Automata for dynamic answer set solving: Preliminary report

Pedro Cabalar[1], Martín Diéguez[2], Susana Hahn[3] and Torsten Schaub[3]

[1]University of Corunna, Spain
[2]University of Angers, France
[3]University of Potsdam, Germany

Abstract

We explore different ways of implementing temporal constraints expressed in an extension of Answer Set Programming (ASP) with language constructs from dynamic logic. Foremost, we investigate how automata can be used for enforcing such constraints. The idea is to transform a dynamic constraint into an automaton expressed in terms of a logic program that enforces the satisfaction of the original constraint. What makes this approach attractive is its independence of time stamps and the potential to detect unsatisfiability. On the one hand, we elaborate upon a transformation of dynamic formulas into alternating automata that relies on meta-programming in ASP. This is the first application of reification applied to theory expressions in *gringo*. On the other hand, we propose two transformations of dynamic formulas into monadic second-order formulas. These can then be used by off-the-shelf tools to construct the corresponding automata. We contrast both approaches empirically with the one of the temporal ASP solver *telingo* that directly maps dynamic constraints to logic programs. Since this preliminary study is restricted to dynamic formulas in integrity constraints, its implementations and (empirical) results readily apply to conventional linear dynamic logic, too.

Keywords

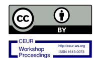
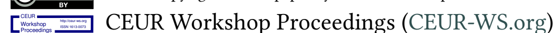
Dynamic Answer Set Programming, Linear Dynamic Logic, Alternating Finite Automaton on Words

CEUR-WS.org/Vol-2970/aspocpinvited1.pdf

1. Introduction

Answer Set Programming (ASP [1]) has become a popular approach to solving knowledge-intense combinatorial search problems due to its performant solving engines and expressive modeling language. However, both are mainly geared towards static domains and lack native support for handling dynamic applications. Rather *change* is accommodated by producing copies of variables, one for each state. This does not only produce redundancy but also leaves the ASP machinery largely uninformed about the temporal structure of the problem.

This preliminary work explores alternative ways of implementing temporal (integrity) constraints in (linear) *Dynamic Equilibrium Logic* (DEL; [2, 3]) by using automata [4]. On the one hand, DEL is expressive enough to subsume more basic systems, like (linear) Temporal Equilibrium Logic [5, 6] or even its metric variant [7]. On the other hand, our restriction to integrity constraints allows us to draw on work in conventional linear dynamic and temporal

14th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'21)
cabalar@udc.es (P. Cabalar); martin.dieguezlodeiro@univ-angers.fr (M. Diéguez); hahnmartinlu@uni-potsdam.de (S. Hahn); torsten@uni-potsdam.de (T. Schaub)

© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

logic (cf. Proposition 1). Although this amounts to using dynamic formulas to filter "stable temporal models" rather than to let them take part in the formation of such models, it allows us to investigate a larger spectrum of alternatives in a simpler setting. Once fully elaborated, we plan to generalize our approach to the full setting. Moreover, we are interested in implementing our approach by means of existing ASP systems, which motivates our restriction to the finite trace variant of DEL, called DEL_f.

In more detail, Section 2 to 4 lay the basic foundations of our approach by introducing DEL, some automata theory, and a translation from dynamic formula into alternating automata. We then develop and empirically evaluate three different approaches. First, the one based on alternating automata from Section 4. This approach is implemented entirely in ASP and relies on meta-programming. As such it is the first application of *gringo*'s reification machinery to user defined language constructs (defined by a theory grammar; cf. [8]). For the second approach we employ our two alternative transformations of dynamic formulas into monadic second order (MSO [9]) formulas proposed in [10], namely *Monadic Second Order Encoding* and *Standard Translation*. These formulas can then be passed to the off-the-shelf automata construction tool MONA [11] that turns them into deterministic automata. And finally, the approach of *telingo* [12, 13], transforming each dynamic constraint directly into a logic program. All three approaches result in a program that allows us to sift out "stable temporal models" satisfying the original dynamic constraints. Usually, these models are generated by another logic program, like a planning encoding and instance.

2. Linear Dynamic Equilibrium Logic

Given a set $\mathcal{P}$ of propositional variables (called *alphabet*), *dynamic formulas* φ and *path expressions* ρ are mutually defined by the pair of grammar rules:

$$\varphi ::= a \mid \bot \mid \top \mid [\rho]\,\varphi \mid \langle\rho\rangle\,\varphi \qquad\qquad \rho ::= \tau \mid \varphi? \mid \rho + \rho \mid \rho\,;\,\rho \mid \rho^*.$$

This syntax is similar to the one of Dynamic Logic (DL; [14]) but differs in the construction of atomic path expressions: While DL uses a separate alphabet for *atomic actions*, LDL has a single alphabet $\mathcal{P}$ and the only atomic path expression is the (transition) constant $\tau \notin \mathcal{P}$ (read as "step"). Thus, each ρ is a regular expression formed with the constant τ plus the test construct $\varphi?$ that may refer to propositional atoms in the (single) alphabet $\mathcal{P}$. As with LDL [15], we sometimes use a propositional formula ϕ as a path expression and let it stand for $(\phi?;\tau)$. This means that the reading of $\top$ as a path expression amounts to $(\top?;\tau)$ which is just equivalent to τ, as we see below. Another abbreviation is the sequence of n repetitions of some expression ρ defined as $\rho^0 \stackrel{def}{=} \top?$ and $\rho^{n+1} \stackrel{def}{=} \rho;\rho^n$.

The above language allows us to capture several derived operators, like the Boolean and

temporal ones [3]:

$$
\begin{array}{rclrclrcl}
\varphi \land \psi & \stackrel{def}{=} & \langle \varphi ? \rangle\, \psi & \varphi \lor \psi & \stackrel{def}{=} & \langle \varphi ? + \psi ? \rangle\, \top & & & \\
\varphi \rightarrow \psi & \stackrel{def}{=} & [\varphi ?]\, \psi & \neg \varphi & \stackrel{def}{=} & \varphi \rightarrow \bot & & & \\
\circ \varphi & \stackrel{def}{=} & \langle \tau \rangle\, \varphi & \widehat{\circ} \varphi & \stackrel{def}{=} & [\tau]\, \varphi & \mathbf{F} & \stackrel{def}{=} & [\tau]\, \bot \\
\Diamond \varphi & \stackrel{def}{=} & \langle \tau^* \rangle\, \varphi & \Box \varphi & \stackrel{def}{=} & [\tau^*]\, \varphi & & & \\
\varphi \,\mathbf{U}\, \psi & \stackrel{def}{=} & \langle (\varphi ?; \tau)^* \rangle\, \psi & \varphi \,\mathbf{R}\, \psi & \stackrel{def}{=} & (\psi \,\mathbf{U}\, (\varphi \land \psi)) \lor \Box \psi & & &
\end{array}
$$

All connectives are defined in terms of the dynamic operators $\langle \cdot \rangle$ and $[\cdot]$. This involves the Booleans $\land$, $\lor$, and $\rightarrow$, among which the definition of $\rightarrow$ is most noteworthy since it hints at the implicative nature of $[\cdot]$. Negation $\neg$ is then expressed via implication, as usual in HT. Then, $\langle \cdot \rangle$ and $[\cdot]$ also allow for defining the future temporal operators $\mathbf{F}$, $\circ$, $\widehat{\circ}$, $\Diamond$, $\Box$, $\mathbf{U}$, $\mathbf{R}$, standing for *final, next, weak next, eventually, always, until,* and *release.* A formula is *propositional*, if all its connectives are Boolean, and *temporal*, if it includes only Boolean and temporal ones. As usual, a *(dynamic) theory* is a set of (dynamic) formulas.

For the semantics, we let $[a..b]$ stand for the set $\{i \in \mathbb{N} \mid a \leq i \leq b\}$ and $[a..b)$ for $\{i \in \mathbb{N} \mid a \leq i < b\}$ for $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\omega\}$. A *trace* of length λ over alphabet $\mathcal{P}$ is then defined as a sequence $(H_i)_{i \in [0..\lambda)}$ of sets $H_i \subseteq \mathcal{P}$. A trace is *infinite* if $\lambda = \omega$ and *finite* otherwise, that is, $\lambda = n$ for some natural number $n \in \mathbb{N}$. Given traces $\mathbf{H} = (H_i)_{i \in [0..\lambda)}$ and $\mathbf{H}' = (H'_i)_{i \in [0..\lambda)}$ both of length λ, we write $\mathbf{H} \leq \mathbf{H}'$ if $H_i \subseteq H'_i$ for each $i \in [0..\lambda)$; accordingly, $\mathbf{H} < \mathbf{H}'$ iff both $\mathbf{H} \leq \mathbf{H}'$ and $\mathbf{H} \neq \mathbf{H}'$.

Although DHT shares the same syntax as LDL, its semantics relies on traces whose states are pairs of sets of atoms. An HT-trace is a sequence of pairs $(\langle H_i, T_i \rangle)_{i \in [0..\lambda)}$ such that $H_i \subseteq T_i \subseteq \mathcal{P}$ for any $i \in [0..\lambda)$. As before, an HT-trace is infinite if $\lambda = \omega$ and finite otherwise. The intuition of using these two sets stems from HT: Atoms in H_i are those that can be proved; atoms not in T_i are those for which there is no proof; and, finally, atoms in $T_i \setminus H_i$ are assumed to hold, but have not been proved. We often represent an HT-trace as a pair of traces $\langle \mathbf{H}, \mathbf{T} \rangle$ of length λ where $\mathbf{H} = (H_i)_{i \in [0..\lambda)}$ and $\mathbf{T} = (T_i)_{i \in [0..\lambda)}$ such that $\mathbf{H} \leq \mathbf{T}$. The particular type of HT-traces that satisfy $\mathbf{H} = \mathbf{T}$ are called *total.*

The overall definition of DHT satisfaction relies on a double induction. Given any HT-trace $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$, we define DHT satisfaction of formulas, namely, $\mathbf{M}, k \models \varphi$, in terms of an accessibility relation for path expressions $\| \rho \|^{\mathbf{M}} \subseteq \mathbb{N}^2$ whose extent depends again on $\models$ by double, structural induction.

Definition 1 (DHT satisfaction; [3])**.** *An* HT-*trace* $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$ *of length* λ *over alphabet* $\mathcal{P}$ satisfies *a dynamic formula* φ *at time point* $k \in [0..\lambda)$*, written* $\mathbf{M}, k \models \varphi$*, if the following conditions hold:*

1. $\mathbf{M}, k \models \top$ *and* $\mathbf{M}, k \not\models \bot$
2. $\mathbf{M}, k \models a$ *if* $a \in H_k$ *for any atom* $a \in \mathcal{P}$
3. $\mathbf{M}, k \models \langle \rho \rangle\, \varphi$ *if* $\mathbf{M}, i \models \varphi$ *for some* i *with* $(k, i) \in \| \rho \|^{\mathbf{M}}$
4. $\mathbf{M}, k \models [\rho]\, \varphi$ *if* $\mathbf{M}', i \models \varphi$ *for all* i *with* $(k, i) \in \| \rho \|^{\mathbf{M}'}$
   for both $\mathbf{M}' = \mathbf{M}$ *and* $\mathbf{M}' = \langle \mathbf{T}, \mathbf{T} \rangle$

where, for any HT*-trace* $\mathbf{M}$*,* $\| \rho \|^{\mathbf{M}} \subseteq \mathbb{N}^2$ *is a relation on pairs of time points inductively defined as follows.*

5. $\| \tau \|^{\mathbf{M}} \stackrel{def}{=} \{(k, k+1) \mid k, k+1 \in [0..\lambda)\}$
6. $\| \varphi? \|^{\mathbf{M}} \stackrel{def}{=} \{(k, k) \mid \mathbf{M}, k \models \varphi\}$
7. $\| \rho_1 + \rho_2 \|^{\mathbf{M}} \stackrel{def}{=} \| \rho_2 \|^{\mathbf{M}} \cup \| \rho_2 \|^{\mathbf{M}}$
8. $\| \rho_1 \,;\, \rho_2 \|^{\mathbf{M}} \stackrel{def}{=} \{(k, i) \mid (k, j) \in \| \rho_1 \|^{\mathbf{M}}$ *and* $(j, i) \in \| \rho_2 \|^{\mathbf{M}}$ *for some* $j\}$
9. $\| \rho^* \|^{\mathbf{M}} \stackrel{def}{=} \bigcup_{n \geq 0} \| \rho^n \|^{\mathbf{M}}$

An HT-trace $\mathbf{M}$ is a *model* of a dynamic theory Γ if $\mathbf{M}, 0 \models \varphi$ for all $\varphi \in \Gamma$. We write $\mathrm{DHT}(\Gamma, \lambda)$ to stand for the set of DHT models of length λ of a theory Γ, and define $\mathrm{DHT}(\Gamma) \stackrel{def}{=} \bigcup_{\lambda=0}^{\omega} \mathrm{DHT}(\Gamma, \lambda)$, that is, the whole set of models of Γ of any length. A formula φ is a *tautology* (or is *valid*), written $\models \varphi$, iff $\mathbf{M}, k \models \varphi$ for any HT-trace $\mathbf{M}$ and any $k \in [0..\lambda)$. The logic induced by the set of all tautologies is called *(Linear) Dynamic logic of Here-and-There* (DHT for short). We distinguish the variants DHT_ω and DHT_f by restricting DHT to infinite or finite traces, respectively.

We refrain from giving the semantics of LDL [15], since it corresponds to DHT on total traces $\langle \mathbf{T}, \mathbf{T} \rangle$ [3]. Letting $\mathbf{T}, k \models \varphi$ denote the satisfaction of φ by a trace $\mathbf{T}$ at point k in LDL, we have $\langle \mathbf{T}, \mathbf{T} \rangle, k \models \varphi$ iff $\mathbf{T}, k \models \varphi$ for $k \in [0..\lambda)$. Accordingly, any total HT-trace $\langle \mathbf{T}, \mathbf{T} \rangle$ can be seen as the LDL-trace $\mathbf{T}$. As above, we denote infinite and finite trace variants as LDL_ω and LDL_f, respectively.

The work presented in the sequel takes advantage of the following result that allows us to treat dynamic formulas in occurring in integrity constraints as in LDL:

Proposition 1. *For any* HT*-trace* $\langle \mathbf{H}, \mathbf{T} \rangle$ *of length* λ *and any dynamic formula* φ*, we have* $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \neg\neg\varphi$ *iff* $\mathbf{T}, k \models \varphi$*, for all* $k \in [0..\lambda)$*.*

We now introduce non-monotonicity by selecting a particular set of traces called *temporal equilibrium models* [3]. First, given an arbitrary set $\mathfrak{S}$ of HT-traces, we define the ones in equilibrium as follows. A total HT-trace $\langle \mathbf{T}, \mathbf{T} \rangle \in \mathfrak{S}$ is an *equilibrium model* of $\mathfrak{S}$ iff there is no other $\langle \mathbf{H}, \mathbf{T} \rangle \in \mathfrak{S}$ such that $\mathbf{H} < \mathbf{T}$. If this is the case, we also say that trace $\mathbf{T}$ is a *stable model* of $\mathfrak{S}$. We further talk about *temporal equilibrium* or *temporal stable models* of a theory Γ when $\mathfrak{S} = \mathrm{DHT}(\Gamma)$. We write $\mathrm{DEL}(\Gamma, \lambda)$ and $\mathrm{DEL}(\Gamma)$ to stand for the temporal equilibrium models of $\mathrm{DHT}(\Gamma, \lambda)$ and $\mathrm{DHT}(\Gamma)$ respectively. Note that stable models in $\mathrm{DEL}(\Gamma)$ are also LDL-models of Γ. Besides, as the ordering relation among traces is only defined for a fixed λ, the set of temporal equilibrium models of Γ can be partitioned by the trace length λ, that is, $\bigcup_{\lambda=0}^{\omega} \mathrm{DEL}(\Gamma, \lambda) = \mathrm{DEL}(\Gamma)$.

(Linear) *Dynamic Equilibrium Logic* (DEL; [2, 3]) is the non-monotonic logic induced by temporal equilibrium models of dynamic theories. We obtain the variants DEL_ω and DEL_f by applying the corresponding restriction to infinite or finite traces, respectively.

As a consequence of Proposition 1, the addition of formula $\neg\neg\varphi$ to a theory Γ enforces that every temporal stable model of Γ satisfies φ. With this, we confine ourselves in Section 4 to LDL_f rather than DEL_f.

In what follows, we consider finite traces only.

3. Automata

A *Nondeterministic Finite Automaton* (NFA; [4]) is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where Σ is a finite nonempty alphabet, Q is a finite nonempty set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function and $F \subseteq Q$ a finite set of final states. A run of an NFA $(\Sigma, Q, Q_0, \delta, F)$ on a word $a_0 \cdots a_{n-1}$ of length n for $a_i \in \Sigma$ is a finite sequence $q_0, \cdots, q_n$ of states such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, a_i)$ for $0 \leq i < n$. A run is accepting if $q_n \in F$. Using the structure of a NFA, we can also represent a *Deterministic Finite Automata* (DFA), where Q_0 contains a single initial state and δ is restricted to return a single successor state. A finite word $w \in \Sigma^*$ is accepted by an NFA, if there is an accepting run on w. The language recognized by a NFA $\mathfrak{A}$ is defined as $\mathcal{L}(\mathfrak{A}) = \{w \in \Sigma^* \mid \mathfrak{A} \text{ accepts } w\}$.

An *Alternating Automaton over Finite Words* (AFW; [16, 15]) is a tuple $(\Sigma, Q, q_0, \delta, F)$, where Σ and Q are as with NFAs, q_0 is the initial state, $\delta : Q \times \Sigma \to B^+(Q)$ is a transition function, where $B^+(Q)$ stands for all propositional formulas built from Q, $\wedge$, $\vee$, $\top$ and $\bot$, and $F \subseteq Q$ is a finite set of final states.

A run of an AFW $(\Sigma, Q, q_0, \delta, F)$ on a word $a_0 \cdots a_{n-1}$ of length n for $a_i \in \Sigma$, is a finite tree T labeled by states in S such that

1. the root of T is labeled by q_0,
2. if node o at level i is labeled by a state $q \in Q$ and $\delta(q, a_i) = \varphi$, then either $\varphi = \top$ or $P \models \varphi$ for some $P \subseteq Q$ and o has a child for each element in P,
3. the run is accepting if all leaves at depth n are labeled by states in F.

A finite word $w \in \Sigma^*$ is accepted by an AFW, if there is an accepting run on w. The language recognized by an AFW $\mathfrak{A}$ is defined as $\mathcal{L}(\mathfrak{A}) = \{w \in \Sigma^* \mid \mathfrak{A} \text{ accepts } w\}$.

AFWs can be seen as an extension of NFAs by universal transitions. That is, when looking at formulas in $B^+(Q)$, disjunctions represent alternative transitions as in NFAs, while conjunctions add universal ones, each of which must be followed. In Section 5.2, we assume formulas in $B^+(Q)$ to be in disjunctive normal form (DNF) and represent them as sets of sets of literals; hence, $\{\emptyset\}$ and $\emptyset$ stand for $\top$ and $\bot$, respectively.

4. LDL_f to AFW

This section describes a translation of dynamic formulas in LDL_f to AFW due to [17]. More precisely, it associates a dynamic formula φ in negation normal form with an AFW $\mathfrak{A}_\varphi$, whose number of states is linear in the size of φ and whose language $\mathcal{L}(\mathfrak{A}_\varphi)$ coincides with the set of all traces satisfying φ. A dynamic formula φ can be put in negation normal form $nnf(\varphi)$ by exploiting equivalences and pushing negation inside, until it is only in front of propositional formulas.

The states of $\mathfrak{A}_\varphi$ correspond to the members of the Fisher-Ladner closure [18] of φ, denoted by $cl(\varphi)$. The alphabet of an AFW $\mathfrak{A}_\varphi$ for a formula φ over $\mathcal{P}$ is $\Sigma = 2^{\mathcal{P} \cup \{last\}}$. It relies on a special proposition $last$ [17], which is only satisfied by the last state of the trace. A finite word over Σ corresponds to a finite trace over $\mathcal{P} \cup \{last\}$.

Definition 2 (LDL$_f$ to AFW[17]). *Given a dynamic formula φ in negation normal form, the corresponding* AFW *is defined as*

$$\mathfrak{A}_\varphi = (2^{\mathcal{P}\cup\{last\}}, \{q_{nnf(\phi)} \mid \phi \in cl(\varphi)\}, q_\varphi, \delta, \emptyset)$$

where transition function δ mapping a state $q_{nnf(\phi)}$ for $\phi \in cl(\varphi)$ and an interpretation $X \subseteq \mathcal{P} \cup \{last\}$ into a positive Boolean formula over the states in $\{q_{nnf(\phi)} \mid \phi \in cl(\varphi)\}$ is defined as follows:

1. $\delta(q_\top, X) \stackrel{def}{=} \top$
2. $\delta(q_\bot, X) \stackrel{def}{=} \bot$
3. $\delta(q_a, X) \stackrel{def}{=} \begin{cases} \top & \text{if } a \in X \\ \bot & \text{if } a \notin X \end{cases}$
4. $\delta(q_{\neg a}, X) \stackrel{def}{=} \begin{cases} \bot & \text{if } a \in X \\ \top & \text{if } a \notin X \end{cases}$
5. $\delta(q_{\langle\tau\rangle\,\varphi}, X) \stackrel{def}{=} \begin{cases} q_\varphi & \text{if } last \notin X \\ \bot & \text{if } last \in X \end{cases}$
6. $\delta(q_{[\tau]\,\varphi}, X) \stackrel{def}{=} \begin{cases} q_\varphi & \text{if } last \notin X \\ \top & \text{if } last \in X \end{cases}$
7. $\delta(q_{\langle\psi?\rangle\,\varphi}, X) \stackrel{def}{=} \delta(q_\psi, X) \wedge \delta(q_\varphi, X)$
8. $\delta(q_{\langle\rho_1+\rho_2\rangle\,\varphi}, X) \stackrel{def}{=} \delta(q_{\langle\rho_1\rangle\,\varphi}, X) \vee \delta(q_{\langle\rho_2\rangle\,\varphi}, X)$
9. $\delta(q_{\langle\rho_1;\rho_2\rangle\,\varphi}, X) \stackrel{def}{=} \delta(q_{\langle\rho_1\rangle\,\langle\rho_2\rangle\,\varphi}, X)$
10. $\delta(q_{\langle\rho^*\rangle\,\varphi}, X) \stackrel{def}{=} \begin{cases} \delta(q_\varphi, X) & \text{if } \rho \text{ is a test} \\ \delta(q_\varphi, X) \vee \delta(q_{\langle\rho\rangle\,\langle\rho^*\rangle\,\varphi}, X) & \text{otherwise} \end{cases}$
11. $\delta(q_{\langle(\psi?)^*\rangle\,\varphi}, X) \stackrel{def}{=} \delta(q_\varphi, X)$
12. $\delta(q_{[\psi?]\,\varphi}, X) \stackrel{def}{=} \delta(q_{nnf(\neg\psi)}, X) \vee \delta(q_\varphi, X)$
13. $\delta(q_{[\rho_1+\rho_2]\,\varphi}, X) \stackrel{def}{=} \delta(q_{[\rho_1]\,\varphi}, X) \wedge \delta(q_{[\rho_2]\,\varphi}, X)$
14. $\delta(q_{[\rho_1;\rho_2]\,\varphi}, X) \stackrel{def}{=} \delta(q_{[\rho_1]\,[\rho_2]\,\varphi}, X)$
15. $\delta(q_{[\rho^*]\,\varphi}, X) \stackrel{def}{=} \begin{cases} \delta(q_\varphi, X) & \text{if } \rho \text{ is a test} \\ \delta(q_\varphi, X) \wedge \delta(q_{[\rho]\,[\rho^*]\,\varphi}, X) & \text{otherwise} \end{cases}$
16. $\delta(q_{[\rho^*]\,\varphi}, X) \stackrel{def}{=} \delta(q_\varphi, X) \wedge \delta(q_{[\rho]\,[\rho^*]\,\varphi}, X)$
17. $\delta(q_{[(\psi?)^*]\,\varphi}, X) \stackrel{def}{=} \delta(q_\varphi, X)$

Note that the resulting automaton lacks final states. This is compensated by the dedicated proposition *last*. All transitions reaching a state, namely $\delta(q_{[\tau]\,\varphi}, X)$ and $\delta(q_{\langle\tau\rangle\,\varphi}, X)$, are subject to a condition on *last*. So, for the last interpretation $X \cup \{last\}$, all transitions end up in $\top$ or $\bot$. Hence, for acceptance, it is enough to ensure that branches reach $\top$.

As an example, consider the formula, φ,

$$\langle([\tau^*]\,b)?\ ;\ \tau\rangle\,a = \Box b \wedge \circ a, \tag{1}$$

stating that b always holds and a is true at the next step. The AFW for φ is $\mathfrak{A}_\varphi = (2^{\{a,b,last\}}, Q^+ \cup Q^-, \delta, \emptyset)$, where $Q^+ = \{q_{\langle([\tau^*]\,b)?\,;\,\tau\rangle\,a}, q_{\langle([\tau^*]\,b)?\,\rangle\,\langle\tau\rangle\,a}, q_{[\tau^*]\,b}, q_{[\tau]\,[\tau^*]\,b}, q_\top, q_b, q_{\langle\tau\rangle\,a}, q_a\}$ and Q^- contains all states stemming from negated formulas in Q^+; all these are unreachable in our case. The alternating automaton can be found in in Figure 1.

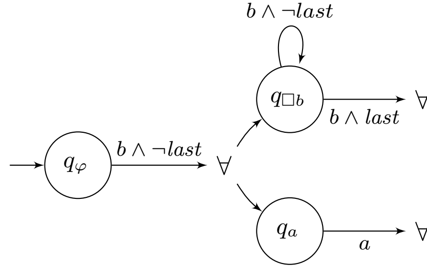


Figure 1: $\mathfrak{A}_\varphi$ showing only the reachable states. The special node type, labeled as $\forall$, represents universal transitions, when the $\forall$-node has no outgoing edges it represents the empty universal constraint $\top$.

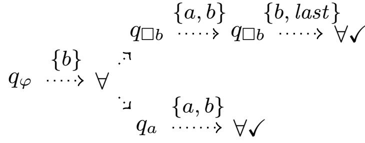


Figure 2: Accepted run for $\{b\} \cdot \{a, b\} \cdot \{b, last\}$.

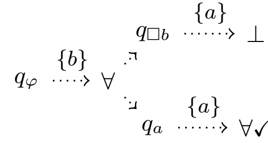


Figure 3: Rejected run for $\{b\} \cdot \{a\} \cdot \{b, last\}$.

5. Using automata for implementing dynamic constraints

Our goal is to investigate alternative ways of implementing constraints imposed by dynamic formulas. To this end, we pursue three principled approaches:

($\mathfrak{T}$) Tseitin-style translation into regular logic programs,
($\mathfrak{A}$) ASP-based translation into alternating automata,
($\mathfrak{M}$) MONA-based translation into deterministic automata, using $\mathfrak{M}_m$ and $\mathfrak{M}_s$ for the Monadic Second Order Encoding and the Standard Translation, respectively.

These alternatives are presented in our systems' workflow [1] from Figure 4. The common idea is to compute all fixed-length traces, or plans, of a dynamic problem expressed in plain ASP (in files `<ins>.lp` and `<enc>.lp`) that satisfy the dynamic constraints in `<dyncon>.lp`. All such constraints are of form `:- not` φ`.` which is the logic programming representation of the formula $\neg\neg\varphi$. Note that these constraints may give rise to even more instances after grounding. The choice of using plain ASP rather than temporal logic programs, as used in *telingo* [6, 12], is motivated by simplicity and the possibility of using existing ASP benchmarks.

For expressing dynamic formulas all three approaches rely on *clingo*'s theory reasoning framework that allows for customizing its input language with theory-specific language constructs that are defined by a theory grammar [8]. The part *telingo* uses for dynamic formulas is given in Listing 1.

```
1  #theory del {
2   formula_body {
3      & : 7,unary; ~ : 5,unary;
4      ? : 4,unary; * : 3,unary; + : 2,binary,left; ;; : 1,binary,left;
```

[1]The source code can be found in https://github.com/potassco/atlingo v1.0.

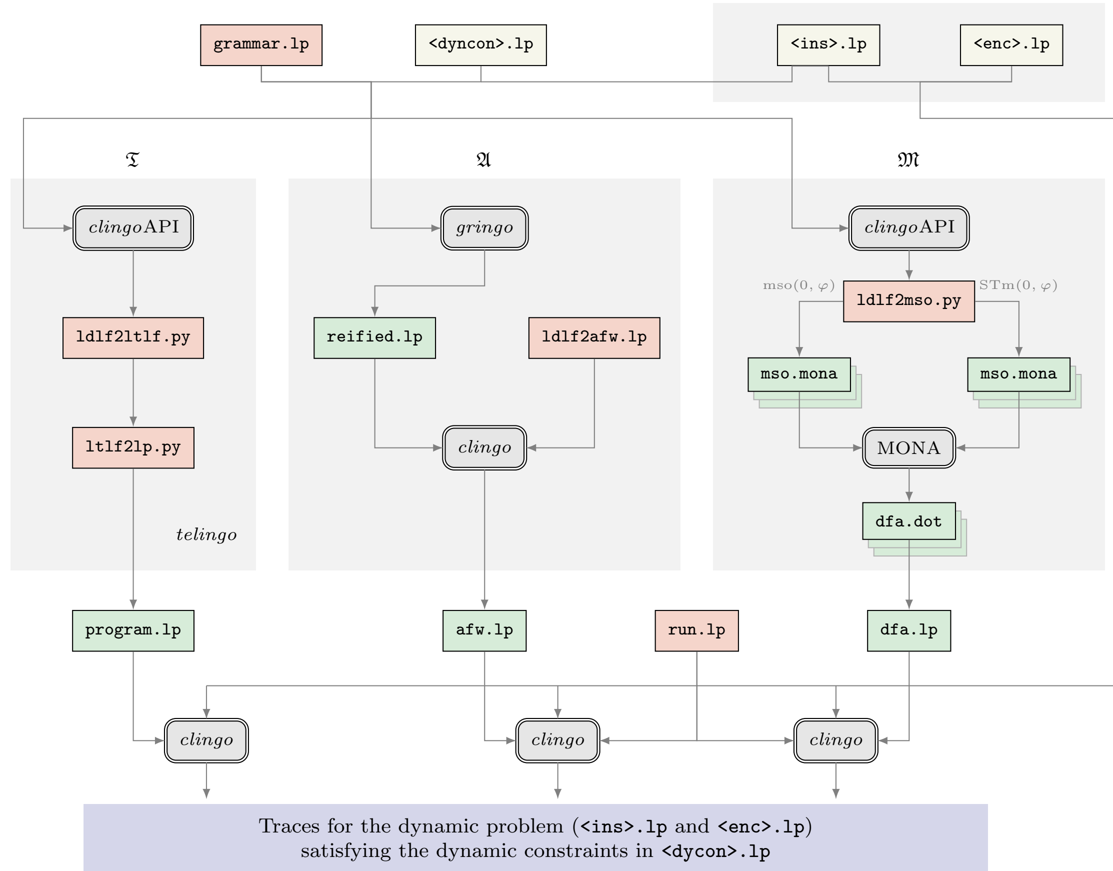


Figure 4: Workflows of our framework. Elements in yellow correspond to user input, green ones are automatically generated, and red ones are provided by the system to solve the problem.

```
5     .>? : 0,binary,right;     .>* : 0,binary,right
6    };
7    &del/0 : formula_body, body
8  }.
```

Listing 1: Theory specification for dynamic formulas (`grammar.lp`)

The grammar contains a single theory term definition for `formula_body`, which consists of terms formed from the theory operators in Line 3 to 5 along with basic *gringo* terms. More specifically, `&` serves as a prefix for logical constants, eg. `&true` and `&t` stand for $\top$ and τ, while `~` stands for negation. The path operators $?, ^*, +, ;$ are represented by `?`, `*`, `+`, `;;`, where `?` and `*` are used as prefixes, and the binary dynamic operators $\langle\cdot\rangle$ and $[\cdot]$ by `.>?` and `.>*`, respectively (extending *telingo*'s syntax `>?` and `>*` for unary temporal operators $\Diamond$ and $\Box$). Such theory terms can be used within the set associated with the (zero-ary) theory predicate `&del/0` defined in Line 7 (cf. [8]). Since we impose our dynamic constraints through integrity constraints, we restrict the occurrence of corresponding atoms to rule bodies, as indicated by the keyword `body`. The representation of our running example $\langle([\tau^*]\, b)?\ ;\ \tau\rangle\, a$ as an integrity constraint is

given in Listing 2.

```
:- not &del{ ? (* &t .>* b) ;; &t .>? a }.
```

Listing 2: Representation of '$\neg\neg\langle([\tau^*]\, b)? \;;\; \tau\rangle\, a$' from (1) (`delex.lp`)

Once such a dynamic formula is parsed by *gringo*, it is processed in a different way in each workflow. At the end, however, each workflow produces a logic program that is combined with the original dynamic problem in `<ins>.lp` and `<enc>.lp` and handed over to *clingo* to compute all traces of length `lambda` satisfying the dynamic formula(s) in `<dyncon>.lp`. We also explored a translation from the alternating automata generated in $\mathfrak{A}$ into an NFA using both ASP and python. This workflow, however, did not show any interesting results, hence, due to space limitations it is omitted.

5.1. Tseitin-style translation into logic programs

The leftmost part of the workflow in Figure 4 relies on *telingo*'s infrastructure [6, 12]: Once grounded, a dynamic formula is first translated into a temporal formula (`ldlf2ltlf.py`), which is then translated into a regular logic program (`ltlf2lp.py`).[2] These translations heavily rely on the introduction of auxiliary variables for subformulas, a technique due to Tseitin [19]. In this way, all integrity constraints in `<dyncon>.lp` get translated into the ground program `program.lp`. In the worst case, this program consists of `lambda` copies of the translated constraint. This approach is detailed in [12, 13].

5.2. ASP-based translation into alternating automata

The approach illustrated in the middle of Figure 4 follows the construction in Section 4. More precisely, it builds the AFW $\mathfrak{A}_\varphi$ for each ground constraint $\neg\neg\varphi$ by taking advantage of Proposition 1. Notably, the approach is fully based on ASP and its meta-programming capabilities: It starts by reifying each $\neg\neg\varphi$ into a set of facts, yielding the single file `reified.lp`. These facts are then turned into one or more AFW $\mathfrak{A}_\varphi$ through logic program `ldlf2afw.lp`. In fact, each $\mathfrak{A}_\varphi$ is once more represented as a set of facts, gathered in file `afw.lp` in Figure 4. Finally, the encoding in `run.lp` makes sure that the trace produced by the encoding of the original dynamic problem is an accepted run of $\mathfrak{A}_\varphi$.

In what follows, we outline these three steps using our running example.

The dynamic constraint in Listing 2 is transformed into a set of facts via *gringo*'s reification option `-output=reify`. The facts provide a serialization of the constraint's abstract syntax tree following the *aspif* format [8]. Among the 42 facts obtained from Listing 2, we give the ones representing subformula $[\tau^*]\, b$, or '`* &t .>* b`', in Listing 3. *Gringo*'s reification format uses integers to identify substructures and to tie them together. For instance, the whole expression '`* &t .>* b`' is identified by `11` in Line 20. Its operator '`.>*`' is identified by `4` and both are mapped to each other in Line 16. The two arguments '`* &t`' and '`b`' are indirectly represented by tuple `2` in Line 17-19 and identified by `9` and `10`, respectively. While '`b`' is directly associated with `10` in Line 15, '`* &t`' is further decomposed in Line 14 into operator '`*`' (cf. Line 11) and its argument '`&t`'. The latter is captured by tuple `1` but not further listed for brevity.

[2] Filenames are of indicative nature only.

```
11  theory_string(5,"*").
12  theory_tuple(1).
13  theory_tuple(1,0,8).
14  theory_function(9,5,1).
15  theory_string(10,"b").
16  theory_string(4,".>*").
17  theory_tuple(2).
18  theory_tuple(2,0,9).
19  theory_tuple(2,1,10).
20  theory_function(11,4,2).
```

Listing 3: Facts `11-20` obtained by a call akin to `gringo -output=reify grammar.lp delex.lp > reified.lp`

The reified representation of the dynamic constraint in Listing 2 is now used to build the AFW in Figure 1 in terms of the facts in Listing 4. As shown in Figure 4, the facts in `afw.lp`

```
1   prop(10,"b"). prop(14,"a"). prop(16,"last").
2   state(0,dia(seq(test(box(str(stp),p(10))),stp),p(14))).
3   state(1,p(14)).
4   state(2,box(str(stp),p(10))).
5   initial_state(0).
6   delta(0,0). delta(0,0,out,16). delta(0,0,in,10).
7               delta(0,0,1). delta(0,0,2).
8   delta(1,0). delta(1,0,in,14).
9   delta(2,0). delta(2,0,out,16). delta(2,0,in,10).
10              delta(2,0,2).
11  delta(2,1). delta(2,1,in,16). delta(2,1,in,10).
```

Listing 4: Generated facts representing the AFW in Figure 1 (`afw.lp`)

are obtained by applying *clingo* to `ldlf2afw.lp` and `reified.lp`, the facts generated in the first step.

An automaton $\mathfrak{A}_\varphi$ is represented by the following predicates:

- `prop/2`, providing a symbol table mapping integer identifiers to atoms,
- `state/2`, providing states along with their associated dynamic formula; the initial state is distinguished by `initial_state/1`, and
- `delta/2,3,4`, providing the automaton's transitions.

The symbol table in Line 1 in Listing 4 is directly derived from the reified format. In addition, the special proposition `last` is associated with the first available identifier. The interpretations over `a`, `b`, `last` constitute the alphabet of the automaton at hand.

More efforts are needed for calculating the states of the automaton. Once all relevant symbols and operators are extracted from the reified format, they are used to build the closure $cl(\varphi)$ of φ in the input and to transform its elements into negation normal form. In the final representation of the automaton, we only keep reachable states and assign them a numerical identifier. The states in Line 2 to 3 correspond to the ones labeled q_φ, q_a and $q_{\Box b}$ in Figure 1.

The transition function is represented by binary, ternary, and quaternary versions of predicate `delta`. The representation is centered upon the conjunctions in the set representation of the DNF of $\delta(q, X)$ (cf. Section 3). Each conjunction `C` represents a transition from state `Q` and is captuted by `delta(Q,C)`. An atom of form `delta(Q,C,Q')` indicates that state `Q'` belongs to conjunction `C` and `delta(Q,C,T,A)` expresses the condition that either $\mathtt{A} \in X$ or $\mathtt{A} \notin X$ depending on whether `T` equals `in` or `out`, respectively. The binary version of `delta` is needed since there may be no instances of the ternary and quaternary ones.

The facts in Line 6 to 7 in Listing 4 capture the only transition from the initial state in Figure 1, viz. $\delta(q_{\varphi}, X) = \{\{q_{\Box b}, q_a\}\}$. Both the initial state and the transition are identified by `0` in Line 6. Line 6 also gives the conditions $last \notin X$ and $b \in X$ needed to reach the successor states given in Line 7. Line 8 accounts for $\delta(q_a, X) = \{\emptyset\}$, reaching $\top$ (ie., an empty set of successor states) from q_a provided $a \in X$. We encounter two possible transitions from state `2`, or $q_{[\tau^*]b}$. Transition `0` in Line 9 to 10 represents the loop $\delta(q_{[\tau^*]b}, X) = \{\{q_{[\tau^*]b}\}\}$ for $last \notin X$ and $b \in X$, while transition `1` in Line 11 captures $\delta(q_{[\tau^*]b}, X) = \{\emptyset\}$ that allows us to reach $\top$ whenever $\{last, b\} \subseteq X$.

Finally, the encoding in Listing 5 checks whether a trace is an accepted run of a given automaton. We describe traces using atoms of form `trace(A,T)`, stating that the atom identified

```
1  node(Q,0) :- initial_state(Q).
2  { select(C,Q,T): delta(Q,C) } = 1 :- node(Q,T), T<=lambda-1.
3  node(Q',T+1) :- select(C,Q,T), delta(Q,C,Q').
4  :- select(C,Q,T), delta(Q,C,in,A), not trace(A,T).
5  :- select(C,Q,T), delta(Q,C,out,A), trace(A,T).
```

Listing 5: Encoding defining the accepted runs of an automaton (`run.lp`).

by `A` is true in the trace at time step `T`. Although such traces are usually provided by the encoding of the dynamic problem at hand, the accepted runs of an automaton can also be enumerated by adding a corresponding choice rule. In addition, the special purpose atom `last` is made true in the final state of the trace.

For verifying whether a trace of length `lambda` is accepted, we build the tree corresponding to a run of the AFW on the trace at hand. This tree is represented by atoms of form `node(S,T)`, indicating that state `S` exists at depth/time `T`[3]. The initial state is anchored as the root in Line 1. In turn, nodes get expanded by depth by selecting possible transitions in Line 2. The nodes are then put in place by following the transition of the selected conjunction in Line 3. Lines 4 and 5 verify the conditions for the selected transition.

5.3. MONA-based translation into deterministic automata

The rightmost part of the workflow in Figure 4 relies on our translations of dynamic formulas into MSOs from [10]. These translations, called Monadic Second Order Encoding and Standard Translation, are defined for a dynamic formula φ and a time point t as $mso(t, \varphi)$ and $stm(t, \varphi)$,

[3]Note that we do not need to represent the edges between nodes as their depth is indicative enough for the acceptance. In the literature, runs of AFW are often represented using directed acyclic graphs instead of trees.

respectively. We use the off-the-shelf tool MONA[4] [11] to translate the resulting MSO formulas into DFAs. More precisely, we use *clingo*'s API to transform each dynamic constraint $\neg\neg\varphi$ in `<dyncon>.lp` either into MSO formula $mso(0,\varphi)$ or $stm(0,\varphi)$. This results in a file ẽxtttmso.mona in MONA's syntax, which is then turned by MONA into a corresponding DFA in `dot` format. All these automata are then translated into facts and gathered in `dfa.lp` in the same format as used for AFWs. The encoding in Listing 5 can be used to find accepted runs of DFAs by adding the following integrity constraint ensuring that runs end in a final state.

```
:- node(Q,lambda), not final_state(Q).
```

6. Evaluation

For our experimental studies, we use benchmarks from the domain of robotic intra-logistics stemming from the *asprilo* framework [20]. As illustrated in Figure 5 and 6, we consider grids of size 7×7 with $n \in \{2,3\}$ robots and $n*2$ orders of single products, each located on a unique shelf. At each timestep, a robot can: (i) *move* in a direction(ii) *pickup* a shelf (iii) *putdown* a shelf or (iv) *wait*. Moreover, a robot will *deliver* an order if it waits at a picking station while carrying a shelf. The goal is to take each shelf to a picking station; in an optimal plan (wrt. trace length) each robot processes two orders.

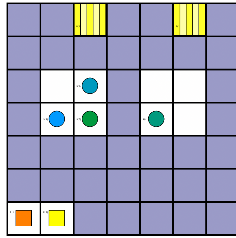


Figure 5: Asprilo visualization for two robots instance.

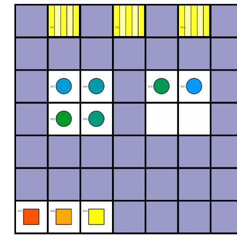


Figure 6: Asprilo visualization for three robots instance.

We consider three different dynamic constraints. The first one restricts plans such that if a robot picks up a shelf, then it must move or wait several times until the shelf is delivered. This is expressed by the dynamic formula φ_1 and represented in Listing 6[5], were $pickup_s$ and $deliver_s$ refer to a specific shelf.

$$\varphi_1 = [\tau^*]\,[pickup_s?]\,\langle(\tau;(move? + wait?))^*; deliver_s?\rangle\,\top$$

The second one, φ_2, represents a procedure where robots must repeat a sequence in which they move towards a shelf, pickup, move towards a picking station, deliver, move to the dropping place and putdown, and finish with waiting until the end of the trace .

$$\varphi_2 = \langle(move^*; pickup^*; move^*; deliver; move^*; putdown)^*; wait^*\rangle\,\mathbf{F}$$

[4] https://www.brics.dk/mona

[5] We start repetitions with τ as `&t`, to cope with movements in *asprilo* starting at time point 1.

```
1  :- not &del{
2        (* &t) .>*
3        ?pickup(robot(R),shelf(S)) .>*
4        *(&t ;; ?move(robot(R)) + ?waits(robot(R))) ;;
5          ?deliver(robot(R),shelf(S)) .>?
6        &true},
7     robot(R), shelf(S).
```

Listing 6: Dynamic constraint for formula φ_1.

Table 1
Automata size for the 3 robots instance showing the number of appearances of each atom.

φ_i	predicate	$\mathfrak{A}$	$\mathfrak{M}_m$	$\mathfrak{M}_s$
φ_1	state/2	36	72	72
	delta/2	162	234	216
φ_2	state/2	24	51	51
	delta/2	60	390	471
φ_3	state/2	45	-	372
	delta/2	189	-	16 503

For our last constraint we use the dynamic formula φ_3. This corresponds to a procedure similar to φ_2 but which relies on a predefined pattern, restricting the direction of movements with $move_r$, $move_l$, $move_u$ and $move_d$ to refer to moving right, left, up and down, respectively. We use the path $\rho = (move_r^* + move_l^*)$ so that robots only move in one horizontal direction. Additionally, each iteration starts by waiting so that whenever a robot starts moving, it fulfills the delivery without intermediate waiting.

$$\varphi_3 = \langle (wait^*; \rho; move_u^*; pickup; \rho; move_u^*; deliver; \rho; move_d^*; putdown)^*; wait^* \rangle \, \mathbf{F}$$

We use these constraints to contrast their implementations by means of our workflows $\mathfrak{A}$, $\mathfrak{T}$, $\mathfrak{M}_m$ and $\mathfrak{M}_s$ with $\lambda \in \{25, \ldots, 31\}$, while using the option of having no constraint, namely NC, as a baseline. The presented results ran using *clingo* 5.4.0 on an Intel Xeon E5-2650v4 under Debian GNU/Linux 9, with a memory of 20 GB and a timeout of 20 min per instance. All times are presented in milliseconds and any time out is counted as 1 200 000 ms in our calculations.

We first compare the size of the automata in Table 1 in terms of the instances of predicates `state/2` and `delta/2`. We see that $\mathfrak{A}$ generates an exponentially smaller automata, a known result from the literature [21]. More precisely, for φ_3 the number of transitions in $\mathfrak{M}_s$ is 90 times larger than for $\mathfrak{A}$. Furthermore, for this constraint, $\mathfrak{M}_m$ reached the limit of nodes for MONA's BDD-based architecture, thus producing no result. This outcome is based on the fact that the MSO formulas computed by $\mathfrak{M}_m$ are significantly larger than those of $\mathfrak{M}_s$.

Next, we give the preprocessing times obtained for the respective translations in Table 2. For the automata-based approaches $\mathfrak{A}$, $\mathfrak{M}_m$ and $\mathfrak{M}_s$ the translation is only performed once and reused in subsequent calls, whereas for $\mathfrak{T}$ the translation is redone for each horizon. The best performing approach is $\mathfrak{A}$, for the subsequent calls the times were very similar with the exception of $\mathfrak{T}$. We see how for φ_2 the $\mathfrak{M}_m$ translation takes considerably longer than for $\mathfrak{M}_s$.

Table 2
Pre-processing time in milliseconds shown as t_1/t_2 were t_1 is the time for the first horizon and t_2 the average over subsequent calls.

φ_i	#r	$\mathfrak{A}$	$\mathfrak{M}_m$	$\mathfrak{M}_s$	$\mathfrak{T}$	NC
φ_1	2	**1 194**/637	5 412/638	5 867/604	2 696/2 992	306/598
	3	**1 991**/600	6 280/671	6 978/610	3 390/3 691	302/617
φ_2	2	2 182/579	33 091/661	4 966/598	**2 107**/2 814	285/577
	3	**1 632**/608	45 303/665	4 973/604	2 718/3 179	318/631
φ_3	2	**2 533**/599	-	12 682/766	3 343/3 280	261/605
	3	**3 112**/600	-	11,001/795	3,278/3,718	272/598

Table 3
Statistics computed by calculating the geometric mean of all horizons.

	φ_i	#r	$\mathfrak{A}$	$\mathfrak{M}_m$	$\mathfrak{M}_s$	$\mathfrak{T}$	NC
clingo time	φ_1	2	3 374	**2 788**	2 975	3 033	21 823
		3	**23 173**	27 866	27 505	23 748	249 737
	φ_2	2	10 840	9 424	9 484	**9 347**	21 378
		3	70 709	**58 739**	83 521	60 765	246 739
	φ_3	2	31 986	-	606 914	**16 145**	21 548
		3	67 287	-	657 633	**48 190**	247 718
rules	φ_1	2	**89 282**	97 396	97 404	96 793	77 832
		3	**172 641**	196 220	190 943	189 637	147 209
	φ_2	2	**84 180**	122 003	126 634	90 178	77 832
		3	**157 525**	214 454	229 063	166 391	147 209
	φ_3	2	**94 653**	-	4 413 056	102 687	77 832
		3	**173 210**	-	3 360 382	185 155	147 209
constraints	φ_1	2	146 999	146 323	146 306	**140 801**	132 370
		3	275 747	274 419	274 382	**260 675**	241 752
	φ_2	2	**138 418**	166 449	171 796	139 909	132 370
		3	**252 023**	295 946	308 204	254 020	241 752
	φ_3	2	153 179	-	3 341 017	**147 123**	132 370
		3	274 851	-	2 743 736	**264 847**	241 752

The results of the final solving step in each workflow are summarized in Table 3, showing the geometric mean over all horizons for obtaining a first solution. First of all, we observe that the solving time is significantly lower when using dynamic constraints, no matter which approach is used. For φ_1 and φ_2 the difference is negligible, whereas for φ_3, $\mathfrak{T}$ is the fastest, followed by $\mathfrak{A}$, which is in turn twenty and ten times faster than $\mathfrak{M}_s$ for 2 and 3 robots, respectively. Furthermore, $\mathfrak{M}_s$ times out for φ_3 with $\lambda = 31$ and $\lambda \in \{30, 31\}$ for 2 and 3 robots, respectively. The size of the program before and after *clingo*'s preprocessing can be read off the number of ground rules and internal constraints, with $\mathfrak{A}$ having the smallest size of all approaches. However, once the program is reduced the number of constraints shows a slight shift in favour of $\mathfrak{T}$.

7. Discussion

To the best of our knowledge, this work presents the first endeavor to represent dynamic constraints with automata in ASP. The equivalence between temporal formulas and automata has been widely used in satisfiability checking, model checking, learning and synthesis [21, 22, 17, 23, 24]. Furthermore, the field of planning has benefited from temporal reasoning to express goals and preferences using an underlying automaton [25, 26, 27]. There exists several systems that translate temporal formulas into automata: SPOT [28] and LTLf2DFA[6] for linear temporal logic; *abstem* [29] and *stelp* [30] for temporal answer set programming. Nonetheless, there have only been a few attempts to use automata-like definitions in ASP for representing temporal and procedural knowledge inspired from GOLOG programs [31, 32].

We investigated different automata-based implementations of dynamic (integrity) constraints using *clingo*. Our first approach was based on alternating automata, implemented entirely in ASP through meta-programming. For our second approach, we employed the off-the-shelf automata construction tool MONA [11] to build deterministic automata. To this aim, we proposed two translations from dynamic logic into monadic second-order logic. These approaches were contrasted with the temporal ASP solver *telingo* which directly maps dynamic constraints to logic programs. We provided an empirical analysis demonstrating the impact of using dynamic constraints to select traces among the ones induced by an associated temporal logic program. Our study showed that the translation using solely ASP to compute an alternating automata yielded the smallest program in the shortest time. While this approach scaled well for more complex dynamic formulas, the MONA-based implementation performed poorly and could not handle one of our translations into second order formulas. The best overall performance was exhibited by *telingo* with the fundamental downside of having to redo the translation for each horizon.

Our future work aims to extend our framework to arbitrary dynamic formulas in DEL_f. Additionally, the automaton's independence of time stamps points to its potential to detect unsatisfiability and to guide an incremental solving process. Finally, we also intend to take advantage of *clingo*'s application programming interface to extend the model-ground-solve workflow of ASP with automata techniques.

References

[1] V. Lifschitz, Answer set planning, in: Proc. of the Int. Conf. on Logic Programming, MIT Press, 1999, pp. 23–37.

[2] A. Bosser, P. Cabalar, M. Diéguez, T. Schaub, Introducing temporal stable models for linear dynamic logic, in: Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning, AAAI Press, 2018, pp. 12–21.

[3] P. Cabalar, M. Diéguez, T. Schaub, Towards dynamic answer set programming over finite traces, in: [33], 2019, pp. 148–162.

[4] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[6]https://github.com/whitemech/LTLf2DFA

[5] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, C. Vidal, Temporal equilibrium logic: a survey, Journal of Applied Non-Classical Logics 23 (2013) 2–24.
[6] P. Cabalar, R. Kaminski, T. Schaub, A. Schuhmann, Temporal answer set programming on finite traces, Theory and Practice of Logic Programming 18 (2018) 406–420.
[7] P. Cabalar, M. Diéguez, T. Schaub, A. Schuhmann, Towards metric temporal answer set programming, Theory and Practice of Logic Programming 20 (2020) 783–798.
[8] R. Kaminski, T. Schaub, P. Wanko, A tutorial on hybrid answer set solving with clingo, in: Proc. of the Int. Summer School of the Reasoning Web, volume 10370 of *LNCS*, Springer, 2017, pp. 167–203.
[9] T. Wolfgang, Languages, automata, and logic, in: Handbook of Formal Languages, volume 3: Beyond Words, Springer, 1997, pp. 389–455.
[10] P. Cabalar, M. Diéguez, S. Hahn, T. Schaub, Automata for dynamic answer set solving: Preliminary report, 2021. arXiv:2109.01782.
[11] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, A. Sandholm, Mona: Monadic second-order logic in practice, in: Proc. of the Int. Workshop on Tools and Algorithms for Construction and Analysis of Systems, volume 1019 of *LNCS*, Springer, 1995, pp. 89–110.
[12] P. Cabalar, R. Kaminski, P. Morkisch, T. Schaub, telingo = ASP + time, in: [33], 2019, pp. 256–269.
[13] P. Cabalar, M. Diéguez, F. Laferriere, T. Schaub, Implementing dynamic answer set programming over finite traces, in: Proc. of the European Conf. on AI, volume 325 of *Frontiers in AI and Applications*, IOS Press, 2020, pp. 656–663.
[14] D. Harel, J. Tiuryn, D. Kozen, Dynamic Logic, MIT Press, 2000.
[15] G. De Giacomo, M. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proc. of the Int. Joint Conf. on AI, IJCAI/AAAI Press, 2013, pp. 854–860.
[16] A. Chandra, D. Kozen, L. Stockmeyer, Alternation, Journal of the ACM 28 (1981) 114–133.
[17] G. De Giacomo, M. Vardi, Synthesis for LTL and LDL on finite traces, in: Proc. of the Int. Joint Conf. on AI, AAAI Press, 2015, pp. 1558–1564.
[18] M. Fischer, R. Ladner, Propositional dynamic logic of regular programs, Journal of Computer and System Sciences 18 (1979) 194–211.
[19] G. Tseitin, On the complexity of derivation in the propositional calculus, Zapiski nauchnykh seminarov LOMI 8 (1968) 234–259.
[20] M. Gebser, P. Obermeier, T. Otto, T. Schaub, O. Sabuncu, V. Nguyen, T. Son, Experimenting with robotic intra-logistics domains, Theory and Practice of Logic Programming 18 (2018) 502–519.
[21] M. Vardi, An automata-theoretic approach to linear temporal logic, in: Logics for Concurrency: Structure versus Automata, volume 1043 of *LNCS*, Springer, 1995, pp. 238–266.
[22] M. Vardi, Alternating automata: Unifying truth and validity checking for temporal logics, in: Proc. of the Int. Conf. on Automated Deduction, volume 1249 of *LNCS*, Springer, 1997, pp. 191–206.
[23] K. Rozier, M. Vardi, Ltl satisfiability checking, in: Int. SPIN Workshop on Model Checking of Software, Springer, 2007, pp. 149–167.
[24] A. Camacho, S. McIlraith, Learning interpretable models expressed in linear temporal logic, in: Proc. of the Int. Conf. on Automated Planning and Scheduling, AAAI Press, 2019,

pp. 621–630.
[25] J. Baier, C. Fritz, M. Bienvenu, S. McIlraith, Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners, in: Proc. of the National Conf. on AI, AAAI Press, 2008, pp. 1509–1512.
[26] G. D. Giacomo, S. Rubin, Automata-theoretic foundations of fond planning for LTLf and LDLf goals., in: Proc. of the Int. Joint Conf. on Artificial Intelligence, ijcai.org, 2018, pp. 4729–4735.
[27] J. Baier, S. McIlraith, Planning with first-order temporally extended goals using heuristic search, in: Proc. of the National Conf. on AI, AAAI Press, 2006, pp. 788–795.
[28] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, L. Xu, Spot 2.0 - A framework for LTL and ω-automata manipulation, in: Proc. of the Int. Symposium on Automated Technology for Verification and Analysis, volume 9938 of *LNCS*, 2016, pp. 122–129.
[29] P. Cabalar, M. Diéguez, Strong equivalence of non-monotonic temporal theories, in: Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning, AAAI Press, 2014.
[30] P. Cabalar, M. Diéguez, STELP — a tool for temporal answer set programming, in: Proc. of the Int. Conf. on Logic Programming and Nonmonotonic Reasoning, volume 6645 of *LNAI*, Springer, 2011, pp. 370–375.
[31] T. Son, C. Baral, T. Nam, S. McIlraith, Domain-dependent knowledge in answer set planning, ACM Transactions on Computational Logic 7 (2006) 613–657.
[32] M. Ryan, Efficiently implementing GOLOG with answer set programming, in: Proc. of the National Conf. on AI, AAAI Press, 2014, pp. 2352–2357.
[33] Proc. of the Int. Conf. on Logic Programming and Nonmonotonic Reasoning, volume 11481 of *LNAI*, Springer, 2019.