Business Process Verification with Constraint Temporal Answer Set Programming

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

Università del Piemonte Orientale
Università di Torino
Italy
Goal

Exploiting Computational Logic for modeling Business Processes and verifying their compliance with business rules and norms

**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 (*data-centric* process models)
- Compliance verification via Bounded Model Checking [Giordano et al. TPLP 13]
Example

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:

$$\square(pn>80000 \land c\neq \text{premium} \land \langle \text{Assess\_Order} \rangle \top \rightarrow \text{solvency\_check\_done})$$
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:

\[
\begin{align*}
\langle \pi \rangle \alpha & \quad \text{there is an execution of } \pi \text{ after which } \alpha \text{ holds} \\
[\pi] \alpha & \quad \alpha \text{ holds after all possible executions of } \pi \\
[a] \alpha & \quad \alpha \text{ holds after } a
\end{align*}
\]

and the usual temporal logic modalities:

\( \Diamond \alpha \) (eventually \( \alpha \)), \( \square \alpha \) (always \( \alpha \)), \( \Diamond \alpha \) (next \( \alpha \))

Their semantics is defined from the one of \( \alpha U^\pi \beta \) 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:

$$\bot \mid T \mid p \mid g \mid \neg \alpha \mid \alpha \lor \beta \mid \alpha U^\pi \beta$$

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 to constraints (relations on variables), then providing an interpretation for constraints: for an assignment $A$ of values to variables, we know whether $\gamma(g)$ is true, which we write:

$$A \models_\gamma g$$
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:

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

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

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

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
Constraint Temporal Answer Sets

Given a set $P$ of rules, we define a **Constraint Temporal Answer Set** combining Temporal AS in [Giordano et al 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 type 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 not true at \(a_1;...;a_k\) (if all true)

reduct = 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.

\[
\begin{align*}
[a] \text{en}_{\text{arc}_{a\_b}} & \leftarrow \text{pn} > 50 \\
[a] \text{en}_{\text{arc}_{a\_c}} & \leftarrow \text{not pn} > 50
\end{align*}
\]
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:

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

Other fluents persist:

\[ [a] \ f \leftarrow f, \text{ not } [a] \neg f \]
In [Giordano et al. 13 TPLP] 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 \([a]f1 \leftarrow f2\):

\[\text{holds}(f1, S') \leftarrow \text{state}(S), \text{next}(S, S'), \text{occurs}(a, S), \text{holds}(f2, S)\]

- \(\text{sat}(\text{Formula}, \text{State})\)

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`

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

becomes

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

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

becomes

\[ \text{value}(x,S') \Leftarrow \text{value}(x,S) \leftarrow \text{state}(S), \text{next}(S,S'), \neg \text{holds}(\text{change}_x,S') \]
The approach in [Giordano et al. 13 TPLP] 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 it exists) [Biere et al. 03,06, Clarke et al. 04, Giordano et al. 12]

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.
Running the translation in clingcon we get (in $\approx 0.1$s)

- $\square (pn>50000 \land \langle Confirm\_Order\rangle T \rightarrow a=true)$  \textcolor{red}{\textbf{valid}}
- $\square (pn>80000 \land c\neq\text{premium} \land \langle Assess\_Order\rangle T \rightarrow solvency\_check\_done)$  \textcolor{red}{\textbf{non valid}}
80000 \leq pn \leq 100000

if \text{clingcon} is asked to provide weak answer sets, for the \text{s} after \text{pn} is assigned, value(\text{pn}, \text{s}) is given the domain \([80001..100000]\)
Scalarility

c_1_i \text{ implies } c_4_i \text{ and } c_{i+1} (k_i > k_{i+1}), \text{ then}

□(a_1 \rightarrow ◊ b_r) \text{ valid}

while

□(d_1 \rightarrow ◊ b_r) \text{ non valid}
Variant with (pure) ASP conditions (run in clingo)
but \(c_4 \land (v_i > k_i'/2)\), with \(k_i' < k_i\), then, \(O(12^r)\) runs and

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

while

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

Variant with (pure) ASP conditions (run in clingo)
r=4 \quad 10^{25} \div 10^{28} \text{ different runs, } 10 \div 100 \text{ s verification time}

r=5 \quad 10^{36} \div 10^{40} \text{ different runs, } 100 \div 10000 \text{ s verif. time}

c_{1i} \text{ is } v>k_i \ (v_i>k_i)

c_{4i} \text{ is } v>k_i/2 \ (v_i>k_i/2)

□ \land (a_i \rightarrow ◊ b_j) \quad \text{valid}

□ \land (d_i \rightarrow ◊ b_j) \quad \text{non valid}
Scalability

- Verification time does not suffer significantly from the integration with constraint solving as long as constraint atoms do not involve several variables.
- The completeness threshold is in general a problem.
Using a technique suitable for $\Box p$ formulae (length of longest loop-free path in the transition system), the thresholds for the example formulae can be found in 20-100 s, and verification is done in < 1 s.
Conclusions

Constraint Temporal Answer Set Programming combines temporal logic with:

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

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