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’.
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.
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.
And the learned model can be used as a benchmark for new branches.
Proposed framework: components

- Observed System
- Trace
- Desired Property
- Monitoring System
- Learning
- Verification
- Compliance
- Actual Properties

1. Observed System
2. Desired Property
3. Trace
4. Verification
5. Compliance
6. Learning
7. Actual Properties
8. Learning to Trace
Proposed framework: tools of choice

System \rightarrow Trace \rightarrow Neural network \rightarrow Verification / Monitoring \rightarrow Compliance

Desired Property \rightarrow Rules \rightarrow Learning \rightarrow Actual Properties

1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9
(Rule-based) monitoring

trace

... d a b, d e c ...

monitoring system

!p1
!p2
!p3
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. ♦ 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.
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.
(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.
(meta)Rules for $\phi \lor \psi$

- $R[\phi \lor \psi], \phi^T, \psi^T \rightarrow \phi \lor \psi^T$
- $R[\phi \lor \psi], \phi^T, \psi? \rightarrow \phi \lor \psi^T$
- $R[\phi \lor \psi], \phi^T, \psi\bot \rightarrow \phi \lor \psi^T$
- $R[\phi \lor \psi], \phi\bot, \psi^T \rightarrow \phi \lor \psi^T$
- $R[\phi \lor \psi], \phi\bot, \psi? \rightarrow \phi \lor \psi^T$
- $R[\phi \lor \psi], \phi\bot, \psi\bot \rightarrow \phi \lor \psi^T$
- $R[\phi \lor \psi], \phi^T, \psi\bot \rightarrow \phi \lor \psi^T$
- $R[\phi \lor \psi], \phi\bot, \psi\bot \rightarrow \phi \lor \psi\bot$
- $R[\phi \lor \psi], \phi\bot, \psi\bot \rightarrow \phi \lor \psi\bot$

\[
\begin{array}{ccc}
V & \psi^T & \psi? & \psi\bot \\
\phi^T & T & T & T \\
\phi? & T & ? & ? \\
\phi\bot & T & ? & \bot \\
\end{array}
\]

\[
\begin{array}{ccc}
V_l & \psi^T & \psi? & \psi\bot \\
\phi^T & T & & \\
\phi? & ? & & \\
\phi\bot & ? & & \\
\end{array}
\]

\[
\begin{array}{ccc}
V_r & \psi^T & \psi? & \psi\bot \\
\phi^T & & T & \\
\phi? & & ? & \\
\phi\bot & & ? & \\
\end{array}
\]

perotti@di.unito.it
Neural-symbolic workshop 2013
August 5, 2013 14 / 26
Rule-based monitoring example: trace and formula

property: \( a \lor \lozenge b \)
Preprocessing: parsing, encoding

V

a

b

rules for V

rules for a

rules for b

rules for

perotti@di.unito.it

Neural-symbolic workshop 2013
August 5, 2013 16 / 26
Running: cell monitoring

```
trace
Z y b x,y x,z y
```

```
V
a F b F
?
V
a F b F
?
V
a F b F
?
V
a F b F
? 
```
Rules as a network

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

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

1) $[\Diamond a]? \rightarrow R[a], R[\Diamond a]$
Example: creating the network

\[ a \lor \diamond b \]

rules for \( a \)
rules for \( b \)
rules for \( \diamond \)
rules for \( V \)
Example: firing the network

\[ a \lor \Diamond b \]
Example: convergence
Example: convergence

trace

z y b x,y x,z y

perotti@di.unito.it
Neural-symbolic workshop 2013
August 5, 2013 22 / 26
The (rule) language of RuleRunner allows for implicit representation of time and involves no 'alternatives'.

Using $\text{CIL}^2 P$, 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