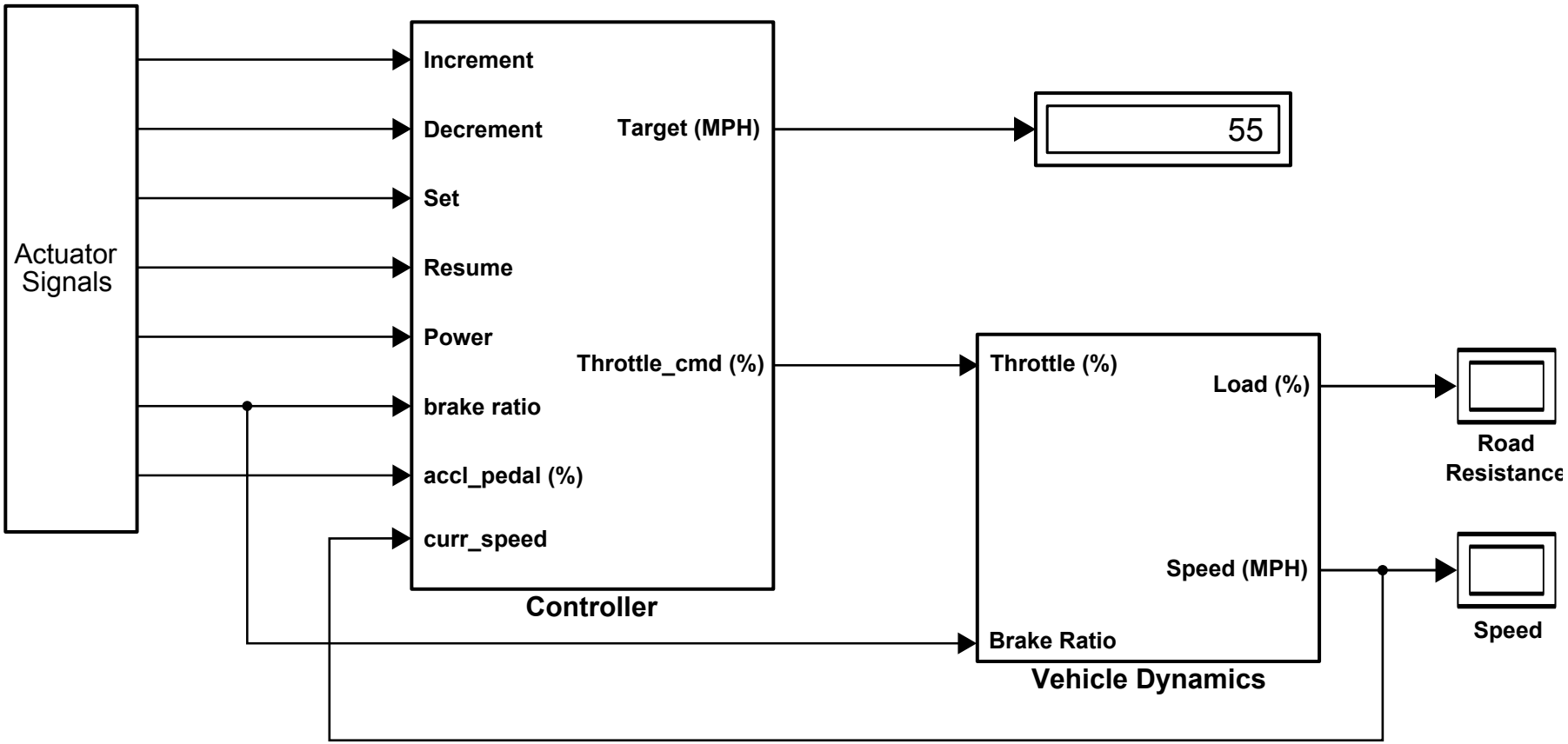
Automatic Requirements Extraction from Test Cases

Chris Ackermann Rance Cleaveland Samuel Huang
Arnab Ray Charles Shelton Elizabeth Latronico

October 1, 2009

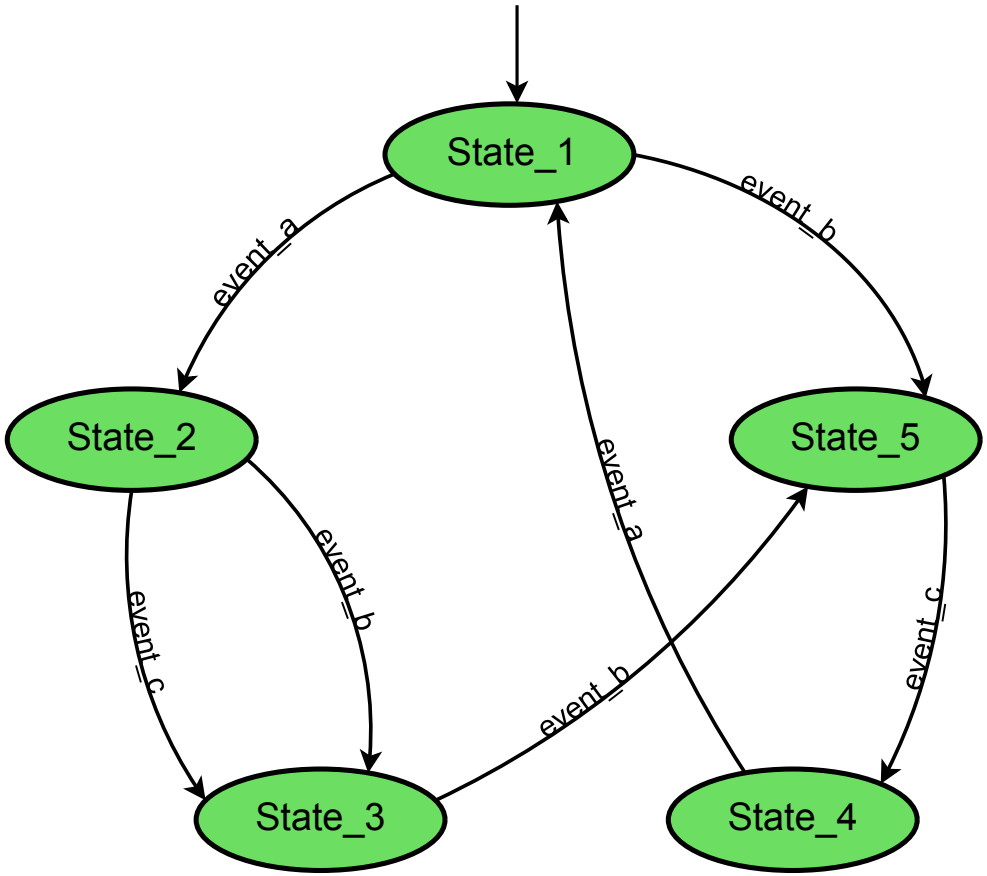
Requirements Extraction - Cruise Control Example



A number of ports and device variables are involved, one might wonder if some were needed or the interface could otherwise be simplified. Perhaps knowing about certain *properties* of the system may help to make such simplifications.

State Machines

The models are often *modal*, and can be abstracted to state machines.

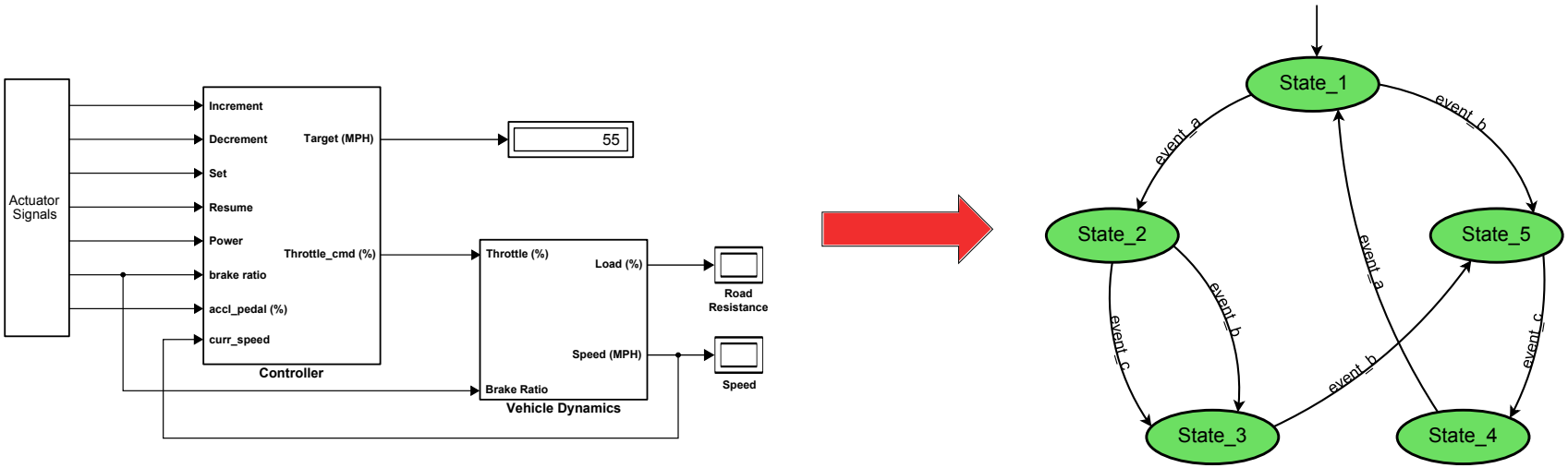


- Edges indicate transitions between states
- Annotations represent properties of nodes and edges
- Sample invariants:

Node If in `state_4`, then annotation `annot_a` applies (not shown)

Edge If in `state_2` and `event_b` occurs, go to `state_3`

Relating Models to State Machines



Now we have invariants like "if gas is applied, cancel cruise control" and "if cruise control is off and the switch is pressed, turn cruise control on."

Motivation

We are oftentimes given designs/models of a system that is said to accomplish some objective. In many of these scenarios, aspects of the model and its behavior are unstated or *implied*.

One Possible Question

"Given model M, does it exhibit property ϕ?"

A Different Question

"Given model M, what is the *set* of properties Φ it exhibits?"

The first question is one of validation, while the second is one of *discovery*.

Problem Statement

Models. . .

- . . . often come with specifications . . .
- . . . some of which are incomplete . . .
- . . . some of which conflict with the implementation.

Can we refine a model (or its specifications) to make implicit properties explicit, or show existing conflicts?

Our Problem

"Given model M with specification S, can we produce a refined specification S' capturing all properties Φ that M exhibits, or detect a conflict if one exists?"

We call such properties *invariants*, as they persist throughout a model's lifetime.

Capturing the Model's Behavior - Test Cases

Approach

We can simulate or otherwise explore a model's properties by examining its behavior; we can inspect behavior by generating test cases from the model itself.

- Several possible software options for generating test cases, we use Reactis.
- Each test case represents an execution sequence on the model, with a specific configuration of inputs and produced outputs (via simulation/execution).
- Would like the test cases to have good coverage over all states of the model.

But now, how do we want to discover invariants from this behavior?

Association Rule Mining

Background

A data mining technique, originates from the challenge to infer relationships between variables in databases, useful for determining which information is more essential or otherwise crucial to know.

Simple association rules

$$
\begin{aligned}
\{\texttt{apples},\texttt{bananas}\} &\Longrightarrow \{\texttt{oranges}\} \\
\{\texttt{onions},\texttt{potatoes}\} &\Longrightarrow \{\texttt{beef}\} \\
\{\texttt{laptop}\} &\Longrightarrow \{\texttt{laptop\_case},\texttt{mouse}\} \\
\{\texttt{diapers},\texttt{male\_customer}\} &\Longrightarrow \{\texttt{beer}\}
\end{aligned}
$$

Using Association Mining

As applied to state machines

Currently, we seek to discover or invalidate edge transitions.

So...

premise	$\Longrightarrow$	consequent	(general AR)
$\bigwedge a_i$	$\Longrightarrow$	b	(Horn Clause)
`diapers & is_male`	$\Longrightarrow$	`father`	(Class AR, or CAR)

Here we try to conclude new state transitions, so we only really care about association rules where the consequent is a **state** variable, which we can consider to be a class attribute.

On the Complexity of Association Mining

- Most formulations of the problem are NP-complete
- Search heuristics are almost always utilized to prune out unlikely paths
- Common search criterion for a rule "$X \longrightarrow y$" ([Web07]):
  - Support: Count of how many tuples have all $X \cup \{y\}$
  - Confidence: MLE of probability $P(y|X)$
  - Lift: $\triangleq conf(X \longrightarrow y)/conf(\emptyset \longrightarrow y)$
  - Leverage: "difference between support and support if X and y were independent"
  - Strength

Results of Association Mining

Tools used

Utilized Magnum Opus, a data mining tool for association mining

```
pressed=1.0 -> new=1.0
pressed=2.0 -> new=7.0
pressed=1.0 & state=1.0 -> new=1.0
state=1.0 & pressed=2.0 -> new=7.0
pressed=1.0 & state=7.0 -> new=1.0
state=1.0 -> new=1.0
pressed=2.0 & requested=7.0 -> new=7.0
state=1.0 -> new=7.0
state=7.0 -> new=1.0
state=1.0 & requested=2.0 -> new=8.0
```

How to validate the association rules?

- Need to determine if any association rules are not invariants, i.e. there are some execution flows that violate the putative rules.
- *The fact that we did not exhaustively enumerate all possible runs allows for types of cases that we can potentially miss.*

Example of a violated association

Maybe "When it rains, it pours" is not quite right, because it could lightly rain, but we concluded this because we only saw thunderstorms.

How to validate the association rules?

- Need to determine if any association rules are not invariants, i.e. there are some execution flows that violate the putative rules.
- *The fact that we did not exhaustively enumerate all possible runs allows for types of cases that we can potentially miss.*

Example of a violated association

Maybe "When it rains, it pours" is not quite right, because it could lightly rain, but we concluded this because we only saw thunderstorms.

How to validate the association rules?

- Need to determine if any association rules are not invariants, i.e. there are some execution flows that violate the putative rules.
- *The fact that we did not exhaustively enumerate all possible runs allows for types of cases that we can potentially miss.*

Example of a violated association

Maybe "When it rains, it pours" is not quite right, because it could lightly rain, but we concluded this because we only saw thunderstorms.

Monitor Models

What it is

Machinery meant to capture a particular association rule in the normal model design notation.

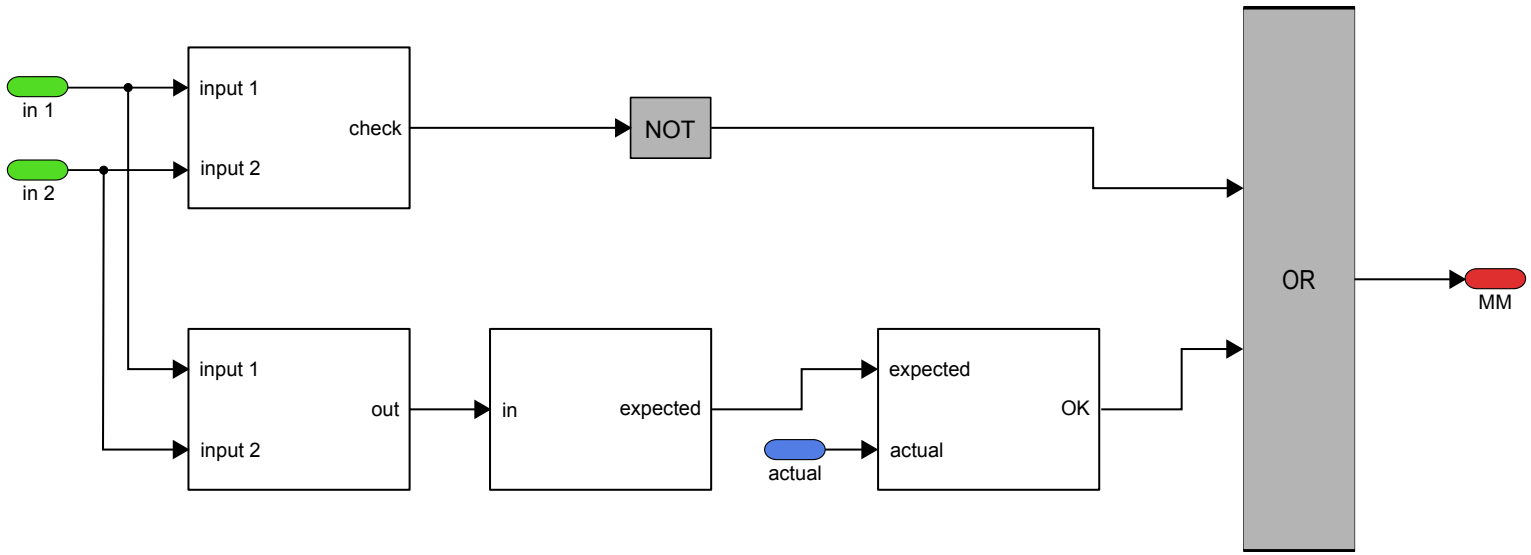


Figure: Example encoding of invariant using `input1` and `input2` as premise, and `actual` as consequent, i.e. "`input1 & input2 -> actual`"

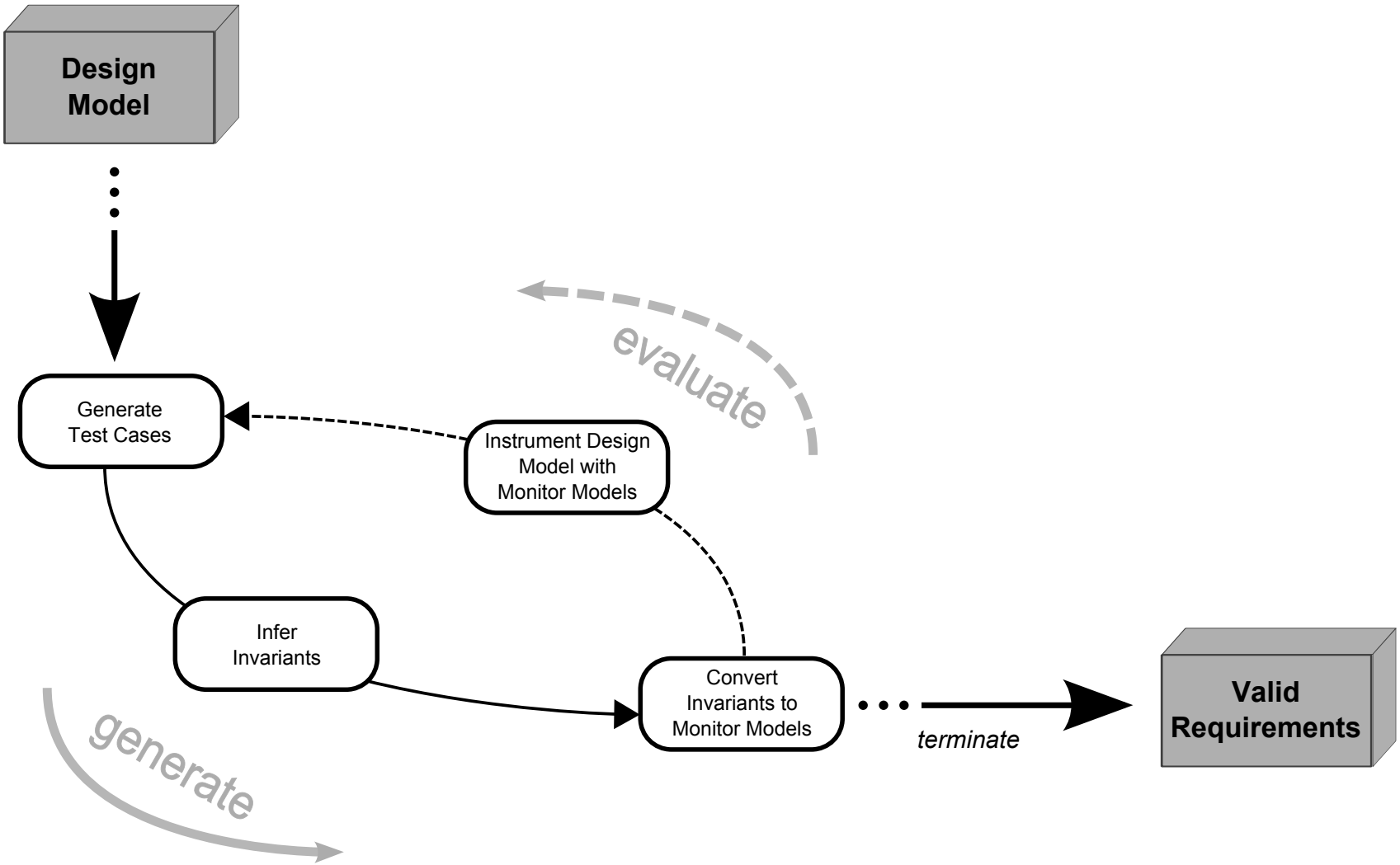
Incorporating Monitor Models into original Model

Once we have built monitor models for our putative invariants, we register them to be interfaced with the original model in Reactis.

Can't use... :(

But now we have a "new" design model, the original plus some new monitor models. Can we repeat the procedure?

Overall Requirement Extraction Process



Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of random test runs, and aggregating them with the first batch.

Each experiment is done 5 times.

Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of random test runs, and aggregating them with the first batch.

Each experiment is done 5 times.

Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of random test runs, and aggregating them with the first batch.

Each experiment is done 5 times.

Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of randomized runs, and aggregating them with the first batch

Each experiment is done 5 times.

Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of randomized runs, and aggregating them with the first batch

Each experiment is done 5 times.

Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of randomized runs, and aggregating them with the first batch

Each experiment is done 5 times.

Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of randomized runs, and aggregating them with the first batch

Each experiment is done 5 times.

Overview of Experiments

The following types of experiments were run:

- Single iteration
  - Run full coverage
  - Run partial coverage
- Second iteration
  - Run full coverage with second iteration, using monitor models
  - Run partial coverage with second iteration
    - This is just generating a second batch of randomized runs, and aggregating them with the first batch

Each experiment is done 5 times.

Raw Results

Full

Run #	Iter 1	# Invalid	Net	Iter 2	# Invalid	Net
1	26	8	18	40	1	39
2	34	6	28	40	2	38
3	30	9	21	38	1	37
4	33	7	26	42	1	41
5	34	7	27	38	0	38
Avg	31.4	7.4	24	39.6	1	38.6

Partial

Run #	Iter 1	# Invalid	Net	Iter 2	# Invalid	Net
1	19	11	8	29	13	16
2	22	11	11	27	10	17
3	26	12	14	34	9	25
4	26	13	13	32	15	17
5	25	6	19	35	3	32
Avg	23.6	10.6	13	31.4	10	21.4

Raw Results

Full

Run #	Iter 1	# Invalid	Net	Iter 2	# Invalid	Net
1	26	8	18	40	1	39
2	34	6	28	40	2	38
3	30	9	21	38	1	37
4	33	7	26	42	1	41
5	34	7	27	38	0	38
Avg	31.4	7.4	24	39.6	1	38.6

Partial

Run #	Iter 1	# Invalid	Net	Iter 2	# Invalid	Net
1	19	11	8	29	13	16
2	22	11	11	27	10	17
3	26	12	14	34	9	25
4	26	13	13	32	15	17
5	25	6	19	35	3	32
Avg	23.6	10.6	13	31.4	10	21.4

Other Statistics

We've computed several statistics on the results, as well as keeping the raw results:

- Total fraction of recovered invariants
  - From estimated ground truth - formed by aggregating all invariants found to not be invalid over any experiment
  - Estimation found 42 true invariants
- False Positive (FP) and False Negative (FN) rates
- Jaccard similarity between invariant sets
  - Unlike the other measures, this looks at similarity *between* invariant sets, so as to compare how similar any two test runs are.

Other Statistics

We've computed several statistics on the results, as well as keeping the raw results:

- Total fraction of recovered invariants
  - From estimated ground truth - formed by aggregating all invariants found to not be invalid over any experiment
  - Estimation found 42 true invariants
- False Positive (FP) and False Negative (FN) rates
- Jaccard similarity between invariant sets
  - Unlike the other measures, this looks at similarity *between* invariant sets, so as to compare how similar any two test runs are.

Other Statistics

We've computed several statistics on the results, as well as keeping the raw results:

- Total fraction of recovered invariants
  - From estimated ground truth - formed by aggregating all invariants found to not be invalid over any experiment
  - Estimation found 42 true invariants
- False Positive (FP) and False Negative (FN) rates
- Jaccard similarity between invariant sets
  - Unlike the other measures, this looks at similarity *between* invariant sets, so as to compare how similar any two test runs are.

Other Statistics

We've computed several statistics on the results, as well as keeping the raw results:

- Total fraction of recovered invariants
  - From estimated ground truth - formed by aggregating all invariants found to not be invalid over any experiment
  - Estimation found 42 true invariants
- False Positive (FP) and False Negative (FN) rates
- Jaccard similarity between invariant sets
  - Unlike the other measures, this looks at similarity *between* invariant sets, so as to compare how similar any two test runs are.

Other Statistics

We've computed several statistics on the results, as well as keeping the raw results:

- Total fraction of recovered invariants
  - From estimated ground truth - formed by aggregating all invariants found to not be invalid over any experiment
  - Estimation found 42 true invariants
- False Positive (FP) and False Negative (FN) rates
- Jaccard similarity between invariant sets
  - Unlike the other measures, this looks at similarity *between* invariant sets, so as to compare how similar any two test runs are.

Other Statistics

We've computed several statistics on the results, as well as keeping the raw results:

- Total fraction of recovered invariants
  - From estimated ground truth - formed by aggregating all invariants found to not be invalid over any experiment
  - Estimation found 42 true invariants
- False Positive (FP) and False Negative (FN) rates
- Jaccard similarity between invariant sets
  - Unlike the other measures, this looks at similarity *between* invariant sets, so as to compare how similar any two test runs are.

Fraction Total, FP, and FN numbers

Full

Run #	Tot Frac$_1$	FP$_1$	FN$_1$	Tot Frac$_2$	FP$_2$	FN$_2$
1	0.43	0.31	0.57	0.93	0.03	0.07
2	0.67	0.18	0.33	0.90	0.05	0.10
3	0.50	0.30	0.50	0.88	0.03	0.12
4	0.62	0.21	0.38	0.98	0.02	0.02
5	0.64	0.21	0.36	0.90	0.00	0.10
Avg	0.57	0.24	0.43	0.92	0.03	0.08

Partial

Run #	Tot Frac$_1$	FP$_1$	FN$_1$	Tot Frac$_2$	FP$_2$	FN$_2$
1	0.19	0.58	0.81	0.38	0.45	0.62
2	0.26	0.50	0.74	0.40	0.37	0.60
3	0.33	0.46	0.67	0.60	0.26	0.40
4	0.31	0.50	0.69	0.40	0.47	0.60
5	0.45	0.24	0.55	0.76	0.09	0.24
Avg	0.31	0.46	0.69	0.51	0.33	0.49

Jaccard Similarity

Summary

Formally, the Jaccard Similarity of two sets A and B is defined to be

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

A measure of set overlap, with values ranging from 0 (no overlap) to 1 (totally equivalent sets).

Why it's a useful metric for us

We'd like to argue that any given run produces "roughly the same" set of invariants to another run. This is hopefully true for our experiments, but not for our baselines (which are more random).

Jaccard Statistics

$full_1$	0	1	2	3	4
0	1	0.53	0.86	0.52	0.67
1		1	0.63	0.64	0.72
2			1	0.62	0.71
3				1	0.61
4					1

$full_2$	0	1	2	3	4
0	1	0.88	0.85	0.90	0.83
1		1	0.92	0.88	0.81
2			1	0.86	0.83
3				1	0.88
4					1

$part_1$	0	1	2	3	4
0	1	0.46	0.47	0.62	0.35
1		1	0.47	0.60	0.58
2			1	0.59	0.43
3				1	0.52
4					1

$part_2$	0	1	2	3	4
0	1	0.74	0.52	0.74	0.50
1		1	0.45	0.70	0.48
2			1	0.56	0.63
3				1	0.48
4					1

	$part_1$	$part_2$	$full_1$	$full_2$
Min	0.35	0.45	0.52	0.81
Avg	0.51	0.58	0.65	0.87
Max	0.62	0.74	0.86	0.92

Contributions

- First known approach of applying data mining to requirement extraction
- Usage of monitor models allow for automatic validation of proposed requirements/invariants
- Established framework, and show benefits of structural coverage and iteration to performance

Future Work

- Assess relative completeness of modeling invariants using association rules - are any interesting classes of invariants undescribable?
- Temporal invariants [YE06]:

  "Within 5 time units of pressing the button, the alarm will sound."

  Will require looking at relations *between* instances of test data - no longer independent

Future Work

- Assess relative completeness of modeling invariants using association rules - are any interesting classes of invariants undescribable?
- Temporal invariants [YE06]:

  "Within 5 time units of pressing the button, the alarm will sound."

  Will require looking at relations *between* instances of test data - no longer independent

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami.

Mining association rules between sets of items in large databases.

In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM.

Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo.

Fast discovery of association rules.

In *Advances in knowledge discovery and data mining*, pages 307–328, Menlo Park, CA, USA, 1996. American Association for Artificial Intelligence.

Bing Liu, Wynne Hsu, and Yiming Ma.

Integrating classification and association rule mining.

pages 80–86, 1998.

Orna Raz.

Helping everyday users find anomalies in data feeds.

PhD thesis, Pittsburgh, PA, USA, 2004.

Chair-Shaw, Mary.

Tobias Scheffer.

Finding association rules that trade support optimally against confidence, 2001.

Geoffrey I. Webb.

Efficient search for association rules.

In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–107, New York, NY, USA, 2000. ACM.

Geoffrey I. Webb.

Discovering significant patterns.

Mach. Learn., 68(1):1–33, 2007.

Geoffrey I. Webb and Songmao Zhang.

K-optimal rule discovery.

Data Min. Knowl. Discov., 10(1):39–79, 2005.

Jinlin Yang and David Evans.

Perracotta: mining temporal api rules from imperfect traces.

In *In 28th Internl. Conf. on Software Engineering (ICSE 2006*, pages 282–291, 2006.