Efficient Conformance Checking of Rich Data-Aware Declare Specifications (Extended)

Jacobo Casas-Ramos[1]✉, Sarah Winkler[2], Alessandro Gianola[3], Marco Montali[2], Manuel Mucientes[1], and Manuel Lama[1]

[1] Universidade de Santiago de Compostela, Santiago de Compostela, Spain
{jacobocasas.ramos,manuel.lama,manuel.mucientes}@usc.es
[2] Free University of Bozen-Bolzano, Bolzano, Italy
{montali,winkler}@inf.unibz.it
[3] INESC-ID/Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
alessandro.gianola@tecnico.ulisboa.pt

Abstract. Despite growing interest in process analysis and mining for data-aware specifications, alignment-based conformance checking for declarative process models has focused on pure control-flow specifications, or mild data-aware extensions limited to numerical data and variable-to-constant comparisons. This is not surprising: finding alignments is computationally hard, even more so in the presence of data dependencies. In this paper, we challenge this problem in the case where the reference model is captured using data-aware Declare with general data types and data conditions. We show that, unexpectedly, it is possible to compute data-aware optimal alignments in this rich setting, enjoying at once efficiency and expressiveness. This is achieved by carefully combining the two best-known approaches to deal with control flow and data dependencies when computing alignments, namely A* search and SMT solving. Specifically, we introduce a novel algorithmic technique that efficiently explores the search space, generating descendant states through the application of repair actions aiming at incrementally resolving constraint violations. We prove the correctness of our algorithm and experimentally show its efficiency. The evaluation witnesses that our approach matches or surpasses the performance of the state of the art while also supporting significantly more expressive data dependencies, showcasing its potential to support real-world applications.

Keywords: Multi-perspective conformance checking · Efficient optimal alignments · Data-aware Declare · Satisfiability modulo theories (SMT).

1 Introduction

Conformance checking [6] is a cornerstone task in process mining. It relates the observed behaviour contained in an event log to the expected behaviour described by a reference process model, with the goal of identifying and reporting deviations. A widely adopted approach substantiates conformance checking in the computation of so-called optimal *alignments*, where each non-conforming log

trace is compared against the closest model trace(s), indicating where discrepancies are located and calculating a corresponding cost [4].

Lifting the computation of alignments to process models integrating multiple perspectives (most prominently data and control-flow) has been tackled with increasing interest [19,2,14], and so has been dealing with other forms of data-aware conformance checking. On the one hand, this reflects a growing prominence of multi-perspective (in particular, data-aware) process models in the foundations of process science. On the other hand, computing multi-perspective alignments provides more informative insights than pure control-flow alignments [19].

Despite this growing interest, alignment-based conformance checking for declarative data-aware process specifications is still an open problem. Existing work mainly focuses only on control-flow, concretely expressed using DCR graphs [8], Declare [18], or Linear Temporal Logic on finite traces (LTLf) [10]. To the best of our knowledge, the only attempt of lifting alignment computation to a data-aware setting is [2], which considers however a very limited data-aware extension of Declare where data are numerical and data conditions are restricted to variable-to-constant comparisons, such as $x > 5$, which also excludes comparisons between variables in different events.

Example 1. To highlight the expressivity of complex data conditions, consider a model for a shipping company that has a Declare *response* constraint between two events: "Package Shipment" (A) and "Delivery Confirmation" (B), such that after a package is shipped, delivery confirmation must be received. The constraint is equipped with a data condition that specifies that the delivery confirmation must be received within 3 days of shipping if the package weighs less than 10 kg and the delivery address is within a specific geographic region. In any other case, it must be received within 10 days of shipping. Our approach can handle this condition, expressed in SMT-LIB2 [1] syntax as `(A.weight < 10.0 and B.region == "Europe") ? (B.time - A.time <= 3d) : (B.time - A.time <= 10d)`.

The fact that previous work did not consider complex constraints like the one shown in the example is not surprising: finding alignments is computationally hard, even more so in the presence of data dependencies. Also, this more expressive setting cannot be addressed relying on previous methods [10,2], based on the construction of an automaton capturing all and only the traces accepted by the reference Declare specification. In fact, this is not possible even for numerical datatypes going beyond mere comparison predicates, due to undecidability [15].

In this paper, we tackle this challenge by casting the alignment problem as a search problem that is solved by repeatedly identifying and repairing constraint violations. To this end we strategically integrate the two most effective methods for handling control flow and data dependencies in alignment computation, respectively: A* search, in the variant adopted by one of the most recent methods for Declare [7], and SMT solving [1], so far employed for aligning data-aware procedural process models [14]. Our technique defines a novel search space that is explored using the A* search algorithm to find an optimal alignment. The initial state of the search space represents the original trace as an SMT formula.

When a state is explored, an SMT solver is used to identify constraint violations, which in turn triggers the generation of child states by repairing the parent state. This process continues until a goal state is found that has minimal cost and no violation to repair, from which the optimal alignment can be reconstructed.

We establish correctness of our algorithm through rigorous proofs and provide an extensive experimental evaluation, showing its ability to operate efficiently even when complex data conditions are employed. Our method matches or surpasses the performance of the state of the art while providing for the first time concrete support for rich datatypes and data conditions.

The remainder of this paper is structured as follows: Sec. 2 we summarize related work. In Sec. 3 we recall the necessary preliminaries about data-aware Declare, and alignments. Sec. 4 is dedicated to our approach to conformance checking for data-aware Declare specifications. We describe its implementation in the tool DADA in Sec. 5. In Sec. 6, we provide a detailed evaluation and comparison with the state-of-the-art. In Sec. 7, we conclude and give some directions for future work. Additional material is available online.[4]

2 Related Work

Although significant research has focused on computing optimal alignments for imperative process models that incorporate the data perspective [19,14,20,11], there is a notable lack of work on data-aware alignment techniques for declarative models. Existing conformance checking methods for declarative models primarily focus on control-flow [17,10,9,7,8,18]. Notably, they lack the capability to handle data conditions, a critical component of real-world processes. Though some approaches have been proposed for conformance checking of data-aware Declare models, they have severe limitations. SQL queries have been employed to filter traces from a database that match the given specification, but this approach only considers exactly matching traces [22,21]. Similarly, [3] uses constraint programming for trace analysis, but only supports global data. More comprehensive results are provided by the analysis framework [5], which reports activations, fulfillments, and violations of constraints. Moreover, importantly, none of the approaches [22,21,3,5] provides alignments, thus failing to offer detailed insights into the nature and extent of deviations between observed and expected behavior: This lack of nuanced analysis hinders the ability to identify root causes of non-conformance and implement targeted improvements, ultimately undermining the effectiveness of conformance checking efforts.

Closest to our work is the planning-based conformance checking approach [2], which does compute optimal alignments of data-aware Declare models. However, their treatment of the data perspective has severe limitations: data conditions can only refer to activation or target events, excluding correlation conditions that link the two. Moreover, data conditions are restricted to simple variable-to-constant comparisons, whereas our approach supports much more expressive data conditions over a wide range of data types including integers, bit-vectors, infinite-precision reals, and arrays. Specifically, we support the complete lan-

[4] https://apps.citius.gal/dada and https://doi.org/10.5281/zenodo.15470077

guage of conditions specified in the SMT-LIB2 standard [1], requiring only that the underlying SMT theory is decidable. Overall, our approach offers a significant improvement over existing methods in terms of expressiveness and efficiency.

3 Preliminaries

In this section we introduce the required background about event logs, Declare with data, and alignments. We start with the data condition language and events.

Data conditions. We consider *sorts* $\Sigma = \{\texttt{bool}, \texttt{int}, \texttt{rat}, \texttt{string}\}$ for data payloads, with associated domains $\mathcal{D}(\texttt{bool}) = \mathbb{B}$, the booleans; $\mathcal{D}(\texttt{int}) = \mathbb{Z}$, the integers; $\mathcal{D}(\texttt{rat}) = \mathbb{Q}$, the rational numbers, and $\mathcal{D}(\texttt{string}) = \mathbb{S}$, finite strings. For a set of variables V and a sort $\sigma \in \Sigma$, V_σ denotes the subset of V of sort σ.

Definition 1. *A* data condition *over a set of variables V is an expression c according to the following grammar:*
$c := v_{\texttt{bool}} \mid b \mid n \geq n \mid r \geq r \mid r > r \mid s = s \mid c \wedge c \mid \neg c \quad s := v_{\texttt{string}} \mid t$
$n := v_{\texttt{int}} \mid z \mid n + n \mid -n \qquad r := v_{\texttt{rat}} \mid q \mid r + r \mid -r$
where $v_\sigma \in V_\sigma$ for $\sigma \in \Sigma$, $b \in \mathbb{B}$, $t \in \mathbb{S}$, $z \in \mathbb{Z}$, and $q \in \mathbb{Q}$. The set of data conditions over a set of variables V is denoted by $\mathcal{C}(V)$.

We use data conditions as in Def. 1 in this paper to have a concrete language to refer to, but our implementation actually allows for arbitrary conditions in the SMT-LIB2 language [1] that is supported by the SMT solver of choice. In the sequel, we assume that the underlying SMT theory is decidable, though; this restriction is required to provide correctness guarantees.

Event logs. Below, we assume an arbitrary infinite set Id of event identifiers; and a set $\mathcal{A}$ of activities, where elements $a \in \mathcal{A}$ are denoted by lower-case letters. We consider the following notions of data-aware events, traces, and event logs:

Definition 2. *An* event *e is a triple $e = (\iota, a, \alpha)$ such that $\iota \in Id$, $a \in \mathcal{A}$ is an activity, and α is a partial assignment that maps variables in V to elements of their domain. Given a set of events E, a* trace $\mathbf{e}$ *is a finite sequence of events in E, that is, $\mathbf{e} \in E^*$; and an* event log *is a multiset of traces. The domain of an assignment α is denoted $dom(\alpha)$.*

Declare. Tab. 1 lists the Declare templates used in this paper. We call a *Declare constraint* an expression that is obtained from a Declare template by substituting the upper-case template variables by activities in $\mathcal{A}$. Constraints based on templates (6)–(9) and (10)–(12) are called *response* and *precedence constraints*, respectively, while constraints using (13)–(15) are *negation constraints*.

Declare templates, as well as the derived constraints, have *activations* and *targets*. Intuitively, an activation is an event whose occurrence imposes the (non) occurrence of other events. These other events are called targets. For the templates in Tab. 1, in all response and negation templates, variable A is the activation and B the target; while in all precedence templates, B is the activation and A the target. In the remaining patterns, both A and B are targets. Given a Declare constraint φ, an activity is an activation (resp. target) activity in φ if it is substituted for an activation (resp. target) variable in the underlying template.

(1)	Existence(n, A):	A occurs at least n times.
(2)	Absence(n, A):	A occurs at most $n-1$ times.
(3)	Init(A):	A is the first activity.
(4)	End(A):	A is the last activity.
(5)	Choice(A, B):	Either A or B, or both, occur.
(6)	RespondedExistence(A, B):	If A occurs, B also occurs.
(7)	Response(A, B):	If A occurs, B follows.
(8)	AlternateResponse(A, B):	If A occurs, B follows without an A in between.
(9)	ChainResponse(A, B):	If A occurs, B is the next activity.
(10)	Precedence(A, B):	If B occurs, A precedes it.
(11)	AlternatePrecedence(A, B):	If B occurs, A precedes it without a B in between.
(12)	ChainPrecedence(A, B):	If B occurs, A is the previous activity.
(13)	NotResponse(A, B):	If A occurs, B does follow.
(14)	NotRespondedExistence(A, B):	If A occurs, B does not.
(15)	NotChainResponse(A, B):	If A occurs, B is not the next activity.

Table 1: Supported Declare templates.

We consider *multi-perspective* Declare constraints that include data conditions. To that end, for the remainder of the paper we fix a set of sorted *process variables* V. Intuitively, these variables are considered the payload of activities; they are maintained along the entire trace, but may change their values. For a set Set, let $V^{Set} = \{v_s \mid v \in V \text{ and } s \in Set\}$ be a set of labelled variables that contains a copy of each variable in V for each element in Set. In particular, $v_a \in V^{\mathcal{A}}$ will be used to represent the value of v while observing activity a.

Definition 3. *A* Declare constraint with data *is a quadruple* $\langle \varphi, c_{act}, c_{tgt}, c_{cor} \rangle$ *consisting of a Declare constraint* φ *and data conditions* c_{act}, c_{tgt} *and* c_{cor}. *Precisely, for* $a \in \mathcal{A}$ *the activation and* $T \subseteq \mathcal{A}$ *the target activities of* φ: *(i)* $c_{act} \in \mathcal{C}(V^{\{a\}})$ *is called the* activation condition*, (ii)* $c_{tgt} \in \mathcal{C}(V^{T})$ *is called the* target condition*, and (iii)* $c_{cor} \in \mathcal{C}(V^{\{a\} \cup T})$ *is the* correlation condition.

Intuitively, c_{act} constrains the data variables while the activation activity is observed, c_{tgt} the data variables while the target activity is observed, and c_{cor} expresses relationships between the data variables of both activities. For Declare constraints φ without activation, we assume that all but c_{tgt} are $\top$. For simplicity of presentation, we assume that the activation and target activity are different, (though in our implementation this is not required). A *Declare specification* $\mathcal{M}$ is a set of Declare constraints with data. In the sequel, if no confusion can arise, we refer to Declare constraints with data simply by *constraints*.

Example 2. As running example, we use the set of variables $V = \{x\}$, activities $\mathcal{A} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and the specification $\mathcal{M}$ that consists of the following two constraints ψ_1 and ψ_2, where for readability we write $a.v$ instead of v_a, for $a \in \mathcal{A}$:

- $\psi_1 = \langle \text{ChainResponse}(\mathsf{a}, \mathsf{c}), \top, \top, \mathsf{c}.x > \mathsf{a}.x \rangle$: This specifies that each occurrence of a must be directly followed by an event with c such that the value of x associated with activity c is greater than the value of x seen with a.
- $\psi_2 = \langle \text{AlternatePrecedence}(\mathsf{c}, \mathsf{b}), \mathsf{b}.x \geq 0, \mathsf{c}.x \neq 0, \mathsf{c}.x < \mathsf{b}.x \rangle$: This states an alternate precedence relationship between the activation b and the target c, demanding that if the value of x seen with b is non-negative, an activity c must occur before activity b, without any other b activities with $x \geq 0$ in between. Furthermore, the x-value of c must be lower than the x-value of b.

The semantics of Declare constraints with data is the same as in [2], we recall it in Sec. 11. The set of all traces that satisfy all constraints in $\mathcal{M}$ is denoted by $runs(\mathcal{M})$. We assume for our approach that $runs(\mathcal{M}) \neq \emptyset$.

Alignments. We aim to design a conformance-checking procedure that, given a trace and a Declare specification $\mathcal{M}$, finds an optimal alignment of $\mathbf{e}$ and a run of $\mathcal{M}$. Typically, when constructing alignments, not all events in the trace can be put in correspondence with an event in a run, and vice versa. Hence we use a "skip" symbol $\gg$ and consider the extended set of events $E^{\gg} = E \cup \{\gg\}$.

For a set E of events, a pair $(e, f) \in {E^{\gg}}^2 \setminus \{(\gg, \gg)\}$ is called *move* iff one of the following cases applies: it is a (i) *log move* if $e \in E$ and $f = \gg$; (ii) *model move* if $e = \gg$ and $f \in E$; (iii) *edit move* if $(e, f) \in E^2$, $(e, f) = ((\iota, a, \alpha), (\iota, a, \alpha'))$, $dom(\alpha) = dom(\alpha')$ and $\exists v \in dom(\alpha)$ such that $\alpha(v) \neq \alpha'(v)$; (iv) *synchronous move* if $(e, f) \in E^2$ and $e = f$. We denote by *Moves* the set of all moves.

For a sequence of moves $\gamma = \langle (e_1, f_1), \ldots, (e_n, f_n) \rangle$, the *log projection* $\gamma|_L$ of γ is the maximal subsequence $e'_1, \ldots, e'_i$ of $e_1, \ldots, e_n$ such that $e'_1, \ldots, e'_i \in E^*$, that is, it contains no $\gg$ symbols. Similarly, the *model projection* $\gamma|_M$ of γ is the maximal subsequence $f'_1, \ldots, f'_j$ of $f_1, \ldots, f_n$ such that $f'_1, \ldots, f'_j \in E^*$.

Definition 4 (Alignment). *Given a Declare model $\mathcal{M}$, a sequence of moves γ is an* alignment *of a trace $\mathbf{e}$ against $\mathcal{M}$ if $\gamma|_L = \mathbf{e}$, and $\gamma|_M \in runs(\mathcal{M})$. The set of alignments for a trace $\mathbf{e}$ wrt. $\mathcal{M}$ is denoted by $Align(\mathcal{M}, \mathbf{e})$.*

Example 3. Consider the trace $\mathbf{e} = \langle (\#_1, \mathsf{a}, \{x = 0\}), (\#_2, \mathsf{b}, \{x = 2\}) \rangle$. The following are two possible alignments for $\mathbf{e}$ against the model from Ex. 2:

$\gamma_1 =$

a	$\{x = 0\}$	$\gg$		b	$\{x = 2\}$
a	$\{x = 0\}$	c	$\{x = 1\}$	b	$\{x = 2\}$

$\gamma_2 =$

a	$\{x = 0\}$	$\gg$		b	$\{x = 2\}$
a	$\{x = 0\}$	c	$\{x = 3\}$	$\gg$	

Each move (e, f) is shown in a column, including e in the first row and f in the second row. Since event identifiers are irrelevant in alignments, we omit them.

A *cost function* is a mapping κ: *Moves* $\to \mathbb{R}^+$ that assigns a cost to every move. It is naturally extended to alignments as follows.

Definition 5 (Alignment cost). *Given $\gamma \in Align(\mathcal{M}, \mathbf{e})$ as before, the* cost *of γ is defined as the sum of the costs of its moves, that is, $\kappa(\gamma) = \sum_{i=1}^{n} \kappa(e_i, f_i)$. Moreover, γ is* optimal for $\mathbf{e}$ and $\mathcal{M}$ *if $\kappa(\gamma)$ is minimal among all alignments for $\mathbf{e}$ and $\mathcal{M}$, namely there is no $\gamma' \in Align(\mathcal{M}, \mathbf{e})$ with $\kappa(\gamma') < \kappa(\gamma)$.*

In this paper, we will use the standard cost function κ that assigns $\kappa(e, f) = 1$ if (e, f) is a log or model move, $\kappa(e, f) = 0$ if (e, f) is a synchronous move, and for an edit move $(e, f) = ((\iota, a, \alpha), (\iota, a, \alpha'))$, $\kappa(e, f) = |\{\alpha(v) \neq \alpha'(v) \mid v \in V\}|$.

4 Data-Aware Declare Aligner

In this section, we outline the conceptual approach of the Data-Aware Declare Aligner. Given a Declare specification $\mathcal{M}$ and a trace $\mathbf{e}$, the aim is to find an optimal alignment of $\mathbf{e}$ wrt. $\mathcal{M}$. To that end, the basic idea is to start with the event sequence in $\mathbf{e}$, and subsequently *repair* it until an event sequence is

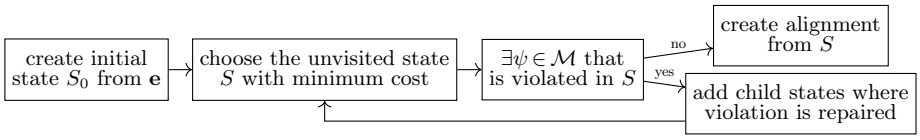

Fig. 1: Overview of the approach.

obtained that satisfies all constraints in $\mathcal{M}$. To navigate through a large search space of possible repairs and respective alignments while ensuring an optimal solution, our approach leverages the A* algorithm.

We use the term *state* to refer to a representation of a candidate alignment. The formal definition of a state is given below; intuitively, each state consists of a partially ordered set of events together with data conditions, effectively acting as a candidate alignment which need not yet satisfy all constraints. Moreover, each state has a *cost*, reflecting the cost of alignments extracted from it.

An overview of the approach is sketched in Fig. 1: the initial state S_0 represents the set of events in the input trace, ordered as in **e**, and with data conditions that reflect the variable assignments. The procedure then selects the previously unvisited state S of minimal cost. It is checked whether there are remaining constraint violations in S. In this case, all possible *repairs* are applied to S creating a new child state from each repair, and another search iteration is performed. Otherwise, an optimal alignment for **e** is reconstructed from S.

4.1 State definition

As mentioned above, a state contains a partially ordered set of events, and data conditions on their payloads. In order to express conditions that involve variables in all events, we need, as a technicality, labelled variables: For an event $e = (\iota, a, \alpha)$, let $V^e = \{v_\iota \mid v \in V\}$ be a copy of the set V where each variable is labelled by the id of e. For a set of events E, let $V^E = \bigcup_{e \in E} V^e$ be the set of variables for all events in E. A state with set of events E can then use data conditions (cf. Def. 1) on V^E to refer to the events' payloads. In the sequel we also assume that V contains a special variable τ of type integer, and τ^ι will denote the timestamp of event with id ι. To reason about partial orderings of events in E, states use *ordering conditions*, defined next:

Definition 6. *An* ordering condition *o for a set of events E is of the following form, where $e, e' \in E$, $a \in \mathcal{A}$ is an activity, and c is a data condition as in Def. 1:*

$$o := e < e' \mid e \ll e' \mid \mathit{first}(e) \mid \mathit{last}(e) \mid e <^a_{[c]} e' \mid \neg o \mid \top$$

Here $e < e'$ expresses that e happens before e', $e \ll e'$ that e happens before e' without any other event in between, $\mathit{first}(e)$ that e is the first, $\mathit{last}(e)$ that e is the last element, and $\top$ is a condition that is always true. Somewhat more complex, $e <^a_{[c]} e'$ expresses that e happens before e' without an event e'' in between that has activity a and satisfies c, where c is supposed to be a data condition over $V^{e''}$. A set of ordering conditions on E in *satisfiable* if there exists a topological sort of E that satisfies all conditions. A set of ordering conditions O is said to

entail an ordering condition q, denoted $O \models q$, if $\bigwedge O \wedge \neg q$ is unsatisfiable. Note that the ordering conditions are defined to closely align with the semantics of the supported Declare templates, as clarified in Def. 8.

Definition 7. *A* state *is a pair $S = \langle E, C \rangle$ where E is a set of events, and C is a set of ordering conditions on E and data conditions over V^E.*

A state $\langle E, C \rangle$ thus represents a set of events E that is partially ordered by the ordering conditions in C, and where payloads of events are constrained by the data conditions in C.

For a trace $\mathbf{e} = \langle e_1, \ldots, e_n \rangle$, let $E(\mathbf{e}) = \{e_1, \ldots, e_n\}$ be its set of events, $O(\mathbf{e}) = \{e_i < e_{i+1} \mid 1 \leq i < n\}$ be the set of ordering conditions that capture the event ordering in $\mathbf{e}$, and $D(\mathbf{e})$ the conjunction of all equations $v_\iota = \alpha(v)$ such that an event $e = (\iota, a, \alpha)$ occurs in $\mathbf{e}$ and $v \in dom(\alpha)$. The *initial state* is $\langle E(\mathbf{e}), O(\mathbf{e}) \cup D(\mathbf{e}) \rangle$, it serves as the starting point for the exploration of the search space. Note that we use formulas that mix ordering and data conditions; we will explain in the next section how standard SMT solvers can be used to perform satisfiability checks of such formulas.

Example 4. Consider the trace $\mathbf{e}$ in Ex. 3 with events $e_1 = (\#_1, \mathsf{a}, \{x = 0\})$ and $e_2 = (\#_2, \mathsf{b}, \{x = 2\})$. If no confusion can arise, we write $\mathsf{a}.x$ rather than $x_{\#_1}$ etc. for readability. The initial state is $S_0 = \langle \{e_1, e_2\}, (e_1 < e_2) \wedge (\mathsf{a}.x = 0) \wedge (\mathsf{b}.x = 2) \rangle$. Here $e_1 < e_2$ expresses that e_1 happens before e_2; the remaining conditions fix the values of x in the two events. Fig. 2 shows most of the search space for $\mathbf{e}$ and the specification from Ex. 2 (the complete search space is shown in Fig. 5). States are shown as boxes, S_0 being the box on top. The events of a state are shown as boxes with activities within the state, and arrows in between them indicate ordering conditions. Here $e_1 < e_2$ is displayed by an arrow $e_1 \rightarrow e_2$, $e_1 \ll e_2$ by $e_1 \twoheadrightarrow e_2$, and the condition $e_1 <^{\mathsf{b}}_{[\mathsf{c}.x < \mathsf{b}.x]} e_2$ obtained from ψ_2 by $e_1 \overset{\psi_2}{\twoheadrightarrow} e_2$. The formulas at the bottom of states specify data conditions. The states S_1–S_8 are obtained from S_0 by applying repairs; we will explain below how this is done.

4.2 Constraint violations

We next define when constraints are violated in a state. To that end, we need some additional notation: given a Declare constraint ψ and events $e, e' \in E$, we denote by $Ord(\psi, e, e')$ the ordering conditions imposed by ψ between an activation event e and a target event e', defined as follows:

Definition 8. *Let e, e' be events and $\psi = (\varphi, c_{act}, c_{tgt}, c_{cor})$ a constraint. For templates φ with an activation, we define $Ord(\psi, e, e')$ as $e < e'$ (resp. $e' < e$) if φ is based on a* Response *(resp.* Precedence*) template, $e \ll e'$ (resp. $e' \ll e$) if it is a* ChainResponse *(resp.* ChainPrecedence*) template, $\neg(e < e')$ for* NotResponse*, $\neg(e \ll e')$ for* NotChainResponse*, $e <^{a}_{[c_{act}]} e'$ for* AlternateResponse*, and $e' <^{a}_{[c_{act}]} e$ for* AlternatePrecedence*. In the last cases, a is the activation activity of φ. For constraints φ without activation, let $Ord(\psi, e)$ be $first(e)$ or $last(e)$ if φ is an* Init *or* Last *constraint, respectively. In all other cases, $Ord(\psi, e) = \top$.*

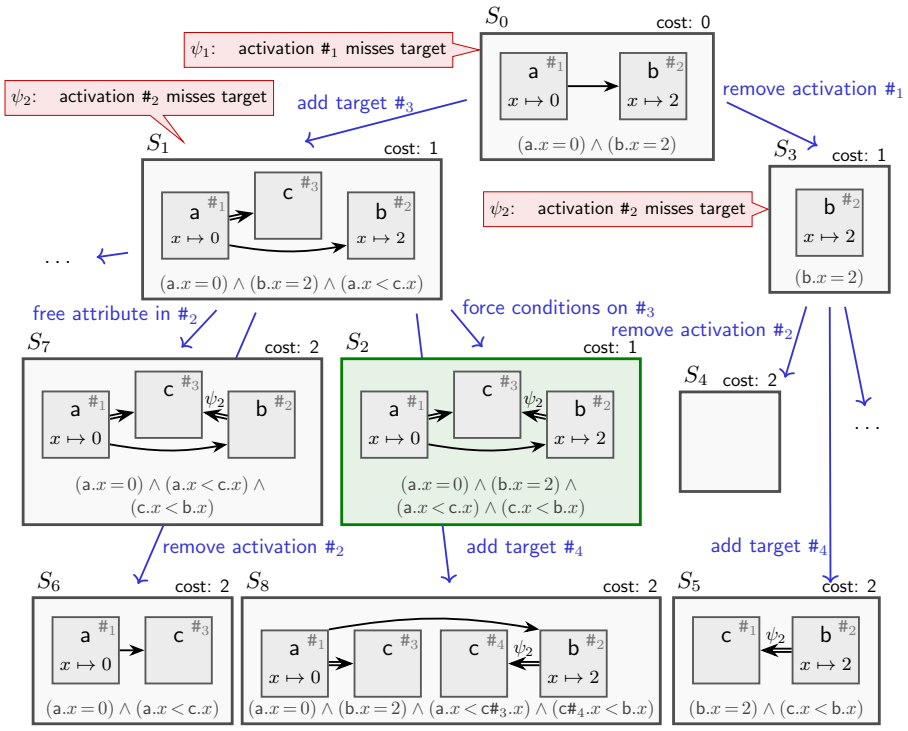

Fig. 2: Search space for the running example.

We also need to instantiate data conditions for events. To that end, given a Declare constraint $\psi = (\varphi, c_{act}, c_{tgt}, c_{cor})$ and an event $e = (\iota, a, \alpha)$ such that a is an activation activity for φ and b a target activity, we denote by $[c_{act}](e)$ (resp. $[c_{tgt}](e)$) the condition obtained from c_{act} (resp. c_{tgt}) by substituting v_a (resp. v_b) with v_ι for each $v \in V$. Similarly, for another event $e' = (\delta, b, \alpha')$, $[c_{tgt} \wedge c_{cor}](e, e')$ denotes the condition obtained from $c_{tgt} \wedge c_{cor}$ by substituting variables v_a by v_ι, and v_b by v_δ for all $v \in V$.

The first kind of violation is a *missing target*; intuitively, it applies if a constraint ψ can be activated but might lack a target that satisfies all conditions.

Definition 9. *A constraint* $(\varphi, c_{act}, c_{tgt}, c_{cor})$ *has a* missing target *violation in state* (E, C) *if one of the following cases applies:*

- φ *is a response or precedence constraint with activation activity* a *and there is an* $e = (\iota, a, \alpha) \in E$ *such that* $C \wedge [c_{act}](e)$ *is satisfiable, but no* $e' \in E$ *with target activity such that* $\bigwedge C \wedge [c_{act}](e) \models Ord(\varphi, e, e') \wedge [c_{tgt} \wedge c_{cor}](e, e')$*; or*
- φ *is of the form* Existence(n, a)*, and* $e_1, \ldots, e_k$ *are all events with activity* a *in* E *but* $\bigwedge C \models \Sigma_{i=1}^{k} ite([c_{tgt}](e_i), 1, 0) \geq n$ *does not hold; or*
- φ *is an* Init*,* End*, or* Choice *constraint and* $e_1, \ldots, e_k$ *are all events with target activity in* E *but* $\bigwedge C \models \bigvee_{i=1}^{k} Ord(\psi, e_i) \wedge [c_{tgt}](e_i)$ *does not hold.*

Here $ite(b, d_1, d_2)$ abbreviates an if-then-else expression. In the first case of Def. 9, e is called *activation event*.

Fig. 2 shows three cases of missing target violations for the constraints in Ex. 2: in state S_0, ψ_1 is activated by the event $\#_1$ with activity a, but no target event with activity c is present. In S_1 and S_3, ψ_2 is violated: in S_3 since no event with activity c occurs, and in S_1 because, even though an event with activity c occurs, namely $\#_3$, its conditions do not entail $\mathsf{c}.x < \mathsf{b}.x$ and $\#_3 <^{a}_{[c_{act}]} \#_2$.

The second kind of violation is dual in that it signals too many targets.

Definition 10. *A constraint $\psi = (\varphi, c_{act}, c_{tgt}, c_{cor})$ has an* excessive target *violation in a state $S = (E, C)$ if one of the following cases applies:*

- *φ is of the form* Absence(n, a) *and there are n events $e_1, \ldots, e_n$ in E with activity a such that $\bigwedge C \wedge \bigwedge_{i=1}^{n} [c_{tgt}](e_i)$ is satisfiable;*
- *φ is a negation constraint, some $e_0, e_1 \in E$ have activation and target activity, resp., and $\bigwedge C \wedge Ord(\psi, e_0, e_1) \wedge [c_{act}](e_0) \wedge [c_{tgt} \wedge c_{cor}](e_0, e_1)$ is satisfiable.*

The events $e_1, \ldots, e_n$ in Def. 10 are called *excessive target events*.

For instance, for the states in Fig. 2, a constraint $\langle \text{NotResponse}(\mathsf{a}, \mathsf{b}), \mathsf{a}.x \geq 0, \top, \mathsf{b}.x > \mathsf{a}.x \rangle$ would have an excessive target violation in states S_0 and S_1, but not in S_3. On the other hand, $\langle \text{Absence}(\mathsf{b}), \top, \mathsf{b}.x = 3 \rangle$ would be violated in state S_7, but not in S_0 where the data conditions exclude $\mathsf{b}.x = 3$.

A constraint ψ is *violated* in a state if it has a missing or excessive target. A state $S = \langle E, C \rangle$ is a *goal state* if no constraint in $\mathcal{M}$ is violated in S and C is satisfiable. In Fig. 2, all leaves of the search tree (S_2 and S_4–S_8) are goal states.

4.3 Repairing violations

Our approach subsequently expands the search space by selecting a state where a constraint is violated, generating *child states* by repairing the violation in different ways. Four kinds of repairs are distinguished: (a) addition of an event, which will be reflected as a model move in the alignment; (b) removal of an event that stems from the trace, corresponding to a log move in the alignment; (c) freeing a data attribute in an event that stems from the trace, corresponding to an edit move; and (d) enforcement of conditions.

The applicable repairs and resulting states depend on the violated constraint $\psi = (\varphi, c_{act}, c_{tgt}, c_{cor}) \in \mathcal{M}$ and current state $S = \langle E, C \rangle$. First, if ψ has an activation event $e_{act} \in E$, the following repairs are applied for both missing and excessive target violations to *disable* the activation:

(1) *Removing an activation event.* This repair only applies if e_{act} stems from the trace $\mathbf{e}$. The resulting state is $S' = \langle E \setminus \{e_{act}\}, C' \rangle$ where C' is like C with conditions involving e_{act} removed.

(2) *Freeing a data attribute.* This applies to an event $e_{act} = (\iota, a, \alpha)$ from the trace $\mathbf{e}$ if α does not satisfy $\neg[c_{act}](e_{act})$. The repair removes an assignment $\alpha(v)$ of e_{act} for some $v \in V$. For $C' = C \setminus \{v_\iota = \alpha(v)\} \cup \{\neg[c_{act}](e_{act})\}$, the new state is $S' = \langle E, C' \rangle$.

(3) *Enforcing the negated activation condition.* The resulting state is $S' = \langle E, C' \rangle$ with $C' = C \cup \{\neg[c_{act}](e_{act})\}$.

If the violation is a missing target for ψ, then a target event can be added, or an existing event with the correct activity can be enforced to satisfy the data conditions, or in some cases events can be removed that block ordering conditions. More precisely, the following repairs apply:

(4) *Adding a target event.* A new state is of the form $S' = \langle E \cup \{e\}, C' \rangle$ where $e = (\iota, a, \emptyset)$ is a new event with fresh identifier ι. If ψ has an activation and e' is the activation event, then $C' = C \cup \{Ord(\psi, e', e), [c_{act}](e'), [c_{tgt} \wedge c_{cor}](e', e)\}$. Otherwise, $C' = C \cup \{Ord(\psi, e), [c_{tgt}](e)\}$.

(5) *Freeing a data attribute.* This applies to events $e = (\iota, a, \alpha)$ in E that stem from the trace $\mathbf{e}$ and have the target activity but do not satisfy $[c_{tgt} \wedge c_{cor}](e', e)$ if ψ has an activation event e', resp. $[c_{tgt}](e)$ otherwise. The repair removes some assignment $\alpha(v)$ for $v \in V$, which can avoid the violation. The new state is $S' = \langle E', C \setminus \{v_\iota = \alpha(v)\} \rangle$ where E' is like E but where e is modifed to (ι,a,α') such that α' is like α except being undefined for v.

(6) *Enforcing conditions.* This applies to an event $e \in E$ with activity a, i.e., a potential target event. The new state is $S = \langle E, C' \rangle$, where if ψ has an activation, and e' is the activation event that caused the missing target, then $C' = C \cup \{Ord(\psi, e', e)\} \cup \{[c_{act}](e'), [c_{tgt} \wedge c_{cor}](e', e)\}$; otherwise, $C' = C \cup \{Ord(\psi, e), [c_{tgt}](e)\}$.

(7) *Removing a blocking event.* This applies if an event $e_t \in E$ with activity a is according to the ordering conditions in C not in the right position to act as target for ψ, but removing another event e from the trace can make room for e_t. E.g., Init constraints delete the first event, and ChainResponse constraints remove the events directly succeeding the activation. The resulting state is $S' = \langle E \setminus \{e\}, C' \rangle$ where C' is like C with conditions on e removed.

If ψ has an excessive target, we can either remove an excessive target event, or change the data conditions such that an event with target activity no longer acts as a target. Precisely, the following fixes apply:

(8) *Removing excessive target events.* This works like (1) above, but removes excessive target events if they stem from the trace.

(9) *Freeing a data attribute.* This repair is similar to (2) above, but it applies if there is an excessive target event $e = (\iota, a, \alpha)$ in E that stems from the input trace. However, we now enforce the negation of target and correlation conditions. The resulting state is $S' = \langle E', C' \rangle$ where E' is like E but where e is modified to (ι, a, α') such that α' is like α except that it is undefined for $v \in V$. Let $\widehat{C} = C \setminus \{v_\iota = \alpha(v)\}$. If there is an activation event e', we set $C' = \widehat{C} \cup \{\neg[c_{tgt} \wedge c_{cor}](e', e)\}$; otherwise, $C' = \widehat{C} \cup \{\neg[c_{tgt}](e)\}$.

(10) *Enforcing negated conditions.* Let $e \in E$ be an excessive target event. There are two resulting states $S' = \langle E, C' \rangle$ and $S'' = \langle E, C'' \rangle$. If there is an activation event e', then $C' = C \cup \{\neg[c_{tgt} \wedge c_{cor}](e', e)\}$; otherwise, $C' = C \cup \{\neg[c_{tgt}](e)\}$. Moreover, $C'' = C \cup \{\neg Ord(e', e)\}$.

Note that *all* applicable repairs are applied in all possible ways. For instance, when freeing a data attribute, a new state is generated for every event e and every variable assignment in e that satisfies the conditions in (2). Also, if there

is a missing target violation and φ is a Choice constraint having two targets, a child state is created for each possible target.

For example, in Fig. 2, S_1 is obtained from S_0 by adding a target event $\#_3$ (repair (4)); S_3 is obtained from S_0 by removing the activation event $\#_1$ (repair (1)); S_7 is obtained from S_1 by freeing the data attribute x in event $\#_2$ (repair (2)); and S_2 is obtained from S_1 by forcing conditions on event $\#_3$ (repair (6)).

4.4 A*-based search

Starting from the initial state that represents the input trace $\mathbf{e}$, our algorithm subsequently chooses a state with a violation and generates child states by applying all possible repairs. By a *search space* for $\mathbf{e}$ and $\mathcal{M}$, we mean below a graph of states where the root is S_0, and all states have as children the states obtained by all possible repairs, if any. To guide the search, the A* algorithm maintains for each state S a cost $cost(S) \in \mathbb{R}$, which can be shown to match exactly the cost of alignments extracted from S. The initial state has cost 0. When expanding a state S, the cost of a child state S' is determined by the applied repair: when adding or removing events, or freeing a data attribute, we have $cost(S') = cost(S) + 1$; when forcing condition satisfaction, $cost(S') = cost(S)$. In Fig. 2, each state is labelled with its respective cost.

Since repairs result in child states of increased cost, by a fair exploration of the search space, the A* algorithm can conclude at some point that all goal states that might possibly be detected in the future will have a higher or equal cost than the goal states found so far. At this point the search terminates, returning a goal state S_g with minimal cost K.

4.5 Alignment extraction

From a goal state $S = \langle E, C \rangle$, we extract an alignment as described by the pseudocode in Alg. 1. The first step is to obtain an SMT model μ of the conditions $\bigwedge C$. This induces a list of model events $\mathbf{f} = \langle f_0, \ldots, f_{m-1} \rangle$ that satisfies all ordering conditions, and where for each $f_j = (\iota_j, a_j, \alpha_j)$ the assignment α_j is given by $\alpha_j(v) = \mu(v_{\iota_j})$ for all $0 \leq j < m$.

Alg. 1 then walks simultaneously along $\mathbf{e}$ and $\mathbf{f}$, using i as an index for $\mathbf{e}$ and j for $\mathbf{f}$, and adds an edit or synchronous move if the current events e_i and f_j share the same id (so f_j stems from a trace event e_i), a model move if the id of the model event f_j does not occur in $\mathbf{e}$, and otherwise a log move. (For an event $e = (\iota, a, \alpha)$, we write $e.id$ to refer to ι.) In Line 6, the number of mismatching assignments in e_i and f_j determines whether the move is an edit or synchronous move. Note that the alignment extracted from a state is in general not unique as there can be multiple SMT models. For instance, by applying Alg. 1 to state S_2 resp. state S_6 in Fig. 2, one obtains the alignments γ_1 resp. γ_2 shown in Ex. 3.

Our correctness result below shows that the alignment extracted from S is optimal with cost K (cf. the proof in Sec. A.2). Our running example illustrates this result: in Fig. 2, the goal state with minimal cost is S_2, and indeed the optimal alignment γ_1 is extracted from it (cf. Ex. 3).

Algorithm 1 Extracting an alignment from a state

Require: State $S = \langle E, C \rangle$, trace $\mathbf{e} = \langle e_0, \ldots, e_{n-1} \rangle$
Ensure: Alignment for $\mathbf{e}$
1: $model \leftarrow$ SMT model of formula $\bigwedge C$
2: $\langle f_0, \ldots, f_{m-1} \rangle \leftarrow$ sort events in E by assignment to ordering conditions in $model$
3: $moves \leftarrow []$, $i \leftarrow 0$, $j \leftarrow 0$
4: **while** $(i < n) \vee (j < m)$ **do**
5: **if** $(i < n) \wedge (j < m) \wedge (e_i.id = f_j.id)$ **then**
6: $moves.append(editOrSynchronousMove(e_i, f_j))$
7: $i \leftarrow i + 1$, $j \leftarrow j + 1$
8: **else if** $(j < m)$ and $f_j.id$ does not occur in $\mathbf{e}$ **then**
9: $moves.append(modelMove(f_j))$
10: $j \leftarrow j + 1$
11: **else**
12: $moves.append(logMove(e_i))$
13: $i \leftarrow i + 1$
14: **return** $moves$

Theorem 1 (Correctness). *If S is a goal state with minimal cost K in a search space for $\mathcal{M}$ and $\mathbf{e}$ then the list of moves γ returned by Alg. 1 on input S and $\mathbf{e}$ is an optimal alignment of $\mathbf{e}$ wrt. $\mathcal{M}$ with cost K.*

5 Implementation

Our approach has been implemented in the tool DADA written in Kotlin, using the SMT solvers Z3 [12] and Yices [13] as backends. DADA requires two inputs: a multi-perspective event log in XES format [24] and a Declare specification with data $\mathcal{M}$. The model format is backward compatible with the one of [2]. Nevertheless, the syntax for data conditions has been significantly enhanced, allowing users to leverage the full expressiveness of the SMT-LIB2 language [1]. Also, the cost function can be customized, providing the cost of log, model, and edit moves as inputs. The Declare constraints language supported by our approach is, in fact, more expressive than initially introduced in Sec. 3. Specifically, branching in Declare constraints is enabled, as described in [7], and all Declare templates listed in [7, Tab. 2] have been implemented. The tool produces an optimal alignment in a human-readable format, similar to Ex. 3. It can also export the search space as a graph, like in Fig. 2.

Encoding. The SMT solver reasons on control flow and data dependencies in tandem, to identify violations and possible repairs for each constraint. We thus need to check satisfiability of formulas that mix ordering and data conditions. Data conditions as in Def. 1, but also much richer conditions, can be directly expressed in SMT-LIB2. Ordering conditions on a set E are encoded as follows: for every event $e \in E$ we use the SMT variable τ_e of integer type that encodes the event's timestamp. Then an ordering constraint $e_1 < e_2$ is directly translated to $\tau_{e_1} < \tau_{e_2}$; $first(e)$ is translated to $\bigwedge_{e' \in E \setminus \{e\}} \tau_e < \tau_{e'}$ and similar for $last(e)$; and $e \ll e'$ is translated to $\tau_{e_1} < \tau_{e_2} \wedge \bigwedge_{e \in E \setminus \{e_1, e_2\}} (\tau_e > \tau_{e_2} \vee \tau_e < \tau_{e_1})$. A constraint

$e_1 <^a_{[c]} e_2$ is translated to $\tau_{e_1} < \tau_{e_2} \wedge \bigwedge_{e \in E_a} (\neg[c](e) \vee \tau_e > \tau_{e_2} \vee \tau_e < \tau_{e_1})$, where E_a is the set of all events in E with activity a, with e_1 and e_2 excluded. Moreover, for efficiency, we use the SMT solver's assumption mechanism to temporarily check conditions, such as those in Defs. 9 and 10. This approach allows us to assert temporary assumptions on top of a core set of formulas, avoiding unnecessary re-computations and improving performance.

Optimizations. We mention the most influential optimizations. The first is *selecting a violation to repair:* as violations can be processed independently, the implementation selects for repair the one that generates the fewest child states, to delay the state explosion. helps avoid state explosion.

The second optimization is about *detecting dead-ends:* in every state, all violations are precomputed, and for every violation it is checked which repairs are applicable. In case no repair is applicable for some violation, the state is a dead end, and the branch can be pruned from the search space.

Finally, we *prune unsatisfiable states.* If a state $S = \langle E, C \rangle$ was generated where $\bigwedge C$ is unsatisfiable, conflicting conditions were added while generating the state. Therefore, the state can be dropped from the search space.

6 Evaluation

In our evaluation we execute all tested algorithms in the same environment, namely a Java Virtual Machine[5] run on an Intel 5220R CPU with 8 GB of RAM. The source code, executable, dataset and raw results are publicly available.[6]

Dataset. The evaluation utilizes a synthetic dataset that systematically varies in complexity, originally introduced in [2]. The complexity of the process models is influenced by the number of constraints (3, 5, 7, or 10) and constraint modifications (replacing 0, 1, 2, or 3 constraints). For each model, multiple event logs with varying trace lengths were generated (10, 15, 20, 25, or 30 events), resulting in 68,000 trace-model pairs. The models feature simple variable-to-constant conditions over `categorical` (with values c1, c2, or c3) and `integer` (ranging from 0 to 100), such as `categorical is c1` or `integer > 10`.

Performance comparison. We compare DADA, using either the Z3 [12] SMT solver or the Yices [13] SMT solver, to Bergami2021 [2], using the original SymBA* [23] planner or the Fast Downward [16] planner. Our experiments measure the execution time for each pair of model and trace. To ensure all alignments are optimal, we validate that the alignment costs produced by DADA-Z3 and DADA-Yices match those generated by Bergami2021-BA and Bergami2021-FD.

Fig. 3 shows how, as the complexity of the trace-model pairs increases, the state-of-the-art algorithms exhibit a sharp increase in execution times, whereas our approach demonstrates better scalability. Notably, our approach aligns any trace-model pair in at most 5 seconds, and DADA-Z3 is on average 2.9 times faster than Bergami2021-FD and 5.9 times faster than Bergami2021-BA

[5] OpenJDK 64-Bit VM Temurin 21.0.6+7-LTS

[6] https://apps.citius.gal/dada and https://doi.org/10.5281/zenodo.15470077

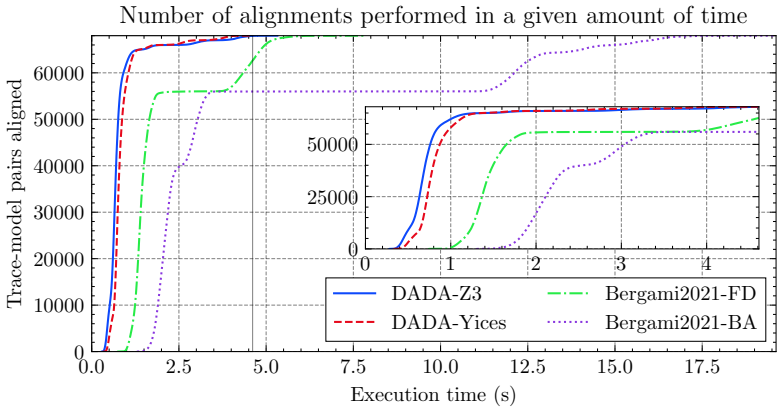

Fig. 3: Number of trace-model pairs aligned within a given time frame by each algorithm. The enlargement on the right highlights the differences for shorter time intervals.

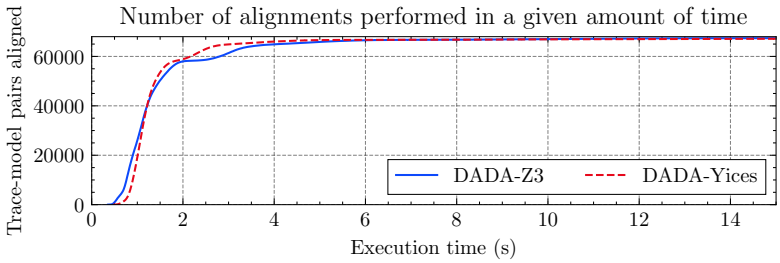

Fig. 4: Performance evaluation incorporating correlation constraints.

Constraint flexibility. While the previous experiment was limited to the data conditions supported by [2], our approach can leverage the power of SMT solvers to define complex data dependencies such as the following correlation conditions. In these conditions, `A` refers to the activation and `T` to the target; `cat` is an abbreviation for the `categorical` attribute, and `timestamp` is the event's time.

(C1) `A.timestamp + A.integer * 1d > T.timestamp + T.integer * 1d`
(C2) `(A.cat - "0") % 10 < (T.cat - "0") % 10`
(C3) `(A.cat == T.cat) ? (T.cat % 2 == 0) : (A.integer > T.integer)`

We create a new dataset by adding random negations, disjunctions and conjunctions of the previous correlation conditions to the original constraints, while retaining the original activation and target conditions, resulting in models like:
`Response[activity 1, activity 2]|...|...|¬(¬C1 or ¬(¬C3 and ¬C2))|`
`Chain Response[activity 3, activity 4] |...|...|¬C1 or ¬C2 or C3|`

The added correlation conditions make the alignment problem even harder by potentially increasing (a) the number of repairs required to reach the optimal alignment, (b) the number of ways in which it is possible to repair them, and (c) the work performed by the SMT solver within each state. For these reasons, the models were simplified by only considering the ceiling of half of the con-

straints generated in this way. Fig. 4 shows that our approach can handle these advanced conditions, with only a small percentage of alignments timing out or running out of memory (0.06% for DADA-Yices and 0.91% for DADA-Z3).

7 Conclusions

This paper presents a novel approach to computing data-aware optimal alignments between event logs and declarative process models, combining A* search and SMT solvers. Our key contributions include a new encoding scheme for the control flow, using an SMT solver to reason about control flow and data conditions, and an efficient A*-based search strategy that resolves constraint violations through repair actions. Notably, the available constraint language is much richer than in earlier work, including a wide range of constructs supported by current SMT solvers. We prove its correctness and demonstrate its efficiency in experiments, matching or surpassing state-of-the-art performance while supporting more expressive data dependencies. Future work includes exploring further optimizations such as advanced pruning strategies and heuristic functions.

Acknowledgments. This work was partially funded by the Spanish Ministerio de Ciencia [grant numbers PID2020-112623GB-I00, PID2023-149549NB-I00, TED2021-130374B-C21], co-funded by the European Regional Development Fund (ERDF). J. Casas-Ramos gratefully acknowledges the support of CiTIUS for funding his research stay. M. Montali was partially supported by the NextGenerationEU FAIR PE0000013 project MAIPM (CUP C63C22000770006) and the PRIN MIUR project PINPOINT Prot. 2020FNEB27. A. Gianola was partly supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT), under project (DOI: 10.54499/UIDB/50021/2020). This work was partially supported by the 'OptiGov' project (DOI: 10.54499/2024.07385.IACDC), fully funded by the 'Plano de Recuperação e Resiliência' (PRR) under the investment 'RE-C05-i08 - Ciência Mais Digital' (measure 'RE-C05-i08.m04'), framed within the financing agreement signed between the 'Estrutura de Missão Recuperar Portugal' (EMRP) and FCT as an intermediary beneficiary.

Disclosure of Interests. The authors have no competing interests to declare.

References

1. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard: Version 2.7. Tech. rep., University of Iowa (2025)
2. Bergami, G., Maggi, F.M., Marrella, A., Montali, M.: Aligning data-aware declarative process models and event logs. In: 19th BPM. pp. 235–251. LNCS (2021)
3. Borrego, D., Barba, I.: Conformance checking and diagnosis for declarative business process models in data-aware scenarios. Expert Syst. Appl. **41**, 5340–5352 (2014)
4. Bose, R.P.J.C., van der Aalst, W.M.P.: Process diagnostics using trace alignment: Opportunities, issues, and challenges. Inf. Syst. **37**(2), 117–141 (2012)
5. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
6. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)

7. Casas-Ramos, J., Lama, M., Mucientes, M.: DeclareAligner: A leap towards efficient optimal alignments for declarative process model conformance checking (2025), arXiv 2503.10479
8. Christfort, A.K.F., Slaats, T.: Efficient optimal alignment between dynamic condition response graphs and traces. In: 21st BPM. LNCS, vol. 14159, pp. 3–19 (2023)
9. De Giacomo, G., Fuggitti, F., Maggi, F.M., Marrella, A., Patrizi, F.: A tool for declarative trace alignment via automated planning. Softw. Imp. **16**, 100505 (2023)
10. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the disruptive effectiveness of automated planning for LTLf-based trace alignment. In: Proc. 31st AAAI. pp. 3555–3561. AAAI Press (2017)
11. De Leoni, M., Van Der Aalst, W.M.: Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In: Proc. 11th BPM. LNCS, vol. 8094, pp. 113–129. Springer (2013)
12. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proc. 14th TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
13. Dutertre, B.: Yices 2.2. In: Proc. 14th CAV. LNCS, vol. 8559, pp. 737–744 (2014)
14. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: Cocomot: Conformance checking of multi-perspective processes via SMT. In: Proc. 21st BPM. LNCS, vol. 12875, pp. 217–234 (2021)
15. Felli, P., Montali, M., Patrizi, F., Winkler, S.: Monitoring arithmetic temporal properties on finite traces. In: Proc. 37th AAAI. pp. 6346–6354. AAAI Press (2023)
16. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research **26**, 191–246 (2006)
17. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: Aligning event logs and declarative process models for conformance checking. In: Proc. 10th BPM. LNCS, vol. 7481, pp. 82–97 (2012)
18. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. Inf. Syst. **47**, 258–277 (2015)
19. Mannhardt, F., de Leoni, M., Reijers, H., van der Aalst, W.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
20. Nagy, Z., Werner-Stark, A.: An alignment-based multi-perspective online conformance checking technique. Acta Polytechnica Hungarica **19**(4), 105–127 (2022)
21. Riva, F., Benvenuti, D., Maggi, F.M., Marrella, A., Montali, M.: An SQL-based declarative process mining framework for analyzing process data stored in relational databases. In: Proc. BPM Forum. LNBIP, vol. 490, pp. 214–231 (2023)
22. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: Proc. 28th CAiSE. LNCS, vol. 9694, pp. 290–305. Springer (2016)
23. Torralba, A., Alcázar, V., Borrajo, D., Kissmann, P., Edelkamp, S.: SymBA*: A symbolic bidirectional A* planner. In: Planning Competition. pp. 105–108 (2014)
24. XES Working Group: IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams. IEEE Std 1849-2023 (2023)

A Appendix

In this appendix, we first define formally the semantics of Declare constraints with data, and then provide a formal correctness proof of our approach.

A.1 Semantics of Declare with Data

The following definition clarifies when a trace satisfies Declare constraints with data. For an assignment α with domain V, we write α^a for some $a \in \mathcal{A}$ for the same assignment on labeled variables, i.e., the assignment with domain $\{v^a \mid v \in V\}$ that sets $\alpha^a(v^a) = \alpha(v)$. Moreover, we write $\alpha \models c$ to express that α satisfies a condition c. For the union of two assignments α, β with disjoint domain we write $\alpha \cup \beta$.

Definition 11. *A constraint* $\psi = \langle \varphi, c_{act}, c_{tgt}, c_{cor} \rangle$ *is satisfied by a trace* $\mathbf{e} = \langle e_0, \ldots, e_{m-1} \rangle$ *if*

- $\varphi = \text{Existence}(n, a)$*, there are* n *distinct events* $e_{i_1}, \ldots, e_{i_n}$ *in* $\mathbf{e}$ *such that for all* $1 \leq j \leq n$ *and if* e_{i_j} *has the form* $e_{i_j} = \langle \iota_j, a, \alpha_j \rangle$ *it holds that* $\alpha_j^a \models c_{tgt}$*;*
- $\varphi = \text{Absence}(n, a)$*, and* $\mathbf{e}$ *does not satisfy* $\langle \text{Existence}(n, a), c_{act}, c_{tgt}, c_{cor} \rangle$*;*
- $\varphi = \text{Init}(a)$*,* $e_0 = \langle \iota, a, \alpha \rangle$*, and* $\alpha^a \models c_{tgt}$*;*
- $\varphi = \text{End}(a)$*,* $e_{m-1} = \langle \iota, a, \alpha \rangle$*, and* $\alpha^a \models c_{tgt}$*;*
- $\varphi = \text{Choice}(a, b)$*, and there is some* $e_i = \langle \iota, d, \alpha \rangle$*,* $1 \leq i < m$*, such that* $d = a$ *and* $\alpha^a \models c_{tgt}$*, or* $d = b$ *and* $\alpha^b \models c_{tgt}$*;*
- $\varphi = \text{RespondedExistence}(a, b)$*, and either there is no* $e_i = \langle \iota, a, \alpha \rangle$*,* $0 \leq i < m$*, such that* $\alpha^a \models c_{act}$*, or there is some* $e_j = \langle \iota', b, \beta \rangle$*, with* $0 \leq j < m$ *and* $i \neq j$*, such that* $\alpha^a \cup \beta^b \models c_{tgt} \wedge c_{cor}$*;*
- $\varphi = \text{Response}(a, b)$*, and either there is no* $e_i = \langle \iota, a, \alpha \rangle$*,* $0 \leq i < m$*, such that* $\alpha^a \models c_{act}$*, or there is some* $e_j = \langle \iota', b, \beta \rangle$*, with* $i < j < m$*, such that* $\alpha^a \cup \beta^b \models c_{tgt} \wedge c_{cor}$*;*
- $\varphi = \text{AlternateResponse}(a, b)$*, and either there is no* $e_i = \langle \iota, a, \alpha \rangle$*,* $0 \leq i < m$*, such that* $\alpha^a \models c_{act}$*, or there is some* $e_j = \langle \iota', b, \beta \rangle$*, with* $i < j < m$*, such that* $\alpha^a \cup \beta^b \models c_{tgt} \wedge c_{cor}$*, and for all* e_k *with* $i < k < j$ *of the form* $e_k = \langle \iota', d, \alpha_k \rangle$ *either* $d \neq a$ *or* $\alpha^a \not\models c_{act}$*;*
- $\varphi = \text{ChainResponse}(a, b)$*, and either there is no* $e_i = \langle \iota, a, \alpha \rangle$*,* $1 \leq i \leq n$*, such that* $\alpha^a \models c_{act}$*, or* $i < m-1$ *and* $e_{i+1} = \langle \iota', b, \beta \rangle$ *and* $\alpha^a \cup \beta^b \models c_{tgt} \wedge c_{cor}$*;*
- $\varphi = \text{Precedence}(a, b)$*, and either there is no* $e_i = \langle \iota, b, \alpha \rangle$*,* $0 \leq i < m$*, such that* $\alpha^b \models c_{act}$*, or* $i > 0$ *and there is some* $e_j = \langle \iota', a, \beta \rangle$*, with* $0 \leq j < i$*, such that* $\alpha^b \cup \beta^a \models c_{tgt} \wedge c_{cor}$*;*
- $\varphi = \text{AlternatePrecedence}(a, b)$*, and either there is no* $e_i = \langle \iota, b, \alpha \rangle$*,* $0 \leq i < m$*, such that* $\alpha^b \models c_{act}$*, or* $i > 0$ *and there is some* $e_j = \langle \iota', a, \beta \rangle$*, with* $0 \leq j < i$*, such that* $\alpha^b \cup \beta^a \models c_{tgt} \wedge c_{cor}$*, and for all* e_k *with* $j < k < i$ *of the form* $e_k = \langle \iota', d, \alpha_k \rangle$ *either* $d \neq b$ *or* $\alpha^b \not\models c_{act}$*;*
- $\varphi = \text{ChainPrecedence}(a, b)$*, and either there is no* $e_i = \langle \iota, b, \alpha \rangle$*,* $0 \leq i < m$*, such that* $\alpha^b \models c_{act}$*, or* $i > 0$ *and* $e_{i-1} = \langle \iota', a, \beta \rangle$ *and* $\alpha^b \cup \beta^a \models c_{tgt} \wedge c_{cor}$*;*
- $\varphi = \text{NotResponse}(a, b)$*, and* $\mathbf{e}$ *does not satisfy* $\langle \text{Response}(n, a), c_{act}, c_{tgt}, c_{cor} \rangle$*;*
- $\varphi = \text{NotRespondedExistence}(a, b)$*, and* $\mathbf{e}$ *does not satisfy the constraint* $\langle \text{RespondedExistence}(n, a), c_{act}, c_{tgt}, c_{cor} \rangle$*; or*

– $\varphi = \text{NotChainResponse}(a, b)$, *and trace* $\mathbf{e}$ *does not satisfy the constraint* $\langle \text{ChainResponse}(n, a), c_{act}, c_{tgt}, c_{cor} \rangle$.

A.2 Correctness Proof

The following is straightforward to show by a case distinction on ψ.

Lemma 1. *Let* $\psi = (\varphi, c_{act}, c_{tgt}, c_{cor})$ *be a constraint. A state* S *does not violate* ψ *if and only if for all* $\gamma \in \Gamma(S)$*, it holds that* $\gamma|_M$ *satisfies* ψ.

We next show that if Alg. 1 returns γ on input S and $\mathbf{e}$ then γ is indeed an alignment for $\mathbf{e}$, though in general only wrt. a subset of $\mathcal{M}$, and the cost of γ coincides with $cost(S)$. Below, for a fixed trace $\mathbf{e}$, we denote by $\Gamma(S)$ the set of alignments that can be extracted from S, i.e., that are possible results of Alg. 1 on input S (recall that the alignment extracted from a state S is in general not unique).

Below, we make the following assumption (†) on the search space generated for $\mathcal{M}$ and $\mathbf{e}$: Repairs that remove an event e are only applied if e was never modified beforehand by a repair that changes the assignment.

Lemma 2 (Soundness). *For each* $\gamma \in \Gamma(S)$*, it holds that* $\gamma|_L = \mathbf{e}$ *and* $\kappa(\gamma) = cost(S)$.

Proof. First of all, it is easy to see that if $S = \langle E, C \rangle$ and $\gamma \in \Gamma(S)$ then $\gamma|_L = \mathbf{e}$ because Alg. 1 adds for every e_i in $\mathbf{e}$ a log or synchronous/edit move, since i is incremented from 0 to $n - 1$.

It remains to show the claim about the cost of γ, which we prove by induction on the depth n at which S occurs in the search tree. The statement holds for the initial state S_0, where an alignment $\gamma \in \Gamma(S)$ consists of only synchronous moves, so $\kappa(\gamma) = 0 = cost(S_0)$.

Let $S' = \langle E', C' \rangle$ be at depth $n+1$ in the search tree. The induction hypothesis is that the claim holds for all states at level n. We perform a case distinction on the repair applied at the parent $S = \langle E, C \rangle$ of S' to create S'.

1. If an event e was added, then e has a fresh id that does not occur in the trace. Each alignment $\gamma' \in \Gamma(S')$ stems from some model μ for C'. By construction of the repair (the constraints remain the same), μ also satisfies C, giving rise to an alignment $\gamma \in \Gamma(S)$. By the induction hypothesis, γ is an alignment of $\mathbf{e}$ with cost $cost(S)$. Alignment γ' must be like γ except for an additional model move (as the added id is fresh, it cannot match an event in the trace), so that $\kappa(\gamma') = \kappa(\gamma) + 1$. Since we have $cost(S') = cost(S) + 1$ and $\kappa(\gamma) = cost(S)$ by the induction hypothesis, the claim holds.
2. If an event was removed, then this event stems from the trace, and was not modified beforehand by assumption (†). Each alignment $\gamma' \in \Gamma(S')$ stems from some assignment μ' for C'. Ordering conditions and data conditions are independent. Thus there is an assignment μ that coincides with μ' on ordering constraints and assigns arbitrary data values compatible with C. Even though data values differ, for all moves except for the one concerning

the removed event, their costs coincide, as required data values are enforced by dedicated conditions. The corresponding alignment $\gamma \in \Gamma(S)$ has a synchronous move where γ' has a log move, so $cost(\gamma') = cost(\gamma) + 1$. Since $cost(S') = cost(S) + 1$ and $cost(\gamma) = cost(S)$ by induction hypothesis, the claim holds.

3. Suppose a data attribute v in an event $e = (\iota, a, \alpha)$ was freed. Each alignment $\gamma' \in \Gamma(S')$ stems from some assignment μ' for C'. Ordering conditions and data conditions are independent. Thus there is an assignment μ for C that coincides with μ' on ordering constraints and assigns data values compatible with C, in particular $\mu(v^\iota) = \alpha(v)$. Even though data values differ, for all moves except for the one concerning e, their costs coincide, as required data values are enforced by dedicated conditions. Since $cost(S') = cost(S) + 1$ and $cost(\gamma) = cost(S)$ by induction hypothesis, the claim holds.
4. Suppose conditions were enforced. Then any model of C is also a model of C' as $C' \subseteq C$, and each alignment $\gamma \in \Gamma(S)$ that stems from some model μ for C is also an alignment in $\Gamma(S)$. Since $cost(S) = cost(S')$, the claim follows from the induction hypothesis. □

Let two alignments γ and γ' for $\mathbf{e}$ and $\mathcal{M}$ be *equivalent* if $|\gamma| = |\gamma'|$ and the move in γ at position i is a log/model/synchronous move if so is the move in γ' at position i, and where edit moves edit the same variables. However, variable values in the model component of γ and γ' that do not match a variable in a trace may differ, as well as event identifiers.

Lemma 3 (Completeness). *Let γ be an optimal alignment for $\mathbf{e}$ and $\mathcal{M}$. Then there is a goal state S_g in the search space such that $\Gamma(S_g)$ contains an alignment equivalent to γ.*

Proof. Let γ be an optimal alignment for $\mathbf{e} = \langle e_1, \ldots, e_n \rangle$ and $\mathcal{M}$. Let $\gamma|_M = \mathbf{f} = \langle f_0, \ldots, f_m \rangle$.

We show that there is a sequence of states $S_0, S_1, S_2, \ldots, S_g$ in the search tree such that, intuitively, by descending along this path the respective alignment gets closer and closer to γ, and S_g is a goal state that satisfies the claim. More precisely, we show that for each state $S_i = \langle E, C \rangle$, there is a correspondence relation $R_{S_i} \subseteq E \times \{f_0, \ldots, f_m\}$ associating some of its events with events in $\mathbf{f}$ that satisfies the following invariants:

(*i*) For every pair $(e, f) \in R_{S_i}$, the two events have the same activity.

(*ii*) For all events $e \in E \setminus \{e_1, \ldots, e_n\}$, there is some f_j in $\mathbf{f}$ such that $(e, f_j) \in R_{S_i}$. That is, all added events have a correspondent in $\mathbf{f}$. Moreover, for all edit or synchronous moves (e, f_j) in γ, f_j has a match in R_{S_i}.

(*iii*) Let $E_R \subseteq E$ be the set of events $\{e \in E \mid \exists f.(e, f) \in R_{S_i}\}$, μ the (partial) assignment on $vars(C)$ that sets, for all $(e, f) \in R_S$ with ι the id of e, $\mu(\tau^\iota) = time(f)$ and $\mu(x^\iota) = \alpha(x)$, where $f = (\iota', a, \alpha)$. Then μ satisfies $(C \setminus C_0)|_{V_R}$ where V_R is the union of all V^ι such that ι is an id of an event in E_R.

(*iv*) Only attributes of events are freed that have via R_S a correspondent in $\mathbf{f}$.

Moreover, the sequence of relations is monotonically increasing, i.e., we have $R_{S_0} \subseteq R_{S_1} \subseteq R_{S_2} \subseteq \dots$.

We first show existence of a sequence $S_0, S_1, S_2, \dots$ that satisfies the invariants, then we reason that it contains a goal state S_g that satisfies the claim.

At depth $k = 0$, we have $S = S_0$ and $E = \{e_1, \dots, e_n\}$. Let R_{S_0} consist of all pairs (e_i, f_j) such that (e_i, f_j) is a synchronous or edit move in γ. The relation R_{S_0} satisfies (i) because in edit and synchronous moves the activity is shared. Item (ii) and (iii) are vacuously satisfied as no events were added, and (iv) because no attributes were freed.

Consider now a state $S = \langle E, C \rangle$ at depth k. If S is a goal state, we are done, so we assume that S has a violation. Suppose S gets repaired for a constraint $\psi = \langle \varphi, c_{act}, c_{tgt}, c_{cor} \rangle \in \mathcal{M}$. Consider first the case of a missing target violation; by a case distinction, we decide a next state S'.

1. Suppose first that ψ does not have an activation. For simplicity, we consider the case of a single target, but the case for Existence is similar. Let f_j be the target of ψ in $\mathbf{f}$.
   (a) If there exists some $e = (\iota, a, \alpha) \in E$ such that $(e, f_j) \in R$ then e and f_j must have the same activity by (i), and we must have $C \not\models \mathit{Ord}(\psi, e) \wedge [c_{tgt}](e)$ $(\star)$, otherwise there would be no violation. Let $f_j = (\iota', a, \alpha')$.
      – Suppose there is some $v \in \mathit{dom}(\alpha)$ such that $\alpha(v) \neq \alpha'(v)$. This is only possible if e stems from the trace, since added events have no fixed values. Thus, let S' be the child state obtained from a repair (5) where attribute v was freed.
      – Otherwise, if $\bigwedge C \wedge \mathit{Ord}(\psi, e) \wedge [c_{tgt}](e)$ is satisfiable, let S' be the result of a condition enforcement repair (6). The repair is applicable because as observed above e and f_j have the same activity, and the resulting state is satisfiable because $C \wedge \mathit{Ord}(\psi, e) \wedge [c_{tgt}](e)$ is satisfiable.
      – Otherwise, $\bigwedge C \wedge \mathit{Ord}(\psi, e) \wedge [c_{tgt}](e)$ is not satisfiable. Since μ satisfies $(C \setminus C_0)|_{V_R}$ and μ satisfies $\mathit{Ord}(\psi, e) \wedge [c_{tgt}](e)$, there must be some event $e' \in E$ from the trace, i.e. which adds conditions to C_0, that causes unsatisfiability. In fact, as assignments in e and f_j have no mismatches, $\bigwedge C \wedge \mathit{Ord}(\psi, e)$ must be unsatisfiable. Precisely, $\mathit{Ord}(\psi, e)$ must be $\mathit{first}(e)$ (resp. $\mathit{last}(e)$), but C contains $e' < e$ (resp. $e' > e$) for some trace event e'. Then e' cannot have a match in R_S because μ satisfies $C \setminus C_0$ restricted to $E|_{R_S}$ by (iii). Hence we can apply repair (7) to remove e', obtaining a state S'. Note that by invariant (iv), no attribute in e' can have been freed because it has no correspondent in R_S.

      In all these cases R_S stays the same and thus satisfies conditions $(i) - -(iii)$. Moreover, in the first case the freed attribute belongs to an event that has a correspondent in R_S, so also (iv) is satisfied.
   (b) Now assume there is no match for f_j in R. Then f_j cannot be in a synchronous or edit move in γ by invariant (ii). So f_j is in a model move, thus let S' be the state obtained from a repair (4) where a target

event e was added. Set $R_{S'} = R_S \cup \{(e, f_j)\}$, which satisfies the invariants $(i) - (iii)$, and (iv) is satisfied because no additional attribute is freed.

2. Now suppose ψ has an activation, let $e_{act} \in E$ be the activation event of the violation.
   (a) Suppose there is some f_j such that $(e_{act}, f_j) \in R$. Suppose first that for $f_j = \langle \iota', a, \alpha' \rangle$, α' does not satisfy c_{act}.
      – If there is some $v \in dom(\alpha)$ such that $\alpha(v) \neq \alpha'(v)$ then let S' be the child state obtained from a repair (2) where attribute v was freed.
      – Otherwise, let S' be the result of a condition enforcement repair (3).

   Second, suppose α' satisfies c_{act}, so it must have a target f_k in $\mathbf{f}$, $k \neq j$.
      i. If there exists some $e_{tgt} = (\nu, b, \beta) \in E$ such that $(e_{tgt}, f_k) \in R_S$ then we must have $C \not\models Ord(\psi, e_{act}, e_{tgt}) \wedge [c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})$ $(\star)$, otherwise there would be no violation. Let $f_k = (\nu', b, \beta')$.
         – If there is some $v \in dom(\beta)$ such that $\beta(v) \neq \beta'(v)$ then e_{tgt} must stem from the trace. Let S' be the child state obtained from a repair (5) where attribute v was freed.
         – Otherwise, if $\bigwedge C \wedge Ord(\psi, e_{act}, e_{tgt}) \wedge [c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})$ is satisfiable, let S' be the result of a condition enforcement repair (6). The repair is applicable because of $(\star)$, and as $\bigwedge C \wedge Ord(\psi, e_{act}, e_{tgt}) \wedge [c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})$ is satisfiable, also the resulting state is satisfiable.
         – Otherwise, if $\bigwedge C \wedge Ord(\psi, e_{act}, e_{tgt}) \wedge [c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})$ is unsatisfiable, this must be because of some trace events in E that have no correspondent in $\mathbf{f}$ via R_S, since $\bigwedge (C \setminus C_0)|_{V_R} \wedge Ord(\psi, e_{act}, e_{tgt}) \wedge [c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})$ is satisfied by μ according to property (iii). In fact, as assignments in e and f_j have no mismatches (this was already excluded above), $\bigwedge C \wedge Ord(\psi, e_{act}, e_{tgt})$ must be unsatisfiable. Precisely, $Ord(\psi, e_{act}, e_{tgt})$ must be $e_{act} \ll e_{tgt}$ but C contains $e_{act} < e'$ and $e' < e_{tgt}$ (or similar for precedence) for some trace event e'. Then e' cannot not have a match in R_S, and apply repair (7) to remove e', obtaining a state S'. Note that by (iv) no attribute in e' has ever been freed because e' has no correspondent in R_S.
      ii. Now assume there is no match for f_k in R_S. Then f_k cannot be in a synchronous or edit move in γ by property (ii). So f_k is in a model move. Let S' be the state obtained from a repair (4) where a target event e was added. Set $R_{S'} = R_S \cup \{(e_{tgt}, f_k)\}$, which satisfies the invariants.
   (b) Suppose there is no $(e_{act}, f_j) \in R_S$. By invariant (ii), e_{act} stems from the trace. We then apply the repair (1) to remove the activation event e_{act}, obtaining a state S'. Note that the activation event cannot have been modified, otherwise there would be a match in R_S by property (iv).

It can be checked that in all cases, the invariants $(i) - (iv)$ remain satisfied.

Second, consider an excessive target violation. Let e_{tgt} be the excessive target event that caused the violation.

3. Suppose first that ψ does not have an activation. For simplicity, we consider the case of a single target, but the case for Absence is similar.
   - Suppose there is some f_j such that $(e_{tgt}, f_j) \in R_S$. Suppose first that for $f_j = \langle \iota', a, \alpha' \rangle$, α' does not satisfy c_{tgt}.
     - If there is some $v \in dom(\alpha)$ such that $\alpha(v) \neq \alpha'(v)$ then e_{tgt} must stem from the trace. Let S' be the child state obtained from a repair (9) where attribute v was freed.
     - Otherwise, let S' be the result of a condition enforcement repair (10).

     It can be checked that in all cases, the invariants $(i) - -(iv)$ remain satisfied.
   - Suppose there is no f_j such that $(e_{tgt}, f_j) \in R_S$. Then e_{tgt} must stem from the trace by property (ii). We apply repair (8) to remove an extra target.
4. Now suppose ψ has an activation, let $e_{act} \in E$ be the activation event of the violation.
   (a) Suppose first there is some f_j such that $(e_{act}, f_j) \in R_S$.
      i. Suppose there is some f_k in $\mathbf{f}$ such that $(e_{tgt}, f_k) \in R_S$. Let $e_{tgt} = (\nu, b, \beta)$ and $f_k = (\nu', b, \beta')$. Since $\mathbf{f}$ satisfies ψ, the assignment μ cannot satisfy $Ord(\psi, e_{act}, e_{tgt}) \wedge [c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})$.
         - If there is some $v \in dom(\beta)$ such that $\beta(v) \neq \beta'(v)$ then e_{tgt} must stem from the trace. Let S' be the child state obtained from a repair (9) where attribute v was freed.
         - Otherwise, we apply a condition enforcement repair (10). If μ does not satisfy $Ord(\psi, e_{act}, e_{tgt})$ then we take the state $S' = \langle E, C' \rangle$ where $C' = C \cup \{\neg Ord(\psi, e_{act}, e_{tgt})\}$. Otherwise, μ does not satisfy $[c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})$, and we take the state $S'' = \langle E, C'' \rangle$ where $C' = C \cup \{\neg[c_{tgt} \wedge c_{cor}](e_{act}, e_{tgt})\}$.
      ii. Suppose there is no f_k in $\mathbf{f}$ such that $(e_{tgt}, f_k) \in R_S$. Then e_{tgt} must stem from the trace. We apply repair (8) to remove the target event.
   (b) Suppose there is no f_j such that $(e_{act}, f_j) \in R_S$. Then e_{act} must stem from the trace. We then apply repair (1) to remove an activation event.

   It can be checked that in all cases, the invariants $(i) - (iv)$ remain satisfied.

This concludes the proof of existence of a sequence $S_0, S_1, S_2, \ldots$.

For $S = \langle E, C \rangle$ a state in this sequence, consider the measure $M(S) = (m - |R_S|, traceEvents(E), bnd(E), viol(S))$, where $|R_S|$ is the number of pairs in R_S, $traceEvents(E)$ is the number of events in E that stem from the trace, $bnd(E)$ the number of bound variables in events in E stemming from the trace, and $viol(S)$ the number of violations in S.

We observe that along the sequence $S_0, S_1, S_2, \ldots$, we have $M(S_0) > M(S_1) > M(S_2) > \ldots$, where we compare tuples lexicographically: Indeed, the measure decreases for repairs where an event was added because we always add an entry to R_S; for all other repairs, R_S stays the same. The measure decreases when removing an event as the number of trace events decreases; while

for all other repairs the number of trace events stays the same. When freeing an attribute, the number of bound variables decreases, and when enforcing constraints, the number of violations decreases (while the number of bound variables is unaffected).

Thus by well-foundedness, we must at some point reach a state $S_g = (E, C)$ with $viol(S_g) = 0$, i.e., a goal state. We show that $m = |R_{S_g}|$: Suppose to the contrary that there is an event f_j in $\mathbf{f}$ with no e such that $(e, f_j) \in R_{S_g}$. There must be a move with f_j in γ, but it cannot be an edit or synchronous move by condition (ii). So it must be a model move. Since μ satisfies $C \setminus \{C_0\}$ restricted to $E|_{R_{S_g}}$, and S_g has no violation, so with Lem. 1 we conclude that also $\langle f_1, \ldots, f_{j-1}, f_{j+1}, \ldots, f_m \rangle$ must satisfy $\mathcal{M}$, which contradicts minimality of γ. Hence $m = |R_{S_g}|$, so every event in $\mathbf{f}$ has a matching event in E. By assumption (iii), assignment μ satisfies all constraints in C. Hence μ can be used in Alg. 1 to obtain an alignment of $\mathbf{e}$ and $\mathcal{M}$ that is equivalent to γ. □

Theorem 1 (Correctness). *If S is a goal state with minimal cost K in a search space for $\mathcal{M}$ and $\mathbf{e}$ then the list of moves γ returned by Alg. 1 on input S and $\mathbf{e}$ is an optimal alignment of $\mathbf{e}$ wrt. $\mathcal{M}$ with cost K.*

Proof. By Lem. 2, every alignment extracted from a goal state is an alignment for $\mathbf{e}$ and $\mathcal{M}$. Moreover, by Lem. 3, for an optimal alignment γ of $\mathbf{e}$ and $\mathcal{M}$ there is some goal state S_g that allows to extract an alignment equivalent to γ, and by Lem. 2 it satisfies $cost(S) = \kappa(\gamma)$. The claim then follows from correctness of A*, i.e., the fact that a state with minimal cost is returned. □

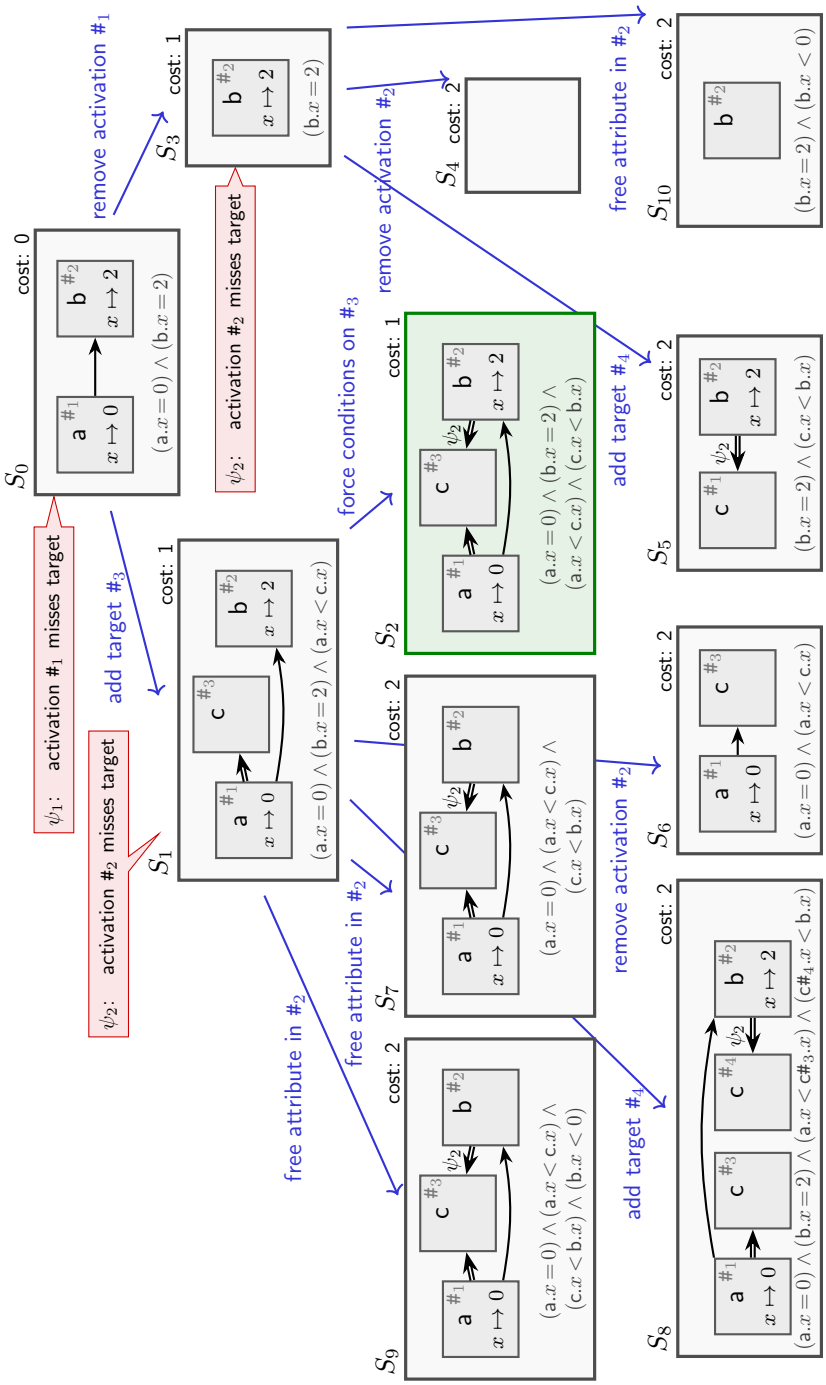

Fig. 5: Complete search space for running example