Asynchronous Learning for Service Composition

Casandra Holotescu

Department of Computer and Software Engineering
Politehnica University of Timişoara

WESOA 2011

To correctly compose a system out of several services

To correctly compose a system out of several services

- we need behavioural models

To correctly compose a system out of several services

- we need behavioural models
- but are they always provided?

...no, they're not.

...no, they're not

So, what if we have at least one black-box service?

...no, they're not

So, what if we have at least one black-box service?

Then, we should:

- **learn** the behavioural model/s for **black-box** service/s
- compose the system
- enjoy

...no, they're not

So, what if we have at least one black-box service?

Then, we should:

- **learn** the behavioural model/s for **black-box** service/s
- compose the system
- enjoy

However... things are not that simple:

- Learning is **hard**. NP-hard.
- $\Rightarrow$ learned models only **approximate** real behaviour
- $\Rightarrow$ can we **safely** build a system using **imprecise** models?

Input-enabled inference

The star: L* (Angluin) algorithm.

- positive and negative sample traces
- can query for trace membership
- hypothesis automaton
- if counterexample $\Rightarrow$ refine hypothesis

Most active learning techniques use a variant of L* for black-box model inference.

Controllability issues

Trace membership query $\Rightarrow$ all events in the trace are **controllable**.

synchronous communication $\surd$
asynchronous ...??

Controllability issues

Trace membership query $\Rightarrow$ all events in the trace are **controllable**.

synchronous communication $\surd$
asynchronous ...??

- black-box receives message – **controllable** $\surd$
- black-box sends message – **uncontrollable** !!

Controllability issues

Trace membership query $\Rightarrow$ all events in the trace are **controllable**.

synchronous communication $\surd$
asynchronous ...??

- black-box receives message – **controllable** $\surd$
- black-box sends message – **uncontrollable** !!

Cannot query an asynchronous black-box for trace membership!!

Cannot use L* for asynchronous black-boxes!!

Problem Statement

a set $\mathcal{W} = \{W_0, W_1, ...W_{n-1}\}$

- of n **black-box** services
- which interact **asynchronously**

want to compose system S

Problem Statement

a set $\mathcal{W} = \{W_0, W_1, ...W_{n-1}\}$

- of n **black-box** services
- which interact **asynchronously**

want to compose system S

S **must comply** to safety property Φ

Problem Statement

a set $\mathcal{W} = \{W_0, W_1, ...W_{n-1}\}$

- of n **black-box** services
- which interact **asynchronously**

want to compose system S

S **must comply** to safety property Φ

goal: a property-enforcing adaptor (a service in the middle to control interactions in the system, s.t. the property is never violated)

Assumptions

A service W_i is associated to a Büchi automaton

$$U_i = \langle Q^i, q_0^i, Q_f^i, \Sigma^i, \delta^i \rangle$$

- Q^i is the state set, $q_0^i \in Q^i$ the initial state
- $Q_f^i = Q^i$ the set of accepting states
- Σ^i the event set (**send**/**receive** events)
- $\delta^i : Q^i \times \Sigma^i \rightarrow \mathcal{P}(Q^i)$ is the transition function

Assumptions

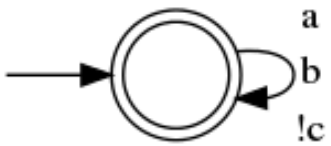
A service W_i is associated to a Büchi automaton

$$U_i = \langle Q^i, q_0^i, Q_f^i, \Sigma^i, \delta^i \rangle$$

- Q^i is the state set, $q_0^i \in Q^i$ the initial state
- $Q_f^i = Q^i$ the set of accepting states
- Σ^i the event set (**send**/**receive** events)
- $\delta^i : Q^i \times \Sigma^i \rightarrow \mathcal{P}(Q^i)$ is the transition function

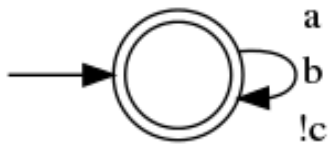
Each W_i is **fair by a bound** θ: if a state q is reached at least θ times, every uncontrollable event $\sigma \in \Sigma^i(q)$ is observed at least once.

Property Φ is also expressed as a Büchi automaton.



Each black-box W_i is associated to a **tentative** Büchi automaton U_i which is **most general** if we have no prior knowledge on its behaviour:

- $Q^i = Q^i_f = \{q^i_0\}$
- $\delta^i\left(q^i_0, \sigma\right) = \{q^i_0\}\, \forall \sigma \in \Sigma^i$.



Each black-box W_i is associated to a **tentative** Büchi automaton U_i which is **most general** if we have no prior knowledge on its behaviour:

- $Q^i = Q^i_f = \{q^i_0\}$
- $\delta^i\left(q^i_0, \sigma\right) = \{q^i_0\}\ \forall \sigma \in \Sigma^i$.

Also assume:

- all events are **observable** (special event *ack*!).
- each service **can be reset** anytime to its initial state (*rst*?).
- the number of states of U_i has an **upper bound** m

BASYL: Black-box Asynchronous Learning

Place an intelligent, proactive monitor in the middle.

Repeat:

- explore possible execution scenarios
- observe black-box reactions
- refine tentative model

until a satisfying refinement is found.

Synthesize property-enforcing controller.

Behaviour exploration at runtime

Let $U_\times$ be the asynchronous product of all models in the system: $U_\times = U_0 \times U_1 \times ... \times U_{n-1}$.

The property automaton Φ is executed synchronously with $U_\times$.

- messages sent are **intercepted**
- and **forwarded** to enable controllable transitions **conforming** to Φ

Thus, **only behaviour that conforms to Φ is explored**.

Exploration stops when no model U_i still contains **unobserved** and correct **controllable** transitions (fixpoint condition).

Model refinement

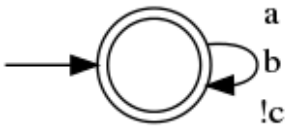
- after trace t, expected event $msg?$ **cannot be enabled** in $W_i \Rightarrow$ trace $t.msg?$ is **removed** from U_i.
- trace t_{bad} that **violates property** Φ is observed at runtime $\Rightarrow$ U_i is refined by **unrolling** t_{bad}.
- state q in U_i has been reached for **more than** θ **times** $\Rightarrow$ unobserved uncontrollable transitions from q are **pruned**.

Model refinement

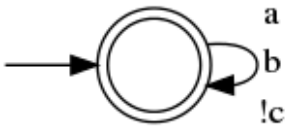
- after trace t, expected event $msg?$ **cannot be enabled** in $W_i \Rightarrow$ trace $t.msg?$ is **removed** from U_i.
- trace t_{bad} that **violates property** Φ is observed at runtime $\Rightarrow$ U_i is refined by **unrolling** t_{bad}.
- state q in U_i has been reached for **more than** θ **times** $\Rightarrow$ unobserved uncontrollable transitions from q are **pruned**.

Model U_i **overapproximates** the real **uncontrollable** behaviour of W_i and **underapproximates** its **controllable** behaviour.

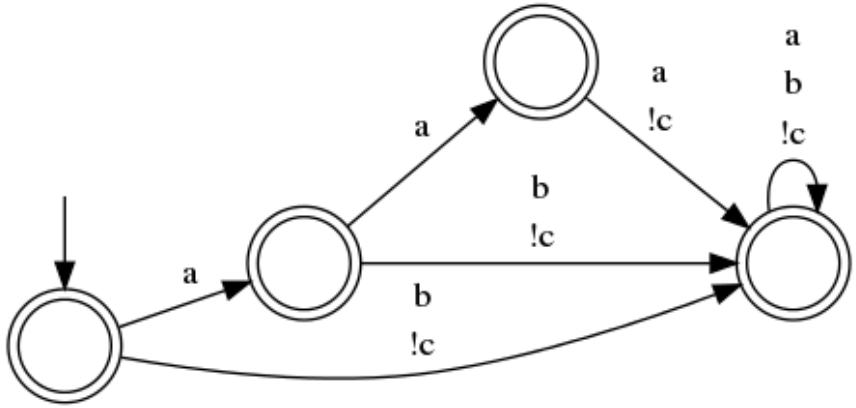
Model refinement



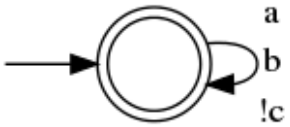
Model refinement



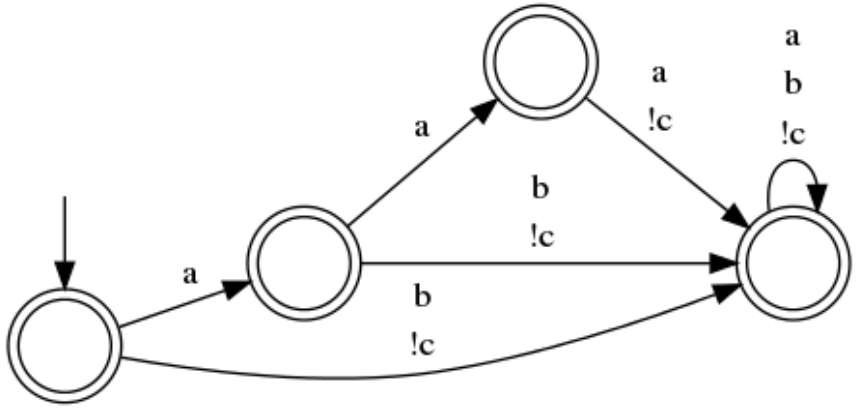
$a?\surd.a?\surd.b?\times$

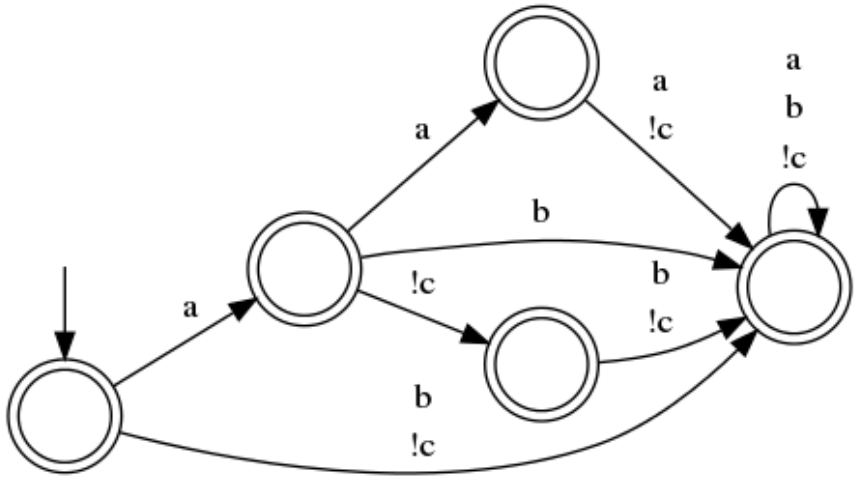


Model refinement



$a?\surd.a?\surd.b?\times$





$a?\surd.c!\surd.a?\times$

Adaptor synthesis

First compute **controller** *Ctrl*, that enforces property Φ over the system plant $U_\times$: $Ctrl = \mathbf{supcon}(U_\times, \Phi)$. [Ramadge and Wonham, 1989]

Adaptor *A* is obtained from **controller** *Ctrl* by **mirroring** its event set (send $\rightarrow$ receive, receive $\rightarrow$ send).

Adaptor synthesis

First compute **controller** *Ctrl*, that enforces property Φ over the system plant $U_\times$: $Ctrl = \mathbf{supcon}(U_\times, \Phi)$. [Ramadge and Wonham, 1989]

Adaptor *A* is obtained from **controller** *Ctrl* by **mirroring** its event set (send $\rightarrow$ receive, receive $\rightarrow$ send).

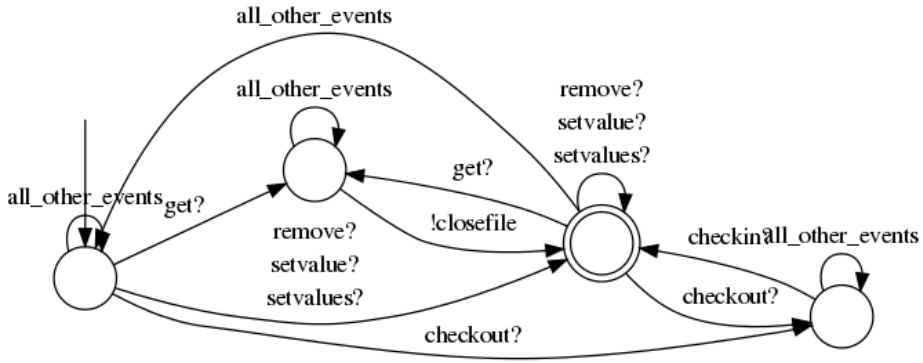
Each U_i **safely approximates** W_i:

overapproximates uncontrollable behaviour

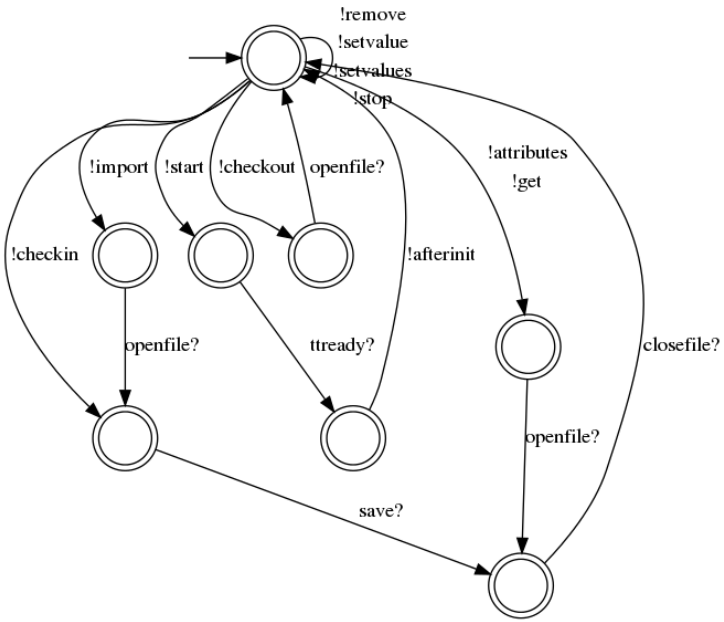
underapproximates controllable behaviour

$\Rightarrow$ *A* **will work for the real system.**

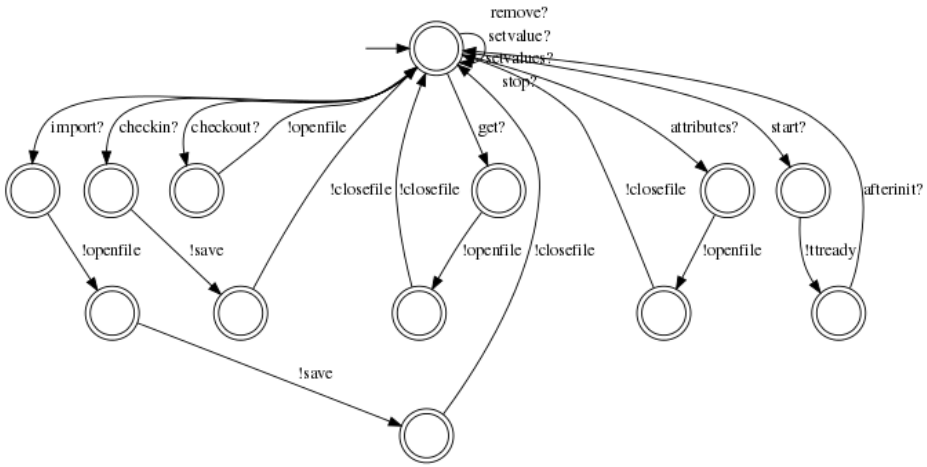
Case Study: CAD and Think Team [Tivoli, 2008]



Property

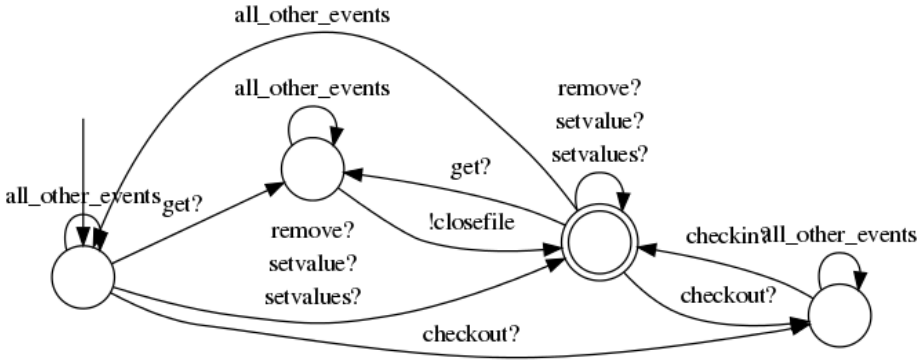


CAD

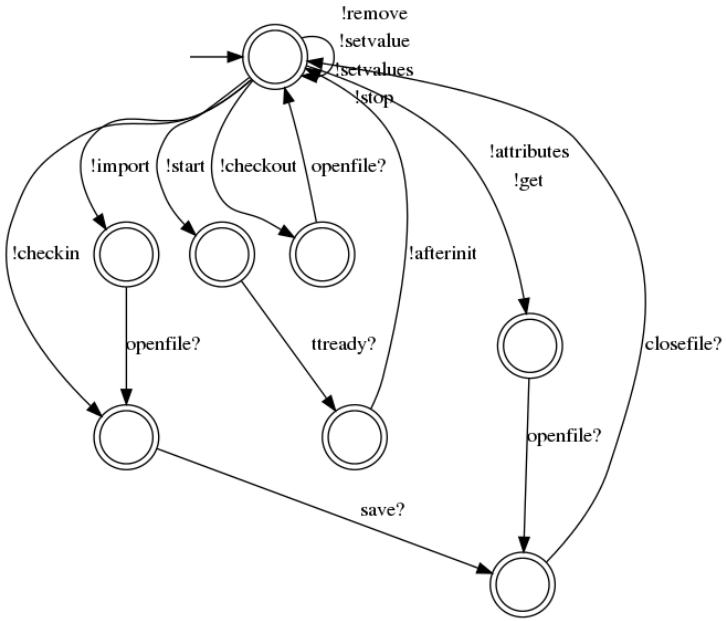


TT

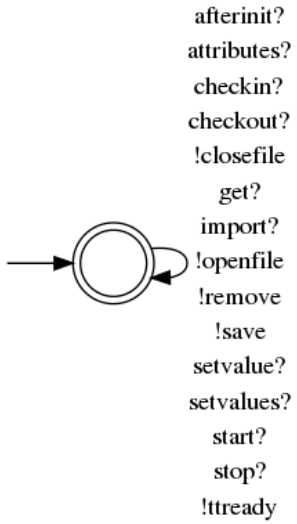
Case Study



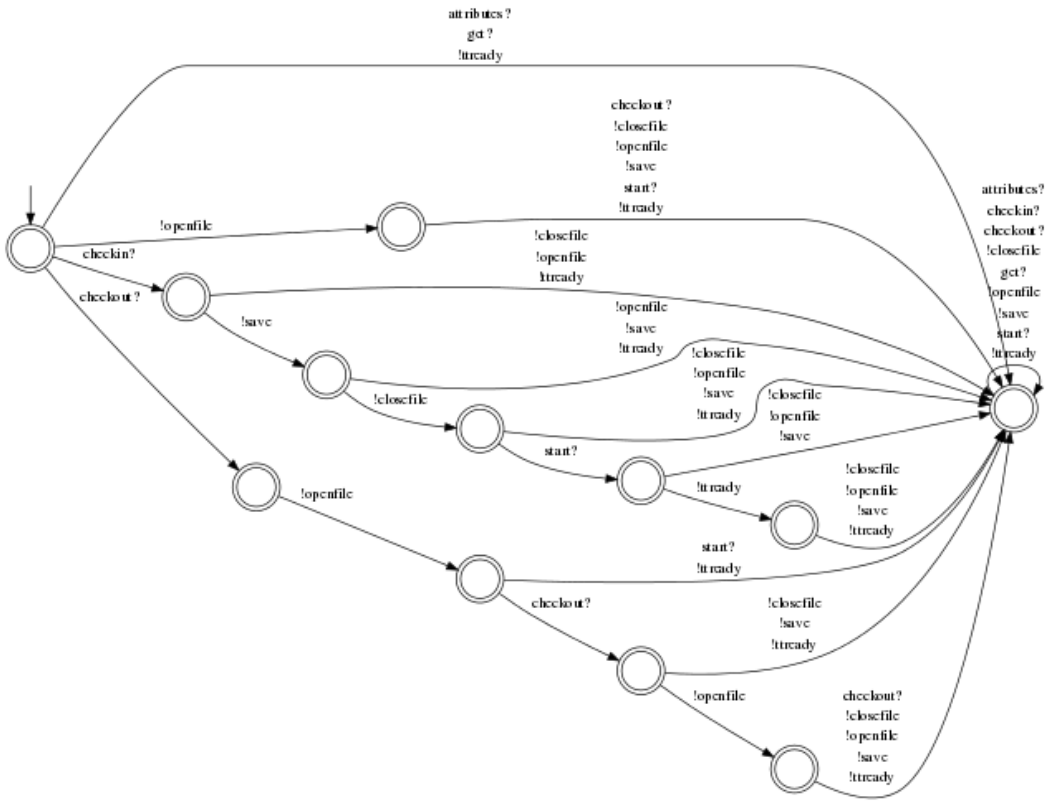
Property



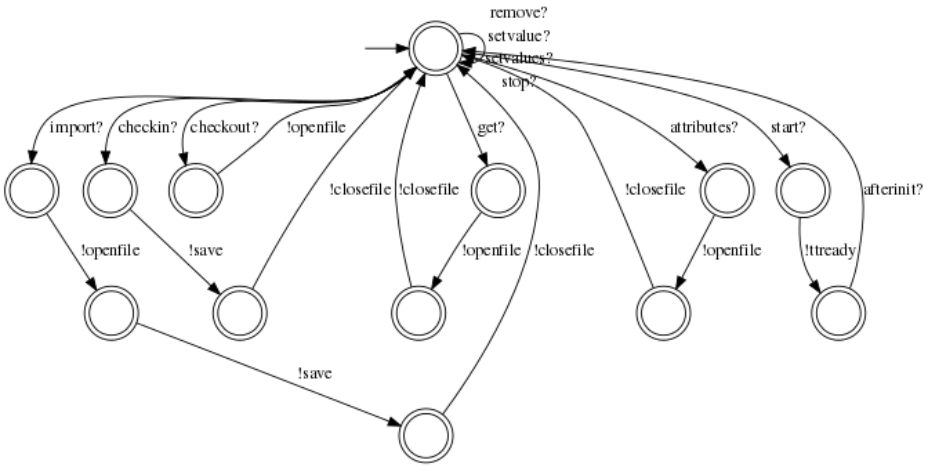
CAD



initial TT



learned TT model



real TT model

Model inference results for maximum 13 states

nr.	size	obs. msg?	feasible msg!	synthesis?	ctrl size	ctrl tr.
0	1	0	4	false	0	0
2	2	0	8	false	0	0
5	7	3	28	false	0	0
10	7	3	28	false	0	0
83	7	3	24	false	0	0
89	11	7	40	false	0	0
90	11	9	40	true	56	380
93	11	10	40	true	56	394
113	11	12	40	true	56	401
125	11	13	40	true	77	547
200	12	13	46	true	77	561
250	12	13	43	true	77	540
400	12	14	40	true	77	519
450	12	15	40	true	84	577
1000	12	15	40	true	84	577

BASYL can obtain more/less precise models upon how permissive we want the system controller to be:

- first possible controller $\Rightarrow$ shorter learning process
- most permissive controller

BASYL can obtain more/less precise models upon how permissive we want the system controller to be:

- first possible controller $\Rightarrow$ shorter learning process
- most permissive controller

max size	min ex. nr.	1st ctrl size	1st ctrl tr.	last ctrl size	last ctrl tr.
5	2	7	58	28	204
13	90	56	380	84	577
20	290	126	841	133	913
real = 13	**real**	**56**	**247**	**56**	**247**

Table: Controller variation by max. model size and permissiveness

Limitations

does not lead to minimal models

the model is never guaranteed to be complete

high complexity:

paths are explored at runtime, individually

uncontrollable events can steer the execution away

$\Rightarrow$ need to reduce the number of explored paths

Summary

- We presented a method to automatically compose an asynchronous system with black-box services.
- It learns only behaviour useful to the composition goal, enabling a safe composition.
- `BASYL` can obtain models precise enough for controller synthesis.

Summary

- We presented a method to automatically compose an asynchronous system with black-box services.
- It learns only behaviour useful to the composition goal, enabling a safe composition.
- `BASYL` can obtain models precise enough for controller synthesis.
- Future Work
  - Learn models locally.
  - Learn when only observing send events.
  - Reduce complexity.

Related Work

- black-box checking: model checking technique for systems with no given model [Peled, 2003]
- `StrawBerry`: learning behaviour for stateless black-box services [Bertolino et al, 2009]
- specification mining: extracting state models for black-boxes [Suman et al, 2010]
- GK-tail: positive traces and invariants to extract EFSMs [Lorenzoli et al, 2008]
- regular inference, Angluin-based [Berg et al, 2008]
- RALT: parametrized Mealy machines [Shabhaz, 2008]
- LearnLib, Libalf (L* extensions and optimizations)

Related Work

- SPY - extracting models for data abstractions [Ghezzi et al, 2009]
- testing for specification mining: extending a test suite by adding/removing method calls [Dallmeier et al, 2009]
- CrystalBall: execution steering and error prediction in distributed systems [Yabandeh et al, 2008]
- assumptions generation: environments for which a component satisfies a property [Păsăreanu and Giannakopoulou, 2008]
- smart play-out: lookahead technique for specification execution [Harel et al, 2007]
- CTL model-checking of systems with an unspecified component [Xie and Dang, 2005]

Model inference results for maximum 20 states

nr.	size	obs. msg?	feasible msg!	synthesis?	ctrl size	ctrl tr.
0	1	0	4	false	0	0
1	1	1	4	false	0	0
2	4	2	16	false	0	0
100	4	2	13	false	0	0
150	4	2	10	false	0	0
290	18	13	66	true	126	841
299	18	14	66	true	126	848
300	18	16	66	true	126	862
322	19	16	70	true	126	862
400	19	17	70	true	133	913
1000	19	17	70	true	133	913

Model inference results for maximum 5 states

nr.	size	obs. msg?	feasible msg!	synthesis?	ctrl size	ctrl tr.
0	1	0	4	false	0	0
1	1	1	4	false	0	0
2	1	2	4	true	7	58
3	2	2	8	true	14	102
5	4	3	16	true	28	197
41	4	3	13	true	28	176
125	4	3	10	true	28	155
452	4	5	10	true	35	220
456	4	7	10	true	28	183
462	4	9	10	true	28	197
467	4	10	10	true	28	204
1000	4	10	10	true	28	204