Combining Runtime Verification and Property Adaptation through Neural-Symbolic Integration

Alan Perotti

Guido Boella
Università di Torino

Artur d'Avila Garcez
City University London

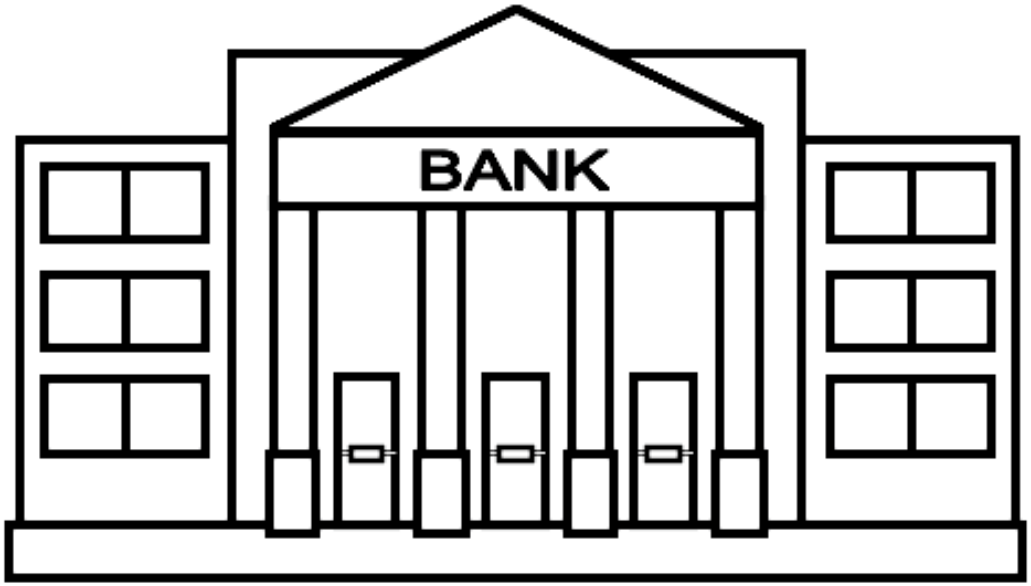
Research question

Is it possible to rigidly verify a system's compliance to a model, modelling at the same time what actually takes place in terms of process management, making it exploitable for other systems?

Quoting the Process Mining Manifesto, 'to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today's (information) systems'.

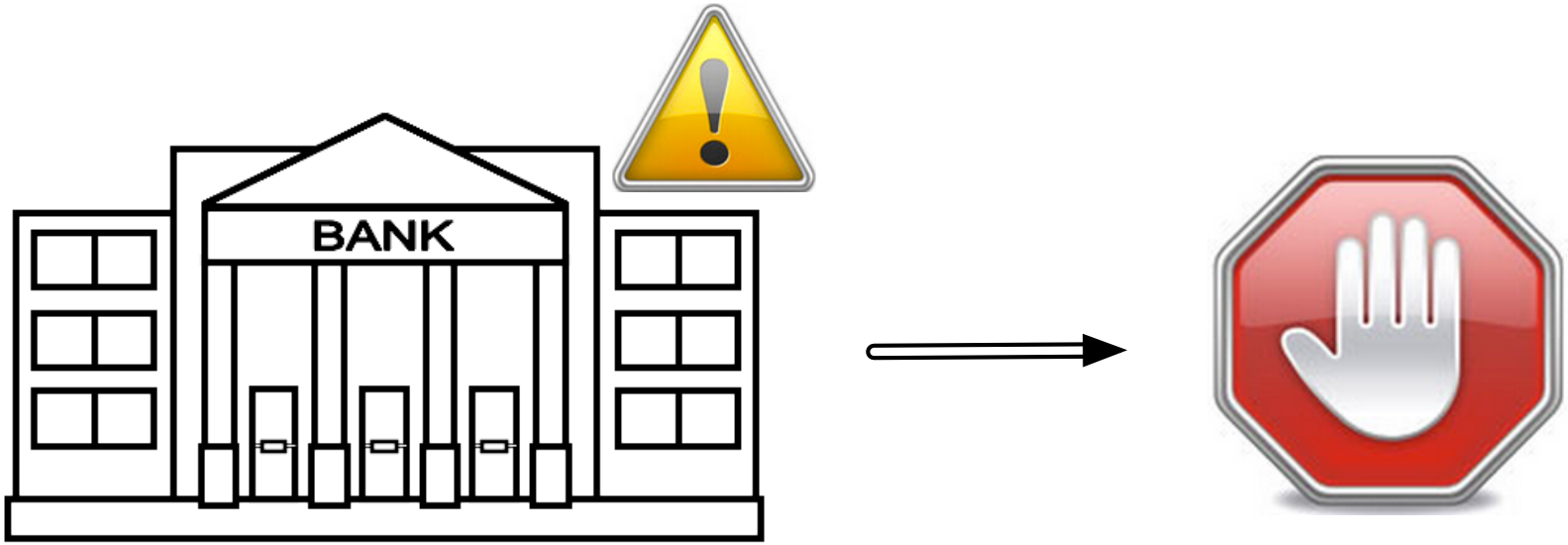
Scenario

When it comes to companies, in many small and medium enterprises (such as a bank's branch) the task execution often differs from the rigid protocol enforcement, due to obstacles (from broken printers to strikes) or by adaptations to specific needs (dynamic resources reallocation).



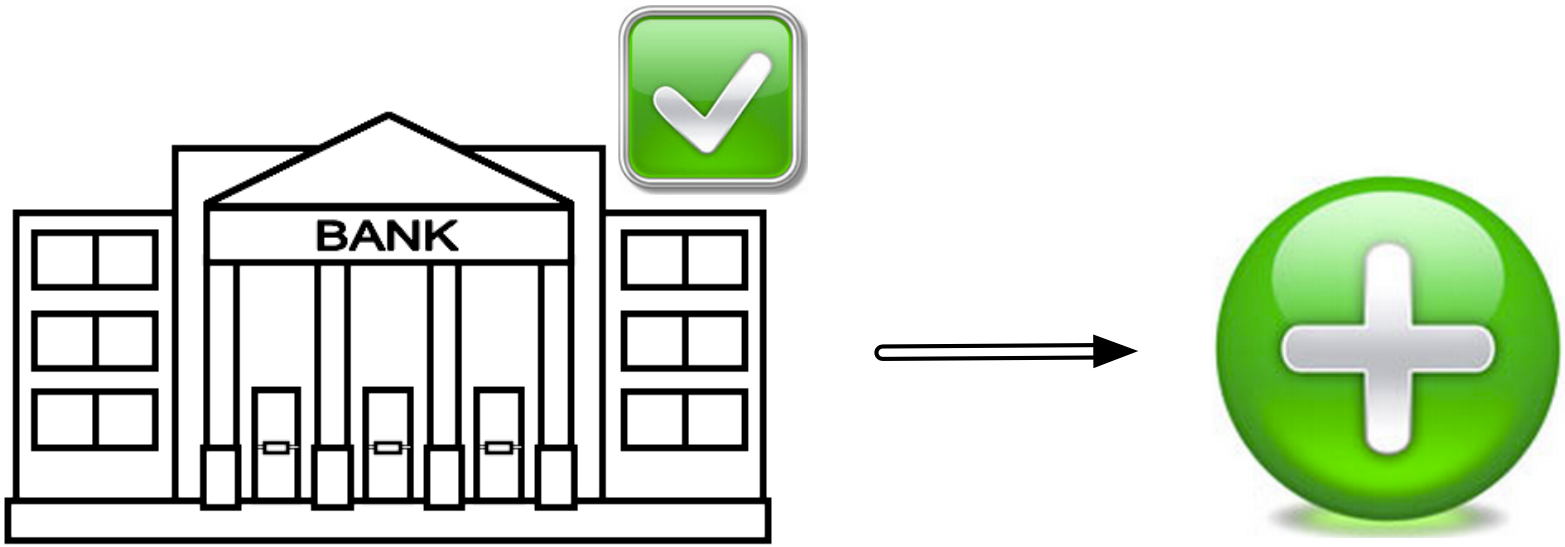
Scenario

Hard violations (security, privacy) have to be detected and prevented.



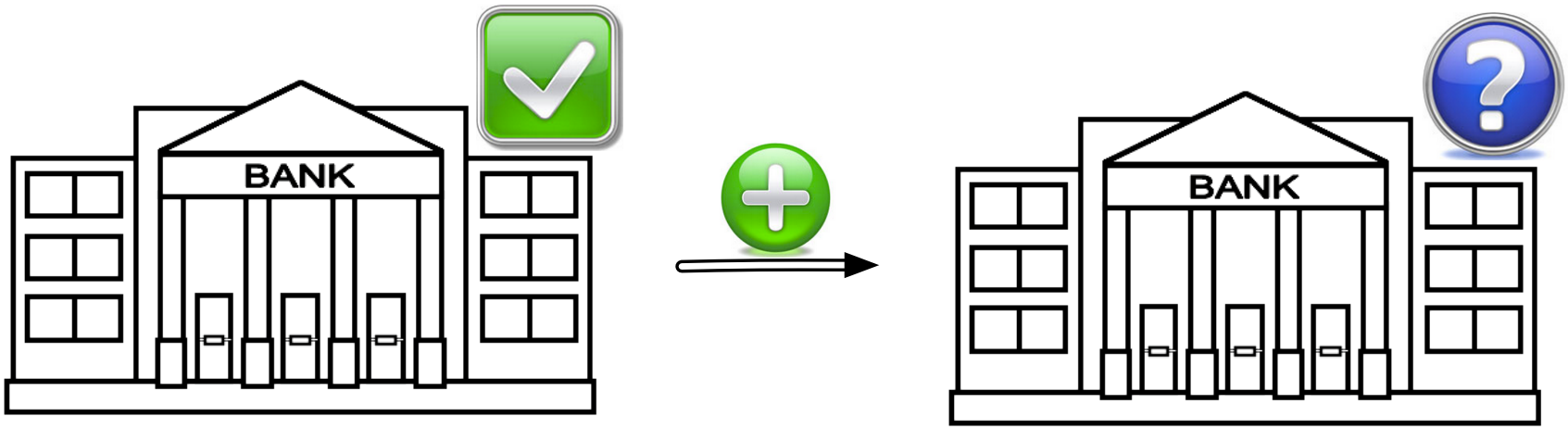
Scenario

Some *soft violations*, that is, discrepancies between high-level directives and real 'implementation', are unavoidable or even necessary. The model should be adapted in order to include them.

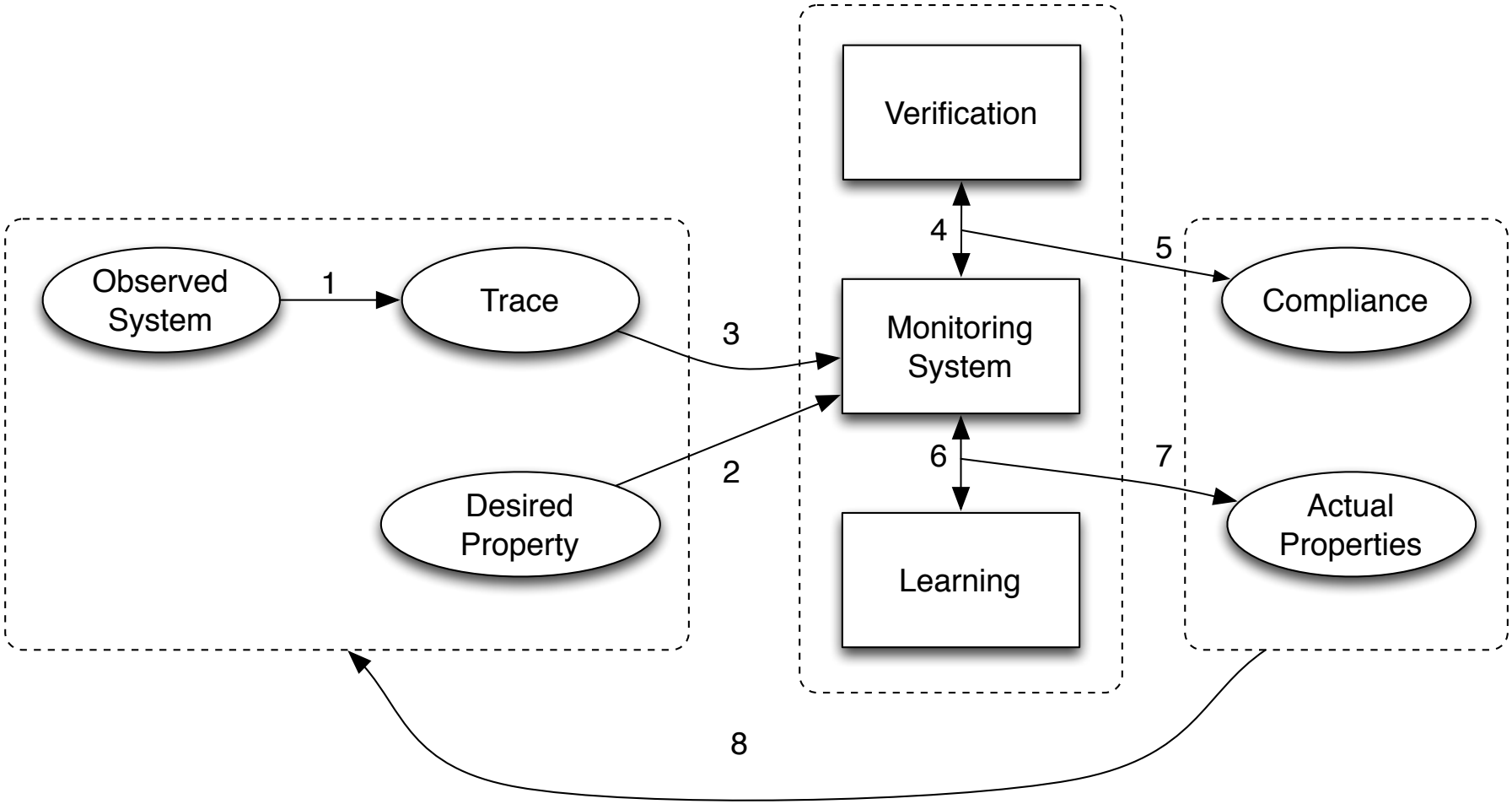


Scenario

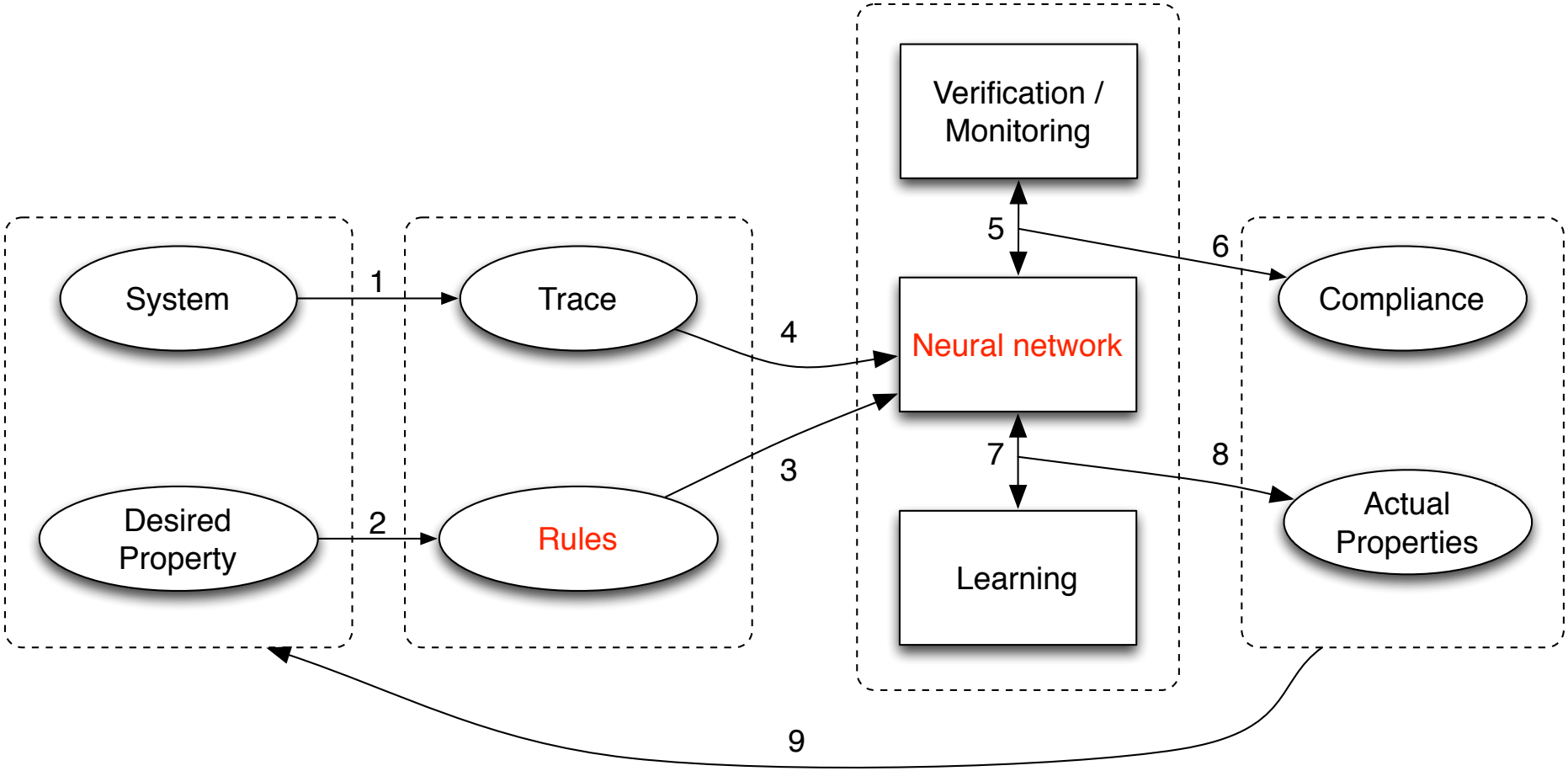
And the learned model can be used as a benchmark for new branches.



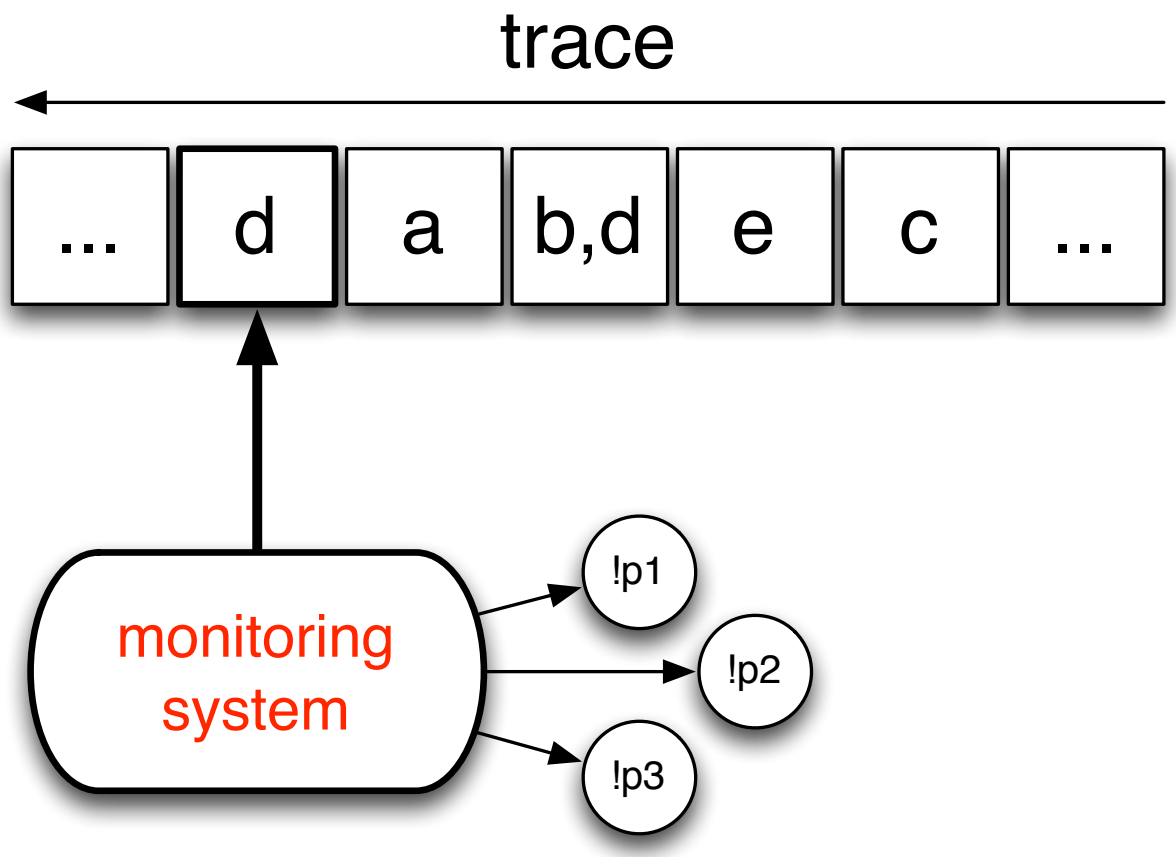
Proposed framework: components



Proposed framework: tools of choice



(Rule-based) monitoring



What kind of model checking is that?

- **blackbox**: we don't know the monitored system, we just OBSERVE its traces (this goes from reading logs of what human employees did to catching Java events with AspectJ)
- **real-time** (a.k.a. 'on the fly'): the trace is fed one cell at a time; and in general one is not allowed to explore the trace back and forth
- **finite traces**: by observing the automaton of a system, one can analyse infinite behaviours; but we can only observe traces. Bauer's LTL3 and RVLTL, for instance, deal with this aspect. Another related field is bounded model checking.
- **propositional LTL** is the language in which the properties we want to check are specified.

Dealing with one cell at a time

You can't peek into the future

Therefore, a third true value, *undecided*, is necessary (e.g. $\Diamond a$ when a is not observed).

You can't look back to check the past

The system needs to 'write down' what happened in the previous cells.

RuleRunner: the approach in a nuthsell

- We stick to the cell-by-cell verification approach.
- We use **rules** to compute the truth value of the encoded property in each cell.
- If some evaluation is pending, it's marked to be repeated; the monitoring loop goes on until the formula is satisfied/verified.

Rules

(simplified) RuleRunner rules for $\Diamond a$ are:

$$R[\Diamond a], [a]\,T \rightarrow [\Diamond a]\,T$$

$$R[\Diamond a], [a]F \rightarrow [\Diamond a]?$$

$$[\Diamond a]? \rightarrow R[a], R[\Diamond a]$$

Evaluation and reactivation rules. Active rules.

Rules for disjunction

(meta)Rules for $\phi \vee \psi$

- $R[\phi \vee \psi],\ \phi^{\top},\ \psi^{\top} \longrightarrow \phi \vee \psi^{\top}$
- $R[\phi \vee \psi],\ \phi^{\top},\ \psi^{?} \longrightarrow \phi \vee \psi^{\top}$
- $R[\phi \vee \psi],\ \phi^{\top},\ \psi^{\bot} \longrightarrow \phi \vee \psi^{\top}$
- $R[\phi \vee \psi],\ \phi^{?},\ \psi^{\top} \longrightarrow \phi \vee \psi^{\top}$
- $R[\phi \vee \psi],\ \phi^{?},\ \psi^{?} \longrightarrow \phi \vee \psi^{?}$
- $R[\phi \vee \psi],\ \phi^{?},\ \psi^{\bot} \longrightarrow \phi \vee \psi^{?}_{l}$
- $R[\phi \vee \psi],\ \phi^{\bot},\ \psi^{\top} \longrightarrow \phi \vee \psi^{\top}$
- $R[\phi \vee \psi],\ \phi^{\bot},\ \psi^{?} \longrightarrow \phi \vee \psi^{?}_{r}$
- $R[\phi \vee \psi],\ \phi^{\bot},\ \psi^{\bot} \longrightarrow \phi \vee \psi^{\bot}$
- $R[\phi \vee \psi]_{l},\ \phi^{\top} \longrightarrow \phi \vee \psi^{\top}$
- $R[\phi \vee \psi]_{l},\ \phi^{?} \longrightarrow \phi \vee \psi^{?}_{l}$
- $R[\phi \vee \psi]_{l},\ \phi^{\bot} \longrightarrow \phi \vee \psi^{\bot}$
- $R[\phi \vee \psi]_{r},\ \psi^{\top} \longrightarrow \phi \vee \psi^{\top}$
- $R[\phi \vee \psi]_{r},\ \psi^{?} \longrightarrow \phi \vee \psi^{?}_{r}$
- $R[\phi \vee \psi]_{r},\ \psi^{\bot} \longrightarrow \phi \vee \psi^{\bot}$

V

	$\psi^{\top}$	$\psi^{?}$	$\psi^{\bot}$
$\phi^{\top}$	T	T	T
$\phi^{?}$	T	?	?
$\phi^{\bot}$	T	?	$\bot$

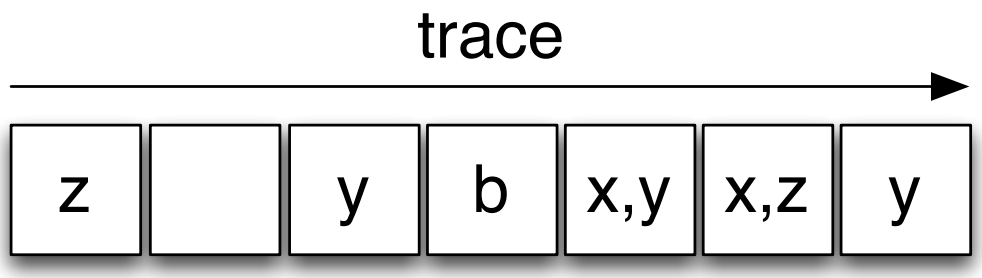
Vl

$\phi^{\top}$	T
$\phi^{?}$	?
$\phi^{\bot}$	$\bot$

Vr

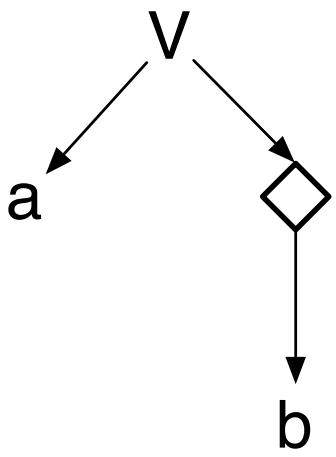
$\psi^{\top}$	T
$\psi^{?}$	?
$\psi^{\bot}$	$\bot$

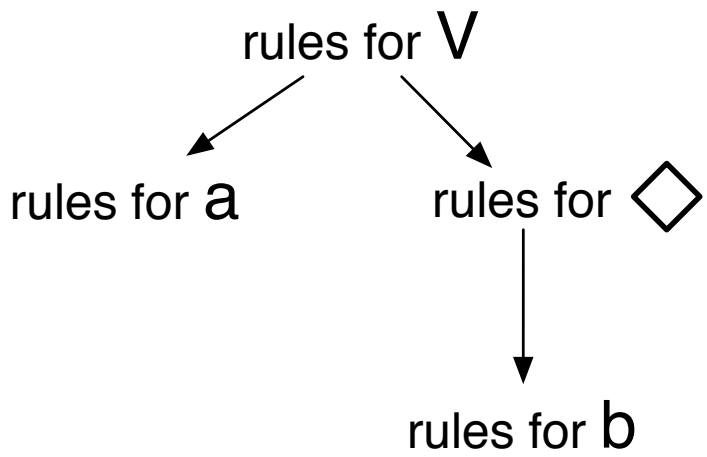
Rule-based monitoring example: trace and formula



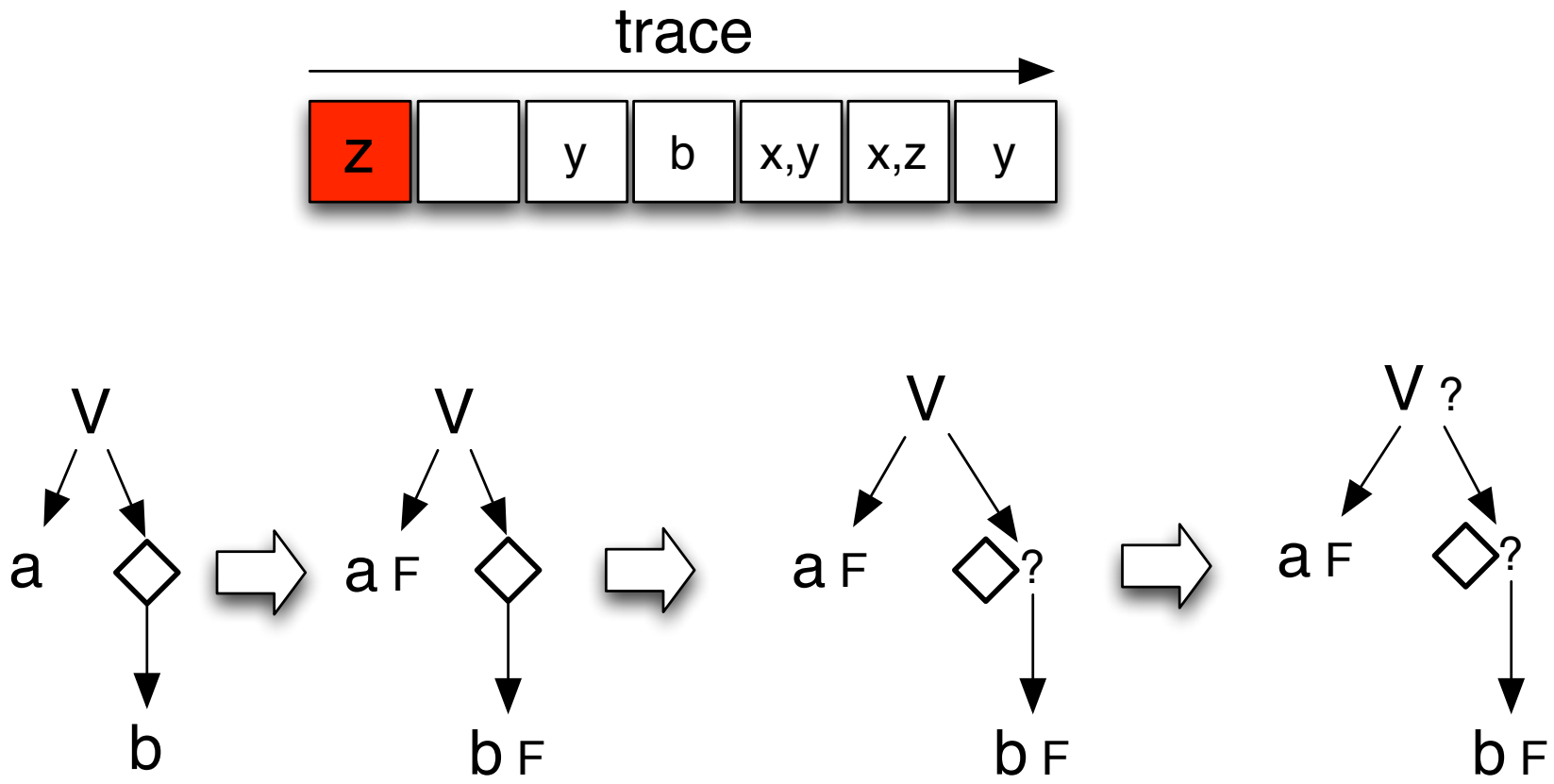
property: $a \vee \Diamond b$

Preprocessing: parsing, encoding





Running: cell monitoring

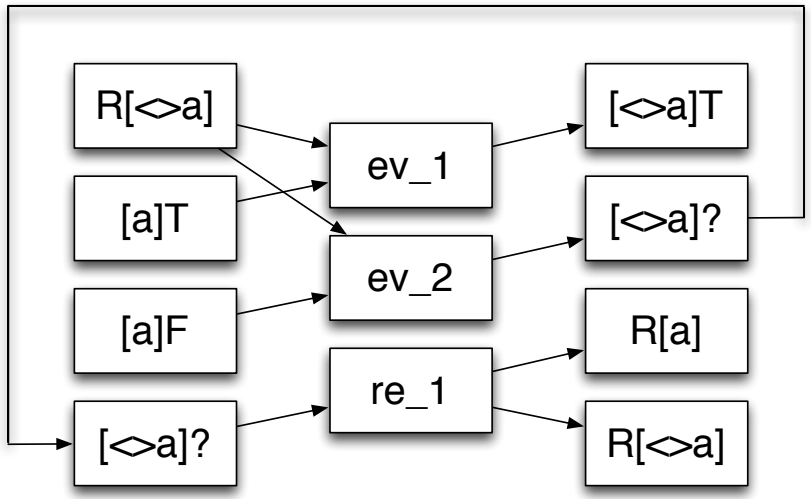


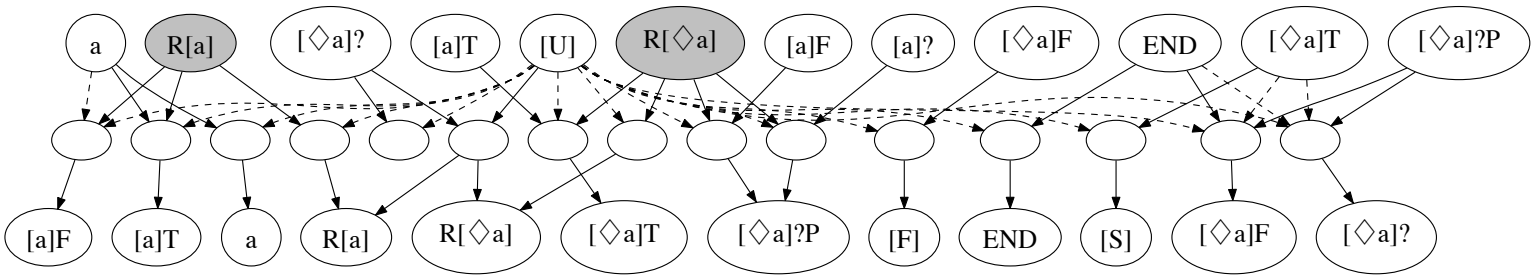
Rules as a network

$ev1)\ R[\Diamond a], [a]T \rightarrow [\Diamond a]T$

$ev2)\ R[\Diamond a], [a]F \rightarrow [\Diamond a]?$

$re1)\ [\Diamond a]? \rightarrow R[a], R[\Diamond a]$





Example: creating the network

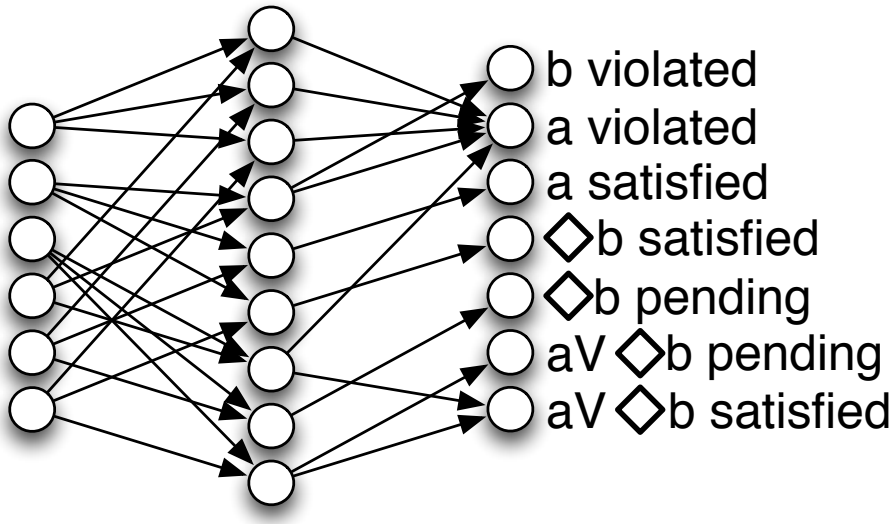
$$a \vee \Diamond b$$

rules for a

rules for b

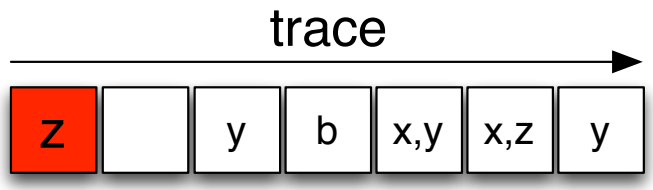
rules for $\Diamond$

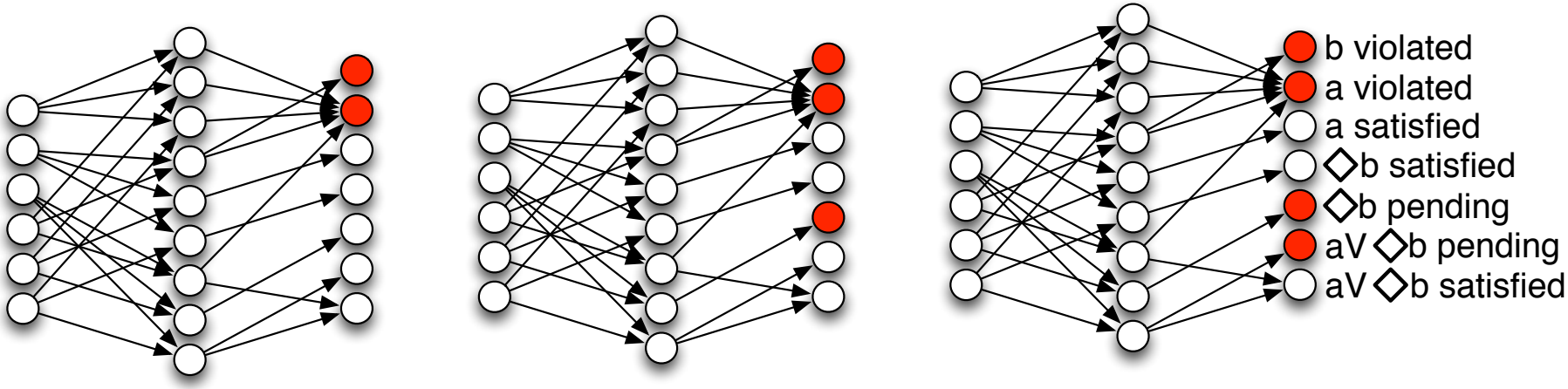
rules for V



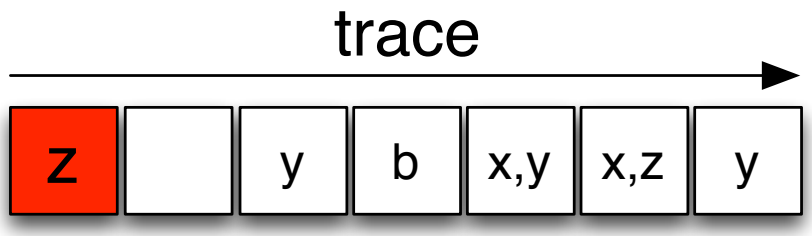
Example: firing the network

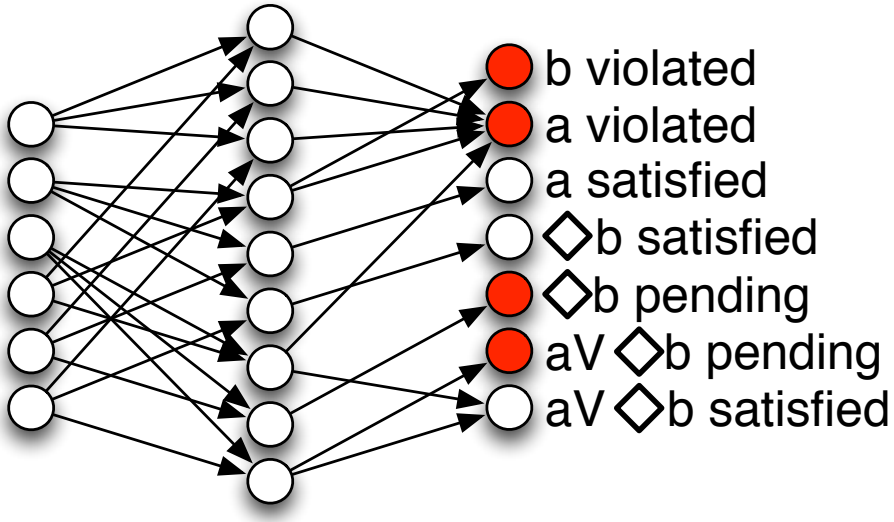
$a \vee \Diamond b$

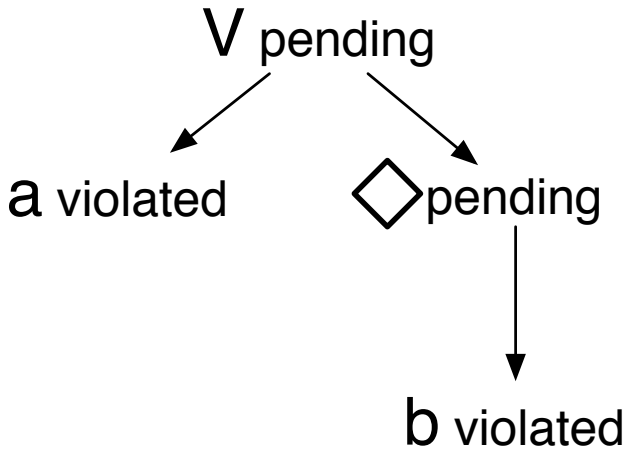




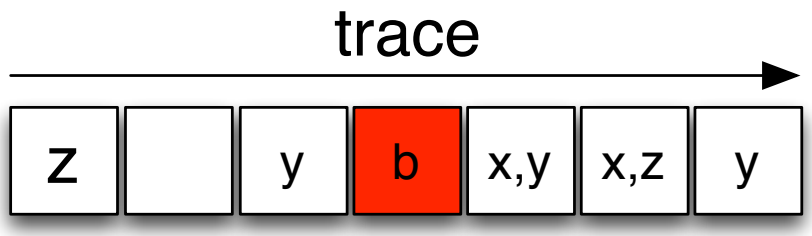
Example: convergence

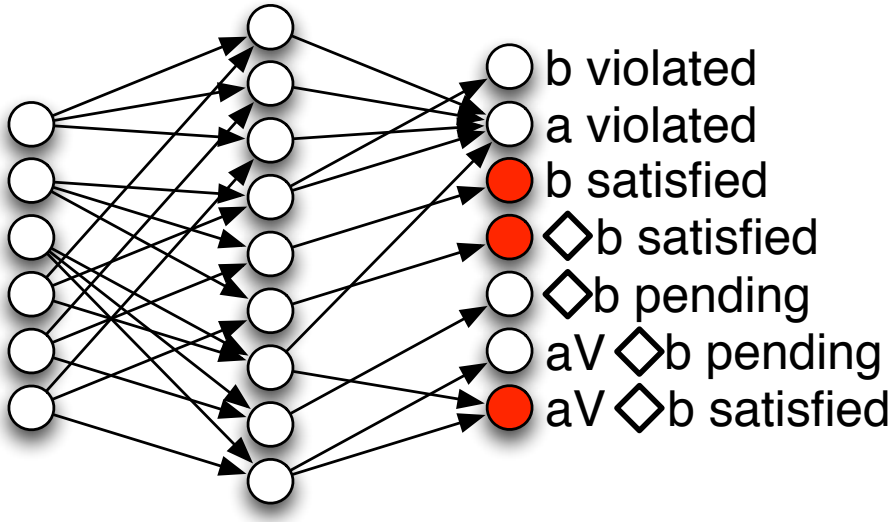


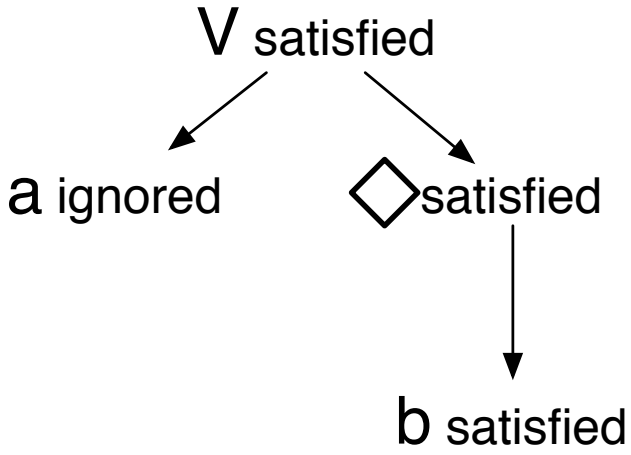




Example: convergence



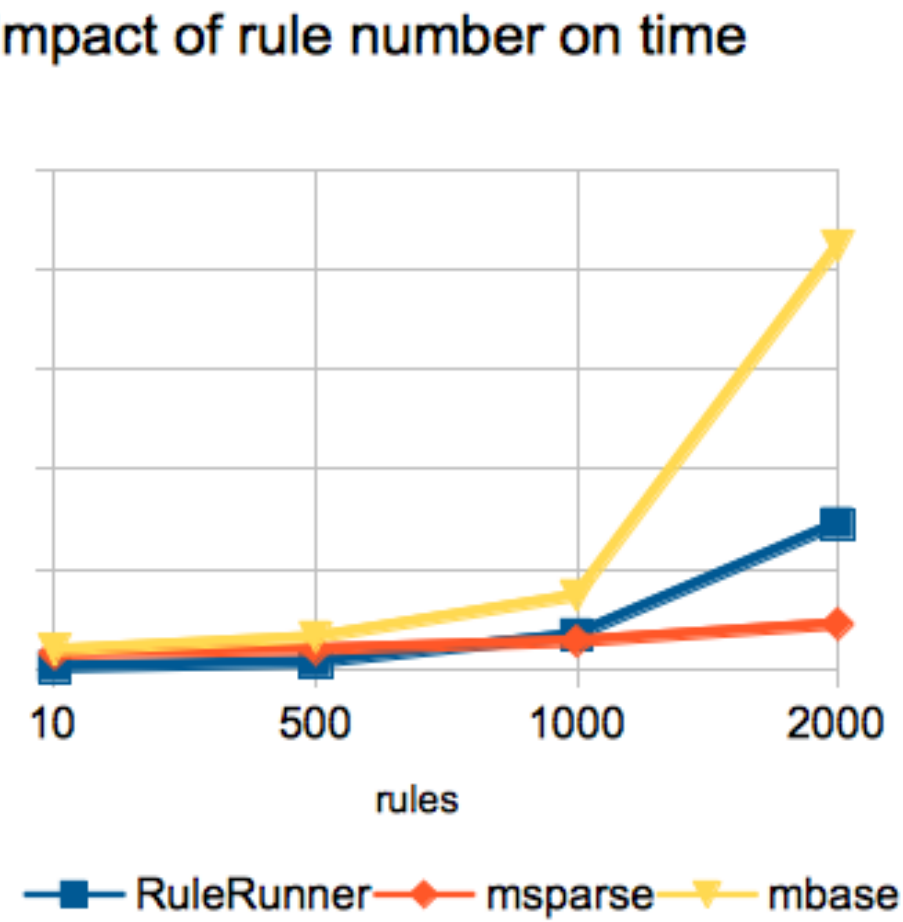




Wrap-up

- The (rule) language of RuleRunner allows for implicit representation of time and involves no 'alternatives'.
- Using CIL^2P, RuleRunner can be encoded in a standard neural network (one-hidden layer, feedforward, with recurrent connections).
- In every cell of the trace, the network converges and the activated output neurons represent the truth evaluations for each subformula of the monitored property (e.g. $[a]T, [\Diamond b]?..$).
- The obtained network is capable of performing LTL runtime verification over finite traces.
- RuleRunner is implemented in java. We implemented the neural encoding and tested various libraries for matrices manipulation (native, Jama, Cern's Colt, JBLAS..) with poor performances. Then we tried to declare the weight matrices as sparse.

Sparse matrices



X axis: number of RuleRunner rules.

Y axis: number of seconds required to monitor 10.000 random cells.

Conclusions

- We are interested in a neural-symbolic system for **combining reasoning and learning about temporal properties**.
- We **developed** an ad-hoc rule system, RuleRunner, that avoids 'possible worlds' and explicit time representation
- With minor adjustments, a RuleRunner rule system can be **encoded** in a standard neural network using CIL^2P
- By explicit sparse matrices manipulation, the neural network implementation **outperformed** the rule-based one.
- We are currently working on **learning and extraction**.

Thank you