# Automata for dynamic answer set solving: Preliminary report

Pedro Cabalar
University of Corunna, Spain

Martín Diéguez
Université d'Angers, France

Susana Hahn and Torsten Schaub
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.

## 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 logic (cf. Proposition 3). 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 $\mathrm{DEL}_f$.

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]). Second, the one elaborated in Section 5, proposing two alternative transformations of dynamic formula into monadic second order formulas. These formulas can then be passed to the off-the-shelf automata construction tool MONA [9] that turns them into deterministic automata. And finally, the approach of *telingo* [10, 11], 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* $\varphi$ and *path expressions* $\rho$ 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; [12]) 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 $\rho$ is a regular expression formed with the constant $\tau$ plus the test construct $\varphi?$ that may refer to propositional atoms in the (single) alphabet $\mathcal{P}$. As with LDL [13], we sometimes use a propositional formula $\phi$ 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 $\tau$, as we see below. Another abbreviation is the sequence of $n$ repetitions of some expression $\rho$ 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]:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\varphi \wedge \psi & \stackrel{def}{=} & \langle \varphi ? \rangle \, \psi & \varphi \vee \psi & \stackrel{def}{=} & \langle \varphi ? + \psi ? \rangle \, \top & & & \\
\varphi \rightarrow \psi & \stackrel{def}{=} & [\varphi ?] \, \psi & \neg \varphi & \stackrel{def}{=} & \varphi \rightarrow \bot & & & \\
\mathsf{O} \varphi & \stackrel{def}{=} & \langle \tau \rangle \, \varphi & \widehat{\mathsf{O}} \varphi & \stackrel{def}{=} & [\tau] \, \varphi & \mathsf{F} & \stackrel{def}{=} & [\tau] \, \bot \\
\Diamond \varphi & \stackrel{def}{=} & \langle \tau^* \rangle \, \varphi & \Box \varphi & \stackrel{def}{=} & [\tau^*] \, \varphi & & & \\
\varphi \, \mathsf{U} \, \psi & \stackrel{def}{=} & \langle (\varphi ?; \tau)^* \rangle \, \psi & \varphi \, \mathsf{R} \, \psi & \stackrel{def}{=} & (\psi \, \mathsf{U} \, (\varphi \wedge \psi)) \vee \Box \psi & & &
\end{array}$$

All connectives are defined in terms of the dynamic operators $\langle \cdot \rangle$ and $[\cdot]$. This involves the Booleans $\wedge$, $\vee$, 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 $\mathsf{F}$, $\mathsf{O}$, $\widehat{\mathsf{O}}$, $\Diamond$, $\Box$, $\mathsf{U}$, $\mathsf{R}$, 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 $\lambda$ 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 $\lambda$, 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 $\lambda$ 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* $\lambda$ *over alphabet* $\mathcal{P}$ satisfies *a dynamic formula* $\varphi$ *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 $\Gamma$ 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 $\lambda$ of a theory $\Gamma$, and define $\mathrm{DHT}(\Gamma) \stackrel{def}{=} \bigcup_{\lambda=0}^{\omega} \mathrm{DHT}(\Gamma, \lambda)$, that is, the whole set of models of $\Gamma$ of any length. A formula $\varphi$ 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 $\mathrm{DHT}_\omega$ and $\mathrm{DHT}_f$ by restricting DHT to infinite or finite traces, respectively.

**Proposition 1.** *For any* $(x,y) \in \mathbb{N} \times \mathbb{N}$*, path expression* $\rho$ *and trace* $\mathbf{M}$*, we have* $(x,y) \in \| \rho \|^{\mathbf{M}}$ *implies* $x \leq y$.

**Proposition 2** ([11, 14])**.** *The following formulas are* $\mathrm{DHT}_f$*-valid.*

1. $[\rho_1 + \rho_2]\,\varphi \leftrightarrow ([\rho_1]\,\varphi \wedge [\rho_2]\,\varphi)$
2. $\langle \rho_1 + \rho_2 \rangle\,\varphi \leftrightarrow (\langle \rho_1 \rangle\,\varphi \vee \langle \rho_2 \rangle\,\varphi)$
3. $[\rho_1; \rho_2]\,\varphi \leftrightarrow ([\rho_1]\,[\rho_2]\,\varphi)$
4. $\langle \rho_1; \rho_2 \rangle\,\varphi \leftrightarrow (\langle \rho_1 \rangle\,\langle \rho_2 \rangle\,\varphi)$
5. $[\rho^*]\,\varphi \leftrightarrow (\varphi \wedge [\rho]\,[\rho^*]\,\varphi)$
6. $\langle \rho^* \rangle\,\varphi \leftrightarrow (\varphi \vee \langle \rho \rangle\,\langle \rho^* \rangle\,\varphi)$

We refrain from giving the semantics of LDL [13], 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 $\varphi$ 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 $\mathrm{LDL}_\omega$ and $\mathrm{LDL}_f$, 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 3.** *For any* HT*-trace* $\langle \mathbf{H}, \mathbf{T} \rangle$ *of length* $\lambda$ *and any dynamic formula* $\varphi$*, 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 $\Gamma$ 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 $\Gamma$. Besides, as the ordering relation among traces is only defined for a fixed $\lambda$, the set of temporal equilibrium models of $\Gamma$ can be partitioned by the trace length $\lambda$, 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 $\mathrm{DEL}_\omega$ and $\mathrm{DEL}_f$ by applying the corresponding restriction to infinite or finite traces, respectively.

As a consequence of Proposition 3, the addition of formula $\neg\neg\varphi$ to a theory $\Gamma$ enforces that every temporal stable model of $\Gamma$ satisfies $\varphi$. With this, we confine ourselves in Section 4 and 5 to $\mathrm{LDL}_f$ rather than $\mathrm{DEL}_f$.

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 $\Sigma$ 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 $\delta$ 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; [15, 13]) is a tuple $(\Sigma, Q, q_0, \delta, F)$, where $\Sigma$ 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 6.2, 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 $\mathrm{LDL}_f$ to AFW

This section describes a translation of dynamic formulas in $\mathrm{LDL}_f$ to AFW due to [16]. More precisely, it associates a dynamic formula $\varphi$ in negation normal form with an AFW $\mathfrak{A}_\varphi$, whose number of states is linear in the size of $\varphi$ and whose language $\mathcal{L}(\mathfrak{A}_\varphi)$ coincides with the set of all traces satisfying $\varphi$. A dynamic formula $\varphi$ 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 closure $cl(\varphi)$ of $\varphi$ defined as the smallest set of dynamic formulas such that [17]

1. $\varphi \in cl(\varphi)$
2. if $\psi \in cl(\varphi)$ and $\psi$ is not of the form $\neg\psi'$ then $\neg\psi \in cl(\varphi)$
3. if $\langle\rho\rangle\,\psi \in cl(\varphi)$ then $\psi \in cl(\varphi)$
4. if $\langle\psi?\rangle\,\psi \in cl(\varphi)$ then $\psi \in cl(\varphi)$
5. if $\langle\rho_1;\rho_2\rangle\,\psi \in cl(\varphi)$ then $\langle\rho_1\rangle\,\langle\rho_2\rangle\,\psi \in cl(\varphi)$
6. if $\langle\rho_1+\rho_2\rangle\,\psi \in cl(\varphi)$ then $\langle\rho_1\rangle\,\psi \in cl(\varphi)$ and $\langle\rho_2\rangle\,\psi \in cl(\varphi)$
7. if $\langle\rho^*\rangle\,\psi \in cl(\varphi)$ then $\langle\rho\rangle\,\langle\rho*\rangle\,\psi \in cl(\varphi)$

The alphabet of an AFW $\mathfrak{A}_\varphi$ for a formula $\varphi$ over $\mathcal{P}$ is $\Sigma = 2^{\mathcal{P}\cup\{last\}}$. It relies on a special proposition *last* [16], which is only satisfied by the last state of the trace. A finite word over $\Sigma$ corresponds to a finite trace over $\mathcal{P} \cup \{last\}$.

**Definition 2** ($\mathrm{LDL}_f$ to AFW[16])**.** *Given a dynamic formula $\varphi$ 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 $\delta$ 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, $\varphi$,

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

stating that $b$ always holds and $a$ is true at the next step. The AFW for $\varphi$ is $\mathfrak{A}_\varphi = (2^{\{a,b,last\}}, Q^+ \cup Q^-, \delta, \emptyset)$, where

$$Q^+ = \{ q_{\langle ([\tau^*] b)? \, ; \, \tau \rangle \, a}, q_{\langle ([\tau^*] b)? \rangle \langle \tau \rangle \, a}, q_{[\tau^*] b}, q_{[\tau] [\tau^*] b}, q_\tau, q_b, q_{\langle \tau \rangle \, a}, q_a \}$$

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.

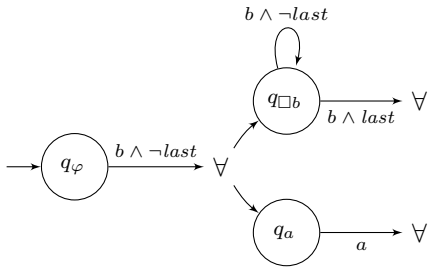


Figure 1: $\mathfrak{A}_\varphi$ showing only the reachable states. The special node type, labeled as $\forall$, represents universal transitions, when the $\forall$-node has no outgoing edges it represents the empty universal constraint $\top$.

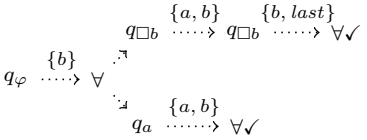


Figure 2: Accepted run for $\{b\} \cdot \{a, b\} \cdot \{b, last\}$.

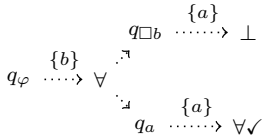


Figure 3: Rejected run for $\{b\} \cdot \{a\} \cdot \{b, last\}$.

# 5 Translating $\mathrm{LDL}_f$ to MSO

It is well-known that while LTL and $\mathrm{LTL}_f$ can be encoded into first-order logic, the case of LDL and $\mathrm{LDL}_f$ is rather different. The encoding of $\mathrm{LDL}_f$ requires the translation of path expressions of the type $\rho^*$ (the reflexive, transitive closure of a relation), which is not first-order representable.

This is why we need to consider a more expressive formalism and Monadic Second Order (MSO) of Linear Order [18] (MSO($<$)) will be our target logic. This logic enhances monadic first-order logic of linear order [19] with second order quantification.

## 5.1 Monadic Second-order of Linear Order

Let $\mathcal{P}$ be an input alphabet [1] , $\mathcal{V}_1$ be a set of first-order variables denoted by bolded lowercase letters and a set $\mathcal{V}_2$ of second-order variables usually denoted by bolded uppercase letters.

Well-formed formulas of MSO($<$) are defined according to the following syntax:

$$\varphi := \mathbf{X}(x) \mid \mathbf{x} < \mathbf{y} \mid \neg\varphi \mid \varphi \vee \psi \mid \exists\, \mathbf{x}.\varphi \mid \exists \mathbf{X}\, \varphi.$$

where $\mathbf{x}, \mathbf{y} \in \mathcal{V}_1$ and $\mathbf{X} \in \mathcal{V}_2$.

The following abbreviations involving logical formulas and orders are valid:

[1] By abuse of notation, we use the symbol $\mathcal{P}$ as for the set of propositional variables in $\mathrm{LDL}_f$, since they will be translated into elements of the alphabet.

$$\begin{array}{ll}
\varphi \wedge \psi \stackrel{def}{=} \neg(\neg\varphi \vee \neg\psi) & \varphi \rightarrow \psi \stackrel{def}{=} \neg\varphi \vee \psi \\
\varphi \leftrightarrow \psi \stackrel{def}{=} (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) & \forall \mathbf{x}\, \varphi \stackrel{def}{=} \neg\exists \mathbf{x}\, \neg\varphi \\
\mathbf{x} \geq \mathbf{y} \stackrel{def}{=} \neg(\mathbf{x} < \mathbf{y}) & \mathbf{x} \leq \mathbf{y} \stackrel{def}{=} \mathbf{y} \geq \mathbf{x} \\
\mathbf{x} = \mathbf{y} \stackrel{def}{=} (\mathbf{x} \leq \mathbf{y}) \wedge (\mathbf{y} \leq \mathbf{x}) & \mathbf{x} \neq \mathbf{y} \stackrel{def}{=} \neg(\mathbf{x} = \mathbf{y}) \\
\mathbf{x} > \mathbf{y} \stackrel{def}{=} \mathbf{y} < \mathbf{x} &
\end{array}$$

Moreover, the following abbreviations involving first-order and second-order variables will be used along this section.

$$\begin{array}{ll}
\mathtt{succ}(\mathbf{x},\mathbf{y}) \stackrel{def}{=} \mathbf{x} < \mathbf{y} \wedge \neg\exists \mathbf{z}\,(\mathbf{x} < \mathbf{z} \wedge \mathbf{z} < \mathbf{y}) & \\
\mathtt{first}(\mathbf{x}) \stackrel{def}{=} \neg\exists \mathbf{y}\; \mathbf{y} < \mathbf{x} & \mathbf{x} \in \mathbf{X} \stackrel{def}{=} \mathbf{X}(\mathbf{x}) \\
\mathtt{last}(\mathbf{x}) \stackrel{def}{=} \neg\exists \mathbf{y}\; \mathbf{y} > \mathbf{x} & \mathbf{X} \neq \mathbf{Y} \stackrel{def}{=} \neg\mathbf{X} = \mathbf{Y} \\
\mathtt{bound}(\mathbf{X},\mathbf{w},\mathbf{v}) \stackrel{def}{=} \forall \mathbf{r}(\mathbf{X}(\mathbf{r}) \rightarrow (\mathbf{w} \leq \mathbf{r} \wedge \mathbf{r} \leq \mathbf{v})) & \mathbf{X} = \mathbf{Y} \stackrel{def}{=} \mathbf{X} \subseteq \mathbf{Y} \wedge \mathbf{Y} \subseteq \mathbf{X} \\
\mathbf{X} \subseteq \mathbf{Y} \stackrel{def}{=} \forall \mathbf{x}\,(\mathbf{x} \in \mathbf{X} \rightarrow \mathbf{x} \in \mathbf{Y}) &
\end{array}$$

A $\mathtt{MSO}(<)$ formula is interpreted over a trace $\mathbf{T}$ of length $\lambda$ with respect to two assignments $v_1 : \mathcal{V}_1 \mapsto \{0, \cdots, \lambda - 1\}$ and $v_2 : \mathcal{V}_2 \mapsto 2^{\{0,\cdots,\lambda-1\}}$. Notice that $v_1$ maps every first-order variable in $\mathcal{V}_1$ into a position in $\mathbf{T}$ while $v_2$ maps each second order variable of $\mathcal{V}_2$ to a set of positions in $\mathbf{T}$. Given a second-order assignment $v_2$, by $v_2[\mathbf{X} := D]$ we refer to an extension of $v_2$ obtained by assigning to the second-order variable $\mathbf{X}$ the set $D \subseteq \{0, \cdots, \lambda - 1\}$. For the case of a first-order assignment $v_1$, by $v_1[\mathbf{x} := d]$ we refer to an extension of $v_1$ obtained by assigning to the first-order variable $\mathbf{x}$ the value $d \in \{0, \cdots, \lambda - 1\}$.

**Definition 3** ($\mathtt{MSO}(<)$ satisfaction)**.** *A trace $\mathbf{T}$ of length $\lambda$ satisfies a $\mathtt{MSO}(<)$ formula $\varphi$ wrt. assignments $v_1$ and $v_2$, written as $\mathbf{T}, v_1, v_2 \models \varphi$ if the following conditions hold:*

1. $\mathbf{T}, v_1, v_2 \models \mathbf{X}(\mathbf{x})$ *iff* $v_1(\mathbf{x}) \in v_2(\mathbf{X})$
2. $\mathbf{T}, v_1, v_2 \models \mathbf{x} < \mathbf{y}$ *iff* $v_1(\mathbf{x}) < v_1(\mathbf{y})$ *holds*
3. $\mathbf{T}, v_1, v_2 \models \neg\varphi$ *iff* $\mathbf{T}, v_1, v_2 \not\models \varphi$
4. $\mathbf{T}, v_1, v_2 \models \varphi \wedge \psi$ *iff* $\mathbf{T}, v_1, v_2 \models \varphi$ *and* $\mathbf{T}, v_1, v_2 \models \psi$
5. $\mathbf{T}, v_1, v_2 \models \varphi \vee \psi$ *iff* $\mathbf{T}, v_1, v_2 \models \varphi$ *or* $\mathbf{T}, v_1, v_2 \models \psi$
6. $\mathbf{T}, v_1, v_2 \models \varphi \rightarrow \psi$ *iff* $\mathbf{T}, v_1, v_2 \not\models \varphi$ *or* $\mathbf{T}, v_1, v_2 \models \psi$
7. $\mathbf{T}, v_1, v_2 \models \exists \mathbf{x}\, \varphi$ *iff* $\mathbf{T}, v_1[\mathbf{x} := d], v_2 \models \varphi$ *for some* $0 \leq d < \lambda$
8. $\mathbf{T}, v_1, v_2 \models \forall \mathbf{x}\, \varphi$ *iff* $\mathbf{T}, v_1[\mathbf{x} := d], v_2 \models \varphi$ *for all* $0 \leq d < \lambda$
9. $\mathbf{T}, v_1, v_2 \models \exists \mathbf{X}\, \varphi$ *iff* $\mathbf{T}, v_1, v_2[\mathbf{X} := D] \models \varphi$ *for some* $D \in 2^{\{0\cdots\lambda-1\}}$
10. $\mathbf{T}, v_1, v_2 \models \forall \mathbf{X}\, \varphi$ *iff* $\mathbf{T}, v_1, v_2[\mathbf{X} := D] \models \varphi$ *for all* $D \in 2^{\{0\cdots\lambda-1\}}$

## 5.2 Standard Translation

In this subsection we extend the so called *standard translation* of $\mathrm{LTL}_f$ [13] to the case of $\mathrm{LDL}_f$. In order to represent $\rho^*$, first-order logic can be equipped with countable infinite disjunctions as proposed in [20]. Conversely, we use a second order quantified predicate $X$ to capture the points where path expressions are satisfied in a trace. As in [20] our standard translation is defined in terms of two translations $ST_m$ and $ST_p$ for dynamic formulas and paths, respectively.

**Definition 4** ($\mathrm{LDL}_f$ Standard Translation)**.** *Given a dynamic formula $\varphi$ and a free variable $\mathbf{w}$ representing the time point in which $\varphi$ is evaluated, $ST_m$ is defined as follows:*

1. $ST_m(\mathbf{w}, p) \stackrel{def}{=} \mathbf{P}(\mathbf{w})$
2. $ST_m(\mathbf{w}, \top) \stackrel{def}{=} \top$
3. $ST_m(\mathbf{w}, \bot) \stackrel{def}{=} \bot$
4. $ST_m(\mathbf{w}, [\rho]\,\varphi) \stackrel{def}{=} \forall \mathbf{v}(ST_p(\mathbf{wv}, \rho) \rightarrow ST_m(\mathbf{v}, \varphi))$
5. $ST_m(\mathbf{w}, \langle\rho\rangle\,\varphi) \stackrel{def}{=} \exists \mathbf{v}(ST_p(\mathbf{wv}, \rho) \wedge ST_m(\mathbf{v}, \varphi))$

*The translation for paths $ST_p$ takes as inputs a path expression $\rho$ and two free variables $\mathbf{w}$ and $\mathbf{v}$ meaning that $\rho$ is satisfied between the time points $\mathbf{w}$ and $\mathbf{v}$, defined as follows:*

6. $ST_p(\mathbf{wv}, \tau) \stackrel{def}{=} \mathbf{v} = \mathbf{w} + 1$
7. $ST_p(\mathbf{wv}, \varphi?) \stackrel{def}{=} ST_m(\mathbf{w}, \varphi) \wedge \mathbf{w} = \mathbf{v}$
8. $ST_p(\mathbf{wv}, \rho_1 + \rho_2) \stackrel{def}{=} ST_p(\mathbf{wv}, \rho_1) \vee ST_p(\mathbf{wv}, \rho_2)$
9. $ST_p(\mathbf{wv}, \rho_1; \rho_2) \stackrel{def}{=} \exists \mathbf{u}(ST_p(\mathbf{wu}, \rho_1) \wedge ST_p(\mathbf{uv}, \rho_2))$
10. $ST_p(\mathbf{wv}, \rho^*) \stackrel{def}{=} \exists \mathbf{X}(\mathbf{X}(\mathbf{w}) \wedge \mathbf{X}(\mathbf{v}) \wedge \mathtt{bound}(\mathbf{X}, \mathbf{w}, \mathbf{v}) \wedge \mathtt{regular}(\mathbf{X}))$

*where $\mathtt{bound}(\mathbf{X}, \mathbf{w}, \mathbf{v})$ is defined in Subsection 5.1 and*

$$\mathtt{regular}(\mathbf{X}) \stackrel{def}{=} \forall \mathbf{x}, \mathbf{y}\ ((\mathtt{succ}(\mathbf{x}, \mathbf{y}) \wedge \mathbf{X}(\mathbf{x}) \wedge \mathbf{X}(\mathbf{y})) \rightarrow ST_p(\mathbf{xy}, \rho))\,.$$

Let $\mathbf{T}$ be a trace of length $\lambda$ and let $v_2$ be an assignment such that each $\mathbf{p} \in \mathcal{P}$, $v_2(\mathbf{P}) = \{x \mid p \in T_x\}$. With this definition we can prove the model correspondence stated in the following theorem.

**Theorem 1.** *Let $\varphi$ be a dynamic formula. Then, for any trace $\mathbf{T}$ of length $\lambda$; $k, d \in [0, \lambda)$ and free variables $\mathbf{x}, \mathbf{y}$, we have*

1. $\mathbf{T}, k \models \varphi$ *iff* $\mathbf{T}, v_1[\mathbf{x} := k], v_2 \models ST_m(\mathbf{x}, \varphi)$
2. $(k, d) \in \| \rho \|^{\mathbf{T}}$ *iff* $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho)$

As an example, the standard translation of the formula $\varphi = \langle([\tau^*]\, b)?\ ;\ \tau\rangle\, a$ with respect to the free variable $\mathbf{t}$ is

$$\begin{aligned} ST_m(\mathbf{t},\varphi) = \exists \mathbf{v_0}(\exists \mathbf{v_1}(\forall \mathbf{v_2}(\exists \mathbf{X}(\mathbf{X}(\mathbf{t}) \wedge \mathbf{X}(\mathbf{v_2}) \wedge \mathtt{bound}(\mathbf{X},\mathbf{t},\mathbf{v_2}) \wedge \\ \forall \mathbf{x},\mathbf{y}((\mathbf{X}(\mathbf{x}) \wedge \mathbf{X}(\mathbf{y}) \wedge \mathtt{succ}(\mathbf{x},\mathbf{y})) \\ \rightarrow (\mathbf{y} = \mathbf{x}+1))) \\ \rightarrow \mathbf{b}(\mathbf{v_2})) \quad (\mathbf{v_0} = \mathbf{v_1}+1)) \wedge \mathbf{a}(\mathbf{v_0})) \end{aligned}$$

## 5.3 Monadic Second Order Encoding

In this subsection we provide an alternative translation into ($\mathtt{MSO}(<)$ where the second-order encoding consists of a sequence of existential monadic second-order quantifiers followed by a single universal first-order quantifier. We extend the translation in [21] from $\mathrm{LTL}_f$ to the case of $\mathrm{LDL}_f$ based on the notion of Fisher-Ladner closure [17].

**Definition 5** ($\mathrm{LDL}_f$ Monadic Second Order Encoding)**.** *Given a dynamic formula $\varphi$ and a free variable $\mathbf{t}$, $mso(\mathbf{t},\varphi)$ states the truth of $\varphi$ at $\mathbf{t}$ as follows:*

$$mso(\mathbf{t},\varphi) \stackrel{def}{=} (\exists \mathbf{Q}_{\theta_0} \cdots \exists \mathbf{Q}_{\theta_m}(\mathbf{Q}_\varphi(\mathbf{t}) \wedge (\forall \mathbf{x}(\wedge_{i=0}^{m} t(\theta_i,\mathbf{x})))))$$

*where, $\theta_i \in cl(\varphi)$, $\mathbf{Q}_{\theta_i}$ is a fresh predicate name and $t(\theta_i,\mathbf{x})$ asserts the truth of every non-atomic subformula $\theta_i$ in $cl(\varphi)$ at time point $\mathbf{x}$, imitating the semantics of* $\mathrm{LDL}_f$*, provided in Table 1.*

| $\boldsymbol{\mu \in cl(\varphi)}$ | $\boldsymbol{t(\mu,\mathbf{x})}$ |
|---|---|
| $\neg\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow \neg\mathbf{Q}_\psi(\mathbf{x})$ |
| $\langle\tau\rangle\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\exists \mathbf{y}(\mathbf{y} = \mathbf{x}+1 \wedge (\mathbf{Q}_\psi(\mathbf{y}))))$ |
| $\langle\mu?\rangle\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_\mu(\mathbf{x}) \wedge \mathbf{Q}_\mu(\mathbf{x}))$ |
| $\langle\rho_1+\rho_2\rangle\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_{\langle\rho_1\rangle\,\psi}(\mathbf{x}) \vee \mathbf{Q}_{\langle\rho_2\rangle\,\psi}(\mathbf{x}))$ |
| $\langle\rho_1;\rho_2\rangle\,\psi$ | $Q_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_{\langle\rho_1\rangle\,\langle\rho_2\rangle\,\psi}(\mathbf{x}))$ |
| $\langle\rho^*\rangle\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_\psi(\mathbf{x}) \vee \mathbf{Q}_{\langle\rho\rangle\,\langle\rho^*\rangle\,\psi}(\mathbf{x}))$ |
| $[\tau]\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\forall \mathbf{y}(\mathbf{y} = \mathbf{x}+1 \rightarrow (\mathbf{Q}_\psi(\mathbf{y}))))$ |
| $[\mu?]\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_\mu(\mathbf{x}) \rightarrow \mathbf{Q}_\mu(\mathbf{x}))$ |
| $[\rho_1+\rho_2]\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_{[\rho_1]\,\psi}(\mathbf{x}) \wedge \mathbf{Q}_{[\rho_2]\,\psi}(\mathbf{x}))$ |
| $[\rho_1;\rho_2]\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_{[\rho_1]\,[\rho_2]\,\psi}(\mathbf{x}))$ |
| $[\rho^*]\,\psi$ | $\mathbf{Q}_\mu(\mathbf{x}) \leftrightarrow (\mathbf{Q}_\psi(\mathbf{x}) \wedge \mathbf{Q}_{[\rho]\,[\rho^*]\,\psi}(\mathbf{x}))$ |

Table 1: MSO Translation for subformulas in the closure.

Let $\mathbf{T}$ be a trace of length $\lambda$ and let $v_2$ be a second-order assignment such that for each $p \in \mathcal{P}$, $v_2(\mathbf{P}) = \{x \mid p \in T_x\}$. With this definition we can prove the model correspondence stated in the following theorem.

**Theorem 2.** *Let $\varphi$ be a dynamic formula. Then, for any trace $\mathbf{T}$ of length $\lambda$ and time point $k \in [0, \lambda)$, we have that $\mathbf{T}, k \models \varphi$ iff $\mathbf{T}, v_1[\mathbf{t} := k], v_2 \models mso(\mathbf{t}, \varphi)$.*

For instance, given $\varphi = \langle([\tau^*]\, b)?\ ;\ \tau\rangle\, a$,

$$\mathrm{mso}(\mathbf{t}, \varphi) = \exists\, \mathbf{Q_0} \exists \mathbf{Q_1}\ \exists \mathbf{Q_2}\ \exists \mathbf{Q_3}\ \exists \mathbf{Q_4}\ (\mathbf{Q_0}(\mathbf{t}) \wedge \forall \mathbf{x}\ (\wedge_{i=0}^{4} t(\mu_i, \mathbf{x})),$$

where each $t(\mu_i, \mathbf{x})$ is defined in Table 2.

| $Q_\mu$ | $\mu \in cl(\varphi)$ | $t(\mu, \mathbf{x})$ |
|---|---|---|
| $\mathbf{Q_0}$ | $\langle([\tau^*]\, b)?\ ;\ \tau\rangle\, a$ | $\mathbf{Q_0}(\mathbf{x}) \leftrightarrow \mathbf{Q_1}(\mathbf{x})$ |
| $\mathbf{Q_1}$ | $\langle([\tau^*]\, b)?\rangle\, \langle\tau\rangle\, a$ | $\mathbf{Q_1}(\mathbf{x}) \leftrightarrow \mathbf{Q_4}(\mathbf{x}) \wedge \mathbf{Q_2}(\mathbf{x})$ |
| $\mathbf{Q_2}$ | $[\tau^*]\, b$ | $\mathbf{Q_2}(\mathbf{x}) \leftrightarrow \mathbf{Q_b}(\mathbf{x}) \wedge \mathbf{Q_3}(\mathbf{x})$ |
| $\mathbf{Q_3}$ | $[\tau]\, [\tau^*]\, b$ | $\mathbf{Q_3}(\mathbf{x}) \leftrightarrow \forall \mathbf{v}\ \mathbf{v} = \mathbf{x} + 1 \rightarrow \mathbf{Q_2}(\mathbf{v})$ |
| $\mathbf{Q_4}$ | $\langle\tau\rangle\, a$ | $\mathbf{Q_4}(\mathbf{x}) \leftrightarrow \exists \mathbf{v}\ \mathbf{v} = \mathbf{x} + 1 \wedge \mathbf{Q_a}(\mathbf{v})$ |

Table 2: MSO Translation for non atomic subformulas of $\varphi$.

# 6 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 [2] 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` $\varphi$`.` 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, 10], 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.

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

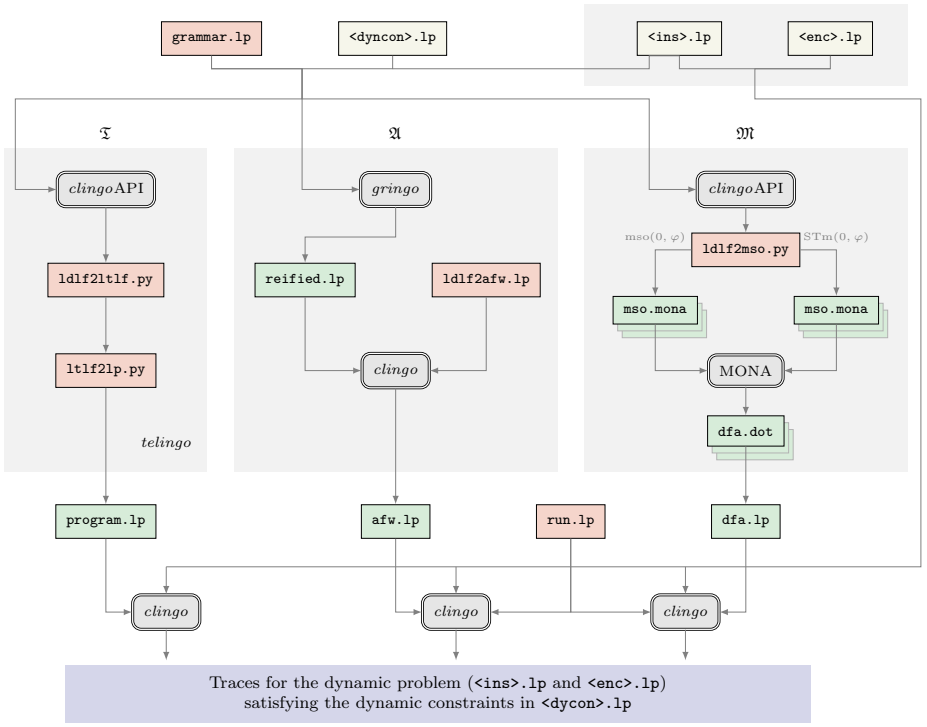


Figure 4: Workflows of our framework. Elements in yellow correspond to user input, green ones are automatically generated, and red ones are provided by the system to solve the problem.

```
1  #theory del {
2   formula_body {
3      & : 7,unary; ~ : 5,unary;
4      ? : 4,unary; * : 3,unary; + : 2,binary,left; ;; : 1,binary,left;
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 $\mathtt{t}$, 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 '$\leftarrow \neg\langle([\tau^*]\, b)?\ ;\ \tau\rangle\, a$' 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.

## 6.1 Tseitin-style translation into logic programs

The leftmost part of the workflow in Figure 4 relies on *telingo*'s infrastructure [6, 10]: 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`).[3] These translations heavily rely on the introduction of auxiliary variables for subformulas, a technique due to Tseitin [22]. 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 [10, 11].

## 6.2 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 3. 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

[3] 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`

'`* &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.

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

```
 1 prop(10,"b").
 2 prop(14,"a").
 3 prop(16,"last").
 4 state(0,dia(seq(test(box(str(stp),p(10))),stp),p(14))).
 5 state(1,p(14)).
 6 state(2,box(str(stp),p(10))).
 7 initial_state(0).
 8 delta(0,0). delta(0,0,out,16). delta(0,0,in,10).
 9             delta(0,0,1). delta(0,0,2).
10 delta(1,0). delta(1,0,in,14).
11 delta(2,0). delta(2,0,out,16). delta(2,0,in,10).
12             delta(2,0,2).
13 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`)

Figure 4, the facts in `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 to 3 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 $\varphi$ 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 4 to 5 correspond to the ones labeled $q_{\varphi}$, $q_a$ 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 `A` $\in X$ or `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 8 to 9 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 8. Line 8 also gives the conditions $last \notin X$ and $b \in X$ needed to reach the successor states given in Line 9. Line 10 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 11 to 12 represents the loop $\delta(q_{[\tau^*]b}, X) = \{\{q_{[\tau^*]b}\}\}$ for $last \notin X$ and $b \in X$, while transition `1` in Line 13 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)`,

```
1  node(Q,0) :- initial_state(Q).

3  { select(C,Q,T): delta(Q,C) } = 1 :- node(Q,T), T<=lambda-1.

5  node(Q',T+1) :- select(C,Q,T), delta(Q,C,Q').

7  :- select(C,Q,T), delta(Q,C,in,A), not trace(A,T).
8  :- select(C,Q,T), delta(Q,C,out,A), trace(A,T).
```

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

stating that the atom identified 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`[4]. The initial state is anchored as the root in Line 1. In turn, nodes get expanded by depth by selecting possible transitions in Line 3. The nodes are then put in place by following the transition of the selected conjunction in Line 5. Lines 7 and 8 verify the conditions for the selected transition.

## 6.3 MONA-based translation into deterministic automata

The rightmost part of the workflow in Figure 4 relies on our translations of dynamic formulas into MSOs in Section 5. We use the off-the-shelf tool MONA[5] [9] 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 `mso.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` (Listing 6) 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).
```

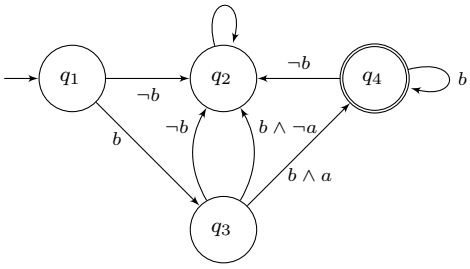


Figure 5: DFA automata computed by MONA for $\varphi$ (1).

```
1  prop(0,"a").
2  prop(1,"b").
3  prop(2,"last").
```

[4]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.

[5]`https://www.brics.dk/mona`

```
4  state(1,"q1").
5  state(2,"q2").
6  state(3,"q3").
7  state(4,"q4").
8  initial_state(1).
9  delta(1,0). delta(1,0,2). delta(1,0,out,1).
10 delta(1,1). delta(1,1,3). delta(1,1,in,1).
11 delta(2,0). delta(2,0,2).
12 delta(3,0). delta(3,0,2). delta(3,0,out,0).
13 delta(3,1). delta(3,1,2).
14             delta(3,1,in,0). delta(3,1,out,1).
15 delta(3,2). delta(3,2,4).
16             delta(3,2,in,0). delta(3,2,in,1).
17 delta(4,0). delta(4,0,2). delta(4,0,out,1).
18 delta(4,1). delta(4,1,4). delta(4,1,in,1).
19 final_state(4).
```

Listing 6: Facts representing the DFA in Figure 5 (`dfa.lp`)

# 7 Evaluation

For our experimental studies, we use benchmarks from the domain of robotic intra-logistics stemming from the *asprilo* framework [23]. As illustrated in Figure 6 and 7, 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.

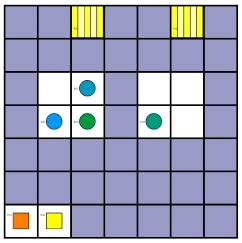

Figure 6: Asprilo visualization for two robots instance.

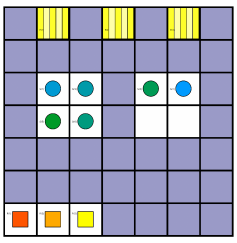

Figure 7: Asprilo visualization for three robots instance.

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 $\varphi_1$ and

represented in Listing 7[6], 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, $\varphi_2$, represents a procedure where robots must repeat a sequence

```
:- not &del{
      (* &t) .>*
      ?pickup(robot(R),shelf(S)) .>*
      *(&t ;; ?move(robot(R)) + ?waits(robot(R))) ;;
        ?deliver(robot(R),shelf(S)) .>?
      &true},
   robot(R), shelf(S).
```

Listing 7: Dynamic constraint for formula $\varphi_1$.

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; it is represented in Listing 8.

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

For our last constraint we use the dynamic formula $\varphi_3$ given in Listing 9. This

```
:- not &del{
           *( *(&t ;; ?move(robot(R))) ;;
              &t ;; ?pickup(robot(R)) ;;
              *(&t ;; ?move(robot(R))) ;;
              &t ;; ?deliver(robot(R));; ?waits(robot(R)) ;;
              *(&t ;; ?move(robot(R)));;
               &t ;; ?putdown(robot(R)))
                   ;; *(&t ;; ?waits(robot(R)))
        .>? &t.>* &false }, robot(R).
```

Listing 8: Dynamic constraint for formula $\varphi_2$.

corresponds to a procedure similar to $\varphi_2$ 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

[6]We start repetitions with $\tau$ as `&t`, to cope with movements in *asprilo* starting at time point 1.

```
:- not &del{
    *(  *(&t ;; ?waits(robot(R))) ;;
        (  *(&t ;; (?move(robot(R),RIGHT))) +
            *(&t ;; (?move(robot(R),LEFT)))
        ) ;;
        *(&t ;; ?move(robot(R),UP) ) ;;

        &t ;; ?pickup(robot(R)) ;;

        (  *(&t ;; (?move(robot(R),RIGHT))) +
            *(&t ;; (?move(robot(R),LEFT)))
        ) ;;
        *(&t ;; ?move(robot(R),UP) ) ;;

        &t ;; ?deliver(robot(R));; ?waits(robot(R)) ;;

        (  *(&t ;; (?move(robot(R),RIGHT))) +
            *(&t ;; (?move(robot(R),LEFT)))
        ) ;;
            *(&t ;; ?move(robot(R),DOWN) ) ;;

        &t ;; ?putdown(robot(R)))
    ;; *(&t ;; ?waits(robot(R)))
.>? &t.>* &false },
   robot(R), up(UP), right(RIGHT), left(LEFT), down(DOWN).
```

Listing 9: Dynamic constraint for formula $\varphi_3$.

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 3 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 [24]. More precisely, for $\varphi_3$ 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 4. 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 $\varphi_2$ the $\mathfrak{M}_m$ translation takes considerably longer than for $\mathfrak{M}_s$.

The results of the final solving step in each workflow are summarized in Table 5, 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 $\varphi_1$

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

| $\varphi_i$ | predicate | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ |
|---|---|---|---|---|
| $\varphi_1$ | state/2 | 36 | 72 | 72 |
| | delta/2 | 162 | 234 | 216 |
| $\varphi_2$ | state/2 | 24 | 51 | 51 |
| | delta/2 | 60 | 390 | 471 |
| $\varphi_3$ | state/2 | 45 | - | 372 |
| | delta/2 | 189 | - | 16 503 |

Table 4: 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.

| $\varphi_i$ | #r | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | NC |
|---|---|---|---|---|---|---|
| $\varphi_1$ | 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 |
| $\varphi_2$ | 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 |
| $\varphi_3$ | 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 5: Statistics computed by calculating the geometric mean of all horizons.

| | $\varphi_i$ | #r | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | $NC$ |
|---|---|---|---|---|---|---|---|
| **clingo time** | $\varphi_1$ | 2 | 3 374 | **2 788** | 2 975 | 3 033 | 21 823 |
| | | 3 | **23 173** | 27 866 | 27 505 | 23 748 | 249 737 |
| | $\varphi_2$ | 2 | 10 840 | 9 424 | 9 484 | **9 347** | 21 378 |
| | | 3 | 70 709 | **58 739** | 83 521 | 60 765 | 246 739 |
| | $\varphi_3$ | 2 | 31 986 | - | 606 914 | **16 145** | 21 548 |
| | | 3 | 67 287 | - | 657 633 | **48 190** | 247 718 |
| | | 3 | 274 851 | - | 2 743 736 | **264 847** | 241 752 |
| **rules** | $\varphi_1$ | 2 | **89 282** | 97 396 | 97 404 | 96 793 | 77 832 |
| | | 3 | **172 641** | 196 220 | 190 943 | 189 637 | 147 209 |
| | $\varphi_2$ | 2 | **84 180** | 122 003 | 126 634 | 90 178 | 77 832 |
| | | 3 | **157 525** | 214 454 | 229 063 | 166 391 | 147 209 |
| | $\varphi_3$ | 2 | **94 653** | - | 4 413 056 | 102 687 | 77 832 |
| | | 3 | **173 210** | - | 3 360 382 | 185 155 | 147 209 |
| **constraints** | $\varphi_1$ | 2 | 146 999 | 146 323 | 146 306 | **140 801** | 132 370 |
| | | 3 | 275 747 | 274 419 | 274 382 | **260 675** | 241 752 |
| | $\varphi_2$ | 2 | **138 418** | 166 449 | 171 796 | 139 909 | 132 370 |
| | | 3 | **252 023** | 295 946 | 308 204 | 254 020 | 241 752 |
| | $\varphi_3$ | 2 | 153 179 | - | 3 341 017 | **147 123** | 132 370 |

and $\varphi_2$ the difference is negligible, whereas for $\varphi_3$, $\mathfrak{T}$ 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 $\varphi_3$ 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 $\mathfrak{T}$.

# 8 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 [24, 25, 16, 26, 27]. Furthermore, the field of planning has benefited from temporal reasoning to express goals and preferences using an underlying automaton [28, 29, 30]. There exists several systems that translate temporal formulas into automata: SPOT [31] and LTLf2DFA[7] for linear temporal logic; *abstem* [32] and *stelp* [33] 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 [34, 35].

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 [9] 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 $\mathrm{DEL}_f$. 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.

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

# References


[1] V. Lifschitz. Answer set planning. In D. de Schreye, editor, *Proceedings of the International Conference on Logic Programming (ICLP'99)*, pages 23–37. MIT Press, 1999.

[2] A. Bosser, P. Cabalar, M. Diéguez, and T. Schaub. Introducing temporal stable models for linear dynamic logic. In M. Thielscher, F. Toni, and F. Wolter, editors, *Proceedings of the Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'18)*, pages 12–21. AAAI Press, 2018.

[3] P. Cabalar, M. Diéguez, and T. Schaub. Towards dynamic answer set programming over finite traces. In Balduccini et al. [36], pages 148–162.

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

[5] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, and C. Vidal. Temporal equilibrium logic: a survey. *Journal of Applied Non-Classical Logics*, 23(1-2):2–24, 2013.

[6] P. Cabalar, R. Kaminski, T. Schaub, and A. Schuhmann. Temporal answer set programming on finite traces. *Theory and Practice of Logic Programming*, 18(3-4):406–420, 2018.

[7] P. Cabalar, M. Diéguez, T. Schaub, and A. Schuhmann. Towards metric temporal answer set programming. *Theory and Practice of Logic Programming*, 20(5):783–798, 2020.

[8] R. Kaminski, T. Schaub, and P. Wanko. A tutorial on hybrid answer set solving with clingo. In G. Ianni, D. Lembo, L. Bertossi, W. Faber, B. Glimm, G. Gottlob, and S. Staab, editors, *Proceedings of the Thirteenth International Summer School of the Reasoning Web*, volume 10370 of *Lecture Notes in Computer Science*, pages 167–203. Springer-Verlag, 2017.

[9] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. volume 1019 of *Lecture Notes in Computer Science*, pages 89–110. Springer-Verlag, 1995.

[10] P. Cabalar, R. Kaminski, P. Morkisch, and T. Schaub. telingo = ASP + time. In Balduccini et al. [36], pages 256–269.

[11] P. Cabalar, M. Diéguez, F. Laferriere, and T. Schaub. Implementing dynamic answer set programming over finite traces. In G. De Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, editors, *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 656–663. IOS Press, 2020.

[12] D. Harel, J. Tiuryn, and D. Kozen. *Dynamic Logic*. MIT Press, 2000.

[13] G. De Giacomo and M. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In F. Rossi, editor, *Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 854–860. IJCAI/AAAI Press, 2013.

[14] D. Harel, D. Kozen, and J. Tiuryn. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 4, pages 99–107. Springer-Verlag, 2001.

[15] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[16] G. De Giacomo and M. Vardi. Synthesis for LTL and LDL on finite traces. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-fourth International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1558–1564. AAAI Press, 2015.

[17] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[18] T. Wolfgang. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. Springer, 1997.

[19] J. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California at Los Angeles, 1968.

[20] H. Jürgen Ohlbach, A. Nonnengart, M. de Rijke, and M. Gabbay. Encoding two-valued nonclassical logics in classical logic. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1403–1486. Elsevier and MIT Press, 2001.

[21] S. Zhu, G. Pu, and M. Vardi. First-order vs. second-order encodings for ltlf-to-automata translation. volume 11436 of *Lecture Notes in Computer Science*, pages 684–705. Springer-Verlag, 2019.

[22] G. Tseitin. On the complexity of derivation in the propositional calculus. *Zapiski nauchnykh seminarov LOMI*, 8:234–259, 1968.

[23] M. Gebser, P. Obermeier, T. Otto, T. Schaub, O. Sabuncu, V. Nguyen, and T. Son. Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming*, 18(3-4):502–519, 2018.

[24] M. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1995.

[25] M. Vardi. Alternating automata: Unifying truth and validity checking for temporal logics. In W. McCune, editor, *Proceedings of the Fourteenth International Conference on Automated Deduction (CADE'97)*, volume 1249 of *Lecture Notes in Computer Science*, pages 191–206. Springer-Verlag, 1997.

[26] K. Rozier and M. Vardi. Ltl satisfiability checking. In *International SPIN Workshop on Model Checking of Software*, pages 149–167. Springer-Verlag, 2007.

[27] A. Camacho and S. McIlraith. Learning interpretable models expressed in linear temporal logic. In J. Benton, N. Lipovetzky, E. Onaindia, D. Smith, and S. Srivastava, editors, *Proceedings of the Twenty-ninth International Conference on Automated Planning and Scheduling (ICAPS'19)*, pages 621–630. AAAI Press, 2019.

[28] J. Baier, C. Fritz, Me. Bienvenu, and S. McIlraith. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In D. Fox and C. Gomes, editors, *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*, pages 1509–1512. AAAI Press, 2008.

[29] G. De Giacomo and S. Rubin. Automata-theoretic foundations of fond planning for LTLf and LDLf goals. In J. Lang, editor, *Proceedings of the Twenty-seventh International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 4729–4735. ijcai.org, 2018.

[30] J. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. In Y. Gil and R. Mooney, editors, *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI'06)*, pages 788–795. AAAI Press, 2006.

[31] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 - A framework for LTL and $\omega$-automata manipulation. In C. Artho, A. Legay, and D. Peled, editors, *Proceedings of Fourteenth International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129, 2016.

[32] P. Cabalar and M. Diéguez. Strong equivalence of non-monotonic temporal theories. In C. Baral, G. De Giacomo, and T. Eiter, editors, *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*. AAAI Press, 2014.

[33] P. Cabalar and M. Diéguez. STELP — a tool for temporal answer set programming. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 370–375. Springer-Verlag, 2011.

[34] T. Son, C. Baral, T. Nam, and S. McIlraith. Domain-dependent knowledge in answer set planning. *ACM Transactions on Computational Logic*, 7(4):613–657, 2006.

[35] M. Ryan. Efficiently implementing GOLOG with answer set programming. In C. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth National Conference on Artificial Intelligence (AAAI'14)*, pages 2352–2357. AAAI Press, 2014.

[36] M. Balduccini, Y. Lierler, and S. Woltran, editors. *Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19)*, volume 11481 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2019.

# A Proofs

*Proof.* Theorem 1
By double induction on $\varphi$ and $\rho$.

- If $\varphi = p$, with $p$ a propositional variable, $ST_m(\mathbf{x}, p) = \mathbf{P}(\mathbf{x})$. If $\mathbf{T}, k \models p$ then $p \in T_k$. By construction of $v_2$, $k \in v_2(\mathbf{P})$ so $\mathbf{T}, v_1[\mathbf{x} := k], v_2 \models \mathbf{P}(\mathbf{x})$. Conversely, if $\mathbf{T}, v_1[\mathbf{x} := k], v_2 \models \mathbf{P}(\mathbf{x})$ then $v_1(\mathbf{x}) = k \in v_2(\mathbf{P})$. By the construction of $v_2$, $p \in T_k$ so $\mathbf{T}, k \models p$.

- The cases $\top$ and $\bot$ are straightforward.

- Negation, disjunction, conjunction and implication are proved directly by using the induction hypothesis.

- If $\varphi = [\rho]\,\psi$, from left to right, assume by contradiction that $\mathbf{T}, k \models \varphi$ but $\mathbf{T}, v_1[\mathbf{x} := k], v_2 \not\models ST_m(\mathbf{x}, \varphi)$. This means that $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho)$ and $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \not\models ST_m(\mathbf{y}, \psi)$ for some $0 \leq d < \lambda$. By induction hypothesis, $(k, d) \in\| \rho \|^{\mathbf{T}}$ and $\mathbf{T}, d \not\models \psi$: a contradiction. Conversely, assume that $\mathbf{T}, k \not\models [\rho]\,\psi$. This means that $(k, d) \in\| \rho \|^{\mathbf{T}}$ and $\mathbf{T}, d \not\models \psi$. By induction, $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho)$ and $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2, \not\models ST_m(\mathbf{y}, \psi)$. Therefore, $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \not\models (ST_p(\mathbf{xy}, \rho) \rightarrow ST_m(\mathbf{y}, \psi))$ so, $\mathbf{T}, v_1[\mathbf{x} := k], v_2 \not\models \forall \mathbf{y}\ (ST_p(\mathbf{xy}, \rho) \rightarrow ST_m(\mathbf{y}, \psi))$: a contradiction.

- If $\varphi = \langle\rho\rangle\,\psi$ then, from left to right, if $\mathbf{T}, k \models \langle\rho\rangle\,\varphi$, there exists $0 \leq d < \lambda$ such that $(k, d) \in\| \rho \|^{\mathbf{T}}$ and $\mathbf{T}, d \models \psi$. By induction, $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho)$ and $\mathbf{T}, v_1[\mathbf{y} := d], v_2 \models ST_m(\mathbf{y}, \psi)$. Therefore, $\mathbf{T}, v_1[\mathbf{x} := k], v_2 \models \exists \mathbf{y}\ (ST_p(\mathbf{xy}, \rho) \wedge ST_m(\mathbf{y}, \psi))$. Conversely, if $\mathbf{T}, v_1[\mathbf{x} := k], v_2 \models ST_m(\mathbf{x}, \varphi)$, then $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho)$ and $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_m(\mathbf{y}, \psi)$. By induction $(k, d) \in\| \rho \|^{\mathbf{T}}$ and $\mathbf{T}, d \models \psi$. Therefore, $\mathbf{T}, k \models \langle\rho\rangle\,\psi$.

Let us consider now the case of path formulas.

- If $\rho = \tau$, from left to right, if $(k, d) \in\| \tau \|^{\mathbf{T}}$ then $d = k+1$. By construction, $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models \mathbf{y} = \mathbf{x} + 1$. Conversely, if $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models \mathbf{y} = \mathbf{x} + 1$ then $d = k + 1$ holds. Consequently, $(k, d) \in\| \tau \|^{\mathbf{T}}$.

- If $\rho = \psi?$ then $ST_p(\mathbf{xy}, \rho) = (\mathbf{x} = \mathbf{y}) \wedge ST_m(\mathbf{y}, \psi)$. It holds that $(k, d) \in\| \rho \|^{\mathbf{T}}$ iff $k = d$ and $\mathbf{T}, d \models \psi$ iff $k = d$ and $\mathbf{T}, v_1[x := k, y := d], v_2, \models ST_m(y, \psi)$ (by induction) iff $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d] \models ST_p(\mathbf{xy}, \rho)$.

- If $\rho = \rho_1 + \rho_2$ then $ST_p(\mathbf{xy}\rho) = ST_p(\mathbf{xy}, \rho_1) \vee ST_p(\mathbf{xy}, \rho_2)$. It holds that $(k, d) \in\| \rho_1 + \rho_2 \|^{\mathbf{T}}$ iff either $(k, d) \in\| \rho_1 \|^{\mathbf{T}}$ or $(k, d) \in\| \rho_2 \|^{\mathbf{T}}$ iff $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho_1)$ or $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho_2)$ (by induction) iff $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy}, \rho)$.

- If $\rho = \rho_1;\rho_2$ then $ST_p(\mathbf{xy},\rho) = \exists \mathbf{u}(ST_p(\mathbf{xu},\rho_1) \wedge ST_p(\mathbf{uy},\rho_2))$. It holds that $(k,d) \in\!\| \rho_1;\rho_2 \|^{\mathbf{T}}$ iff there exists $d'$ such that $(w,d') \in\!\| \rho_1 \|^{\mathbf{T}}$ and $(d',v) \in\!\| \rho_2 \|^{\mathbf{T}}$ iff $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{u} := d'], v_2 \models ST_p(\mathbf{xu},\rho_1)$ and $\mathbf{T}, v_1[\mathbf{u} := d', \mathbf{y} := d], v_2 \models ST_p(\mathbf{uy},\rho_2)$ (by induction) iff $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d] \models ST_p(\mathbf{xy},\rho)$.

- If $\rho = \rho^*$ then, from left to right, we will prove that for all $n \geq 0$ and for all $(k,d) \in \mathbb{N}\times\mathbb{N}$ if $(k,d) \in\!\| \rho^n \|^{\mathbf{T}}$ then $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy},\rho^*)$ by induction on $n$

  - If $n = 0$ then $k = d$. Let $v_2' = v_2[\mathbf{X} := \{k\}]$. Clearly, $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \models \mathbf{X}(\mathbf{x})$ and $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \models \mathtt{bound}(\mathbf{X},\mathbf{x},\mathbf{x})$ and $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \models \mathtt{regular}(\mathbf{X})$ since $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \not\models \mathtt{succ}(\mathbf{x},\mathbf{x})$. Thanks to the second-order semantics we conclude $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy},\rho^*)$.
  - Assume that the claim holds for all $n \geq 0$ and let us prove it for $n{+}1$. If $(k,d) \in\!\| \rho^{n+1} \|^{\mathbf{T}}$ then, by definition, $(k,u) \in\!\| \rho \|^{\mathbf{T}}$ and $(u,d) \in\!\| \rho^n \|^{\mathbf{T}}$ for some $0 \leq u < \lambda$. By Proposition 1, $0 \leq k \leq u \leq d < \lambda$. By induction on $\rho$ we get $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := u], v_2 \models ST_p(\mathbf{xy},\rho)$. By induction on $n$ we get $\mathbf{T}, v_1[\mathbf{x} := u, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy},\rho^*)$. From the previous result it follows $\mathbf{T}, v_1[\mathbf{x} := u, \mathbf{y} := d], v_2' \models \mathbf{X}(\mathbf{x}) \wedge \mathbf{X}(\mathbf{y}) \wedge \mathtt{bound}(\mathbf{X},\mathbf{x},\mathbf{y}) \wedge \mathtt{regular}(\mathbf{X})$, where $v_2'$ is an extension of $v_2$ for which $v_2'(\mathbf{X})$ is defined.

    Let $v_2''$ be an extension of $v_2$ such that $v_2''(\mathbf{X}) \stackrel{def}{=} v_2'(\mathbf{X}) \cup \{k\}$ and $v_2'' = v_2$ for the rest of second-order variables. Notice that $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2'' \models \mathbf{X}(\mathbf{x})$ and $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2'' \models \mathbf{X}(\mathbf{y})$. From $k \leq u$ and the definition of $v_2''(\mathbf{X})$, we get that $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2'' \models \mathtt{bound}(\mathbf{X},\mathbf{x},\mathbf{y})$. To prove that $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2'' \models \mathtt{regular}(\mathbf{X})$, let us take $d_1, d_2 \in v_2''(\mathbf{X})$. We consider three cases:

    * If $d_1, d_2 \in v_2'(\mathbf{X})$ we use the fact that $\mathbf{T}, v_1[\mathbf{x} := u, y := d], v_2' \models \mathtt{regular}(\mathbf{X})$ to conclude that $\mathbf{T}, v_1[\mathbf{x} := u, \mathbf{y} := d, \mathbf{a} := d_1, \mathbf{b} := d_2], v_2'' \models (\mathtt{succ}(\mathbf{a},\mathbf{b}) \wedge \mathbf{X}(\mathbf{a}) \wedge \mathbf{X}(\mathbf{b})) \rightarrow ST_p(\mathbf{ab},\rho))$
    * $d_1 = d_2 = k$ then $\mathbf{T}, v_1[\mathbf{x} := u, \mathbf{y} := d, \mathbf{a} := d_1, \mathbf{b} := d_2], v_2'' \models (\mathtt{succ}(\mathbf{a},\mathbf{b}) \wedge \mathbf{X}(\mathbf{a}) \wedge \mathbf{X}(\mathbf{b})) \rightarrow ST_p(\mathbf{ab},\rho))$ since $\mathbf{T}, v_1[\mathbf{x} := u, \mathbf{y} := d, \mathbf{a} := d_1, \mathbf{b} := d_2], v_2'' \not\models (\mathtt{succ}(\mathbf{a},\mathbf{b})$.
    * $d_1 = k$ and $d_2 \neq w$. Necessarily, $d_1 = u$. In this case, $\mathbf{T}, v_1[\mathbf{x} := u, \mathbf{y} := d, \mathbf{a} := d_1, \mathbf{b} := d_2], v_2'' \models (\mathtt{succ}(\mathbf{a},\mathbf{b}) \wedge \mathbf{X}(\mathbf{a}) \wedge \mathbf{X}(\mathbf{b})) \rightarrow ST_p(\mathbf{ab},\rho))$ because $\mathbf{T}, v_1[\mathbf{x} := u, \mathbf{y} := d, \mathbf{a} := d_1, \mathbf{b} := d_2], v_2'' \models ST_p(\mathbf{ab},\rho)$.

    Thus, we conclude $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2'' \models \mathtt{regular}(\mathbf{X})$.

  for the converse direction, if $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2 \models ST_p(\mathbf{xy},\rho^*)$ then there exists an assignment $v_2'$ that extends $v_2$ with an assignment for the second-order variable $\mathbf{X}$. By definition, it holds that

  1. $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \models \mathbf{X}(\mathbf{x})$

2. $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \models \mathbf{X}(\mathbf{y})$
3. $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \models \mathtt{bound}(\mathbf{X}, \mathbf{x}, \mathbf{y})$
4. $\mathbf{T}, v_1[\mathbf{x} := k, \mathbf{y} := d], v_2' \models \mathtt{regular}(\mathbf{X})$

From all those items we get that there exists $u_0, u_1, \cdots, u_n$ in $v_2(\mathbf{X})$ such that $u_0 = k$, $u_n = d$ and for all $0 \leq i < n$, $(u_i, u_{i+1}) \in \| \rho \|^{\mathbf{T}}$. By using the definition of $\| \rho^* \|^{\mathbf{T}}$, we conclude that $(k, d) \in \| \rho^* \|^{\mathbf{T}}$.

□

*Proof.* Theorem 2

If $\varphi$ is propositional atom $p$ then $\mathrm{mso}(\mathbf{t}, p) = P(\mathbf{t})$. It is true that $\mathbf{T}, k \models \varphi$ iff $\mathbf{T}, v_1[\mathbf{t} := k], v_2 \models \mathrm{mso}(\mathbf{t}, p)$. If $\varphi$ is a nonatomic formula, we prove this theorem in two directions.

Suppose first that $\mathbf{T}, k \models \varphi$. Let $v_2'$ be a second-order assignment such that $v_2'(\mathbf{Q}_{\theta_\mathbf{i}}) = \{\{x \mid \mathbf{T}, x \models \Theta_i\}\}$ for each $\Theta_i \in cl(\varphi)$ and $v_2'(\mathbf{P}) = v_2(\mathbf{P})$ otherwise. By assumption, $\mathbf{T}, v_1[\mathbf{t} := k], v_2' \models \mathbf{Q}_\varphi(\mathbf{t})$. It remains to prove that $\mathbf{T}, v_1[\mathbf{t} := k], v_2' \models \forall \mathbf{x}.\, t(\Theta_i, \mathbf{x})$ for each nonatomic subformula $\Theta_i \in cl(\varphi)$, which we prove by induction over $\Theta_i$

- If $\Theta_i = \neg\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \big(\mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \neg\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x})\big)$. This holds, since $v_2'(\mathbf{Q}_{\mathbf{\neg\Theta_j}}) = \{x \mid \mathbf{T}, x \not\models \Theta_j\}$ and $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$.
- If $\Theta_i = \Theta_j \wedge \Theta_k$ , then $t(\Theta_i, \mathbf{x}) = \big(\mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \wedge \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x})\big)\big)$. This holds since $v_2'(\mathbf{Q}_{\mathbf{\Theta_j \wedge \Theta_k}}) = \{x \mid \mathbf{T}, x \models \Theta_j \text{ and } \mathbf{T}, x \models \Theta_k\}$, $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$ and $v_2'(\mathbf{Q}_{\mathbf{\Theta_k}}) = \{x \mid \mathbf{T}, x \models \Theta_k\}$.
- If $\Theta_i = \Theta_j \vee \Theta_k$ , then $t(\Theta_i, \mathbf{x}) = \big(\mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \vee \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x})\big)\big)$. This holds since $v_2'(\mathbf{Q}_{\mathbf{\Theta_j \vee \Theta_k}}) = \{x \mid \mathbf{T}, x \models \Theta_j \text{ or } \mathbf{T}, x \models \Theta_k\}$, $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$ and $v_2'(\mathbf{Q}_{\mathbf{\Theta_k}}) = \{x \mid \mathbf{T}, x \models \Theta_k\}$.
- If $\Theta_i = \Theta_j \rightarrow \Theta_k$ , then $t(\Theta_i, \mathbf{x}) = \big(\mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \rightarrow \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x})\big)\big)$. This holds since $v_2'(\mathbf{Q}_{\mathbf{\Theta_j \rightarrow \Theta_k}}) = \{x \mid \mathbf{T}, x \not\models \Theta_j \text{ or } \mathbf{T}, x \models \Theta_k\}$, $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$ and $v_2'(\mathbf{Q}_{\mathbf{\Theta_k}}) = \{x \mid \mathbf{T}, x \models \Theta_k\}$.
- If $\Theta_i = [\Theta_j?]\,\Theta_k$ we proceed as in the case of implication.
- If $\Theta_i = \langle\Theta_j?\rangle\,\Theta_k$ we proceed as in the case of conjunction.
- If $\Theta_i = [\tau]\,\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\forall \mathbf{y}\ (\mathbf{y} = \mathbf{x} + 1) \rightarrow \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{y})\big)$. This holds since

$$\begin{aligned} v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) &= \{x \mid \mathbf{T}, x \models [\tau]\,\Theta_k\} \\ &= \{x \mid \text{ for all } (x, y) \text{ if } (x, y) \in \| \tau \|^{\mathbf{T}} \text{ then } \mathbf{T}, y \models \Theta_j\} \\ &= \{x \mid \text{ for all y, if} y = x + 1 \text{ then } \mathbf{T}, y \models \Theta_j\}. \end{aligned}$$

and $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$.

- If $\Theta_i = \langle\tau\rangle\,\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\exists \mathbf{y}\ (\mathbf{y} = \mathbf{x} + 1) \rightarrow \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{y})\big)$. This holds because

$$\begin{aligned} v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) &= \{x \mid \mathbf{T}, x \models \langle\tau\rangle\,\Theta_j\} \\ &= \{x \mid \text{ there exits } (x,y) \in \|\,\tau\,\|^{\mathbf{T}} \text{ such that } \mathbf{T}, y \models \Theta_j\} \\ &= \{x \mid \text{ there exits } y = x+1 \text{ such that } \mathbf{T}, y \models \Theta_j\} \end{aligned}$$

  and $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$.

- If $\Theta_i = [\rho_1;\rho_2]\,\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{[\rho_1]\,[\rho_2]\,\mathbf{\Theta_j}}(\mathbf{x})\big)$. This holds because $v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) = \{x \mid \mathbf{T}, x \models [\rho_1;\rho_2]\,\Theta_j\} = \{x \mid \mathbf{T}, x \models [\rho_1]\,[\rho_2]\,\Theta_j\}$ by Proposition 2 (item 3) and $v_2'(\mathbf{Q}_{[\rho_1]\,[\rho_2]\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models [\rho_1]\,[\rho_2]\,\Theta_j\}$.

- If $\Theta_i = \langle\rho_1;\rho_2\rangle\,\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \mathbf{Q}_{\mathbf{\Theta_i}}(x) \leftrightarrow \big(\mathbf{Q}_{\langle\rho_1\rangle\,\langle\rho_2\rangle\,\mathbf{\Theta_j}}(\mathbf{x})\big)$. This holds because $v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) = \{x \mid \mathbf{T}, x \models \langle\rho_1;\rho_2\rangle\,\Theta_j\} = \{x \mid \mathbf{T}, x \models \langle\rho_1\rangle\,\langle\rho_2\rangle\,\Theta_j\}$ thanks to Proposition 2 (item 4) and $v_2'(\mathbf{Q}_{\langle\rho_1\rangle\,\langle\rho_2\rangle\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \langle\rho_1\rangle\,\langle\rho_2\rangle\,\Theta_j\}$.

- If $\Theta_i = [\rho_1 + \rho_2]\,\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{[\rho_1]\,\mathbf{\Theta_j}}(\mathbf{x}) \wedge \mathbf{Q}_{[\rho_2]\,\mathbf{\Theta_j}}(\mathbf{x})\big)$. This holds because $v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) = \{x \mid \mathbf{T}, x \models [\rho_1 + \rho_2]\,\Theta_j\} = \{x \mid \mathbf{T}, x \models [\rho_1]\,\Theta_j \wedge [\rho_2]\,\Theta_j\}$ by Proposition 2 (item 1), $v_2'(\mathbf{Q}_{[\rho_1]\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models [\rho_1]\,\Theta_j\}$ and $v_2'(\mathbf{Q}_{[\rho_2]\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models [\rho_2]\,\Theta_j\}$.

- If $\Theta_i = \langle\rho_1 + \rho_2\rangle\,\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\langle\rho_1\rangle\,\mathbf{\Theta_j}}(\mathbf{x}) \vee \mathbf{Q}_{\langle\rho_2\rangle\,\mathbf{\Theta_j}}(\mathbf{x})\big)$. This holds because $v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) = \{x \mid \mathbf{T}, x \models \langle\rho_1 + \rho_2\rangle\,\Theta_j\} = \{x \mid \mathbf{T}, x \models \langle\rho_1\rangle\,\Theta_j \vee \langle\rho_2\rangle\,\Theta_j\}$ by Proposition 2 (item 2), $v_2'(\mathbf{Q}_{\langle\rho_1\rangle\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \langle\rho_1\rangle\,\Theta_j\}$ and $v_2'(\mathbf{Q}_{\langle\rho_2\rangle\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \langle\rho_2\rangle\,\Theta_j\}$.

- If $\Theta_i = [\rho^*]\,\Theta_j$, then $t(\Theta_i, \mathbf{x}) = \mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \wedge \mathbf{Q}_{[\rho]\,[\rho^*]\,\mathbf{\Theta_j}}(\mathbf{x})\big)$. This holds since $v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) = \{x \mid \mathbf{T}, x \models [\rho^*]\,\Theta_j\} = \{x \mid \mathbf{T}, x \models \Theta_j \wedge [\rho]\,[\rho^*]\,\Theta_j\}$ by Proposition 2 (item 5), $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$ and $v_2'(\mathbf{Q}_{[\rho]\,[\rho^*]\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models [\rho]\,[\rho^*]\,\Theta_j\}$.

- If $\Theta_i = \langle\rho^*\rangle\,\Theta_j$, then $t(\Theta_i, x) = \mathbf{Q}_{\mathbf{\Theta_i}}(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \vee \mathbf{Q}_{\langle\rho\rangle\,\langle\rho^*\rangle\,\mathbf{\Theta_j}}(\mathbf{x})\big)$. This holds since $v_2'(\mathbf{Q}_{\mathbf{\Theta_i}}) = \{x \mid \mathbf{T}, x \models \langle\rho^*\rangle\,\Theta_j\} = \{x \mid \mathbf{T}, x \models \Theta_j \vee \langle\rho\rangle\,\langle\rho^*\rangle\,\Theta_j\}$ by Proposition 2 (item 6), $v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \Theta_j\}$ and $v_2'(\mathbf{Q}_{\langle\rho\rangle\,\langle\rho^*\rangle\,\mathbf{\Theta_j}}) = \{x \mid \mathbf{T}, x \models \langle\rho\rangle\,\langle\rho^*\rangle\,\Theta_j\}$.

Assume now that $\mathbf{T}, v_1[\mathbf{t} := k], v_2 \models \mathrm{mso}(\mathbf{t}, \varphi)$. This means that there is an assignment $v_2'$ that extends $v_2$ by defining $v_2'(\mathbf{Q}_{\theta_i})$ for each predicate $\mathbf{Q}_{\theta_i}$ with $\theta_i$ being a non-atomic formula in $cl(\varphi)$ and satisfying $\mathbf{T}, v_1[\mathbf{t} := k], v_2' \models \mathbf{Q}_{\varphi}(\mathbf{t}) \wedge (\forall \mathbf{x}(\wedge_{i=0}^{m} t(\theta_i, \mathbf{x}))$. We now prove by induction on $\varphi$ that if $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models \mathbf{Q}_{\varphi}(\mathbf{x})$ then $\mathbf{T}, d \models \varphi$ for all $0 \leq d < \lambda$ so $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := k], v_2' \models \mathbf{Q}_{\varphi}(\mathbf{x})$ indicates that $\mathbf{T}, k \models \varphi$.

- If $\varphi = \neg\Theta_j$, then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_{\varphi}(\mathbf{x}) \leftrightarrow \neg\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x})\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it holds that $d \in v_2'(\mathbf{Q}_{\varphi})$ iff $d \notin v_2'(\mathbf{Q}_{\mathbf{\Theta_j}})$. By induction we get $\mathbf{T}, d \not\models \Theta_j$. Thus, $\mathbf{T}, d \models \varphi$.

- If $\varphi = \Theta_j \wedge \Theta_k$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \wedge \mathbf{Q}_{\mathbf{\Theta_k}}(\mathbf{x})\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{\mathbf{\Theta_k}}) \cap v_2'(\mathbf{Q}_{\mathbf{\Theta_j}})$. By induction $\mathbf{T}, d \models \Theta_j$ and $\mathbf{T}, d \models \Theta_k$.

- If $\varphi = \Theta_j \vee \Theta_k$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \vee \mathbf{Q}_{\mathbf{\Theta_k}}(\mathbf{x})\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{\mathbf{\Theta_k}}) \cup v_2'(\mathbf{Q}_{\mathbf{\Theta_j}})$. By induction $\mathbf{T}, d \models \Theta_j$ or $\mathbf{T}, d \models \Theta_k$.

- If $\varphi = \Theta_j \to \Theta_k$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \to \mathbf{Q}_{\mathbf{\Theta_k}}(\mathbf{x})\big)\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $v_1(\mathbf{x}) \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2(\mathbf{Q}_{\mathbf{\Theta_k}}) \cup v_2'(\mathbf{Q}_{\mathbf{\Theta_j}})$. By induction $\mathbf{T}, d \not\models \Theta_j$ and $\mathbf{T}, d \models \Theta_k$ meaning that $\mathbf{T}, d \models \Theta_j \to \Theta_k$.

- If $\varphi = [\Theta_k?]\, \Theta_j$ we proceed as for implication.

- If $\varphi = \langle \Theta_k? \rangle\, \Theta_j$ we proceed as for conjunction.

- If $\varphi = [\tau]\, \Theta_j$ then $t(\varphi, \mathbf{x}) = \big(Q_\varphi(x) \leftrightarrow \big(\forall \mathbf{y}.; \mathbf{y} = \mathbf{x} + 1 \to \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{y})\big)\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff for all $\mathbf{y}$, $\mathbf{y} = \mathbf{d} + 1$ implies $\mathbf{y} \in v_2'(\mathbf{Q}_{\mathbf{\Theta_j}})$. By induction it follows that either $d + 1 = \lambda$ or $\mathbf{T}, d + 1 \models \Theta_j$ so $\mathbf{T}, d \models \varphi$.

- If $\varphi = \langle \tau \rangle\, \Theta_j$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \big(\exists \mathbf{y}\ \big(\mathbf{y} = \mathbf{x} + 1 \wedge \mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{y})\big)\big)\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff there exists $\mathbf{y} = d + 1$ and $\mathbf{y} \in v_2'(\mathbf{Q}_{\mathbf{\Theta_j}})$. By induction it follows that $d + 1 < \lambda$ and $\mathbf{T}, d + 1 \models \mathbf{\Theta_j}$ so $\mathbf{T}, d \models \varphi$.

- If $\varphi = [\rho_1; \rho_2]\, \Theta_j$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \mathbf{Q}_{[\rho_1]\,[\rho_2]\,\mathbf{\Theta_j}}(\mathbf{x})\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{[\rho_1]\,[\rho_2]\,\mathbf{\Theta_j}})$. By induction it follows that there $\mathbf{T}, d \models [\rho_1]\, [\rho_2]\, \Theta_j$ and, by Proposition 2 (item 3) so $\mathbf{T}, d \models \varphi$.

- If $\varphi = \langle \rho_1; \rho_2 \rangle\, \Theta_j$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \mathbf{Q}_{\langle \rho_1 \rangle\, \langle \rho_2 \rangle\, \mathbf{\Theta_j}}(\mathbf{x})\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{\langle \rho_1 \rangle\, \langle \rho_2 \rangle\, \mathbf{\Theta_j}})$. By induction it follows that there $\mathbf{T}, d \models \langle \rho_1 \rangle\, \langle \rho_2 \rangle\, \Theta_j$ and, by Proposition 2 (item 4) so $\mathbf{T}, d \models \varphi$.

- If $\varphi = [\rho_1 + \rho_2]\, \Theta_j$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{[\rho_1]\, \mathbf{\Theta_j}}(\mathbf{x}) \wedge \mathbf{Q}_{[\rho_2]\, \mathbf{\Theta_j}}(\mathbf{x})\big)\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{[\rho_1]\, \mathbf{\Theta_j}}) \cap v_2'(\mathbf{Q}_{[\rho_2]\, \mathbf{\Theta_j}})$. By induction it follows that there $\mathbf{T}, d \models [\rho_1]\, \Theta_j \wedge [\rho_2]\, \Theta_j$ and, by Proposition 2 (item 1) so $\mathbf{T}, d \models \varphi$.

- If $\varphi = \langle \rho_1 + \rho_2 \rangle\, \Theta_j$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\langle \rho_1 \rangle\, \mathbf{\Theta_j}}(\mathbf{x}) \vee \mathbf{Q}_{\langle \rho_2 \rangle\, \mathbf{\Theta_j}}(\mathbf{x})\big)\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{\langle \rho_1 \rangle\, \mathbf{\Theta_j}}) \cup v_2'(\mathbf{Q}_{\langle \rho_2 \rangle\, \mathbf{\Theta_j}})$. By induction, it follows that there $\mathbf{T}, d \models \langle \rho_1 \rangle\, \Theta_j \vee \langle \rho_2 \rangle\, \Theta_j$ and, by Proposition 2 (item 2) so $\mathbf{T}, d \models \varphi$.

- If $\varphi = [\rho^*]\,\Theta_j$ then $t(\varphi, \mathbf{x}) = \big(\mathbf{Q}_\varphi(\mathbf{x}) \leftrightarrow \big(\mathbf{Q}_{\mathbf{\Theta_j}}(\mathbf{x}) \wedge \mathbf{Q}_{[\rho]\,[\rho^*]\,\mathbf{\Theta_j}}(\mathbf{x})\big)\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) \cap v_2'(\mathbf{Q}_{[\rho]\,[\rho^*]\,\mathbf{\Theta_j}})$. By induction it follows that there $\mathbf{T}, d \models \Theta_j \wedge [\rho]\,[\rho^*]\,\Theta_j$. By Proposition 2 (item 5) so $\mathbf{T}, d \models \varphi$.
- If $\varphi = \langle\rho^*\rangle\,\Theta_j$ then $t(\varphi, x) = \big(Q_\varphi(x) \leftrightarrow \big(Q_{\Theta_j}(x) \vee Q_{\langle\rho\rangle\,\langle\rho^*\rangle\,\Theta_j}(x)\big)\big)$. Since $\mathbf{T}, v_1[\mathbf{t} := k, \mathbf{x} := d], v_2' \models t(\varphi, \mathbf{x})$ for all $0 \leq d < \lambda$, it follows that $d \in v_2'(\mathbf{Q}_\varphi)$ iff $d \in v_2'(\mathbf{Q}_{\mathbf{\Theta_j}}) \cup v_2'(\mathbf{Q}_{[\rho]\,[\rho^*]\,\mathbf{\Theta_j}})$. By induction it follows that there $\mathbf{T}, d \models \Theta_j \vee \langle\rho\rangle\,\langle\rho^*\rangle\,\Theta_j$. By Proposition 2 (item 6) so $\mathbf{T}, d \models \varphi$.

□

# B Detailed results tables

In the following tables, lambdas appear crossed out when the instance was UNSAT with the corresponding constraint. The results are for finding the first model and the best performance excluding NC is found in bold.

Table 6: Statistics for constraint $\varphi_1$ and the 2 robots instance.

| | $\lambda$ | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | $NC$ |
|---|---|---|---|---|---|---|
| **translation time** | ~~25~~ | **1 194** | 5 412 | 5 867 | 2 696 | 305 |
| | ~~26~~ | 557 | **451** | 725 | 3 407 | 397 |
| | 27 | 837 | 914 | **638** | 2 683 | 558 |
| | 28 | **589** | 648 | 617 | 3 218 | 454 |
| | 29 | 508 | 670 | **448** | 2 873 | 799 |
| | 30 | 663 | **445** | 472 | 2 197 | 768 |
| | 31 | **672** | 701 | 724 | 3 574 | 614 |
| **clingo time** | ~~25~~ | 3 395 | **2 387** | 2 775 | 2 657 | 10 104 |
| | ~~26~~ | 3 900 | 3 732 | **3 016** | 3 375 | 14 131 |
| | 27 | 4 497 | 2 335 | **2 128** | 3 769 | 31 167 |
| | 28 | **2 153** | 3 779 | 3 509 | 2 434 | 29 118 |
| | 29 | 3 032 | 2 826 | **2 796** | 3 280 | 25 184 |
| | 30 | 2 700 | **2 183** | 2 332 | 3 730 | 24 176 |
| | 31 | 4 744 | 2 700 | 5 059 | **2 348** | 29 871 |
| **choices** | ~~25~~ | 15 197 | 14 196 | 16 509 | **11 267** | 51 711 |
| | ~~26~~ | 24 236 | 21 258 | 17 956 | **14 982** | 91 708 |
| | 27 | 27 412 | **15 038** | 15 995 | 19 005 | 120 714 |
| | 28 | 17 473 | 23 631 | 41 320 | **12 952** | 124 386 |
| | 29 | 25 394 | 20 823 | 20 373 | **17 310** | 121 999 |
| | 30 | 23 611 | **16 924** | 18 829 | 21 436 | 125 925 |
| | 31 | 37 079 | **20 815** | 36 333 | 21 725 | 150 685 |
| **conflicts** | ~~25~~ | 5 916 | 5 927 | 6 933 | **4 986** | 31 833 |
| | ~~26~~ | 9 397 | 9 302 | 7 627 | **6 551** | 40 969 |
| | 27 | 10 591 | **5 608** | 5 710 | 7 832 | 80 625 |
| | 28 | 5 255 | 9 671 | 8 523 | **4 241** | 77 873 |
| | 29 | 7 281 | 7 353 | 6 550 | **6 029** | 71 170 |
| | 30 | 6 391 | **5 065** | 5 577 | 7 012 | 71 750 |
| | 31 | 11 411 | 6 690 | 13 001 | **3 443** | 84 896 |
| **rules** | ~~25~~ | **77 980** | 85 216 | 85 224 | 84 688 | 67 749 |
| | ~~26~~ | **81 860** | 89 396 | 89 404 | 88 842 | 71 213 |
| | 27 | **85 740** | 93 576 | 93 584 | 92 996 | 74 677 |
| | 28 | **89 620** | 97 756 | 97 764 | 97 150 | 78 141 |
| | 29 | **93 500** | 101 936 | 101 944 | 101 304 | 81 605 |
| | 30 | **97 380** | 106 116 | 106 124 | 105 458 | 85 069 |
| | 31 | **101 260** | 110 296 | 110 304 | 109 612 | 88 533 |
| **constraints** | ~~25~~ | 125 800 | 124 948 | 124 932 | **120 596** | 113 198 |
| | ~~26~~ | 133 109 | 132 321 | 132 305 | **127 561** | 119 809 |
| | 27 | 140 418 | 139 694 | 139 678 | **134 526** | 126 420 |
| | 28 | 147 727 | 147 067 | 147 051 | **141 491** | 133 031 |
| | 29 | 155 036 | 154 440 | 154 424 | **148 456** | 139 642 |
| | 30 | 162 345 | 161 813 | 161 797 | **155 421** | 146 253 |
| | 31 | 169 654 | 169 186 | 169 170 | **162 386** | 152 864 |

Table 7: Statistics for constraint $\varphi_1$ and the 3 robots instance.

| | $\lambda$ | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | $NC$ |
|---|---|---|---|---|---|---|
| **translation time** | ~~25~~ | **1 991** | 6 280 | 6 978 | 3 390 | 301 |
| | ~~26~~ | **474** | 477 | 689 | 4 123 | 477 |
| | ~~27~~ | 670 | 937 | **632** | 3 316 | 611 |
| | ~~28~~ | **633** | 634 | 637 | 3 578 | 388 |
| | ~~29~~ | 574 | 712 | **476** | 3 240 | 751 |
| | 30 | 628 | 501 | **490** | 3 601 | 842 |
| | 31 | **623** | 767 | 738 | 4 289 | 629 |
| **clingo time** | ~~25~~ | 12 069 | 11 375 | 11 196 | **10 687** | 49 097 |
| | ~~26~~ | **12 487** | 15 689 | 13 500 | 13 588 | 77 250 |
| | ~~27~~ | **16 865** | 26 711 | 20 568 | 17 957 | 193 067 |
| | ~~28~~ | 38 528 | **28 985** | 29 708 | 38 543 | 530 237 |
| | ~~29~~ | **49 117** | 52 888 | 60 487 | 49 148 | 796 953 |
| | 30 | 36 833 | 62 928 | 55 439 | **36 765** | 508 369 |
| | 31 | **20 253** | 28 378 | 38 456 | 23 454 | 385 140 |
| **choices** | ~~25~~ | 59 635 | 49 477 | 117 691 | **45 287** | 173 746 |
| | ~~26~~ | 64 653 | 64 926 | 62 331 | **59 828** | 245 978 |
| | ~~27~~ | 91 121 | 239 786 | 84 908 | **71 137** | 417 294 |
| | ~~28~~ | 311 375 | 252 341 | **98 109** | 237 270 | 912 562 |
| | ~~29~~ | 144 001 | 350 208 | 151 183 | **132 747** | 2 561 235 |
| | 30 | **145 468** | 464 029 | 156 648 | 285 323 | 3 058 806 |
| | 31 | 381 453 | **130 569** | 425 412 | 264 940 | 2 508 802 |
| **conflicts** | ~~25~~ | 21 663 | 19 473 | 21 694 | **18 113** | 106 940 |
| | ~~26~~ | **21 750** | 26 160 | 24 368 | 24 112 | 154 251 |
| | ~~27~~ | **28 000** | 46 206 | 34 613 | 29 302 | 284 788 |
| | ~~28~~ | 55 972 | 48 958 | **41 437** | 55 795 | 670 627 |
| | ~~29~~ | **64 615** | 76 098 | 70 879 | 65 578 | 956 430 |
| | 30 | 55 678 | 92 900 | 67 725 | **54 496** | 753 632 |
| | 31 | **35 980** | 46 615 | 58 403 | 38 162 | 576 474 |
| **rules** | ~~25~~ | **151 147** | 172 162 | 167 464 | 166 327 | 128 422 |
| | ~~26~~ | **158 522** | 180 413 | 175 517 | 174 322 | 134 873 |
| | ~~27~~ | **165 897** | 188 664 | 183 570 | 182 317 | 141 324 |
| | ~~28~~ | **173 272** | 196 915 | 191 623 | 190 312 | 147 775 |
| | ~~29~~ | **180 647** | 205 166 | 199 676 | 198 307 | 154 226 |
| | 30 | **188 022** | 213 417 | 207 729 | 206 302 | 160 677 |
| | 31 | **195 397** | 221 668 | 215 782 | 214 297 | 167 128 |
| **constraints** | ~~25~~ | 236 285 | 234 560 | 234 524 | **223 456** | 206 819 |
| | ~~26~~ | 249 887 | 248 306 | 248 270 | **236 284** | 218 864 |
| | ~~27~~ | 263 489 | 262 052 | 262 016 | **249 112** | 230 909 |
| | ~~28~~ | 277 091 | 275 798 | 275 762 | **261 940** | 242 954 |
| | ~~29~~ | 290 693 | 289 544 | 289 508 | **274 768** | 254 999 |
| | 30 | 304 295 | 303 290 | 303 254 | **287 596** | 267 044 |
| | 31 | 317 897 | 317 036 | 317 000 | **300 424** | 279 089 |

Table 8: Statistics for constraint $\varphi_2$ and the 2 robots instance.

| | $\lambda$ | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | $NC$ |
|---|---|---|---|---|---|---|
| **translation time** | ~~25~~ | 2 182 | 33 091 | 4 966 | **2 107** | 285 |
| | ~~26~~ | 529 | **522** | 670 | 3 006 | 444 |
| | 27 | **548** | 904 | 624 | 2 150 | 560 |
| | 28 | **592** | 620 | 633 | 2 372 | 377 |
| | 29 | 504 | 719 | **478** | 2 533 | 729 |
| | 30 | 653 | 488 | **476** | 3 254 | 807 |
| | 31 | **650** | 714 | 711 | 3 572 | 577 |
| **clingo time** | ~~25~~ | 5 815 | **5 712** | 6 921 | 7 081 | 9 845 |
| | ~~26~~ | 8 918 | 10 307 | 12 229 | **7 465** | 13 729 |
| | 27 | 13 902 | 9 206 | **8 105** | 11 135 | 30 361 |
| | 28 | 11 543 | 11 859 | 9 095 | **6 658** | 28 158 |
| | 29 | 10 057 | **7 346** | 10 258 | 7 486 | 25 309 |
| | 30 | 13 708 | **10 204** | 10 993 | 16 829 | 23 784 |
| | 31 | 15 338 | 13 696 | **9 805** | 12 628 | 29 339 |
| **choices** | ~~25~~ | **33 023** | 39 073 | 42 068 | 33 878 | 51 711 |
| | ~~26~~ | 45 972 | 50 647 | 62 204 | **39 592** | 91 708 |
| | 27 | 61 491 | 55 236 | **50 548** | 53 343 | 120 714 |
| | 28 | 59 980 | 65 021 | 57 297 | **38 631** | 124 386 |
| | 29 | 57 594 | 188 241 | 67 603 | **42 736** | 121 999 |
| | 30 | 71 743 | **66 340** | 72 332 | 80 171 | 125 925 |
| | 31 | 80 675 | 75 532 | 285 296 | **69 654** | 150 685 |
| **conflicts** | ~~25~~ | 18 989 | 19 514 | **18 501** | 19 431 | 31 833 |
| | ~~26~~ | 27 180 | 27 906 | 33 399 | **20 912** | 40 969 |
| | 27 | 38 262 | 26 728 | **24 532** | 30 634 | 80 625 |
| | 28 | 34 453 | 33 157 | 26 112 | **18 682** | 77 873 |
| | 29 | 31 597 | **18 412** | 28 473 | 20 295 | 71 170 |
| | 30 | 40 803 | **28 115** | 30 613 | 43 978 | 71 750 |
| | 31 | 44 385 | 35 627 | **23 957** | 35 219 | 84 896 |
| **rules** | ~~25~~ | **73 406** | 106 969 | 111 079 | 78 734 | 67 749 |
| | ~~26~~ | **77 106** | 112 126 | 116 414 | 82 663 | 71 213 |
| | 27 | **80 806** | 117 283 | 121 749 | 86 592 | 74 677 |
| | 28 | **84 506** | 122 440 | 127 084 | 90 521 | 78 141 |
| | 29 | **88 206** | 127 597 | 132 419 | 94 450 | 81 605 |
| | 30 | **91 906** | 132 754 | 137 754 | 98 379 | 85 069 |
| | 31 | **95 606** | 137 911 | 143 089 | 102 308 | 88 533 |
| **constraints** | ~~25~~ | **118 320** | 142 056 | 146 621 | 119 776 | 113 198 |
| | ~~26~~ | **125 251** | 150 471 | 155 306 | 126 717 | 119 809 |
| | 27 | **132 182** | 158 886 | 163 991 | 133 658 | 126 420 |
| | 28 | **139 113** | 167 301 | 172 676 | 140 599 | 133 031 |
| | 29 | **146 044** | 175 716 | 181 361 | 147 540 | 139 642 |
| | 30 | **152 975** | 184 131 | 190 046 | 154 481 | 146 253 |
| | 31 | **159 906** | 192 546 | 198 731 | 161 422 | 152 864 |

Table 9: Statistics for constraint $\varphi_2$ and the 3 robots instance.

| | $\lambda$ | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | $NC$ |
|---|---|---|---|---|---|---|
| **translation time** | ~~25~~ | **1 632** | 45 303 | 4 973 | 2 718 | 317 |
| | ~~26~~ | 567 | **479** | 718 | 3 389 | 450 |
| | ~~27~~ | **557** | 889 | 635 | 2 676 | 533 |
| | ~~28~~ | 599 | 652 | **558** | 2 779 | 523 |
| | ~~29~~ | 564 | 718 | **490** | 3 160 | 800 |
| | ~~30~~ | 680 | 495 | **455** | 3 289 | 888 |
| | 31 | **683** | 762 | 769 | 3 786 | 592 |
| **clingo time** | ~~25~~ | 17 910 | **14 564** | 16 993 | 14 647 | 47 761 |
| | ~~26~~ | 33 219 | **26 979** | 36 179 | 29 358 | 75 161 |
| | ~~27~~ | 40 397 | 42 075 | 50 820 | **35 071** | 195 261 |
| | ~~28~~ | 74 244 | **59 909** | 78 488 | 69 510 | 539 328 |
| | ~~29~~ | 110 278 | 104 918 | 122 672 | **91 111** | 786 940 |
| | ~~30~~ | **177 537** | 197 182 | 402 643 | 186 005 | 496 808 |
| | 31 | 252 945 | **117 736** | 234 057 | 172 171 | 376 699 |
| **choices** | ~~25~~ | 79 355 | 89 839 | 82 854 | **68 478** | 173 746 |
| | ~~26~~ | 126 138 | 117 782 | 137 992 | **113 770** | 245 978 |
| | ~~27~~ | 141 902 | 167 256 | 178 598 | **131 983** | 417 294 |
| | ~~28~~ | 222 615 | 215 795 | 215 028 | **199 998** | 912 562 |
| | ~~29~~ | 284 038 | 300 505 | 279 271 | **251 921** | 2 561 235 |
| | ~~30~~ | **388 048** | 464 021 | 1 733 075 | 401 158 | 3 058 806 |
| | 31 | 539 032 | **348 336** | 1 475 025 | 430 008 | 2 508 802 |
| **conflicts** | ~~25~~ | 41 110 | 39 086 | 37 373 | **33 142** | 106 940 |
| | ~~26~~ | 72 849 | **58 218** | 69 422 | 61 248 | 154 251 |
| | ~~27~~ | 81 352 | 86 506 | 95 104 | **71 633** | 284 788 |
| | ~~28~~ | 137 837 | 117 090 | 120 923 | **117 023** | 670 627 |
| | ~~29~~ | 181 467 | 180 692 | 163 538 | **151 996** | 956 430 |
| | ~~30~~ | 259 733 | 294 888 | 434 690 | **258 368** | 753 632 |
| | 31 | 364 780 | **203 024** | 302 537 | 260 559 | 576 474 |
| **rules** | ~~25~~ | **137 614** | 188 131 | 201 109 | 145 489 | 128 422 |
| | ~~26~~ | **144 449** | 197 159 | 210 695 | 152 663 | 134 873 |
| | ~~27~~ | **151 284** | 206 187 | 220 281 | 159 837 | 141 324 |
| | ~~28~~ | **158 119** | 215 215 | 229 867 | 167 011 | 147 775 |
| | ~~29~~ | **164 954** | 224 243 | 239 453 | 174 185 | 154 226 |
| | ~~30~~ | **171 789** | 233 271 | 249 039 | 181 359 | 160 677 |
| | 31 | **178 624** | 242 299 | 258 625 | 188 533 | 167 128 |
| **constraints** | ~~25~~ | **215 577** | 253 048 | 263 456 | 217 522 | 206 819 |
| | ~~26~~ | **228 144** | 267 841 | 278 888 | 230 104 | 218 864 |
| | ~~27~~ | **240 711** | 282 634 | 294 320 | 242 686 | 230 909 |
| | ~~28~~ | **253 278** | 297 427 | 309 752 | 255 268 | 242 954 |
| | ~~29~~ | **265 845** | 312 220 | 325 184 | 267 850 | 254 999 |
| | ~~30~~ | **278 412** | 327 013 | 340 616 | 280 432 | 267 044 |
| | 31 | **290 979** | 341 806 | 356 048 | 293 014 | 279 089 |

Table 10: Statistics for constraint $\varphi_3$ and the 2 robots instance.

| | $\lambda$ | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | $NC$ |
|---|---|---|---|---|---|---|
| **translation time** | ~~25~~ | **2 533** | - | 12 682 | 3 343 | 260 |
| | ~~26~~ | **517** | - | 926 | 3 717 | 436 |
| | ~~27~~ | 606 | - | **578** | 2 781 | 599 |
| | ~~28~~ | **623** | - | 834 | 2 864 | 381 |
| | ~~29~~ | **547** | - | 739 | 3 429 | 761 |
| | 30 | **654** | - | 754 | 3 276 | 822 |
| | 31 | **652** | - | - | 3 613 | 630 |
| **clingo time** | ~~25~~ | 12 253 | - | 229 981 | **9 291** | 9 904 |
| | ~~26~~ | 25 188 | - | 359 347 | **13 359** | 14 406 |
| | ~~27~~ | 41 912 | - | 575 544 | **12 090** | 30 541 |
| | ~~28~~ | 42 680 | - | 898 255 | **16 686** | 28 792 |
| | ~~29~~ | 46 213 | - | 837 573 | **18 315** | 24 138 |
| | 30 | 36 066 | - | 706 322 | **17 020** | 23 975 |
| | 31 | 37 229 | - | - | **36 629** | 29 704 |
| **choices** | ~~25~~ | 54 885 | - | 15 989 429 | **44 122** | 51 711 |
| | ~~26~~ | 96 641 | - | 26 420 540 | **57 103** | 91 708 |
| | ~~27~~ | 140 631 | - | 53 728 668 | **53 232** | 120 714 |
| | ~~28~~ | 140 053 | - | 75 682 776 | **70 231** | 124 386 |
| | ~~29~~ | 148 264 | - | 70 859 664 | **78 123** | 121 999 |
| | 30 | 130 536 | - | 29 387 312 | **71 563** | 125 925 |
| | 31 | 133 751 | - | - | **131 548** | 150 685 |
| **conflicts** | ~~25~~ | 38 229 | - | 42 422 | **28 841** | 31 833 |
| | ~~26~~ | 70 619 | - | 54 901 | **38 380** | 40 969 |
| | ~~27~~ | 106 770 | - | 91 584 | **35 137** | 80 625 |
| | ~~28~~ | 105 411 | - | 123 025 | **47 733** | 77 873 |
| | ~~29~~ | 110 880 | - | 120 513 | **52 747** | 71 170 |
| | 30 | 95 068 | - | 106 837 | **46 265** | 71 750 |
| | 31 | 96 986 | - | - | **93 448** | 84 896 |
| **rules** | ~~25~~ | **82 732** | - | 3 970 094 | 89 873 | 67 749 |
| | ~~26~~ | **86 824** | - | 4 151 638 | 94 270 | 71 213 |
| | ~~27~~ | **90 916** | - | 4 333 182 | 98 667 | 74 677 |
| | ~~28~~ | **95 008** | - | 4 514 726 | 103 064 | 78 141 |
| | ~~29~~ | **99 100** | - | 4 696 270 | 107 461 | 81 605 |
| | 30 | **103 192** | - | 4 877 814 | 111 858 | 85 069 |
| | 31 | **107 284** | - | - | 116 255 | 88 533 |
| **constraints** | ~~25~~ | 130 874 | - | 2 856 055 | **126 120** | 113 198 |
| | ~~26~~ | 138 567 | - | 3 057 106 | **133 359** | 119 809 |
| | ~~27~~ | 146 260 | - | 3 258 157 | **140 598** | 126 420 |
| | ~~28~~ | 153 953 | - | 3 459 208 | **147 837** | 133 031 |
| | ~~29~~ | 161 646 | - | 3 660 259 | **155 076** | 139 642 |
| | 30 | 169 339 | - | 3 861 310 | **162 315** | 146 253 |
| | 31 | 177 032 | - | - | **169 554** | 152 864 |

Table 11: Statistics for constraint $\varphi_3$ and the 3 robots instance.

| | $\lambda$ | $\mathfrak{A}$ | $\mathfrak{M}_m$ | $\mathfrak{M}_s$ | $\mathfrak{T}$ | $NC$ |
|---|---|---|---|---|---|---|
| **translation time** | ~~25~~ | **3 112** | - | 11 001 | 3 278 | 271 |
| | ~~26~~ | **481** | - | 864 | 3 895 | 441 |
| | ~~27~~ | **621** | - | 851 | 3 314 | 546 |
| | ~~28~~ | **627** | - | 760 | 3 375 | 449 |
| | ~~29~~ | **531** | - | 708 | 3 855 | 777 |
| | ~~30~~ | **679** | - | - | 3 351 | 815 |
| | ~~31~~ | **663** | - | - | 4 519 | 555 |
| **clingo time** | ~~25~~ | 23 083 | - | 407 903 | **14 376** | 49 111 |
| | ~~26~~ | 29 884 | - | 364 093 | **21 753** | 74 915 |
| | ~~27~~ | 36 847 | - | 472 354 | **30 569** | 195 389 |
| | ~~28~~ | 60 786 | - | 476 299 | **45 751** | 526 867 |
| | ~~29~~ | 107 503 | - | 1 105 621 | **78 207** | 796 914 |
| | ~~30~~ | 151 375 | - | - | **108 197** | 500 901 |
| | ~~31~~ | 248 384 | - | - | **163 082** | 378 597 |
| **choices** | ~~25~~ | 98 424 | - | 36 202 304 | **60 086** | 173 746 |
| | ~~26~~ | 110 269 | - | 26 575 984 | **86 056** | 245 978 |
| | ~~27~~ | 111 872 | - | 32 814 318 | **103 601** | 417 294 |
| | ~~28~~ | 170 755 | - | 297 764 | **142 450** | 912 562 |
| | ~~29~~ | 259 262 | - | 57 960 092 | **200 553** | 2 561 235 |
| | ~~30~~ | 310 797 | - | - | **262 182** | 3 058 806 |
| | ~~31~~ | 460 825 | - | - | **350 260** | 2 508 802 |
| **conflicts** | ~~25~~ | 63 019 | - | 97 082 | **36 166** | 106 940 |
| | ~~26~~ | 72 644 | - | 81 166 | **53 689** | 154 251 |
| | ~~27~~ | 75 839 | - | 129 245 | **66 361** | 284 788 |
| | ~~28~~ | 116 876 | - | 136 306 | **93 099** | 670 627 |
| | ~~29~~ | 184 608 | - | 249 421 | **137 604** | 956 430 |
| | ~~30~~ | 230 970 | - | - | **185 500** | 753 632 |
| | ~~31~~ | 347 612 | - | - | **254 287** | 576 474 |
| **rules** | ~~25~~ | **151 581** | - | 3 083 856 | 162 198 | 128 422 |
| | ~~26~~ | **159 003** | - | 3 225 088 | 170 074 | 134 873 |
| | ~~27~~ | **166 425** | - | 3 366 320 | 177 950 | 141 324 |
| | ~~28~~ | **173 847** | - | 3 507 552 | 185 826 | 147 775 |
| | ~~29~~ | **181 269** | - | 3 648 784 | 193 702 | 154 226 |
| | ~~30~~ | **188 691** | - | - | 201 578 | 160 677 |
| | ~~31~~ | **196 113** | - | - | 209 454 | 167 128 |
| **constraints** | ~~25~~ | 235 091 | - | 2 435 143 | **227 044** | 206 819 |
| | ~~26~~ | 248 801 | - | 2 594 044 | **240 073** | 218 864 |
| | ~~27~~ | 262 511 | - | 2 752 945 | **253 102** | 230 909 |
| | ~~28~~ | 276 221 | - | 2 911 846 | **266 131** | 242 954 |
| | ~~29~~ | 289 931 | - | 3 070 747 | **279 160** | 254 999 |
| | ~~30~~ | 303 641 | - | - | **292 189** | 267 044 |
| | ~~31~~ | 317 351 | - | - | **305 218** | 279 089 |