Business Process Verification with Temporal Answer Set Programming

Laura Giordano, Alberto Martelli, Matteo Spiotta, Daniele Theseider Dupré

Università del Piemonte Orientale
Università di Torino
Italy
Cross-fertilize Business Process Modeling with contributions from **Reasoning about actions and change in AI** and **Answer Set Programming**

- Declarative or procedural process model
- Modeling direct effects of activities as well as side effects with **causal rules** (background knowledge)
- Flexible modeling of obligations
- Modeling data
- Verification of compliance with norms and business rules
Our contribution

• Declarative / Procedural description of Processes
• Action effects
• Norms

expressed in a **Temporal** extension of **Answer Set Programming** (a nonmonotonic knowledge representation & reasoning framework)

Verification of temporal logic properties performed as **Bounded Model Checking** in Temporal ASP
The ICT4LAW project


2009-2013, funded by Regione Piemonte

Several partners, including, for procedure compliance with norms:

• ITTIG and LOA (CNR)
• Augeos, SSB Progetti (private companies)
• Our universities
Action theories

**Fluents**: their truth value describes the state of the world, e.g. informed (C): customer C is informed on the firm policies

**Action/event**: performed by an agent or “internal” in the system; has direct and possibly indirect effects on fluents, causing some state change (unless effect already true)

**Action laws**: direct effects of actions

**Causal laws**: fluent dependencies, and then indirect effects of actions

**Precondition laws**: action can happen only if preconditions hold
Temporal modalities in DLTL

A **program** is built from actions using “+” (or), “;” (sequence) and “*” (iteration):

\[
\begin{align*}
    a & | \pi_1 + \pi_2 | \pi_1 ; \pi_2 | \pi^*
\end{align*}
\]

Temporal formulae include:

- \(\langle \pi \rangle \alpha\): there is an execution of \(\pi\) after which \(\alpha\) holds
- \([\pi] \alpha\): \(\alpha\) holds after all possible executions of \(\pi\)
- \([a] \alpha\): \(\alpha\) holds after \(a\)

and the usual temporal logic modalities:

- \(\Diamond \alpha\) (eventually \(\alpha\)), \(\Box \alpha\) (always \(\alpha\)), \(\bigcirc \alpha\) (next \(\alpha\))

(Their semantics is defined from the one of \(\alpha \mathcal{U}^\pi \beta\)
which means: there is an execution of \(\pi\) after which \(\beta\) holds, and \(\alpha\) holds in all previous states)
Action laws

General form ($l_0$ fluent literal - a fluent or its negation, $l_1$, $\ldots$, $l_n$ fluent literals or temporal literals $[a] l$):

$$\Box ([a] l_0 \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n)$$

means that:
always, if $a$ is executed in a state where $l_1$, $\ldots$, $l_m$ hold, and there is no reason for $l_{m+1}$, $\ldots$, $l_n$ to hold, then $l_0$ holds in the resulting state

Example:

$$\Box (\text{[inform]informed})$$

Persistency laws:

$$\Box ([a] l \leftarrow l, \text{not } [a] \neg l)$$
Action laws

Nondeterministic action laws:

\[ \Box ([a] (l_0 \lor ... \lor l_k) \leftarrow l_{k+1}, ..., l_m, \text{not } l_{m+1}, ..., \text{not } l_n) \]

i.e., one of \( l_0, ..., l_k \) becomes true.

They are not primitive, can be mapped to a set of action laws using default negation.
Causal laws

General form (for static causal laws):

□ (l ← l₁, ..., lₘ, not lₘ₊₁, ..., not lₙ)

model dependencies and indirect effects: if l₁, ..., lₘ already hold or are caused to hold, and lₘ₊₁, ..., lₙ do not hold or are caused not to hold, then “l” also holds (if its complement holds before, it does not persist)

Example:

□ (¬ confirmed ← deleted)

where “confirmed” means “order confirmed for the seller” – but the customer can withdraw its order, making “deleted” true as a direct effect, and “confirmed” false as a side effect
**Dynamic causal laws:**

\[ \square \left( O l \leftarrow t_1, \ldots, t_m, \neg t_{m+1}, \ldots, \neg t_n \right) \]

The \( t_i \)'s can be of the forms \( l_i \) or \( O l_i \)

Then we can represent side effects of changes of fluents, e.g.:

\[ \neg f, O f \]

i.e. \( f \) becomes true
Ramifications & BPs


The intended states after an action are those:

- where direct effects hold
- where the background axioms are satisfied
- that differ minimally from the state before the action

But one of their examples is:

insurance claim accepted when accepted by reviewer A and by reviewer B.
Ramifications & BPs

If this is modeled as the material implication:

\[ \text{claimAccRevA} \land \text{claimAccRevB} \supset \text{claimAccepted} \]

and the PMA is used, if A already accepted and B accepts, this either makes \( \text{claimAccepted} \) true ... or \( \text{claimAccRevA} \) false !!!

The static causal rule

\[ \text{claimAccepted} \leftarrow \text{claimAccRevA} , \text{claimAccRevB} \]

can be used to have only \( \text{claimAccepted} \) change as a side effect, while still intending that the implication holds
Ramifications & BPs

The implication may be false if e.g. we allow the acceptance to be overridden later by a supervisor.

In this case dynamic laws are appropriate:

\[ \varnothing \text{claimAccepted} \leftarrow \varnothing \text{claimAccRevA}, \neg \text{claimAccRevB}, \varnothing \text{claimAccRevB} \]

i.e., if the conjunction of acceptances becomes true, we have the side effect, which:

- remains true by default persistence
- may be made false while its original cause remains true
Modeling Business Processes

A domain description in [Giordano et al TPLP 2012] is a pair (Action and causal laws, DLTL constraints)

The control flow of a business process can be modeled in several ways (not mutually exclusive)

**Option 1**: a program (regular expression) in a DLTL constraint: $\langle \pi \rangle T$
(only structured, sequential programs)

**Option 2**: use general DLTL constraints, e.g.

- $\square [a] \langle b \rangle T$
- $\square [a] \diamond \langle b \rangle T$

after a is executed, immediately/eventually b is executed
Modeling Business Processes

**Option 3**: use ConDec constraints (given their LTL correspondent)

**Option 4**: use effects of some actions as preconditions of other actions

**Option 5**: use a «classical» graphical workflow notation (BPMN, YAWL) and define a translation to temporal ASP

Actually, given that we translate temporal ASP to plain ASP, we defined a direct translation from a YAWL subset to ASP
Data

Actions that acquire a value for a variable in a finite domain: nondeterministic action laws

[verify\_status\] status(gold) ∨ status(silver) ∨ status(normal)

For numerical data, we can use the abstraction technique in [Knuplesch et al 2010] which uses thresholds in the model (XOR-splits) and in the formula to be verified to reduce the domain to a small set of abstract values.
We use the notation (from some ASP solvers)

\[ 1\{ [a]R(X) \mid P(X) \} \]

after \( a \), \( R \) is true for exactly one \( X \) such that \( P(X) \), short for:

\[
[a] R(X) \leftarrow \text{not } [a] \rightarrow R(X), P(X) \\
[a] \rightarrow R(X) \leftarrow [a] R(Y), P(X), P(Y), X \neq Y
\]

Then «select a shipper \( S \), among the available ones, that is compatible with the product \( P \)» is:

\[ 1\{ \text{select\_shipper}(P) \} \text{shipper}(S) \mid \text{available\_shipper}(S) \} \]

\[ \bot \leftarrow [\text{select\_shipper}(P)] \text{shipper}(S), \text{not\_compatible}(P, S) \]
Compliance

Several norms and business rules have the form «if A happens/is true, than B shall happen/be true»

This may mean verifying the formula:

\[ \Box (A \rightarrow \Diamond B) \]

but this does not allow the «obligation» to B to be cancelled later: e.g. (obligation to send goods cancelled if order cancelled)

We then have an explicit notion of commitment \( C(\alpha) \):

\[ \Box ([\text{order}]C(\text{goods\_sent})) \]
Compliance

The commitment may be canceled:

□ ([cancel_order] → C(goods_sent))

and it is discharged when fulfilled:

□ (∅ → C(α) ← C(α), ∅ α)

(dynamic causal rule)

Now, the formula to be verified is, for all C(α):

□ (C(α) → ◊ ¬ C(α))
Compliance

Other business rules (not related to commitments) can be verified, e.g.:

"Premium customer status shall only be offered after a prior solvency check"

□(solvency check done ∨ ¬ ⟨offer premium status⟩ ⊤)
Compliance

Verification is performed with Bounded Model Checking in Answer Set Programming [Giordano et al, TPLP 2012, KR 2012]

The Temporal ASP representation is translated to standard ASP, and BMC is represented in ASP (following [Heljanko & Niemelä 2003])

In particular, we used the ASP systems DLV and Clingo
Conclusions

We showed some contributions from Reasoning about actions and change in AI and Answer Set Programming to Business Process modeling and verification:

- Modeling direct effects of activities as well as side effects with causal rules
- Flexible modeling of obligations
- Modeling data
- Declarative or procedural process model
- Verification of compliance with norms and business rules in ASP
Thanks!