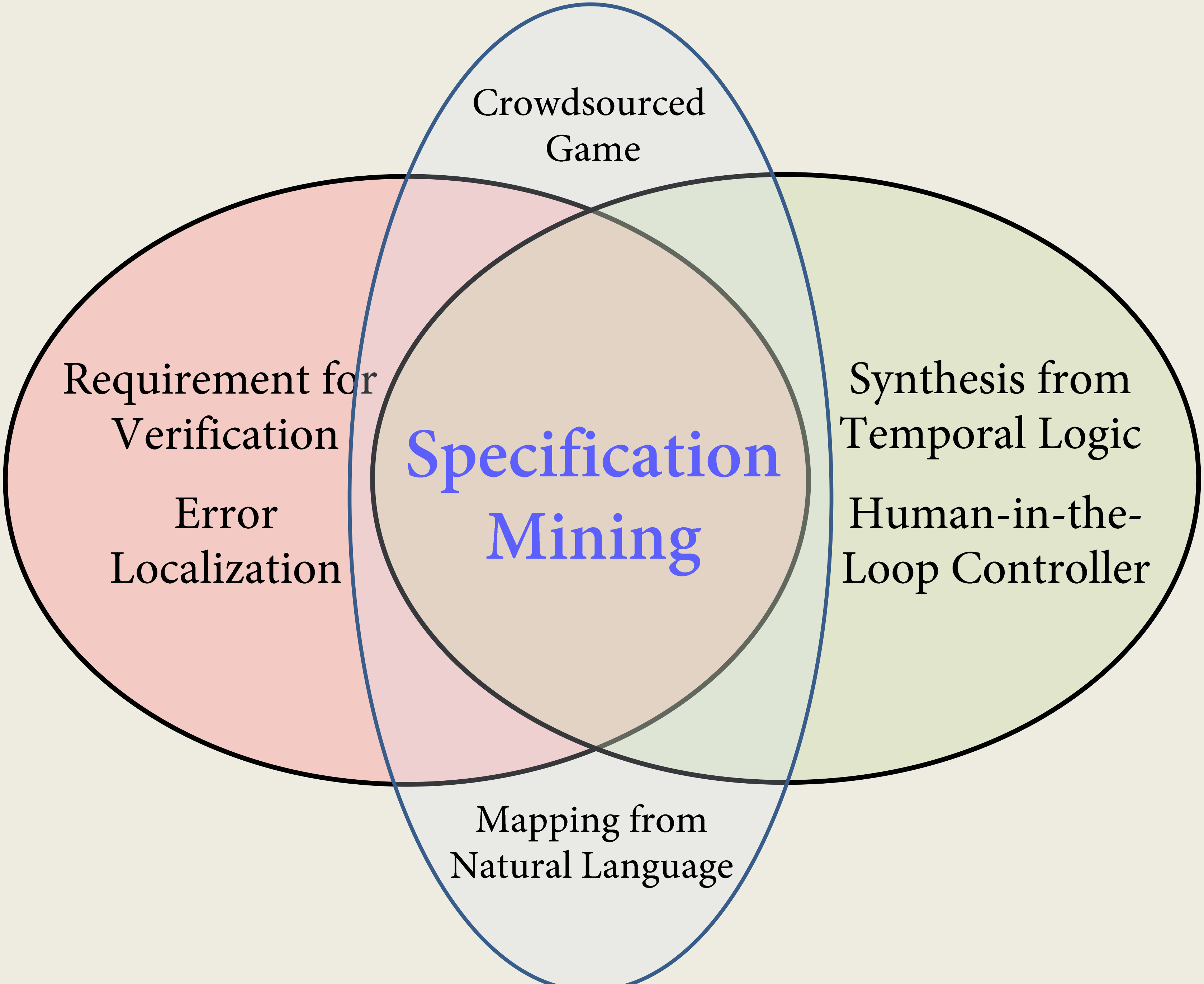
Basis Subtrace   Version-Space Learning

Algorithms

Automata-Based   Sparse Coding   Counterstrategy-Guided

Applications

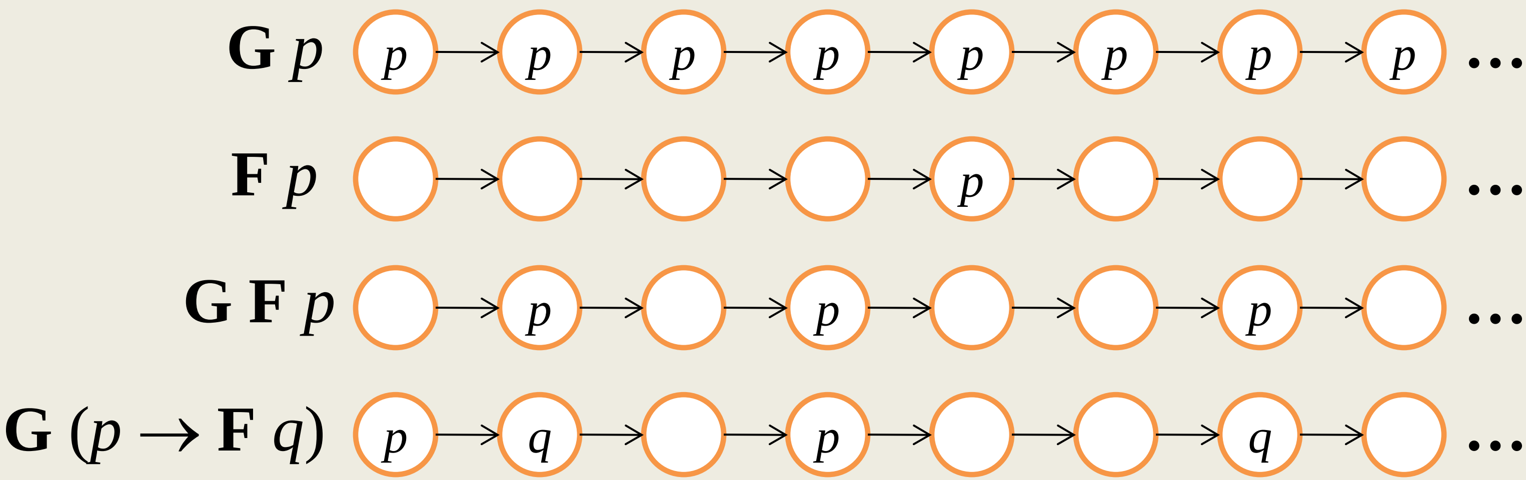
Bug Localization   LTL Synthesis   Human-in-the-Loop Controller


Crowdsourced
Game
Requirement for
Verification
Error
Localization
Specification
Mining
Synthesis from
Temporal Logic
Human-in-the-
Loop Controller
Mapping from
Natural Language

Linear Temporal Logic

Formal specification: behavior description supported by logic-based languages

$$\psi ::= p \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\ \psi \mid \psi\ \mathbf{U}\ \psi$$



$\mathbf{G}\ (\mathrm{req} \rightarrow \mathbf{F}\ \mathrm{grant})$:
Every request must be followed by a grant.

Specification Mining with Templates

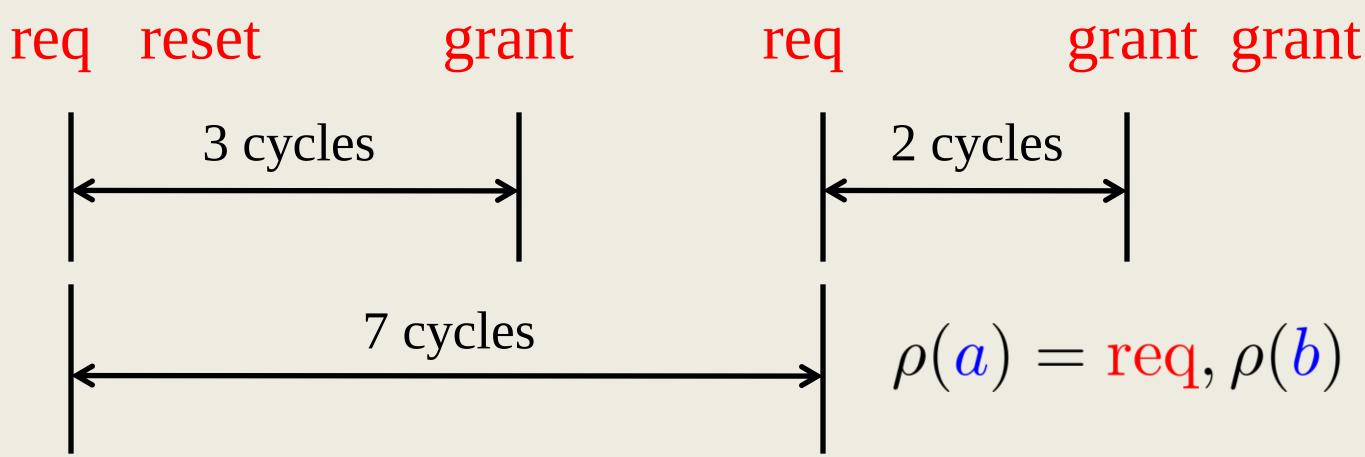
Example specification $\psi(a, b)$:

(1) every a is followed by a b within 3 cycles;

(2) every two as are separated by at least 7 cycles.

$\Sigma = \{a, b\}$ $\qquad$ $\Sigma' = \{\text{req}, \text{grant}, \text{reset}\}$

Find (all) mapping $\rho : \Sigma \rightarrow \Sigma'$, s.t. $\psi(\rho(a), \rho(b))$ is **true** w.r.t. *some evidence*.

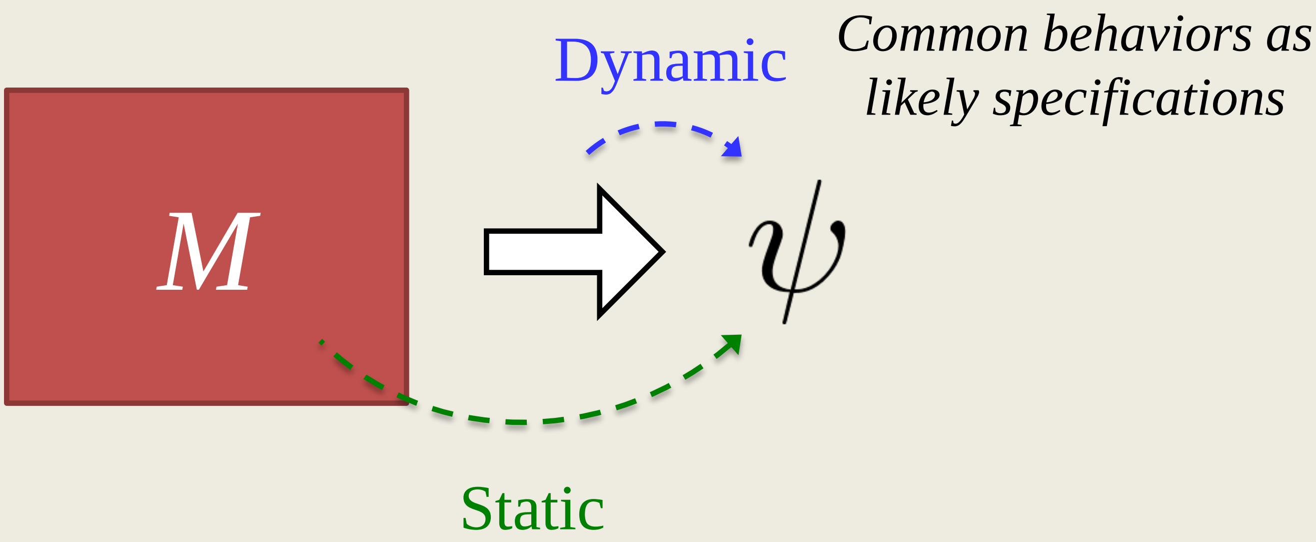


$\rho(a) = \text{req}, \rho(b) = \text{grant}$

Part I

Requirement Generation and Error Localization

Requirement Generation



Static: Infer specification directly from the description of the design, e.g. synthesis of interface specification for Java classes [Alur et. al., 2005]

Dynamic: Infer *likely specification* from simulation /execution traces, e.g. DAIKON [Ernst et. al., 2000]

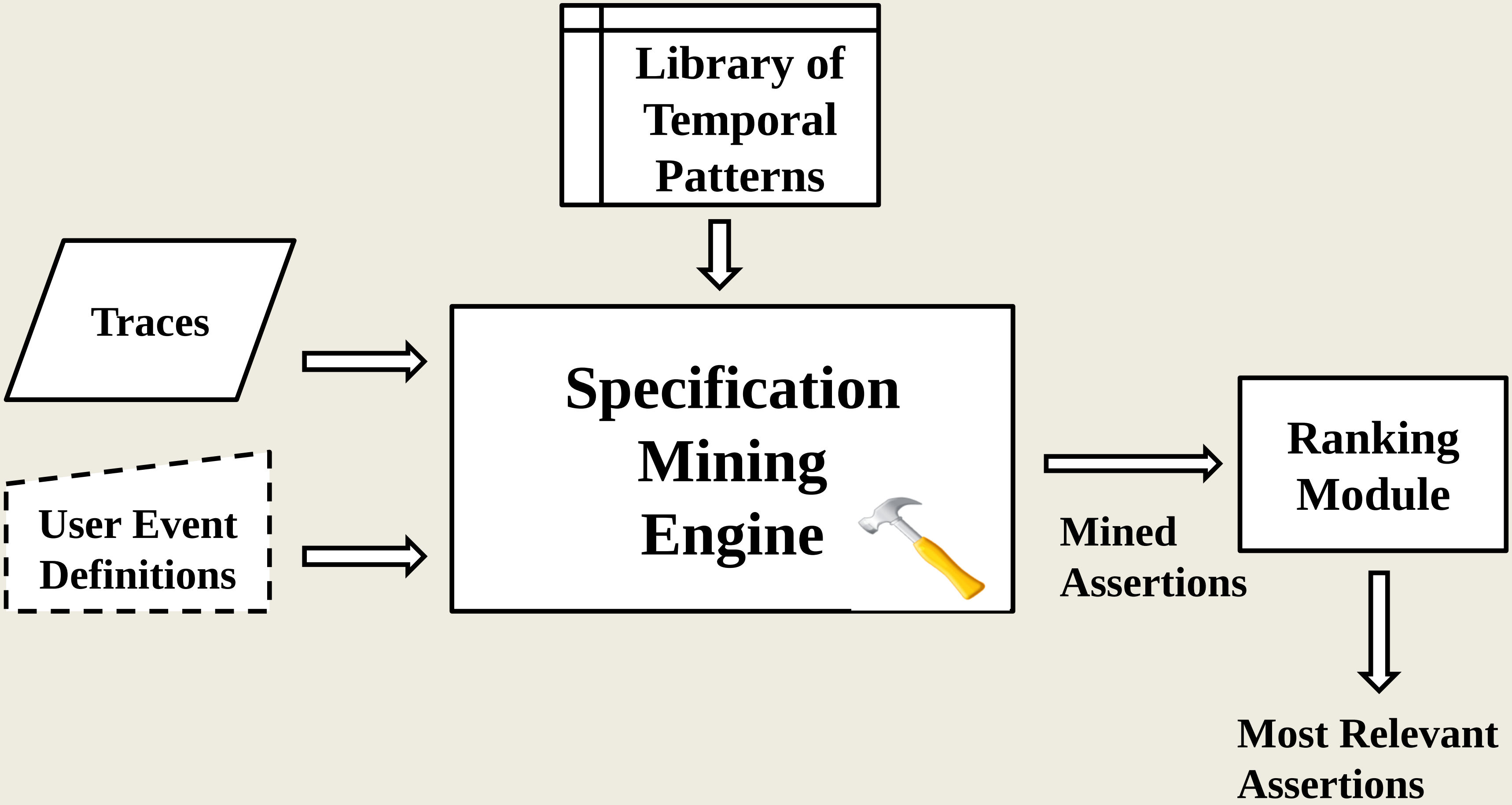
Automata-based [DAC'10] **Sparse Coding** [RV'12]

Specification Mining:

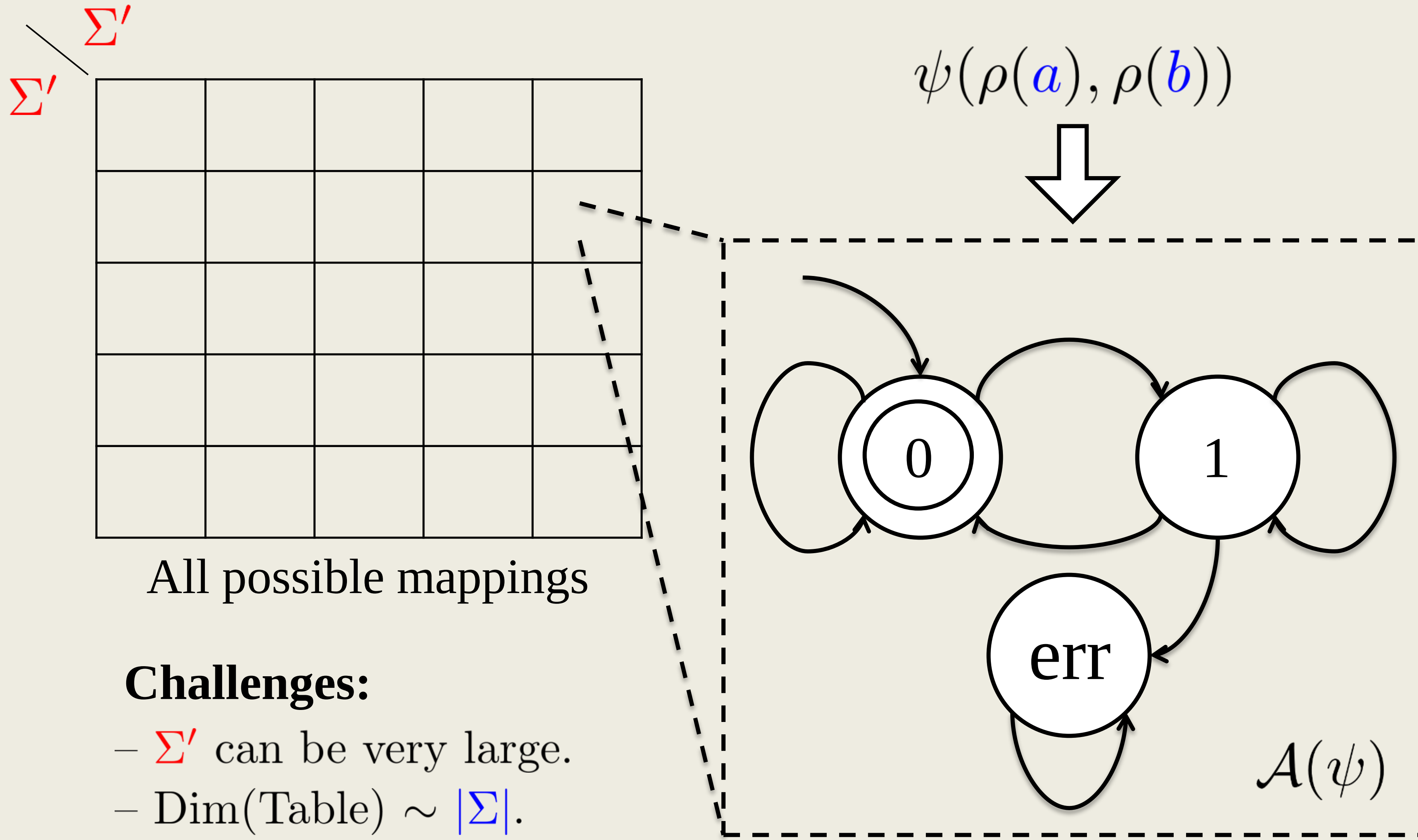
An *Automata-based Monitoring* Approach

Mining Temporal Properties

With a focus on hardware traces



[Li et al., Scalable Specification Mining for Verification and Diagnosis. DAC 2010]



All possible mappings

Evaluate $\mathcal{A}(\psi)$ over traces

Challenges:

– Σ' can be very large.
– $\mathrm{Dim(Table)} \sim |\Sigma|$.

Solutions:

– Design ψ s.t. evaluating transitions are sufficient.
– Small Σ but use inference rules to merge ψ.

Requirement Generation:

eMIPS - 278 modules and more than 20,000 signals

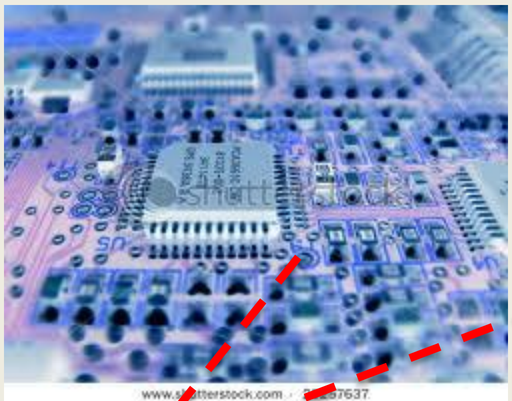
Design	$\|T\|$	$\|T^{\Delta}_{m}\|$	n_m	$\|S\|$	$\|S_{merged}\|$	Runtime (s)
eMIPS	5 mil	5408	108	2079	1028	51
Router	0.23 mil	12420	28	120	74	13
I2C	1.6 mil	20904	33	389	308	9
CAN	26 mil	36100	175	3272	1356	71

Summary:

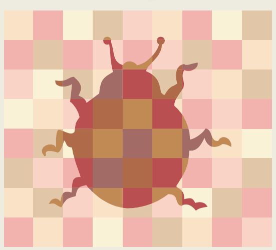
- Industrial-size designs;
- Traces of millions of cycles;
- Mine relevant temporal properties efficiently.

Research Question

Can we use the many simple mined specifications to *localize* complex bugs?



010101010101
011011010101
010111111010
10101



Where?

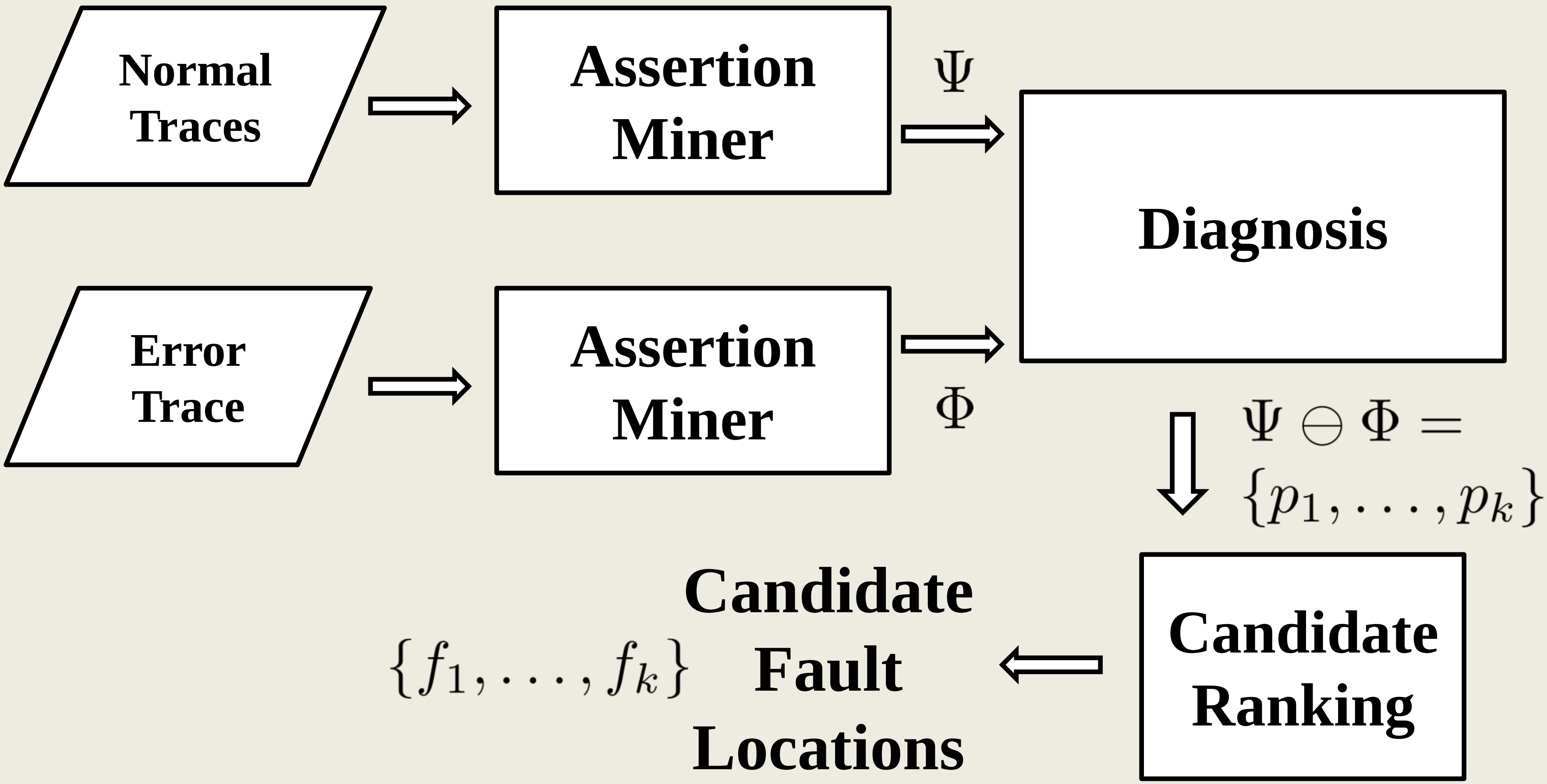
Post-Si Challenges:

- Limited observability
- Long error detection latency
- Transient and hard-to-reproduce bugs

Expensive: $1 million to redesign the masks [Ying et al., 2005]; 3:1 headcount for design vs. post-Si validation [Patra et al., 2007]; post-Si validation consumes 35% of chip development time on average [Abramovici et al., 2006]

Proposed Solution

Mine *distinguishing patterns* between good and bad traces over module interfaces



Error Localization:

CMP router; localize to within 15 cycles for transient faults

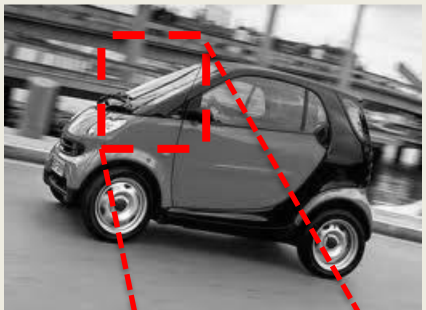
Type of Fault	Fault Coverage %	Time Localization %	Module Localization %
Stuck-at	100	-	100
Erroneous Transition	100	-	100
Erroneous Assignment	100	-	57
Transient	100	81	56

Summary:

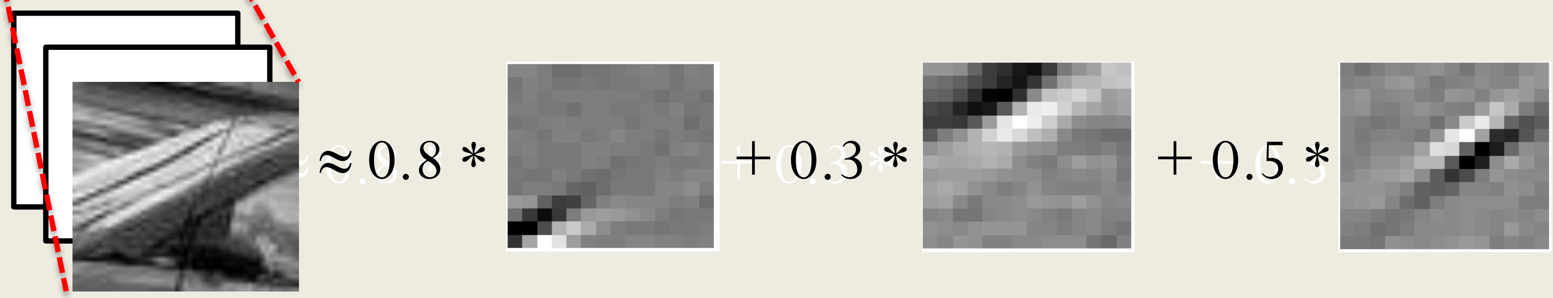
- eMIPS: effectively localize different design bugs.
- CMP router: effectively localize transient bugs also.
- Mining *simple distinguishing* patterns can help to localize *complex* bugs.

Research Question

Can we learn specifications w/o assuming forms?



Sparse Coding:
Sparsity helps to uncover latent structure
e.g. finding edge detectors in an unsupervised setting



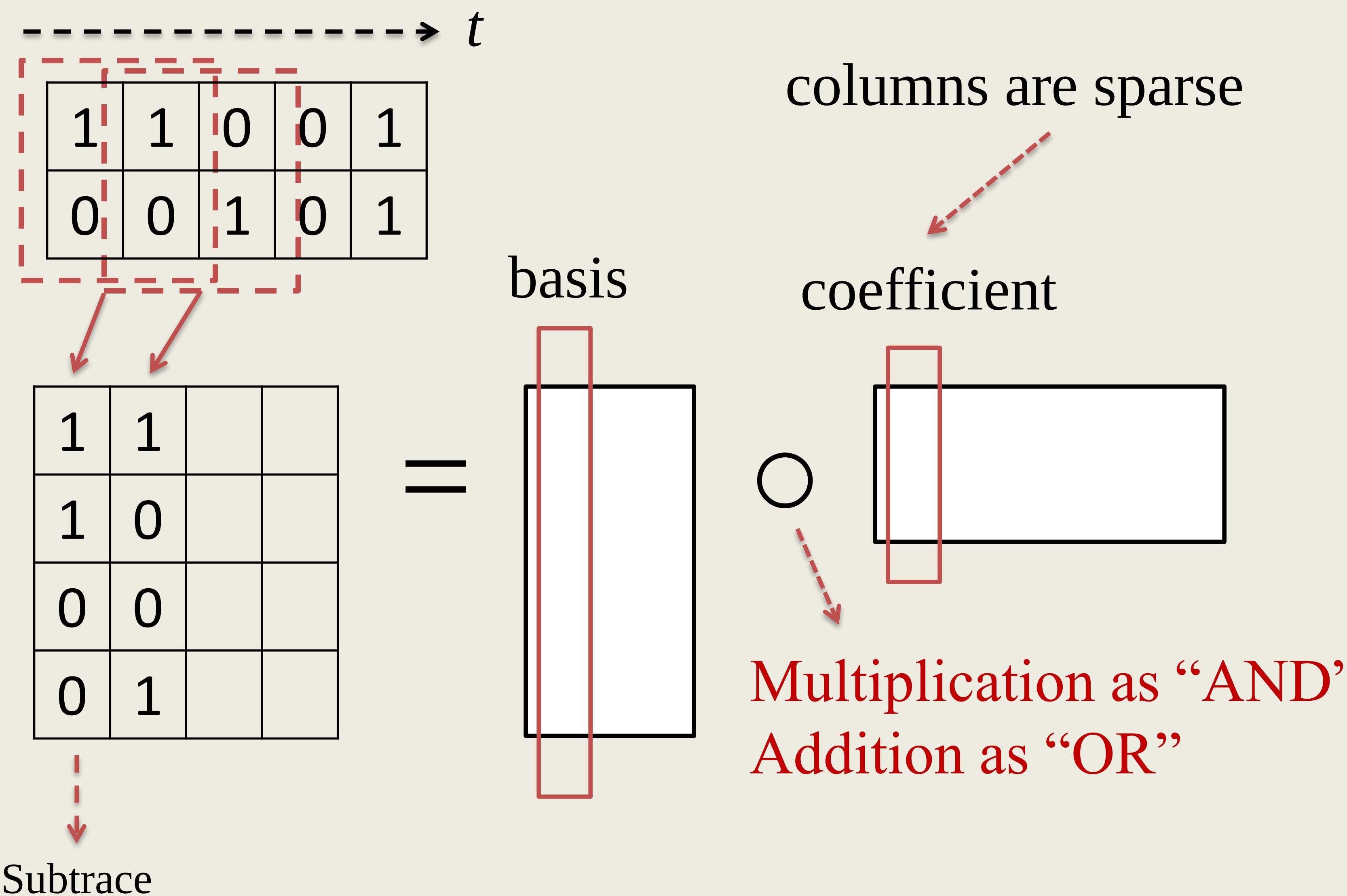
Specification formalism: Express each subtrace as a Boolean combination of a few “basis subtraces”– a (sparsity-constrained) Boolean matrix factorization problem.

[Li and Seshia. Sparse Coding for Specification Mining and Error Localization. RV 2012]

Specification Mining:

A *Sparse Coding* Approach

Problem Formulation



Sparsity-Constrained Boolean Factorization

Given a data matrix $X \in \boldsymbol{B}^{m \times n}$ and a positive integer C, the *sparsity-constrained Boolean factorization problem* is to find k, $B = \boldsymbol{B}^{m \times k}$ and $S = \boldsymbol{B}^{k \times n}$ such that

$$X = \boldsymbol{B} \circ \boldsymbol{S}$$

$$\text{and } \left\| \boldsymbol{S}_{\cdot,i} \right\|_1 \leq \boldsymbol{C}, \forall_i$$

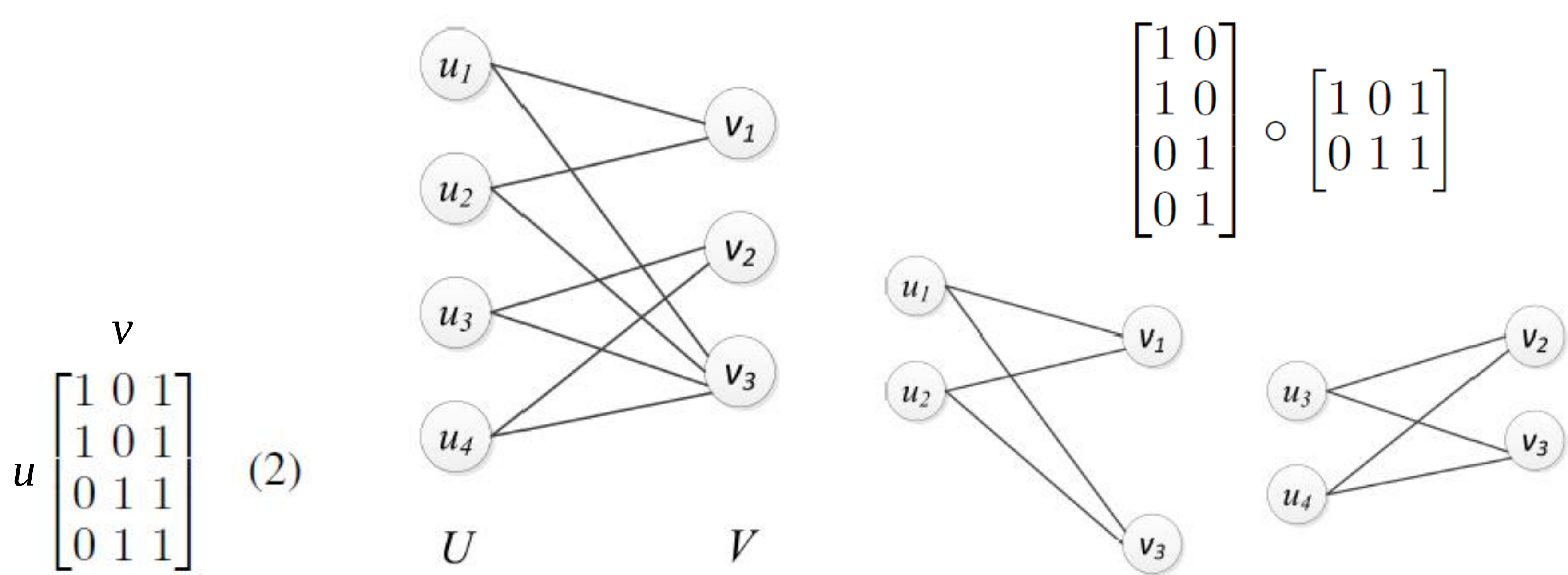
(and $\sum_i \sum_j \boldsymbol{S}_{i,j}$ is maximized).

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \qquad C = 2$$

X B S

Algorithm Idea

- Observe that the data matrix X can be viewed as the adjacency matrix for a bipartite graph.
- **Idea:** factorization → biclique cover (biclique ↔ basis subtrace)



(a) Matrix form (b) Bipartite graph (c) Biclique edge cover

Error Localization

- Error localization and explanation based on reconstruction:

A subtrace has an error if it cannot be reconstructed from the basis subtraces

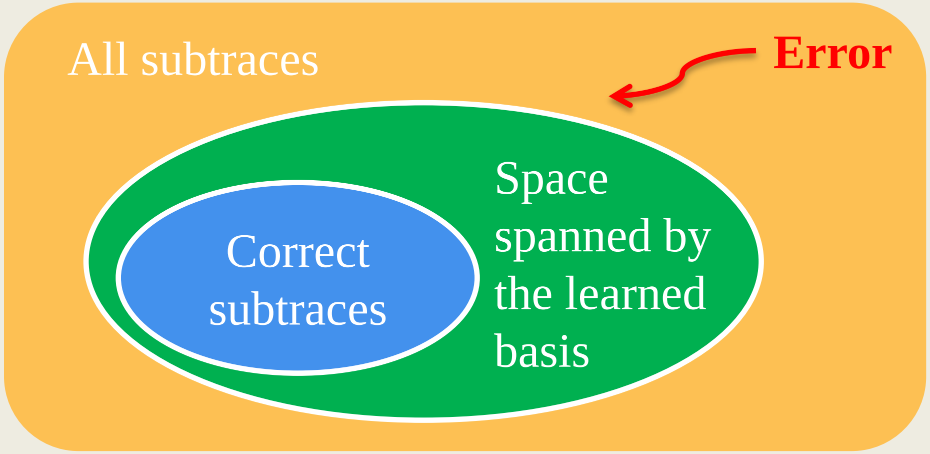
0	1	0	1	1	0	…	…
1	0	0	1	1	1	…	…
0	1	0	0	1	0	…	…
1	0	0	1	0	1	…	…

Columns 1–3: $X_{\cdot,1}$; columns 4–6: $X_{\cdot,2}$

$$\underset{S_{\cdot,i}}{\text{Minimize}} \left\| X_{\cdot,i} \oplus (B \circ S_{\cdot,i}) \right\|_1$$

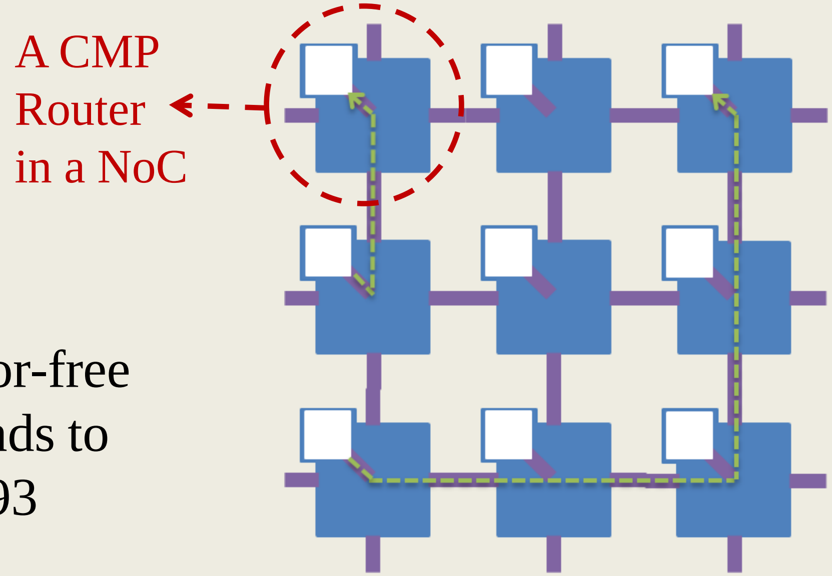
$$\text{Subject to } \left\| S_{\cdot,i} \right\| \leq C$$

- A subtrace is error-free if $\left\| X_{\cdot,i} \oplus (B \circ S_{\cdot,i}) \right\|_1 = 0$



Experimental Results

- <u>Chip Multiprocessor Router:</u>
  - Observe 14 control signals
  - Subtrace width: 2 cycles
  - Learn the basis from a single error-free trace of 1000 cycles: 0.243 seconds to obtain 189 basis subtraces from 93 distinct subtraces



[Source: Daniel Holcomb]

- <u>Error Localization:</u>
  - Inject a single bit flip at a random cycle for each of 99 error traces
  - Localize the error to the subtrace (out of 999) where it was injected
- <u>Comparisons:</u>
  - Baseline approach (1): hash all distinct subtraces – report error even before an error is injected for the 99 traces
  - Baseline approach (2): use unit basis – 0% localization
  - Sparse Coding: 55.6% localization

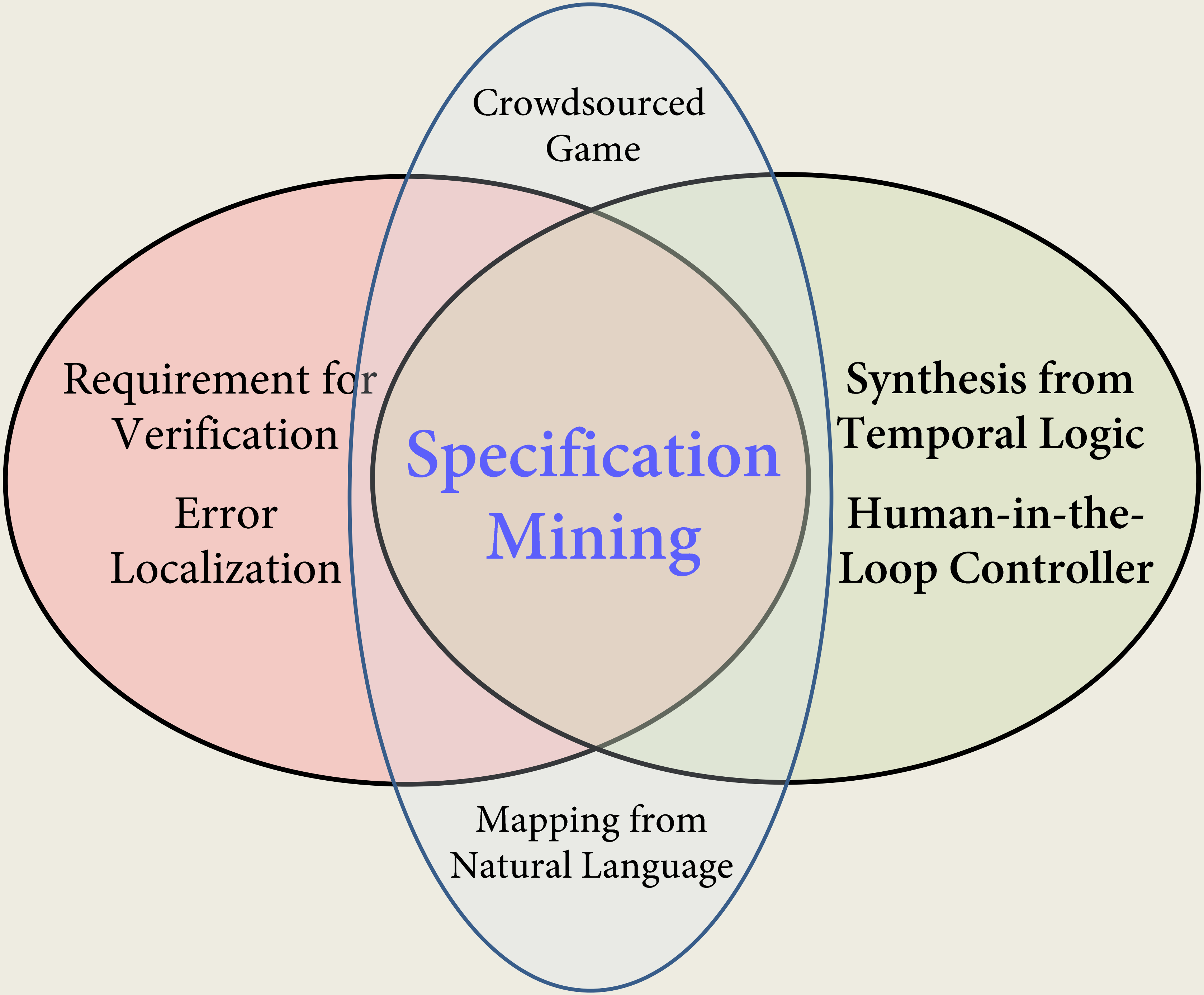
Part I: Contributions

Automata-based: [Li et al., 2010]

- An efficient algorithm for mining *temporal properties* from traces of digital designs.
- Effective algorithm for *localizing bugs* in hardware using *distinguishing patterns*.

Sparse Coding: [Li et al., 2012]

- A novel formalism of specification based on the notion of *basis subtraces*.
- An *unsupervised* algorithm for learning basis subtraces.
- An effective way of using basis subtraces to *localize bugs*.

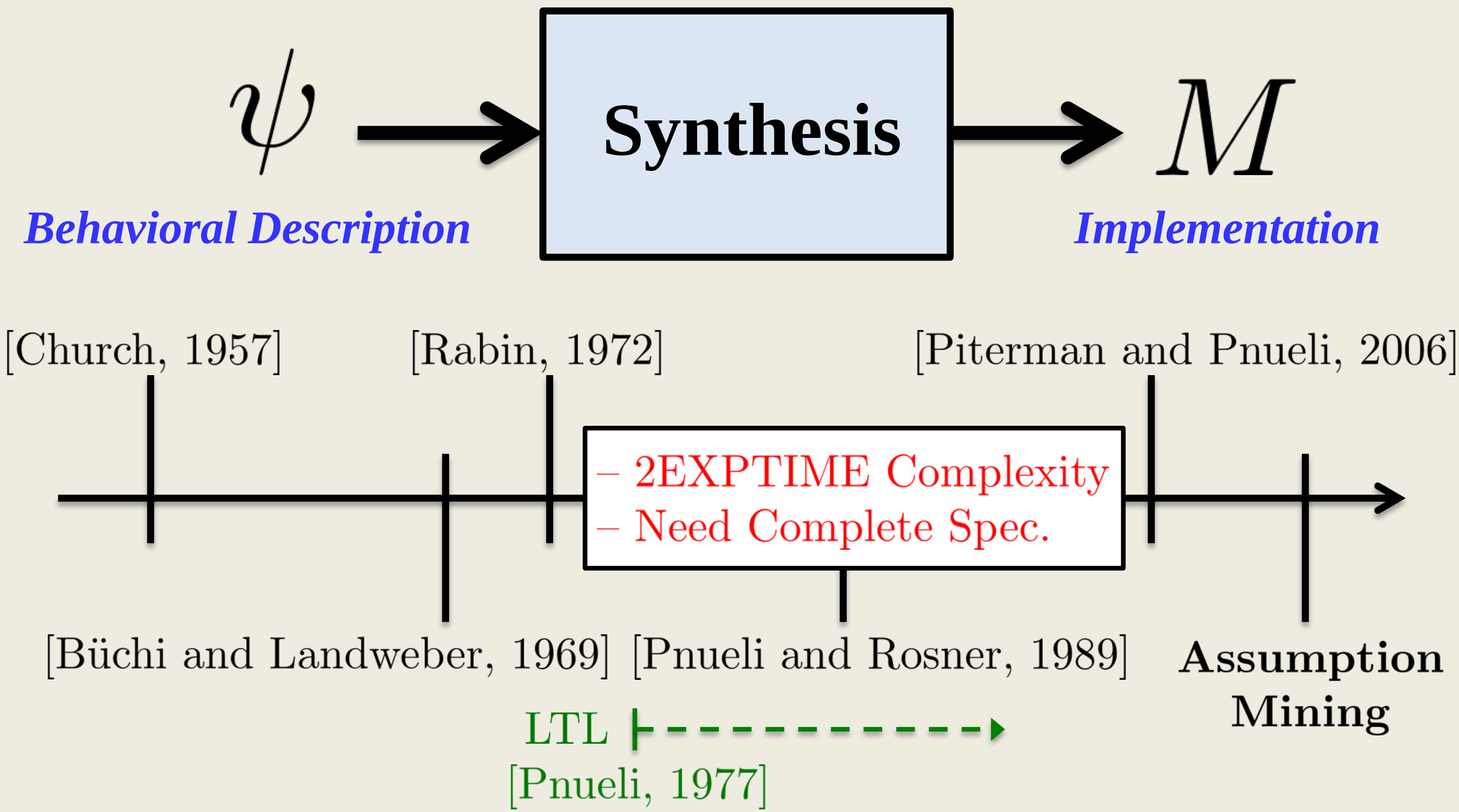

Crowdsourced
Game
Requirement for
Verification
Error
Localization
Specification
Mining
Synthesis from
Temporal Logic
Human-in-the-
Loop Controller
Mapping from
Natural Language

Part II

Assumption Mining for LTL Synthesis

Temporal Logic Synthesis

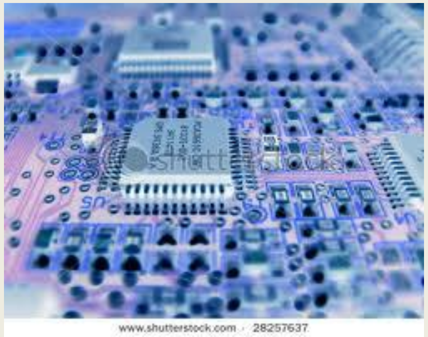
*Automatically construct an **implementation** that is guaranteed to satisfy its **behavioral description**.*



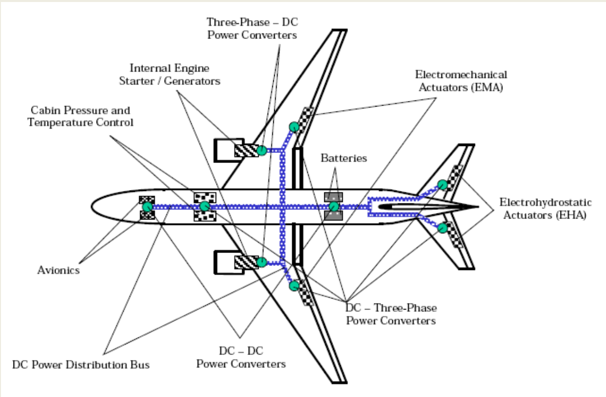
PSL

Δ

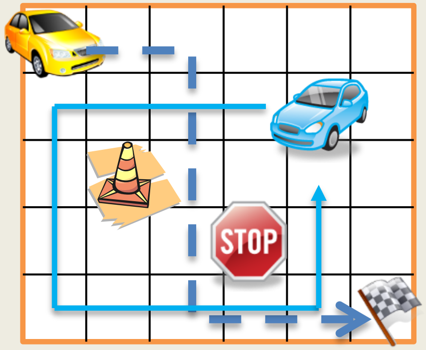
[Bloem et al., 2007]



Digital Circuit

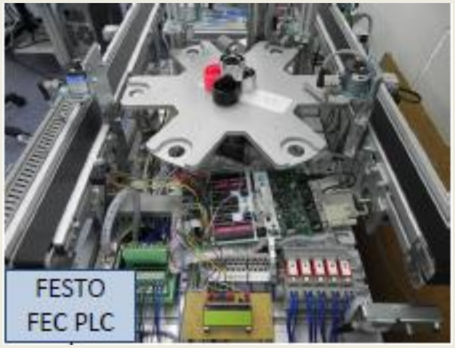


[Xu, 2013]



[Kress-Gazit, 2008]

[Wongpiromsarn et al., 2011]



[Cheng et al., 2012]

Main advantage:
Correct-by-construction

Caveat: Complete specification!

"Writing a complete formal specification for the arbiter was not trivial. Many aspects of the arbiter are not defined in ARM's standard." [Bloem et al., 2007]

Assumption Mining:

A *Counterstrategy-Guided* Approach

GR(1) Specifications

$$\psi^e \to \psi^s$$

Require ψ^l for $l \in \{e, s\}$ to be conjunctions in the following forms:

ψ^l_i: a Boolean formula that characterizes the initial states.

ψ^l_t: a LTL formula that describes the transition, in the form $\mathbf{G}\, f$, where f is a Boolean combination of variables in $X \cup Y$ and expressions $\mathbf{X}\, u$ where $u \in X$ if $l = e$ and $u \in X \cup Y$ if $l = s$.

ψ^l_f: a LTL formula that describes fairness, in the form $\mathbf{G}\,\mathbf{F}\, f$, where f is a Boolean formula over variables in $X \cup Y$.

Advantage: Can find an implementation in $O((2^{|X|+|Y|})^3)$ time.

[Piterman and Pnueli, 2006]

Given $\psi = \psi^e \rightarrow \psi^s$ with input x and output y

$$\psi^e_f = \mathbf{G}\ (\mathbf{F}\ (\neg x));$$

$$\psi^s_t = \mathbf{G}\ ((\neg x) \rightarrow (\neg y));$$

$$\psi^s_f = \mathbf{G}\ (\mathbf{F}\ (y));$$

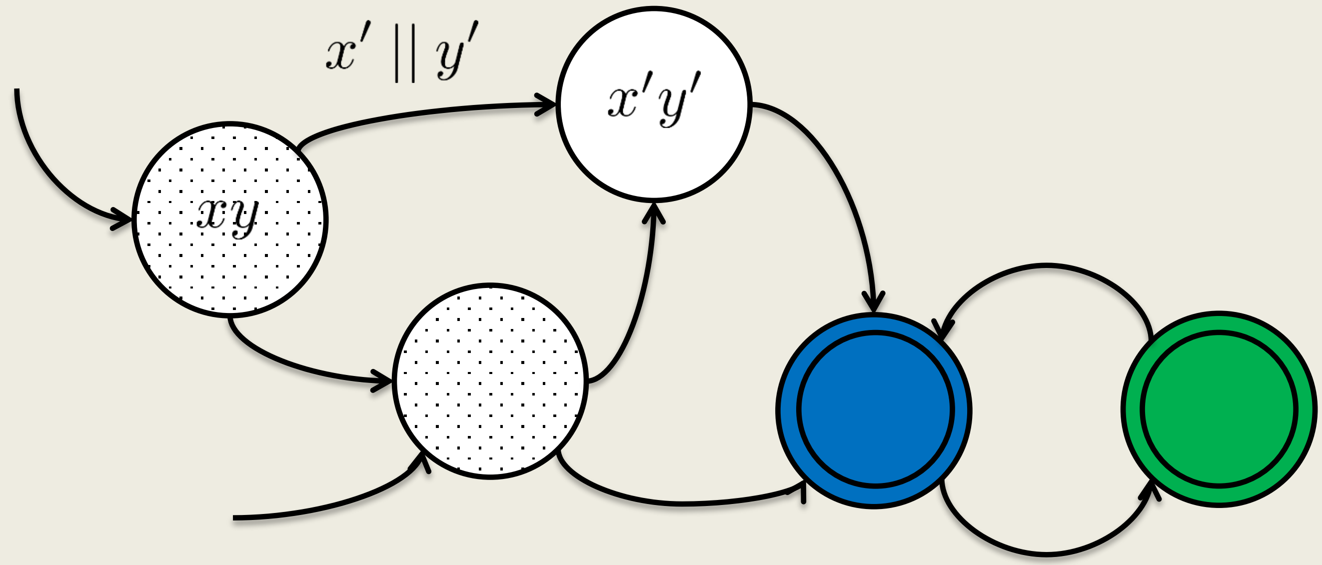
$$x: \ 0\,1\,0\,1\,0\,1\ \ldots$$
$$y: \ 0\,1\,0\,1\,0\,1\ \ldots$$

Satisfiable

$$x: \ 0\,0\,0\,0\,0\,0\ \ldots$$
$$y: \ 0\,0\,0\,0\,0\,0\ \ldots$$

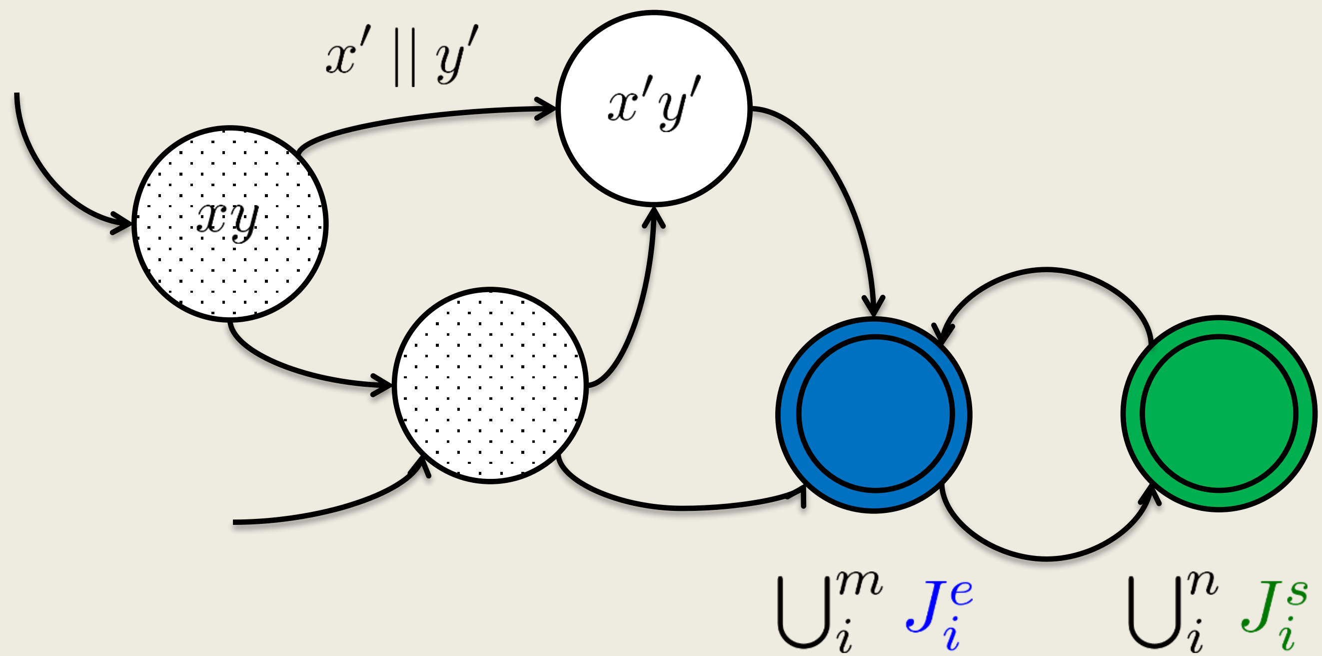
Unrealizable

Decide if $\exists\, M$ s.t. $M \models \psi$: Game Solving



A game structure $\mathcal{G}$ is a tuple $(X, Y, Q, \theta, \rho^e, \rho^s, Win)$, where

- X: a set of input variables controlled by e.
- Y: a set of output variables controlled by s.
- $Q \subseteq 2^X \times 2^Y$: state space.
- θ: a Boolean formula over $X \cup Y$ that defines the initial states.
- $\rho^e \subseteq Q \times 2^X$: environment transition relation.
- $\rho^s \subseteq Q \times 2^X \times 2^Y$: system transition relation.
- Win: winning condition of the game.



Given GR(1) specifications $\psi_i^e, \psi_i^s, \psi_t^e, \psi_t^s, \psi_f^e, \psi_f^s$,

- $\theta = \psi_i^e \wedge \psi_i^s$
- $\rho^e = \psi_t^e$ replacing $(\mathbf{X}\, u)$ by u'
- $\rho^s = \psi_t^s$ replacing $(\mathbf{X}\, u)$ by u'
- Win is given by $\psi^e \rightarrow \psi^s$

$$\bigwedge_i \psi_{f,i}^e \rightarrow \bigwedge_j \psi_{f,j}^s$$

Remark: The mapping also works for specifications given as deterministic Büchi automata.

GR(1) Synthesis ~ Games

Compute winning regions $W^s \subseteq Q$ using a nested fixpoint formula.

[Piterman and Pnueli, 2006]

$W^s \xrightarrow{\text{extract}}$ strategy $\mathcal{S}^s = (\Gamma^s, \gamma_0^s, \eta^s) \xrightarrow{\text{compute}}$ circuit consistent with $\mathcal{S}^s$

(if $Q_0 \subseteq W^s$)

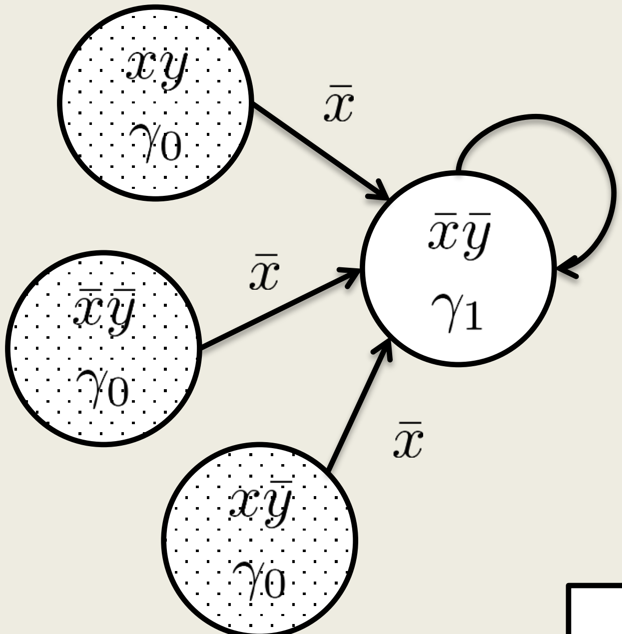
Dually, compute winning regions $W^e = Q \setminus W^s$ using fixpoint formula.

$W^e \xrightarrow{\text{extract}}$ *counterstrategy* $\mathcal{S}^e = (\Gamma^e, \gamma_0^e, \eta^e)$ [Könighofer et al., 2009]

Problem: ψ not *realizable* if $Q_0 \cap W^e \neq \emptyset$

$$\mathcal{S}^e \begin{cases} \Gamma^e = \mathcal{I} \times \mathcal{J} \\ \gamma_0^e \in \Gamma^e \\ \eta^e \subseteq Q \times \Gamma \times 2^X \times \Gamma \end{cases}$$

Key idea: Mine additional assumption ϕ to prohibit $\mathcal{S}^e$



Given $\psi = \psi^e \rightarrow \psi^s$ with input x and output y

$\psi^e_f = \mathbf{G}\,(\mathbf{F}\,(\neg x));$

$\psi^s_t = \mathbf{G}\,((\neg x) \rightarrow (\neg y));$

$\psi^s_f = \mathbf{G}\,(\mathbf{F}\,(y));$

Unrealizable

Counterstrategy $\models$ $\mathbf{F}\,(\mathbf{G}\,(\neg x))$

Candidate assumption:

$\phi = \neg(\mathbf{F}\,(\mathbf{G}\,(\neg x))) = \mathbf{G}\,(\mathbf{F}\,(x))$

$\Rightarrow \psi_{new} = \phi \wedge \psi^e \rightarrow \psi^s$

Realizable

A counterstrategy graph G^c is a discrete transition system $(V, V_0 \subseteq V, T \subseteq V \times V)$, where

- $V \subseteq Q \times \Gamma^e$: *state space*,
- $V_0 = Q_0 \times \gamma_0^e$: initial states
- $T = \eta^e \wedge \rho^s$: *transition relation*

G^c contains all game states where env. e adheres to $\mathcal{S}^e = (\Gamma^e, \gamma_0^e, \eta^e)$

General Solution:

Given a candidate assumption ϕ and a counterstrategy graph G^c, $\psi^e_{new} := \phi \wedge \psi^e$ if $\phi \wedge \psi^e \neq \texttt{false}$ and $G^c \models \neg\phi$ (*model checking*).

(a) Consistency: $\phi \wedge \psi^e \neq \texttt{false}$

(b) Done if $\psi^e_{new} \to \psi^s$ is realizable;

(c) Otherwise, iterate with new candidate ϕ_{new} and new G^c_{new}.

Question: How to pick ϕ?

Mining with Templates

What kind of assumptions?

– Efficient: In GR(1) to take advantage of $O(|Q|^3)$ algorithm.

– User-friendly: Simple formulas are easier to understand.

– Representative: Cover ϕ_i, ϕ_t, ϕ_f.

Assumption Templates:

– ϕ_a: $\mathbf{G}\,\mathbf{F}\,u$ or $\mathbf{G}\,\mathbf{F}\,(u \vee v)$ where u and v are literals over X.

– ϕ_b: $\mathbf{G}\,u$ or $\mathbf{G}\,(u \vee v)$, where u and v are literals over X.

– ϕ_c: $\mathbf{G}\,(u \rightarrow (\mathbf{X}\,v))$, where u and v are literals over X.

Related: Assumptions for LTL synthesis as a monolithic Büchi automaton. [Chatterjee et al., 2008]

Iterative Search

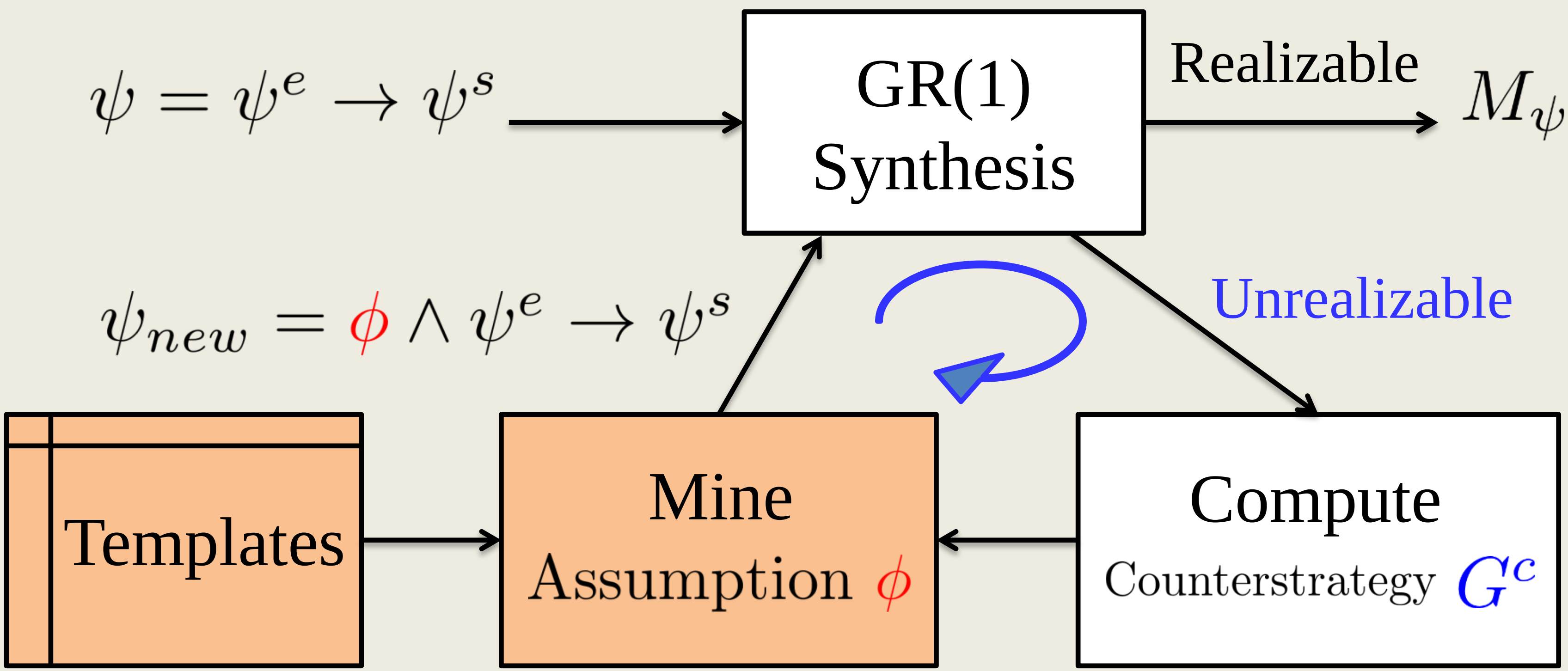
Problem1: Redundant Checks

Problem2: Restricted by Templates

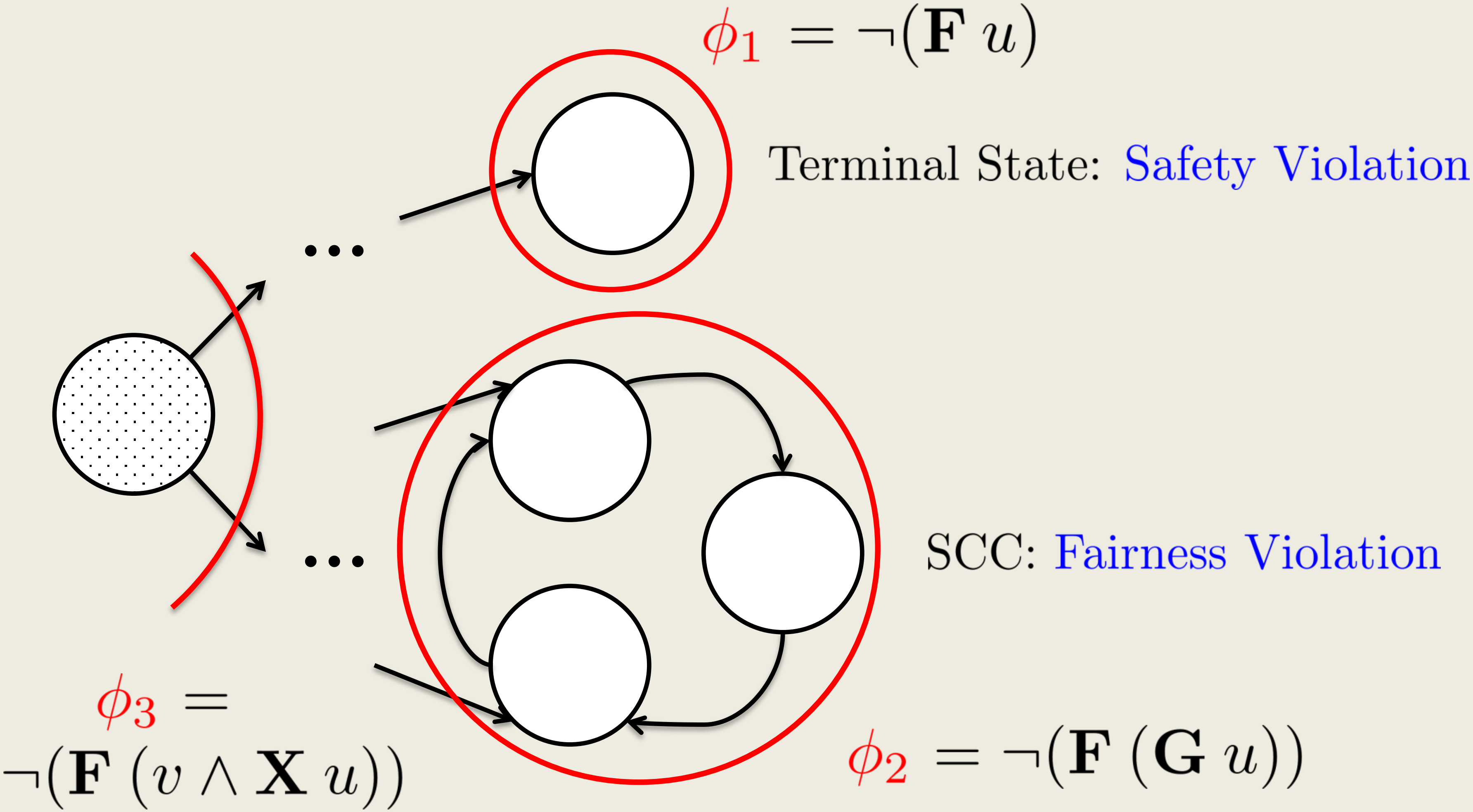
Idea:

– Check one type of assumption in GR(1) at a time.

– Use a random determinization of G^c.



Search Optimizations



Boolean formula u over X and v over $X \cup Y$

Compute ϕ_1, ϕ_2, ϕ_3 given *symbolic* representation of the counterstrategy graph $G^c = (V, V_0, T)$.

Lemma1: $\phi_1 \wedge \phi_2$ is a minimal assumption in GR(1) syntax that removes the counterstrategy.

Lemma2: The optimized algorithm produces a *nontrivial* ϕ, i.e. $\phi \wedge \psi^e \neq \texttt{false}$.

Theorem: Given a *satisfiable* GR(1) specification $\psi = \psi^e \rightarrow \psi^s$ and a G^c that represents *all* moves by the environment to force a violation of ψ, the optimized algorithm computes a *nontrivial* and *minimal* environment assumption ϕ in GR(1) such that $\phi \wedge \psi^e \rightarrow \psi^s$ is *realizable*.

Experimental Evaluation

Benchmarks:

– IBM Gen. Buffer, AMBA AHB Bus. [Bloem et al., 2007]

– Simple robotic controller.

Setup:

– Remove a single assumption from a realizable specification.

– Mine ϕ s.t. ψ is realizable.

Result Highlight:

– Recover the missing assumption in most cases.

– Reasonable replacement?

HLOCK[0]: locked access

HBUSREQ[0]: bus request

AMBA AHB Example:

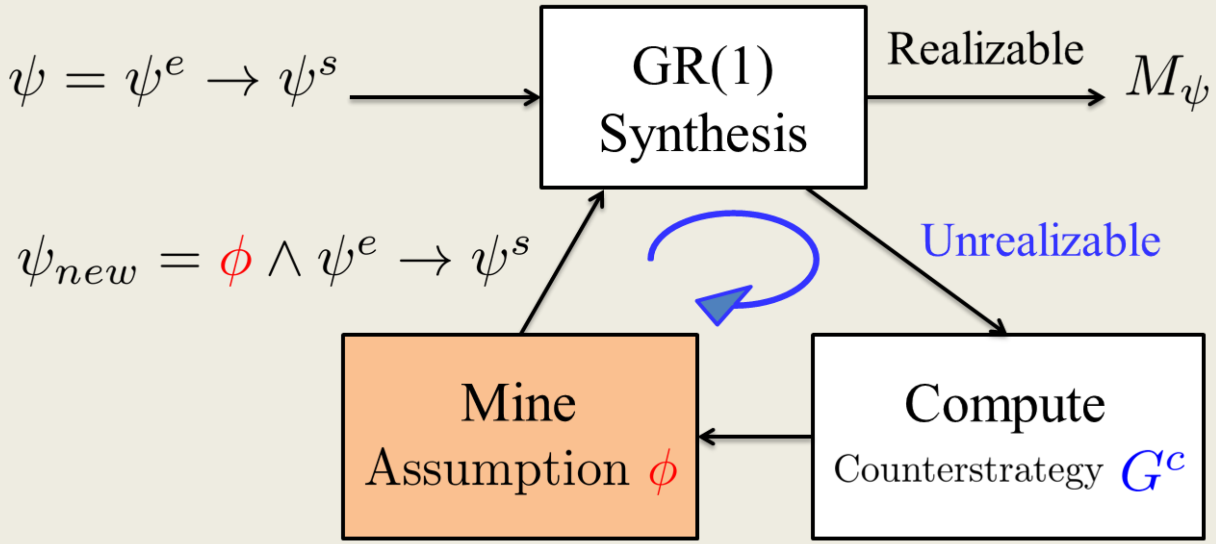
$$\phi_{\text{original}} = \mathbf{G}\,(\text{HLOCK}[0] \rightarrow \text{HBUSREQ}[0])$$

$$\phi_{\text{mined}} = \mathbf{G}\,(\mathbf{F}\,\neg\text{HBUSREQ}[0]))$$

[Li et al., Mining Assumptions for Synthesis. MEMOCODE 2011]

Summary of Contributions

- First counterstrategy-guided synthesis framework



- An efficient algorithm with theoretical guarantees for assumption generation – a key problem in correct-by-construction synthesis from temporal logic.

Assumption Mining:

Synthesizing *Human-in-the-Loop* Controllers

Many *safety-critical* systems interact with humans. The correctness of such systems depend on both the correctness of **autonomous controller**, **actions of the human** and **their interaction**.

"Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to **monitor for changes** in those conditions requiring **transition back to driver** control. The driver is expected to be available for **occasional control**, but with **sufficiently comfortable transition time**."

Level 0: **No Automation**: Driver is in complete control

Level 1: **Function Specific Automation** *Pre-charged Brakes*

Level 2: **Combined Function Automation** *Cruise Control* *Lane Keeping*

Level 3: **Limited Self Driving Automation**

Level 4: **Full Self Driving Automation**

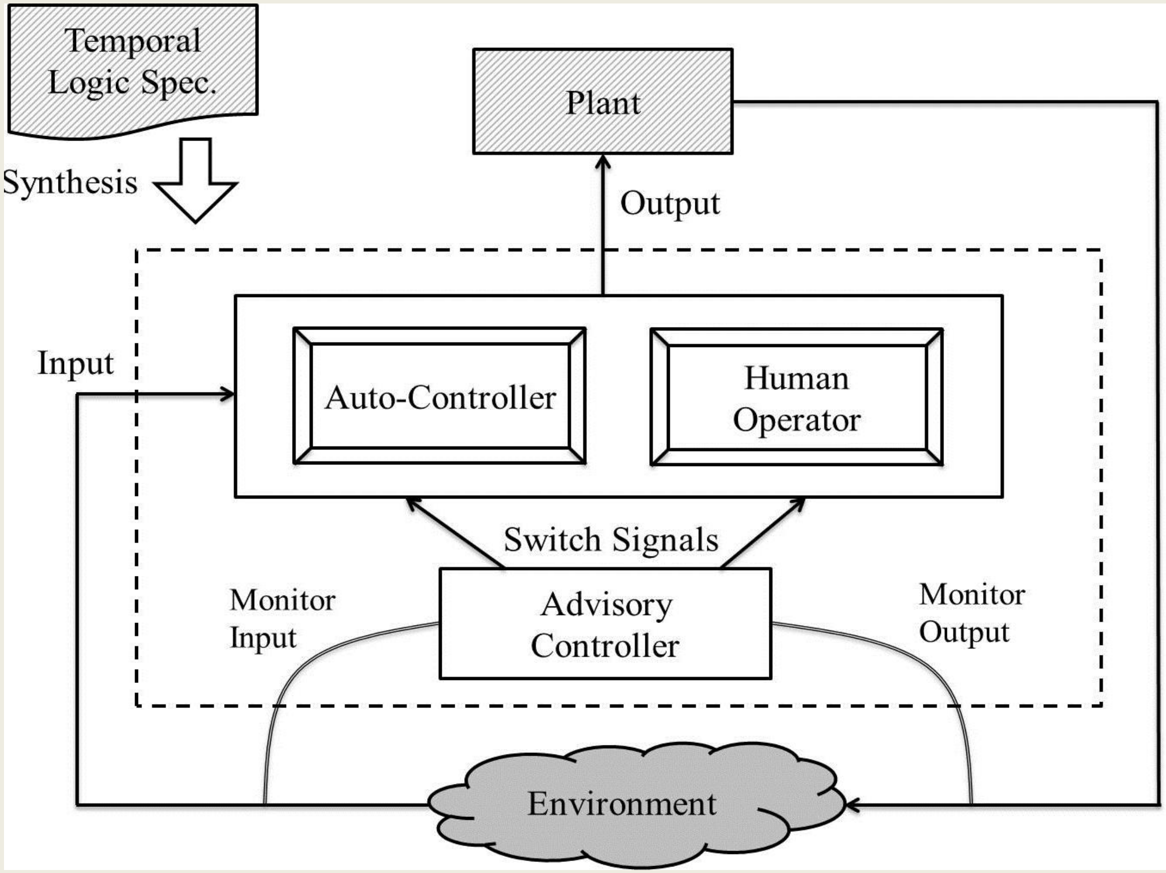
Source: National Highway Traffic Safety Administration. Preliminary statement of policy concerning automated vehicles, May 2013.

Research Question

**When autonomous controller fails,
can human safely take over control?**

MIT Cornell Crash during DARPA Urban Challenge, 2007

Human-in-the-Loop Controllers



Criteria:

- Monitoring

  Determine control switch based on monitored information

- Minimally Intervening

  Low probability of human control needed

- Prescient

  Notify danger ahead of time

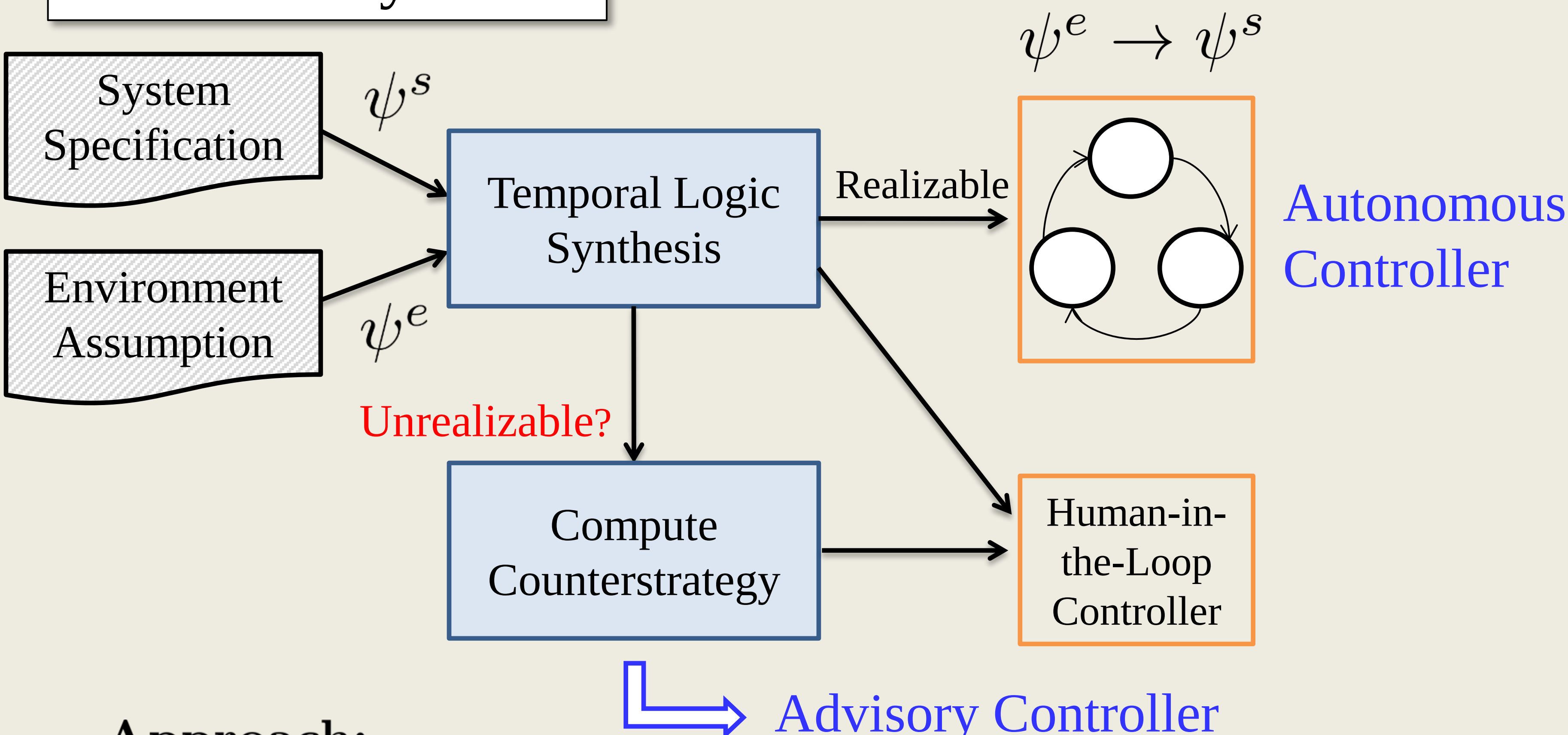
- Conditionally Correct

  Safe until human takes over control

Composition of Auto-Controller, Human Operator and Advisory Controller

[Li et al., Synthesis of Human-in-the-Loop Controllers. UCB Tech. Report 2013]

Controller Synthesis



Approach:

- Mine *transition* assumptions ϕ_3 to monitor
- Modify G^c to account for human response time
- Assign probability and early intervention penalty to G^c
- Find *s-t cut* in the weighted G^c

Theoretical Guarantees

Theorem: Given a GR(1) specification ψ, and a response time parameter T, the algorithm is guaranteed to either produce a fully autonomous controller satifying ψ, or a HuIL controller, modeled as a composition of an auto-controller, a human operator and an advisory controller that is monitoring, prescient (with parameter T), minimally intervening and conditionally correct.

Assumptions: System cannot fail within T steps.

Remark: The human operator can be replaced by a controller that maintains critical functionalities.

[Li et al., Synthesis of Human-in-the-Loop Controllers. UCB Tech. Report 2013]

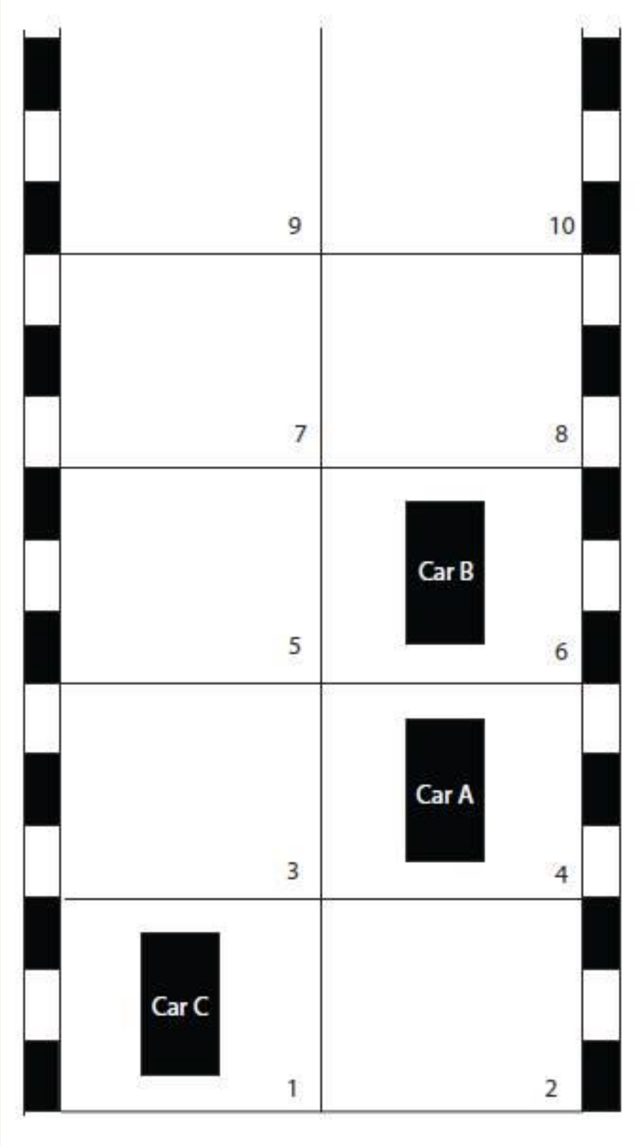
A Car Following Example

Autonomous car: A
Environment cars: B & C

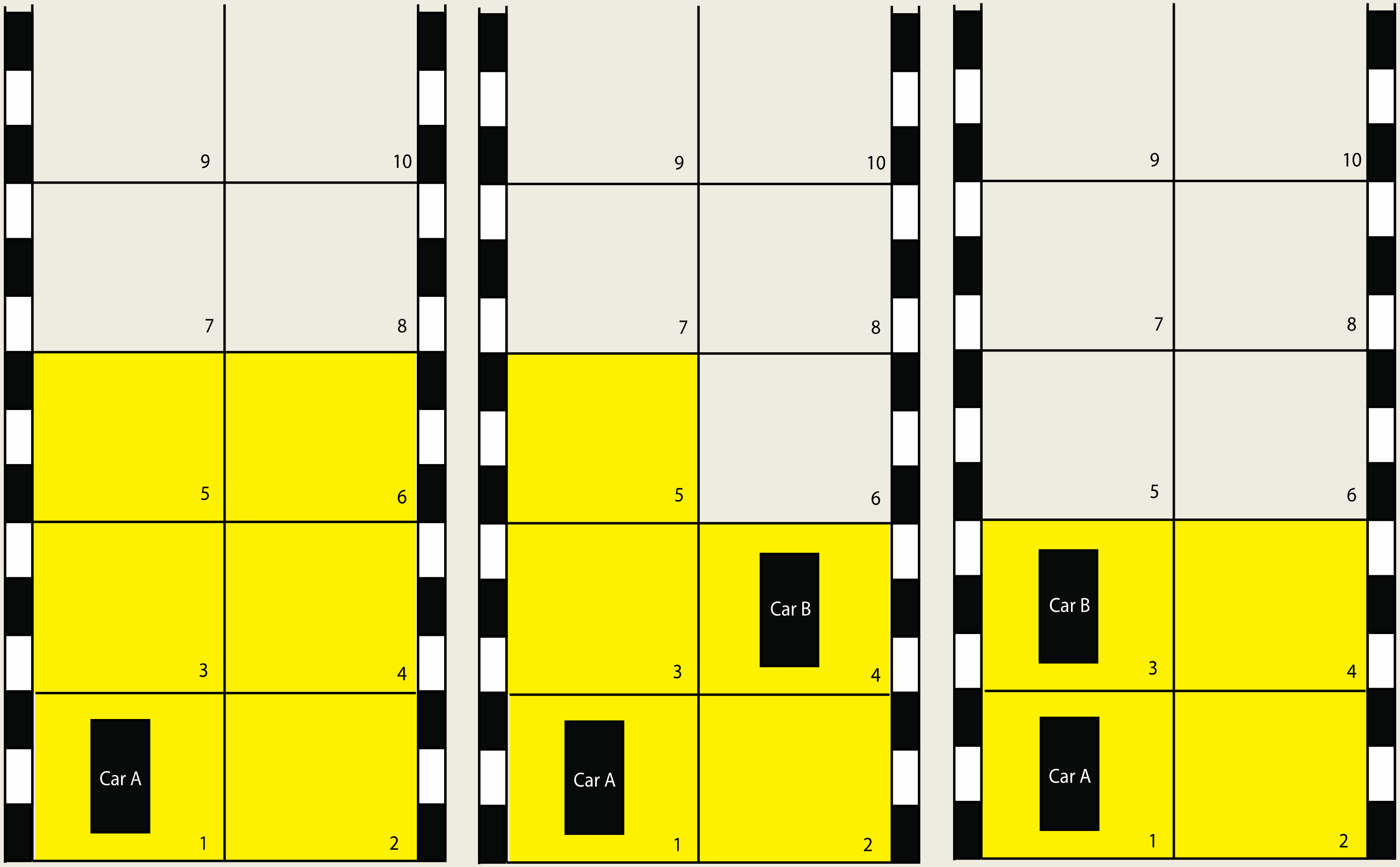
Objective: A follows B, and when this is not achievable, *switches control* to the human driver with *sufficient time* for her to respond.

Follow := move to a square where A can still sense B

Given specs encoding movement rules and $T = 1$.



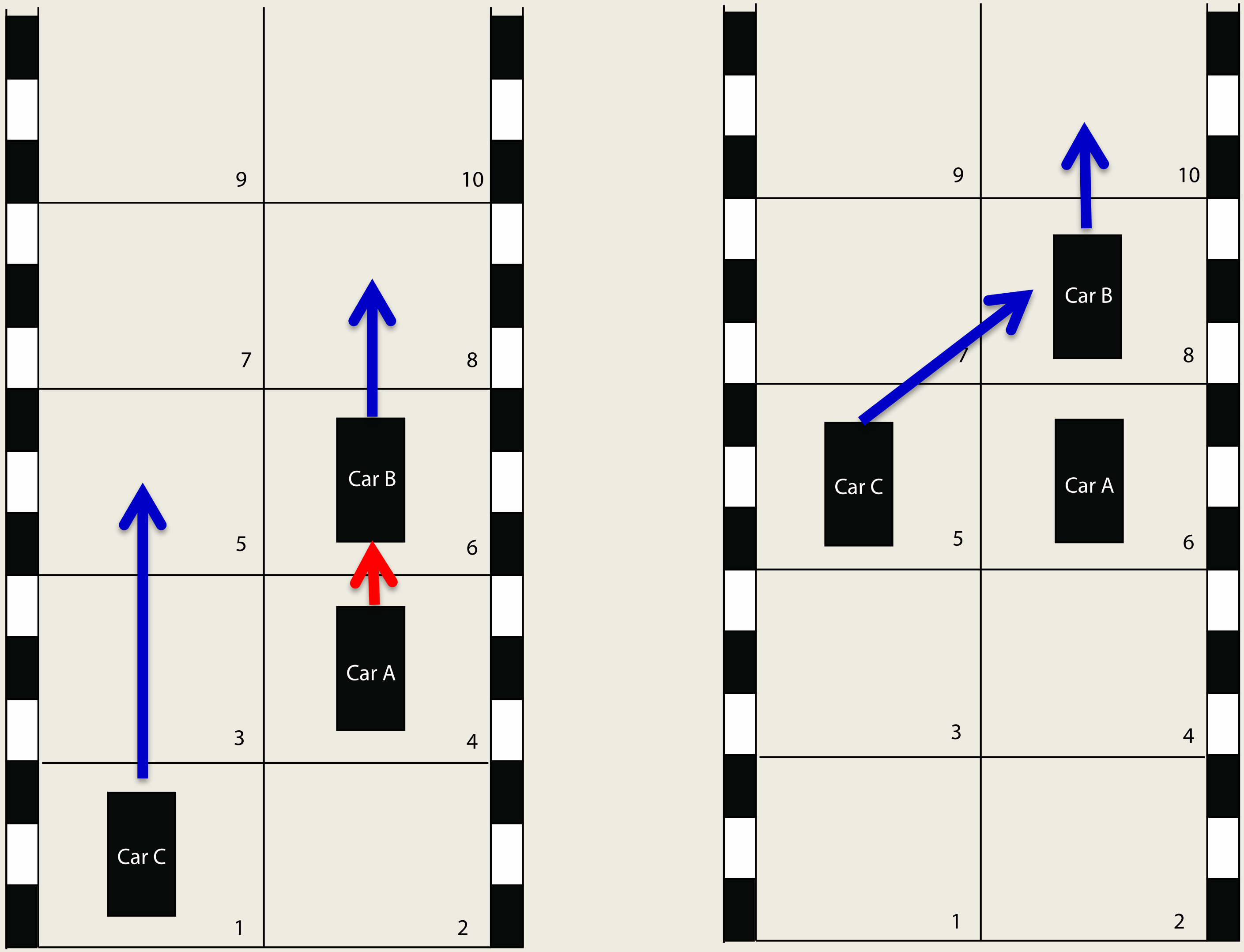
Sening Regions:



Assume given a finite-state abstraction.

[Kloetzer and Belta, 2008] [Bhatia, 2011] [Wolff et al., 2013]

Failure Scenario:



Step 1 Step 2

A Car Following Example

Mined Assumptions:

$$\varphi_{env} = \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_c = 1)) \rightarrow \right.$$

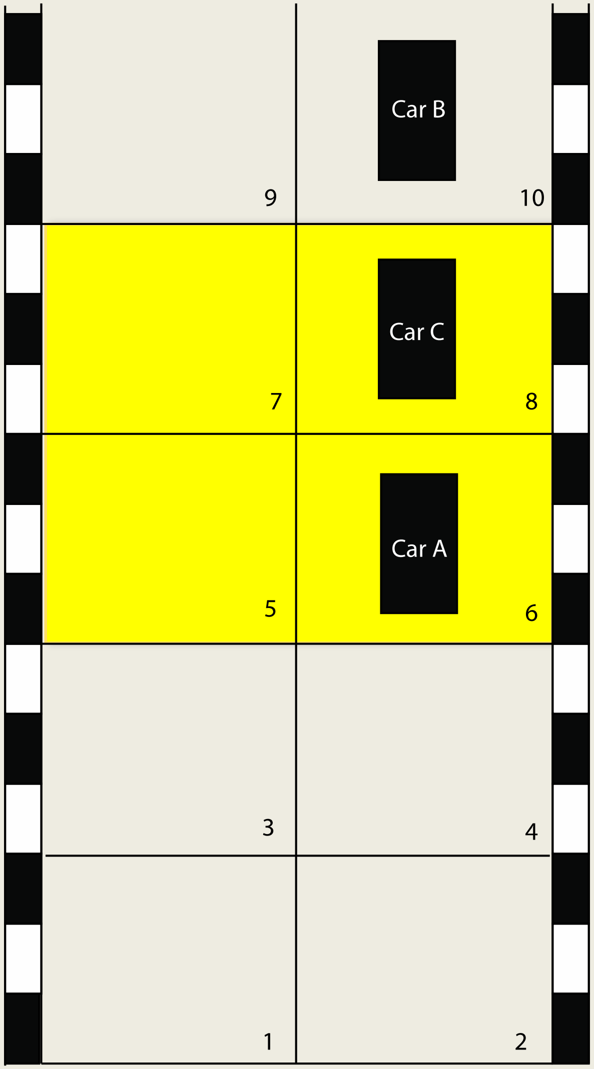
$$\left. \mathbf{X}\left((p_B \neq 8) \wedge (p_C \neq 5)\right)\right) \bigwedge$$

$$\mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_c = 1)) \rightarrow \right.$$

$$\left. \mathbf{X}\left((p_B \neq 6) \wedge (p_C \neq 3)\right)\right) \bigwedge$$

$$\mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_c = 1)) \rightarrow \right.$$

$$\left. \mathbf{X}\left((p_B \neq 6) \wedge (p_C \neq 5)\right)\right) \bigwedge$$



[Li et al., Synthesis of Human-in-the-Loop Controllers. UCB Tech. Report 2013]

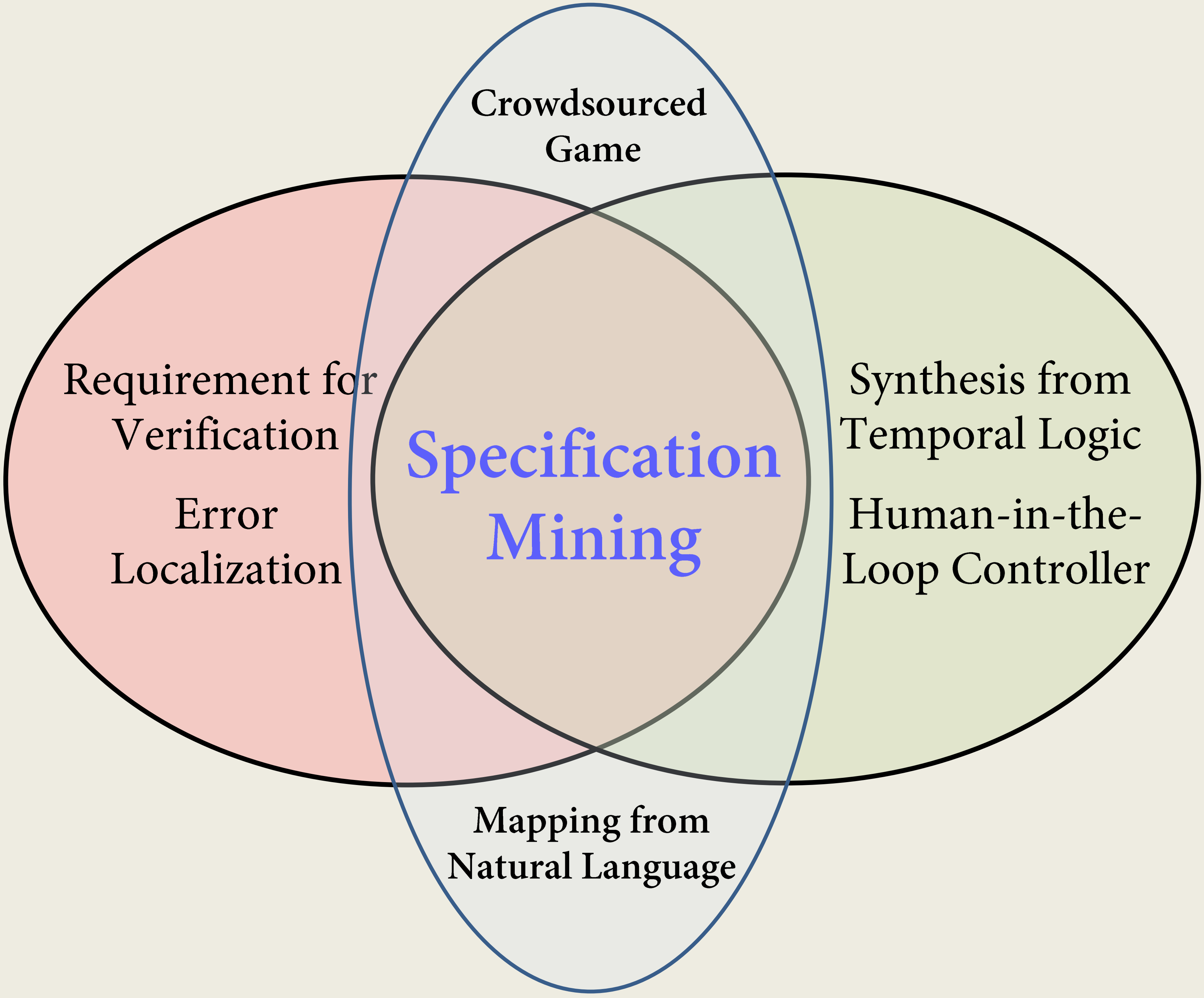
Part 2: Contributions

Assumption Mining: [Li et al., 2011]

- First *counterstrategy-guided* approach for synthesis from temporal logic.
- An efficient algorithm with theoretical guarantees for *mining assumptions* for GR(1) synthesis.

Human-in-the-Loop Controllers: [Li et al., 2013]

- A novel formalism of *human-in-the-loop controllers*.
- Identify criteria with application to *driving automation*.
- An algorithm for *synthesizing* human-in-the-loop controllers that automatically satisfy these criteria, from temporal logic specifications.

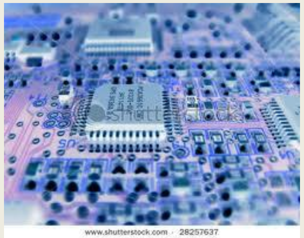

Crowdsourced Game
Requirement for Verification
Error Localization
Specification Mining
Synthesis from Temporal Logic
Human-in-the-Loop Controller
Mapping from Natural Language

Human Inputs:

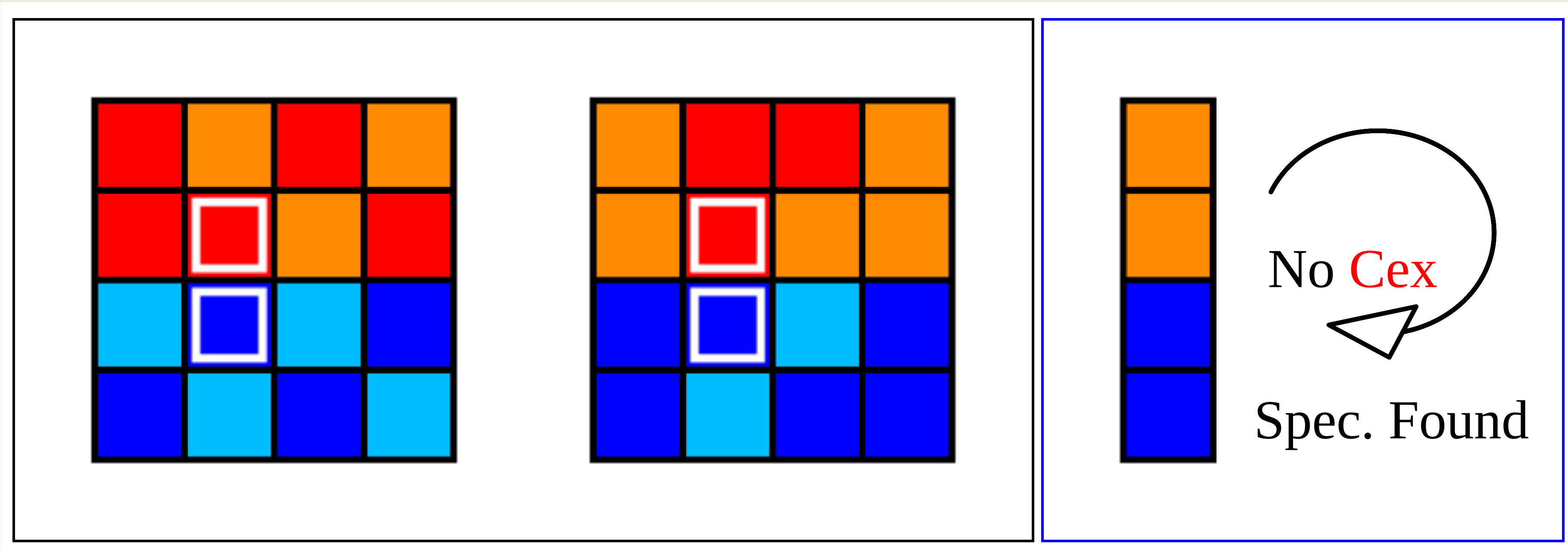
CrowdMine: Gamification and Crowdsourcing

CrowdMine

Traces



Web-based Game Prototype



Two Sampled Subtraces

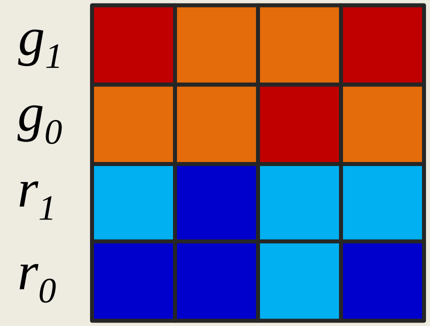
Counterexample

Selected Patterns → LTL Formulas → Model Checker

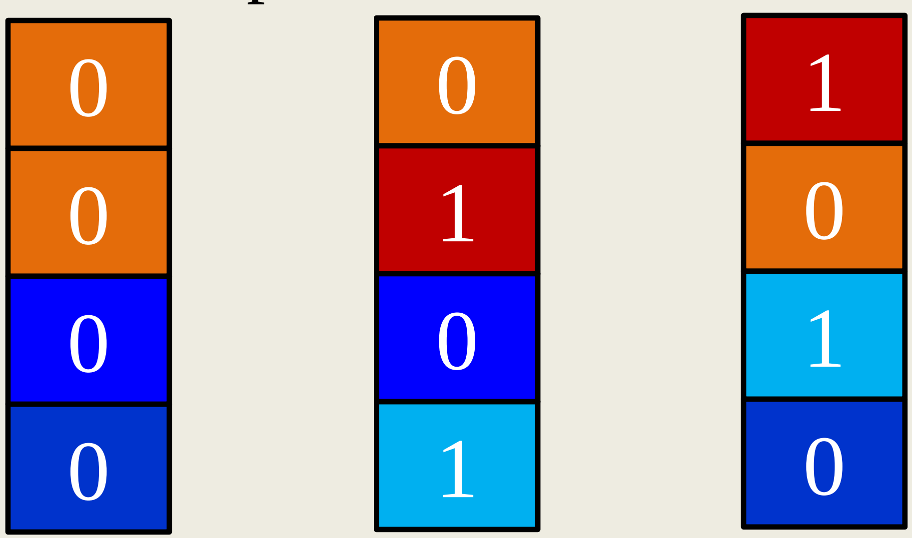
[URL: http://verifun.eecs.berkeley.edu/crowdmine2/]

Preliminary Results

- Circuit: I/O traces from a 2-input 2-output arbiter.



- Top ranked patterns:



"When r_1 is high and there is no competing r_0, g_1 is high at the same cycle.

Discussion

Remark: w/o model checker in the loop.

- What are humans good at?
  - Visual recognition?

    Most frequently identified common patterns correspond to desired behaviors of the circuit.
  - Randomness?

    165 different patterns out of 283 hits (mostly EECS students)
    Top rank patterns have counts 31, 16 and 7.
- What problems do we crowdsource?

  Problems that require human input and insight, or ones that are hard to formally define.

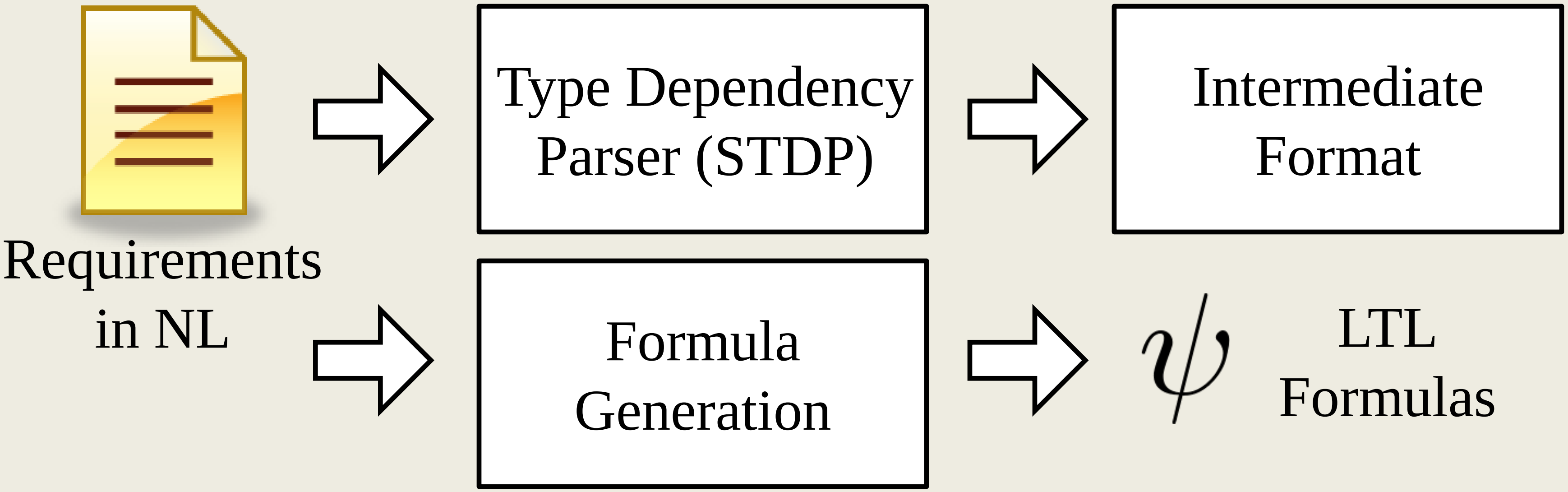
  - E.g. specification, diagnosis, repair.
  - *Not* purely computationally intractable problems.

Related Work: FunSAT/Human EDA [DeOrio and Bertacco, DAC 2009]

Human Inputs:

Mapping *Natural Language* to Temporal Logic

Natural Language → LTL Specification



Result highlights:

- FAA-Isolette requirements from NL to LTL.
- Assumption mining discovered a missing assumption.

Source: D. L. Lempia and S. P. Miller. Requirements engineering management handbook. Final Report DOT/FAA/AR-08/32, Federal Aviation Administration, June 2009.

Conclusion

Formal specifications can be mined in a systematic way to improve the effectiveness of verification and synthesis.

Formalisms
- Basis Subtrace
- Version-Space Learning

Algorithms
- Automata-Based
- Sparse Coding
- Counterstrategy-Guided

Applications
- Bug Localization
- LTL Synthesis
- Human-in-the-Loop Controller

Future Work

- Combine automata-based and sparse coding-based approaches for mining specifications.
- Improve the scalability of the sparse coding-based approach.
- Mining assumptions in contract-based synthesis.
- Evaluate human-in-the-loop controller synthesis in real setting.
- Human studies of CrowdMine for large designs.
- More robust NL→LTL techniques.

Thank you!