Compliance Verification of Business Processes with (Constraint) Temporal Answer Set Programming

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

Università del Piemonte Orientale
Università di Torino
Italy
based on


both build on:

Goal

Cross-fertilize Business Process modeling with Reasoning about actions and change in AI and (Constraint) Answer Set Programming

(Constraint) Temporal Answer Set Programming combines (Constraint) ASP with Temporal Logic (DLTL) useful for:

• Declarative or procedural process model
• Modeling background knowledge: direct effects of activities and side effects
• Constraint solving on numeric process data
• Compliance verification via Bounded Model Checking based on [Giordano, Martelli & T.D. TPLP 13]
Norm 1: “The firm shall provide the investor adequate information on its policies before any contract is signed”

Norm 2: “If an investor signs a contract, the firm shall provide him a copy of the contract”
Order-delivery process adapted from [Knuplesch 2010], branching depends on variables, esp. piece number
• After confirming an order, goods have to be shipped eventually
• An order shall either be confirmed or declined
• Orders with $pn>50000$ shall be approved before they are confirmed
• For orders of a non-premium customer with $pn>80000$ a solvency check is necessary before assessing the order
For orders of a non-premium customer with \( pn > 80000 \) a solvency check is necessary before assessing the order:

\[
\Box (pn > 80000 \land c \neq \text{premium} \land \langle \text{Assess\_Order} \rangle \; \top \rightarrow \; \text{solvency\_check\_done})
\]
Representation languages

Then, we use:

• An **action language**, used to describe a domain, where effects of atomic actions and their executability conditions may involve constraints

• A **temporal logic (with constraints)**, used to express (at least) formulae to be verified

Constraints (e.g. $pn>80000$, or $x+y>k$) will be treated as atoms at the temporal logic level and the answer set level (as in [Gebser et al 09])
DLTL (with constraints)

DLTL [Henriksen & Thiagarajan 99] extends LTL: temporal operators can be indexed with regular expressions (programs) \( \pi \)

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 \( \pi \)
- \( [\alpha] \alpha \) \( \alpha \) holds after \( \alpha \)

and the usual temporal logic modalities:

- \( \Diamond \alpha \) (eventually \( \alpha \)), \( \Box \alpha \) (always \( \alpha \)), \( \lozenge \alpha \) (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
DLTL (with constraints)

DLTL formulae with constraints are then:

\[
\bot \mid T \mid p \mid g \mid \neg \alpha \mid \alpha \lor \beta \mid \alpha \mathcal{U}^{\pi} \beta
\]

where the $p$s are atomic propositions and the $g$s are constraints in a constraint language (which we assume to have a finite domain).

As in [Gebser et al 09], a function $\gamma$ maps constraint atoms (syntax) to constraints (relations on variables), then providing an interpretation for constraints (e.g. the usual interpretation for arithmetic ops and rels), $A \models_{\gamma} g$ means: $\gamma(g)$ is true for the assignment $A$ of values to variables, e.g. $A \models_{\gamma} x+y>20$ if $A(x)=15$ and $A(y)=10$
DLTL with constraints

A model is $M = (\sigma, V, v)$ where $\sigma$ is an infinite sequence of actions, $V$ and $v$ provide, for each prefix $\tau$ of $\sigma$ (the state reached after $\tau$) an interpretation of atomic propositions, and an assignment for constraint variables. Then:

$M, \tau \models p$ iff $p \in V(\tau)$
$M, \tau \models g$ iff $v(\tau) \not\models g$
$M, \tau \models \alpha U^{\pi} \beta$ iff in $\sigma$, after $\tau$, there is an execution $\tau'$ of $\pi$ such that $M, \tau \tau' \models \beta$ and for all intermediate states $\tau \tau''$, $M, \tau \tau'' \models \alpha$
Temporal action language

\[ \square ( l_0 \leftarrow l_1, \ldots, l_m, \neg l_{m+1}, \ldots, \neg l_n ) \]

\( l_0 \) is a fluent literal or temporal fluent literal ( [a]l or ◯l)

The \( l_i \) can be:

- fluent literals,
- constraint literals,
- temporal (constraint or fluent) literals,
- dynamic constraint literals, i.e. constraint literals also involving variables \( x^\circ \), i.e. ‘\( x \) in the next state’

with some restriction ensuring that successor states only depend on current state

Action laws, causal laws, persistence can be expressed
Action laws, causal laws

Example of action law:

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

Persistence:

- $([a] \perp \perp l, \text{not } [a] \rightarrow l)$

**Static** causal laws model dependencies within the same state and then also side effects, e.g.:

- $(\neg \text{confirmed} \leftarrow \text{deleted})$

where “deleted” = “order deleted by customer” and “confirmed” = “order confirmed for the seller”
Causal laws

**Dynamic** causal laws:

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

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
[Weber et al. 2010] use clauses (in classical logic) to model dependencies, and the Possible Models Approach [Winslett 1988] to deal with ramifications. 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:

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

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:

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

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
Constraint Temporal Answer Sets

Given a set $P$ of rules, we define a **Constraint Temporal Answer Set** combining Temporal AS in [Giordano, Martelli & T.D.13] and Constraint AS in [Gebser et al 09]

It is a partial temporal interpretation $(\sigma, S)$ where $S$ is a set of temporal literals of the form $[a_1;...;a_k]$ where $a_1;...;a_k$ is a prefix of $\sigma$

It is defined relative to an assignment $\nu$ to constraint variables at each prefix of $\sigma$

Then, we define for the various types of literals their being satisfied by $(\sigma, S)$ at the prefix $a_1;...;a_k$ given $\nu$
Constraint Temporal Answer Sets

Given an interpretation \((\sigma,S)\), for each prefix \(a_1;...;a_k\) we compute a different constraint reduct, a set of rules

\[
[a_1;...;a_k] \ H \leftarrow \ \text{Body}
\]

obtained from rules in \(P\):

• eliminating constraint literals true at \(a_1;...;a_k\) given \(v\) (if all true)
• and extended literals \(\text{not } l\) true at \(a_1;...;a_k\) (if all true)

\(\text{reduct} = \text{union of reducts for all prefixes}\)

\((\sigma,S)\) is a constraint temporal answer set wrt \(v\) if \(S\) is minimal among the \(R\) such that \((\sigma,R)\) satisfies the rules in the reduct
Extensions

Given a domain description \((P,Q)\) where \(P\) is a set of rules, and \(Q\) is a set of (constraint) DLTL formulas, its extensions (i.e. models) are constraint temporal answer sets of \(P\) whose corresponding temporal model satisfies formulae in \(Q\).

Validity of a formula \(\alpha\) for a d.d. \((P,Q)\) corresponds to verifying that there is no extension of \((P, \{Q \land \neg \alpha\})\).
Modeling Business Processes

The control flow of a business process can be modeled in several ways

- a **program** (regular expression) in a DLTL constraint: \( \langle \pi \rangle T \) (only structured, sequential programs) [Giordano et al. CLIMA 10]
- **declarative** temporal constraints (e.g. ConDec/Declare from van der Aalst et al)
- «classical» graphical **workflow notation** (BPMN, YAWL)
Modeling Business Processes

We used a **translation from basic workflow constructs** of YAWL to the temporal action language, based on the *enabling* of actions and arcs.

An action precondition is its being enabled.

Causal laws define enabling of action based on enabling of incoming arcs (one/all for XOR/AND)

Actions enable and disable arcs

\[
[a] \text{en}_{\text{arc}}_{a\_b} \leftarrow \text{pn} > 50
\]

\[
[a] \text{en}_{\text{arc}}_{a\_c} \leftarrow \text{not pn} > 50
\]
Modeling Business Processes

The model provides information on which actions have a variable as output:

\[ [a] \ x \in [0..1000000] \]

Across other actions, the value of \( x \) persists, we model this via a fluent \( \text{change}_x \) which is non persistent and false by default:

\[
\begin{align*}
\text{x}^\circ &= \text{x} \leftarrow \circ \neg \text{change}_x \\
[a] \ \text{change}_x \\
\neg \text{change}_x &\leftarrow \text{not change}_x
\end{align*}
\]

Other fluents persist:

\[ [a] \ f \leftarrow f, \text{not} [a] \neg f \]
Verification

In [Giordano, Martelli & T.D. TPLP13] we defined a translation of domain descriptions to ASP and an encoding in ASP of Bounded Model Checking (following [Heljanko & Niemelä 03])

- BMC, given a system and a formula, searches for a model
- Infinite paths are represented as finite paths of length k with a loop back from state k to a previous state
- The search proceeds iteratively, increasing k until a model is found (if one exists)
Translation

Our translation is defined so that extensions of domain descriptions correspond to (constraint) answer sets of the translation

- `occurs(Action,State)` (State is a number)
- `holds(Literal,State)`

E.g. for `[a]f1 ← f2`:

```
holds(f1,S') ← state(S), next(S,S'), occurs(a,S), holds(f2,S)
```

- `sat(Formula,State)`

Defined inductively on the structure of the DLTL Formula
**Translation**

**Constraint literals** are represented using CSP variables `value(x,s)` for the value of process variable `x` at state `s`.

\[ a \] \(\text{en}_\text{arc}_a\_b \leftarrow \text{pn} \succ 50\)

becomes

\[
\text{holds(en}_\text{arc}_a\_b,S') \leftarrow \\
\text{state}(S),\text{next}(S,S'),\text{occurs}(a,S),\text{value}(pn,S) \succ 50
\]

\[
x \circ = x \leftarrow \circ \neg \text{change}_x
\]

becomes

\[
\text{value}(x,S') \preceq \text{value}(x,S) \leftarrow \\
\text{state}(S),\text{next}(S,S'),\neg \text{holds(change}_x,S')
\]
The approach in [Giordano, Martelli & T.D. TPLP 13] is suitable for verifying systems with infinite computations (and finite state space)

In BPs only executions that reach the end are considered sound.

Finite executions can be represented as infinite ones with a final dummy action.

In practice, we restrict to finite traces.
Completeness of BMC

BMC is in general a partial decision procedure.

Completeness can be obtained for special classes of formulae, or for general formulae, computing a **completeness threshold** $t$ (using bounds up to $t$ is enough to find a model if one exists) [Biere et al. 03,06, Clarke et al. 04, Giordano et al. KR12]

But computing the threshold may be unfeasible.

For loop-free workflows the length of the longest run can be used as threshold.
Orders with $pn>50000$ shall be approved before they are confirmed.

For orders of a non-premium customer with $pn>80000$ a solvency check is necessary before assessing the order.
BP Verification

Running the translation in **clingcon** we get (in ≈ 0.1s)

- \( \Box (pn>50000 \land \langle Confirm\_Order\rangle \top \rightarrow a=true) \) **valid**
- \( \Box (pn>80000 \land c\neq premium \land \langle Assess\_Order\rangle \top \rightarrow solvency\_check\_done) \) **non valid**
if clingcon is asked to provide weak answer sets, for the \( s \) after \( pn \) is assigned, value(\( pn, s \)) is given the domain \([80001..100000]\)
Scalability

c_i \implies c_{i+1} \ (k_i > k_{i+1}), \text{ then } \square (a_1 \rightarrow \Diamond b_r) \quad \text{valid}

while

\square (d_1 \rightarrow \Diamond b_r) \quad \text{non valid}
Variants with (pure) ASP conditions (run in clingo)

Scalability
but $c_{4r}$ is $\land (v_i > k_i'/2)$, with $k_i' < k_i$, then, $O(12^r)$ runs and

$\Box(\land a_i \rightarrow \Diamond b_r)$ \hspace{1cm} valid

while

$\Box(\land d_i \rightarrow \Diamond b_r)$ \hspace{1cm} non valid
Scalability

Variant with (pure) ASP conditions (run in clingo)
Scalability

c1i is v>k

r=5

10^{36} ÷ 10^{40} different runs, 100 ÷ 10000 s verif. time

c4i is v>k/2

r=4

10^{25} ÷ 10^{28} different runs, 10 ÷ 100 s verification time
Conclusions

Constraint Temporal Answer Set Programming combines temporal logic with:

- Nonmonotonic knowledge representation of actions and change [Giordano, Martelli & T.D. TPLP 13], also suitable for flexible modeling of obligations [Giordano et al ICAIL 13]
- Constraint reasoning

We have shown that current (C)ASP technology already makes the framework useful for verifying compliance of business processes (also) involving conditions on numerical data