Knowledge Compilation Using Interval Automata and Applications to Planning

Alexandre Niveau[1] and **Hélène Fargier**[2] and **Cédric Pralet**[1] and **Gérard Verfaillie**[1]

Abstract. Knowledge compilation [7, 5, 17, 9] consists in transforming a problem offline into a form which is tractable online. In this paper, we introduce new structures, based on the notion of *interval automaton* (IA), adapted to the compilation of problems involving both discrete and continuous variables, and especially of decision policies and transition tables, in the purpose of controlling autonomous systems.

Interval automata can be seen as a generalization of binary decision diagrams (BDDs) insofar as they are rooted DAGs with variable-labelled nodes, with the differences that interval automata are non-deterministic structures whose edges are labelled with closed intervals and whose nodes can have a multiplicity greater than two.

This paper studies the complexity of the queries and transformations classically considered when examining a new compilation language. We show that a particular subset of interval automata, the *focusing* ones (FIAs), have theoretical capabilities very close to those of DNNFs; they notably support in polytime the main operations needed to handle decision policies online. Experimental results are presented in order to support these claims.

1 INTRODUCTION

Autonomous systems are required to make decisions automatically, depending on the current observations and goals. Performing the decision-making tasks completely *online*, with the embedded computational capabilities only, can compromise the reactivity of the system. On the other hand, the limited size of embedded memory does not allow to record the potentially huge set of different alternatives (all decisions to be made in every possible situation).

A possible way of solving this contradiction is to use *knowledge compilation*, which consists in transforming offline a problem, thanks to some *target compilation language*, in such a way that its online resolution becomes tractable. In our context of autonomous system control, the offline transformation can be, for example, to directly express the transition relation of the problem in a target language, as well as to solve the problem entirely, retrieve a decision policy, and then express it in a target language. In all cases, the compiled form must be both as compact as possible, so that embedded memory constraints are respected, and as tractable as possible, so that relevant operations (depending on what we need to do) can be quickly processed online.

Efficient target compilation languages were proposed for planning domains involving variables with Boolean or enumerated domains (*e.g.* OBDDs [4], finite-state automata [19], DNNFs [8], etc.). However, in many cases, controlling an autonomous system involves variables with continuous or large enumerated domains, such as time or energy; it would be interesting to represent them without having to discretize them.

All in all, the goal of this paper is to define new target compilation languages, namely the *interval automata* family, suited to this particular application; that is to say, they must be applicable to mixed problems (involving both continuous and discrete features) and support all operations needed. We will focus on the representation and online exploitation of decision policies and transition relations.

We first formally define interval automata in Section 2. We then study several operations in Section 3. The way interval automata can be built is presented in Section 4. Last, experimental results are provided in Section 5. Proofs are gathered in Appendix A.

2 INTERVAL AUTOMATA

2.1 Structure and Semantics

Definition 1 (Interval automaton)**.** *An* interval automaton *(IA) is a couple* $\phi = \langle X, \Gamma \rangle$*, with*

- X *(denoted* $\mathrm{Var}(\phi)$*) a finite and totally ordered set of real variables, whose domains are representable by the union of a* finite *number of* closed *intervals from* $\mathbb{R}$*;*
- Γ *a directed acyclic graph with at most one root and at most one leaf (the* sink*), whose non-leaf nodes are labelled by a variable of* X *or by the disjunctive symbol* $\veebar$ *(that we shall treat as a peculiar variable), and whose edges are labelled by a closed interval from* $\mathbb{R}$*. Edges going out of* $\veebar$*-labelled nodes can only be labelled by either* $\mathbb{R}$ *or* $\emptyset$*.*

This definition allows an interval automaton[3] to be empty (no node at all) or to contain only one node (together root and sink), and ensures that every edge belongs to at least one path from the root to the sink. Figure 1 gives an example of interval automaton.

For $x \in X$, $\mathrm{Dom}(x) \subseteq \mathbb{R}$ denotes the *domain* of x, which can either be enumerated ($\mathrm{Dom}(x) = \{1, 3, 56, 4.87\}$)

[1] ONERA/DCSD, France, email: {alexandre.niveau, cpralet, verfail}@onera.fr
[2] IRIT/RPDMP, France, email: fargier@irit.fr

[3] Note that our interval automata have no relationship with the single-clock timed automata that go by the same name.

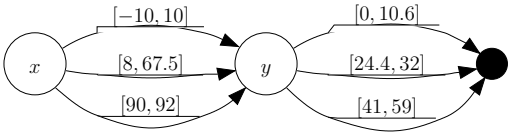

Figure 1. An example of interval automaton. Its model set (see Definition 2), represented as a union of boxes, is $[-10, 10] \times [0, 10.6] \cup [-10, 10] \times [24.4, 32] \cup [-10, 10] \times [41, 59] \cup [8, 67.5] \times [0, 10.6] \cup [8, 67.5] \times [24.4, 32] \cup [8, 67.5] \times [41, 59] \cup [90, 92] \times [0, 10.6] \cup [90, 92] \times [24.4, 32] \cup [90, 92] \times [41, 59]$.

or continuous ($\mathrm{Dom}(x) = [1, 7] \cup [23.4, 28]$). By convention, $Dom(\veebar) = \mathbb{R}$. We call "*box*" a cartesian product of intervals. For $Y = \{y_1, \ldots, y_k\} \subseteq X$, such that the y_i are sorted in ascending order, $\mathrm{Dom}(Y)$ denotes $\mathrm{Dom}(y_1) \times \cdots \times \mathrm{Dom}(y_k)$, and $\vec{y}$ denotes a *Y-assignment* of variables from Y, *i.e.* $\vec{y} \in \mathrm{Dom}(Y)$. When $Y \cap X = \emptyset$, $\vec{x} \,.\, \vec{y}$ is the *concatenation* of $\vec{x}$ and $\vec{y}$. Last, $\vec{y}(y_i)$ denotes the value assigned to y_i in $\vec{y}$.

Let $\phi = \langle X, \Gamma \rangle$ be an interval automaton, N a node and E an edge in Γ. We can then define the following elements:

- $\mathrm{Root}(\phi)$ the root of Γ and $\mathrm{Sink}(\phi)$ its sink;
- $|\phi|$ the *size* of ϕ, *i.e.* the number of edges of Γ plus the number of intervals needed to represent the domains of the variables;
- $\mathrm{Out}_\phi(N)$ (resp. $\mathrm{In}_\phi(N)$) the set of outgoing (resp. incoming) edges of N;
- $\mathrm{Var}_\phi(N)$ the variable labelling N (by convention $\mathrm{Var}_\phi(\mathrm{Sink}(\phi)) = \veebar$);
- $\mathrm{Src}_\phi(E)$ the node from which E comes and $\mathrm{Dest}(E)$ the node to which E points;
- $\mathrm{Itv}_\phi(E)$ the interval labelling E;
- $\mathrm{Var}_\phi(E) = \mathrm{Var}_\phi(\mathrm{Src}(E))$ the variable associated with E.

When there is no ambiguity, we forget to use the ϕ subscript.

An IA can be seen as a compact representation of a Boolean function over discrete or continuous variables. This function is the *interpretation function* of the interval automaton:

Definition 2 (Semantics of an interval automaton)**.** *An interval automaton ϕ on X (*i.e. *we denote $X = \mathrm{Var}(\phi)$) represents a function from $\mathrm{Dom}(X)$ to $\{\top, \bot\}$. This function, called its* interpretation function $\mathrm{I}(\phi)$*, is defined as follows: for every X-assignment $\vec{x}$, $\mathrm{I}(\phi)(\vec{x}) = \top$ if and only if there exists a path p from the root to the sink of ϕ such that for each edge E along p, either $\mathrm{Var}(E) = \veebar$ and $\mathrm{Itv}(E) \neq \emptyset$, or $\vec{x}(\mathrm{Var}(E)) \in \mathrm{Itv}(E)$.*

We say that $\vec{x}$ is a model of ϕ whenever $\mathrm{I}(\phi)(\vec{x}) = \top$. $\mathrm{Mod}(\phi)$ denotes the set of models of ϕ.

ϕ is said to be equivalent *to another IA ψ (denoted $\phi \equiv \psi$) iff $\mathrm{Mod}(\phi) = \mathrm{Mod}(\psi)$.*

Note that the interpretation function of the empty automaton always returns $\bot$, since an empty IA contains no path from the root to the sink. Conversely, the interpretation function of the one-node automaton always returns $\top$, since in the one-node IA, the only path from the root to the sink contains no edge. We can now introduce useful definitions:

Definition 3 (Consistency, validity, context)**.** *Let ϕ be an interval automaton on X.*

ϕ is said to be consistent *(resp.* valid*) if and only if $\mathrm{Mod}(\phi) \neq \emptyset$ (resp. $\mathrm{Mod}(\phi) = \mathrm{Dom}(X)$).*

A value $\omega \in \mathbb{R}$ is said to be consistent *for a variable $y \in X$ in ϕ if and only if there exists an X-assignment $\vec{x}$ in $\mathrm{Mod}(\phi)$ such that $\vec{x}(y) = \omega$.*

The set of all consistent values for y in ϕ is called the context *of y in ϕ and denoted $\mathrm{Ctxt}_\phi(y)$.*

We will see in the following that deciding whether an IA is consistent is not tractable. One of the reasons is that the intervals along a path do not have a nested structure: on a given path, the intervals related to the same variable can enlarge after having shrunk, and conversely. They can even be conflicting, hence the intractability of the consistency request. We will therefore consider focusing IAs, *i.e.* IAs in which intervals can only shrink from the root to the sink.

Definition 4 (Focusing interval automata)**.** *A* focusing *edge in an interval automaton ϕ is an edge E such that all edges E' on a path from the root of ϕ to $\mathrm{Src}(E)$ such that $\mathrm{Var}(E) = \mathrm{Var}(E')$ verify $\mathrm{Itv}(E) \subseteq \mathrm{Itv}(E')$.*

A focusing *interval automaton (FIA) is an IA containing only focusing edges.*

An example of FIA can be found on Fig. 2.

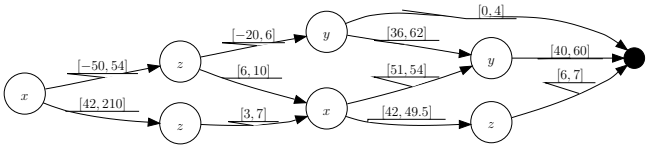

Figure 2. An example of focusing interval automaton. Variable domains are as follows: $\mathrm{Dom}(x) = [0, 100]$, $\mathrm{Dom}(y) = [0, 100]$ and $\mathrm{Dom}(z) = \{0, 3, 7, 10\}$.

As suggested by Fig. 1, the size of the automaton can be exponentially lower than the size of its extended model set (described as an union of boxes). This is notably due to the fact that IAs can be *reduced* by suppressing redundancies, in the manner of BDDs and NNFs. Before detailing this reduction operation, let us enlighten the relationships between IAs and these kinds of structures.

2.2 Relationships with BDDs and other target languages

Introduced by Bryant in [4], Binary Decision Diagrams (BDDs) are rooted directed acyclic graphs that represent Boolean functions of Boolean variables. They have exactly two leaves, respectively labelled $\top$ and $\bot$; their non-leaf nodes are labelled by a Boolean variable and have exactly two outgoing edges, also respectively labelled $\top$ and $\bot$ (or equivalently 1 and 0). A free BDD (FBDD) is a BDD that satisfies the read-once property (each path contains at most one occurrence of each variable). Whenever a same order is imposed on the variables along every path, we get an ordered BDD (OBDD).

Interval automata can be understood as a generalization of BDDs. The interpretation function of BDDs is indeed similar to the one of IAs: for a given assignment of the variables, the function's value is $\top$ if and only if there exists a path from the root to the $\top$-labelled leaf such that the given assignment is coherent with each edge along the path.

We see that, when interpreting a BDD, it is possible to ignore the $\bot$-labelled leaf. Now, if we remove this leaf, a BDD is an IA the intervals of which are $[0, 0]$ or $[1, 1]$:

Proposition 5 (Correspondence between IAs and BDDs)**.** *Any BDD can be expressed in the form of an equivalent IA, in time linear in the BDD's size.*

This *linear translatability* will help prove further propositions. It can also be used to translate any FBDD or OBDD in the FIA framework:

Proposition 6 (Correspondence between FIAs and FBDDs)**.** *Any FBDD (and thus any OBDD) can be expressed in the form of an equivalent FIA, in time linear in the FBDD's size.*

The main difference between the IA family and the BDD family (including ADDs) is that IAs are not required to be deterministic (the same solution can be checked over several paths of the automaton, which potentially allows gain in space), and obviously that IAs are not limited to Boolean variables. Vempaty's automata [19, 1], SLDDs [22] and signed logic [2] also support non Boolean domains, but are restricted to finite domains. Vempaty's automata are moreover ordered structures, just like OBDDs or interval diagrams [18].

To compile Boolean functions over continuous variables, one could use the spatial access method "R*-tree" [3], which is a tree (not a graph) whose nodes are labelled by boxes. However, since it has not been introduced as a target compilation language, the feasibility of useful operations (conditioning, forgetting...) have not been studied yet.

Interestingly, FIA are not decomposable structures in the sense of DNNFs [8], but keep the essence of the decomposability property: they are linkless [14] — in a FIA, a variable restriction can be repeated on a path (in terms of NNFs, on the two sides of an AND node), but the restrictions cannot conflict (with the noticeable exception of $\emptyset$-marked edges, that are typically removed when reducing the automaton).

2.3 Reduction

Like a BDD, an interval automaton can be reduced in size without changing its semantics by merging some nodes or edges. The reduction operations introduced thereafter are based on the notions of isomorphic, stammering and undecisive nodes, and of contiguous and unreachable edges. Some of these notions are straightforward generalizations of definitions introduced in the context of BDDs [4], while others are specific to interval automata.

Definition 7 (Isomorphic nodes)**.** *Two non-leaf nodes N_1, N_2 of an IA ϕ are* isomorphic *if and only if*

- $\mathrm{Var}(N_1) = \mathrm{Var}(N_2)$*;*
- *there exists a bijection σ from* $\mathrm{Out}(N_1)$ *onto* $\mathrm{Out}(N_2)$*, such that* $\forall E \in \mathrm{Out}(N_1), \mathrm{Itv}(E) = \mathrm{Itv}(\sigma(E))$ *and* $\mathrm{Dest}(E) = \mathrm{Dest}(\sigma(E))$.

Isomorphic nodes are redundant, as they represent the same function; only one of them is necessary (see Figure 3).

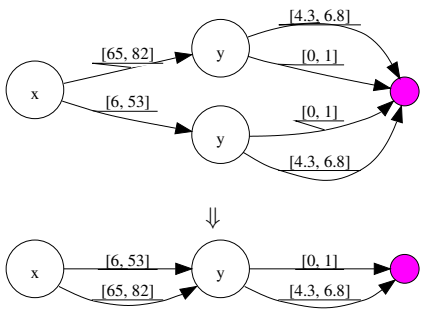

Figure 3. Merging of isomorphic nodes.

Definition 8 (Stammering node)**.** *A non-root node N of an IA ϕ is* stammering *if and only if all parent nodes of N are labelled by* $\mathrm{Var}(N)$*, and either* $|\mathrm{Out}(N)| = 1$ *or* $|\mathrm{In}(N)| = 1$.

Stammering nodes are useless, since the information they bring could harmlessly be deported to their parents (see Figure 4).

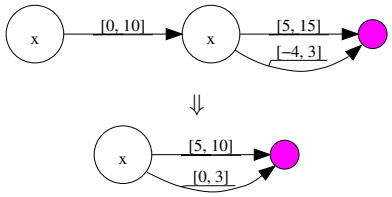

Figure 4. Merging of stammering nodes.

Definition 9 (Undecisive node)**.** *A node N of an IA ϕ is* undecisive *if and only if* $|\mathrm{Out}(N)| = 1$ *and* $E \in \mathrm{Out}(N)$ *is such that* $\mathrm{Dom}(\mathrm{Var}(E)) \subseteq \mathrm{Itv}(E)$.

An undecisive node does not restrict the solutions corresponding to the paths it is in; it is "automatically" crossed (see Figure 5).

Figure 5. Elimination of undecisive nodes.

Definition 10 (Contiguous edges)**.** *Two edges E_1, E_2 of an IA ϕ are* contiguous *if and only if*

- $\mathrm{Src}(E_1) = \mathrm{Src}(E_2)$*;*
- $\mathrm{Dest}(E_1) = \mathrm{Dest}(E_2)$*;*
- *there exists an interval $I \subseteq \mathbb{R}$ such that* $I \cap \mathrm{Dom}(\mathrm{Var}(E_1)) = (\mathrm{Itv}(E_1) \cup \mathrm{Itv}(E_2)) \cap \mathrm{Dom}(\mathrm{Var}(E_1))$.

Contiguous edges both come from the same node, both point to the same node and are not disjoint (modulo the domain of their variable): they could be replaced by a single edge (see Figure 6). For example, in the case of an integer-valued variable, a couple of edges labelled $[0, 3]$ and $[4, 8]$ respectively is equivalent to a single edge labelled $[0, 8]$.

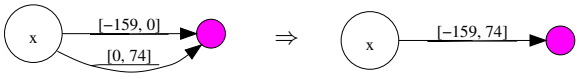

Figure 6. Merging of contiguous edges.

Definition 11 (Unreachable edge)**.** *An edge E of an IA ϕ is* unreachable *if and only if* $\mathrm{Itv}(E) \cap \mathrm{Dom}(\mathrm{Var}(E)) = \emptyset$.

An unreachable edge will never be crossed, as no value in its label is coherent with the variable domain (see Figure 7).

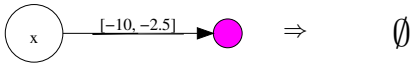

Figure 7. Elimination of unreachable edges (here $\mathrm{Dom}(x) = \mathbb{R}_+$).

Definition 12 (Reduced interval automaton)**.** *An interval automaton ϕ is said to be* reduced *if and only if*

- *no node of ϕ is isomorphic to another, stammering, or undecisive;*
- *no edge of ϕ is contiguous to another or unreachable.*

In the following, we can consider only reduced IAs since reduction can be done in time polynomial in the size of the structure.

Proposition 13 (Reduction of an IA)**.** *There exists a polytime algorithm that transforms any IA ϕ into an equivalent reduced IA ϕ' such that $|\phi'| \leq |\phi|$.*

The first result we get on FIAs is that they are not harder to reduce than IAs: our reduction algorithm maintains the focusing property when applied on a FIA.

Proposition 14 (Reduction of a FIA)**.** *There exists a polytime algorithm that transforms any FIA ϕ into an equivalent reduced FIA ϕ' such that $|\phi'| \leq |\phi|$.*

3 REQUESTS ON INTERVAL AUTOMATA

As previously said, an interval automaton represents a function from some set of variables to $\{\bot, \top\}$. This section formalizes the main queries and transformations that could be useful in a planning context. For the sake of exhaustivity, we also introduce requests that are classically studied when evaluating the facilities of a compilation language [8].

3.1 Useful Operations for Planning

Compilation of decision policies In a planning context, we first want to represent by an interval automaton a decision policy δ produced by some planning algorithm. In this case, δ is a function which holds on two sets of variables, the set S of state variables and the set D of decision variables. For any S-assignment $\vec{s}$ and any D-assignment $\vec{d}$, $\delta(\vec{s} \,.\, \vec{d}) = \top$ if and only if $\vec{d}$ is a suitable decision in state $\vec{s}$.

In order to exploit a decision policy δ online, two basic operations are required. First, each time a new state instantiation $\vec{s}$ is observed, we need to determine the set of decisions suiting $\vec{s}$ according to δ. This operation corresponds to *conditioning* δ by $\vec{s}$. One of the suitable decisions must then be extracted, to be executed. This operation corresponds to *model extraction*. Both operations will be defined formally in the sequel.

Concerning the elaboration of a decision policy, consider that it is built incrementally by some planning algorithm, until it covers the whole set of reachable states. In this case, incrementally building δ means adding in δ new pairs $(\vec{s}, \vec{d})$ such that decision $\vec{d}$ covers state $\vec{s}$. To do so, if δ is represented at each step by an interval automaton, we need to perform operations of the form $\delta := \delta \vee (\vec{s} \,.\, \vec{d})$, that is *disjunctions*.

It is worth noticing that in the final policy, all the possible decisions for a given state are of equal interest (relative plausibilities are not expressed in IA), even if the original problem is stochastic. This does not prevent to build such a policy from a stochastic problem. Once a decision policy has been built, be the initial problem stochastic or not, fully observable or not, it can be compiled into an IA.

Compilation of transition relations IAs can also be used to represent the basic data involved in a planning domain: the set of possible initial states, of goal states, and the *transition relation* defining the possible transitions of a given system. Let us consider the example of a non-stochastic[4] transition relation T. Such a relation holds on three sets of variables: the set S of variables representing the current state, the set D of variables representing a decision made, and the set S' of variables representing the state after the decision is applied. For any $S \cup D \cup S'$-assignment $\vec{s} \,.\, \vec{d} \,.\, \vec{s'}$, $\delta(\vec{s} \,.\, \vec{d} \,.\, \vec{s'}) = \top$ means that $\vec{s'}$ is a possible successor state when decision $\vec{d}$ is applied in state $\vec{s}$.

Several operations may be needed to efficiently manipulate transition relations compiled as IAs. Notably, in forward approaches of planning, it may be useful to efficiently compute, for a current state $\vec{s}$ and a decision $\vec{d}$, the set S' of possible successors of $\vec{s}$, that is S'-instantiations $\vec{s'}$ such that $T(\vec{s}, \vec{d}, \vec{s'}) = \top$. This requires the operations of *conditioning*, to assign $\vec{s}$ and $\vec{d}$ in T, and of *model enumeration*, to get all possible successors $\vec{s'}$. When actions have a deterministic effect, the transition relation T becomes a transition function and *model extraction* suffices to get the only possible successor state $\vec{s'}$. Manipulation of deterministic transition functions cover practical deterministic planning problems, in which the objective is to build offline a controller able to face any possible initial situation (an alternative to the planning/replanning approach).

All operations interesting in a planning context, as well as other standard requests, are formally defined in the following.

3.2 Operations on Interval Automata

Let us detail the operations[5] we will focus on, and check whether they can be performed efficiently on the compiled form. We first introduce the *queries*, that is, the operations which return information about an IA.

Definition 15 (Queries)**.** *Let* `L` *denote a subset of the* `IA` *language.*

- `L` *satisfies*[6] **CO** *(resp.* **VA***) iff there exists a polytime algorithm that maps every automaton ϕ from* `L` *to* 1 *if ϕ is*

[4] IAs do not express plausibilities. Yet, using IAs for compiling stochastic transition relations (and policies) is a natural extension of our work. This extension can be achieved by adding probabilities on the edges, thus making valued IAs, closer to SLDDs.

[5] **CO** stands for "COnsistency", **VA** for "VAlidity", **EQ** for "EQuivalence", **MC** for "Model Checking", **MX** for "Model eXtraction", **ME** for "Model Enumeration", **CX** for "Context Extraction", **CD** for "ConDitioning", **FO** for "FOrgetting", **EN** for "ENsuring", **SCD**, **SFO**, **SEN** for "Single **CD**, **FO**, **EN**", $\wedge$**C**, $\vee$**C** for "$\wedge$, $\vee$-Closure", $\wedge$**BC**, $\vee$**BC** for "$\wedge$, $\vee$-Binary Closure", and $\wedge$**tC** for "Closure under conjunction with a term".

[6] One can also use "supports".

consistent (resp. valid), and to 0 *otherwise.*

- L *satisfies* **EQ** *iff there exists a polytime algorithm that maps every pair of automata* (ϕ, ϕ') *from* L *to* 1 *if* $\phi \equiv \phi'$ *and to* 0 *otherwise.*
- L *satisfies* **MC** *iff there exists a polytime algorithm that maps every automaton* ϕ *from* L *and any* $\mathrm{Var}(\phi)$*-assignment* $\vec{x}$ *to* 1 *if* $\vec{x}$ *is a model of* ϕ *and to* 0 *otherwise.*
- L *satisfies* **MX** *iff there exists a polytime algorithm that maps every automaton* ϕ *in* L *to one model of* ϕ *if there is one, and stops without returning anything otherwise.*
- L *satisfies* **ME** *iff there exists a polynomial* $p(;)$ *and an algorithm that outputs, for any automaton* ϕ *from* L*, a set* $\mathcal{B}$ *of non-empty boxes whose union is equal to* $\mathrm{Mod}(\phi)$ *in time* $p(|\phi|; |\mathcal{B}|)$*.*
- L *satisfies* **CX** *iff there exists a polytime algorithm that outputs, for any* ϕ *in* L *and any* $y \in \mathrm{Var}(\phi)$*,* $\mathrm{Ctxt}_\phi(y)$*.*

We will now define a number of *transformations* on IAs, (*i.e.* operations that return a modified IA); we first present the semantic operations on which they are based.

Definition 16. *Let* I, J *be the interpretation functions on* $\mathrm{Var}(I), \mathrm{Var}(J)$ *of some automata.*

- *The conjunction (resp. disjunction) of* I *and* J *is the function* $\mathrm{I} \wedge \mathrm{J}$ *(resp.* $\mathrm{I} \vee \mathrm{J}$*) on the variables in* $X = \mathrm{Var}(\mathrm{I}) \cup \mathrm{Var}(\mathrm{J})$ *defined by* $(\mathrm{I} \wedge \mathrm{J})(\vec{x}) = \mathrm{I}(\vec{x}) \wedge \mathrm{J}(\vec{x})$ *(resp.* $(\mathrm{I} \vee \mathrm{J})(\vec{x}) = \mathrm{I}(\vec{x}) \vee \mathrm{J}(\vec{x})$*).*
- *The existential projection of* I *on* $Y \subseteq \mathrm{Var}(\mathrm{I})$ *is the function* $\mathrm{I}^{\downarrow Y}$ *on the variables of* Y *defined by:* $\mathrm{I}^{\downarrow Y}(\vec{y}) = \top$ *iff there exist a* Z*-assignment* $\vec{z}$ *(with* $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$*) s.t.* $\mathrm{I}(\vec{z}\ .\ \vec{y}) = \top$*. The "forgetting" operation is the dual one:* $\mathrm{forget}(\mathrm{I}, Y) = \mathrm{I}^{\downarrow \mathrm{Var}(\mathrm{I}) \setminus Y}$*.*
- *The universal projection of* I *on* $Y \subseteq \mathrm{Var}(\mathrm{I})$ *is the function* $\mathrm{I}^{\Downarrow Y}$ *on the variables of* Y *defined by:* $\mathrm{I}^{\Downarrow Y}(\vec{y}) = \top$ *iff for any* Z*-assignment* $\vec{z}$ *(with* $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$*),* $\mathrm{I}(\vec{z}\ .\ \vec{y}) = \top$*. The "ensuring" operation is the dual one:* $\mathrm{ensure}(\mathrm{I}, Y) = \mathrm{I}^{\Downarrow \mathrm{Var}(\mathrm{I}) \setminus Y}$*.*
- *Given an assignment* $\vec{y}$ *of some set of variables* $Y \subseteq \mathrm{Var}(\mathrm{I})$*, the conditioning of* I *by* $\vec{y}$ *is the function* $\mathrm{I}_{|\vec{y}}$ *on the variables in* $Z = \mathrm{Var}(\mathrm{I}) \setminus Y$ *defined by:* $\mathrm{I}_{|\vec{y}}(\vec{z}) = \mathrm{I}(\vec{y}\ .\ \vec{z})$*.*

And now for the knowledge compilation-oriented transformations:

Definition 17 (Transformations)**.** *Let* L *denote a subset of the* IA *language.*

- L *satisfies* **CD** *iff there exists a polytime algorithm that maps every automaton* ϕ *in* L *and every assigment* $\vec{y}$ *of* $Y \subseteq \mathrm{Var}(\phi)$ *to an automaton* ϕ' *in* L *such that* $\mathrm{I}(\phi') = \mathrm{I}(\phi)_{|\vec{y}}$*.*
- L *satisfies* **FO** *(resp.* **EN***) iff there exists a polytime algorithm that maps every automaton* ϕ *from* L *and every* $Y \subseteq \mathrm{Var}(\phi)$ *to an automaton* ϕ' *in* L *such that* $\mathrm{I}(\phi') = \mathrm{forget}(\mathrm{I}(\phi), Y)$ *(resp.* $\mathrm{I}(\phi') = \mathrm{ensure}(\mathrm{I}(\phi), Y)$*).*
- L *satisfies* **SCD** *(resp.* **SFO***, resp.* **SEN***) iff it satisfies* **CD** *(resp.* **FO***, resp.* **EN***) when limited to a single variable (*i.e. $\mathrm{Card}(Y) = 1$*).*
- L *satisfies* $\wedge$**C** *(resp.* $\vee$**C***) iff there exists a polytime algorithm that maps every finite set of automata* $\Phi = \{\phi_1, \ldots, \phi_k\}$ *from* L *to an automaton* ϕ *in* L *such that* $\mathrm{I}(\phi) = \mathrm{I}(\phi_1) \wedge \cdots \wedge \mathrm{I}(\phi_k)$ *(resp.* $\mathrm{I}(\phi) = \mathrm{I}(\phi_1) \vee \cdots \vee \mathrm{I}(\phi_k)$*).*
- L *satisfies* $\wedge$**BC** *(resp.* $\vee$**BC***) iff it satisfies* $\wedge$**C** *(resp.* $\vee$**C***) when limited to a pair of automata (*i.e. $\mathrm{Card}(\Phi) = 2$*)*
- L *satisfies* $\wedge$**tC** *iff there exists a polytime algorithm that maps every automaton* ϕ *from* L*, any set of variables* $\{y_1, \ldots, y_k\} \subseteq \mathrm{Var}(\phi)$ *and any sequence* $(A_1, \ldots, A_k)$ *of closed intervals, to an automaton* ϕ' *in* L *such that* $\mathrm{I}(\phi') = \mathrm{I}(\phi) \wedge f_{y_1, A_1} \wedge \cdots \wedge f_{y_k, A_k}$*, where* $f_{x,A}$ *is the function defined on* $Y = \{x\}$ *by* $f_{x,A}(\vec{y}) = \top \Leftrightarrow \vec{y}(x) \in A$*.*

3.3 Complexity Results

L	CO	VA	EQ	MC	MX	CX	ME	CD	SCD	$\wedge$tC	FO	SFO	EN	SEN	$\veebar$C	$\veebar$BC	$\vee$C	$\vee$BC
IA	∘	∘	∘	√	∘	∘	∘	√	√	√	∘	√	∘	√	√	√	√	√
FIA	√	∘	∘	√	√	√	√	√	√	√	√	√	∘	∘	∘	∘	√	√

Table 1. Results about queries and transformations. √ means "satisfies", ∘ means "does not support, unless P = NP".

Proposition 18. *The results of Table 1 hold.*

It appears that performance of interval automata is weak with respect to most of the queries, and in particular with respect to **CO**, **MX** and **VA**, which is not surprising since BDDs are IAs. Imposing the restrictive focusing property makes most of the queries tractable, including **CO** and **MX**. The main reason is that every path from the root to the sink of a reduced FIA is coherent, since no edge along it conflicts with any other (similarly to FBDDs).

This is also why (added to the fact that we allow $\veebar$-nodes) FIAs support **CD** and **FO** : it is roughly sufficient to replace all concerned nodes by $\veebar$-nodes and their edges' labels by $\mathbb{R}$ or $\emptyset$.

Proposition 18 shows that FIAs are suitable for compilation of decision policies, as well as transition relations to be used in a forward approach.

It also proves that neither IAs' nor FIAs' reduced form is canonical (if it were, **EQ** would be polytime), and that IAs are of course not polynomially translatable into FIAs (FIA supports operations not supported by IA).

4 BUILDING INTERVAL AUTOMATA

We have shown that FIA allows in polytime operations that are useful for planning. Let us briefly cite two possible algorithmic approaches for their construction.

Union of Boxes It is straightforward to convert a union of boxes into a FIA. This can be done in polytime, thanks to $\vee$**C**. We can then easily compile into FIA any policy or transition table that is given in this form: either a discrete one, obtained for example by an algorithm returning DNFs, or a continuous one, obtained for example by an interval-based constraint solver.

Trace of RealPaver We can also adopt a process similar to [11], using the trace of a search algorithm as a convenient way to transform a CSP into an FIA [15]. This process consists in creating new nodes and edges as soon as a solution is found by the search algorithm, and in fusioning them with the current FIA recording the solutions found so far. Here, we will

use this method on the interval-based solver RealPaver [10] to create an interval automaton representing an approximation of the solution set of a constraint network.

5 RESULTS

problem	red time (ms)	size (edges)	% edges/ input	% edges/ OBDD	**CD** (ms)	**MX** (ms)
obsmem2	1102	100	74	66	1	5
obsmem3	2168	197	75	69	4	11
obsmem4	4729	342	75	70	4	11
obsmem5	5657	546	76	70	7	19
obsmem6	9433	820	76	76	11	35
porobot	4035	56	97	36	0	1
forobot	52767	60	99	31	0	3
ring7	92	13	75	71	0	1
ring8	185	13	78	75	0	1
ring9	92	13	80	75	0	1
ring10	82	13	81	75	0	2
drone10	46732	453	95	47	11	23
drone20	947174	763	97	44	30	61
drone30	2850715	944	98	43	21	48
drone40	5721059	944	98	45	15	29
drone10	104373	16820	35	×	7143	110
drone20	418885	38076	35	×	16970	193
drone30	1850326	53917	36	×	23597	612

Table 2. Application results.

Table 2 presents a few results of our first implementation for a number of discrete and continuous problems, consisting in policies or transition tables. The `obsmem` problem manages connections between the observation device and the mass memory of a satellite. The `robot` problem deals with a robot exploring an area, and the `ring` domain is a standard benchmark for planning with non-determinism. In the `drone` problem, a drone must achieve different goals on a number of zones in limited time; this latter problem is used in a discrete and a hybrid version, in which the continuous variable is the remaining time. See Appendix B for more details.

In Table 2, the last three instances are transition tables involving a continuous variable (thus not comparable with OBDDs), obtained by following the trace of RealPaver. All the others are discrete decision policies, obtained by compiling disjunctions of boxes given by the algorithm described in [16]. For each instance, we state the time needed for reducing the compiled FIA, the size of the reduced FIA, the reduction rate (0% meaning no reduction) w.r.t. the input (number of boxes × number of variables), the reduction rate w.r.t. the equivalent OBDD (obtained by converting enumerated variables into Boolean by log encoding [21]), and the mean time taken by a single conditioning or model extraction operation on a standard laptop[7].

Those results show that FIAs can be favourably compared to OBDDs concerning the size of the graph, and that our implementation of the requests is worth being improved.

6 CONCLUSION

In this paper, we introduced interval automata, a new knowledge compilation language dealing with Boolean functions holding on enumerated or continuous variables. We identified a subclass of interval automata, the focusing ones, for which several requests useful in a planning context were proven to be tractable. We showed the significant gains obtained regarding the size of the compiled structure compared to OBDDs, IAs being moreover able to model continuous domains without requiring discretization. In the future, we plan to compare FIAs to other enumerated domains target languages (Vempaty's automata, SLDDs...), to study other interesting fragments of IAs, to extend the IA language with valuations (thus allowing to represent stochastic policies, and to use approximate compilation [20]), and to define other compilation languages suited to the management of planning domains.

[7] Mobile Turion 64 X2 TL-56, 1.80 GHz, 2 Go RAM.

REFERENCES

[1] J. Amilhastre, P., and M.-C. Vilarem, 'Fa Minimisation Heuristics for a Class of Finite Languages', in *WIA*, pp. 1–12, (1999).

[2] Bernhard Beckert, Reiner Hähnle, and Felip Manyà, 'Transformations between Signed and Classical Clause Logic', in *ISMVL*, pp. 248–255, (1999).

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, 'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles', in *SIGMOD Conference*, pp. 322–331, (1990).

[4] R.E. Bryant, 'Graph-Based Algorithms for Boolean Function Manipulation', *IEEE Transactions on Computers*, **35**(8), 677–691, (1986).

[5] M. Cadoli and F.M. Donini, 'A Survey on Knowledge Compilation', *AI Communications*, **10**(3–4), 137–150, (1998).

[6] Alessandro Cimatti and Marco Roveri, 'Conformant Planning via Symbolic Model Checking', *JAIR*, **13**, 305–338, (2000).

[7] A. Darwiche, 'Decomposable Negation Normal Form', *Journal of the ACM*, **48**(4), 608–647, (2001).

[8] A. Darwiche and P. Marquis, 'A Knowledge Compilation Map', *JAIR*, **17**, 229–264, (2002).

[9] A. del Val, 'Tractable Databases: How to Make Propositional Unit Resolution Complete Through Compilation', in *Proc. of KR'94*, pp. 551–561, (1994).

[10] L. Granvilliers and F. Benhamou, 'Algorithm 852: RealPaver: an Interval Solver Using Constraint Satisfaction Techniques', *ACM Trans. Math. Softw.*, **32**(1), 138–156, (2006).

[11] J. Huang and A. Darwiche, 'DPLL with a Trace: From SAT to Knowledge Compilation', in *IJCAI*, pp. 156–162, (2005).

[12] Michel Lemaître, Cédric Pralet, and Gérard Verfaillie, 'Programme commun Onera/CNES AGATA Autonomie des Systèmes Spatiaux — Tâche 2.1 Synthèse de contrôleur', Technical report, Onera/CNES, (2009).

[13] Christoph Meinel and Thorsten Theobald, *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*, Springer, 1998.

[14] N. V. Murray and E. Rosenthal, 'Tableaux, Path Dissolution, and Decomposable Negation Normal Form for Knowledge Compilation', in *TABLEAUX*, pp. 165–180, (2003).

[15] Alexandre Niveau, Hélène Fargier, Cédric Pralet, and Gérard Verfaillie, 'Handling the Output of Interval-Based Constraint Solvers by Interval Automata Compilation', in *IntCP Workshop on Interval Analysis and Constraint Propagation for Applications, CP*, (2009).

[16] Cédric Pralet, Gérard Verfaillie, Michel Lemaître, and Guillaume Infantes, 'Constraint-based Controller Synthesis in Non-Deterministic and Partially Observable Domains', in *ECAI*, (2010).

[17] B. Selman and H.A. Kautz, 'Knowledge Compilation and Theory Approximation', *Journal of the ACM*, **43**, 193–224, (1996).

[18] K. Strehl and L. Thiele, 'Symbolic Model Checking of Process Networks Using Interval Diagram Techniques', in *Proc. of the 1998 IEEE/ACM international conference on Computer-aided design*, pp. 686–692, (1998).

[19] N. R. Vempaty, 'Solving Constraint Satisfaction Problems Using Finite State Automata', in *AAAI*, pp. 453–458, (1992).
[20] Alberto Venturini and Gregory Provan, 'Incremental Algorithms for Approximate Compilation', in *AAAI*, pp. 1495–1499, (2008).
[21] Toby Walsh, 'SAT v CSP', in *CP*, pp. 441–456, (2000).
[22] Nic Wilson, 'Decision Diagrams for the Computation of Semiring Valuations', in *IJCAI*, pp. 331–336, (2005).

A PROOFS

In all of these proofs, we consider for short that for an interpretation function I with $Y = \mathrm{Var(I)}$, a set of variables Z s.t. $Z \cap Y = \emptyset$, $\vec{y}$ a Y-assignment and $\vec{z}$ a Z-assignment, $\mathrm{I}(\vec{y} \,.\, \vec{z})$ is simply defined as $\mathrm{I}(\vec{y})$.

Let ϕ be an IA on X and $\vec{x}$ an X-assignment. A path p from the root to the sink of ϕ is said to be *compatible with* $\vec{x}$ if and only if for each edge E along p, either $\mathrm{Var}(E) = \trianglerighteq$ and $\mathrm{Itv}(E) \neq \emptyset$, or $\vec{x}(\mathrm{Var}(E)) \in \mathrm{Itv}(E)$.

For any I, we denote $\vec{x} \models \mathrm{I}$ the fact that $\vec{x}$ is a model of I, and $\vec{x} \models \phi$ iff $\vec{x} \models \mathrm{I}(\phi)$. We also denote $\vec{x} \models p$ if p is a path compatible with $\vec{x}$.

Proof of Proposition 5. As Boolean variables can be represented by real variables with domain $\{0, 1\}$, a BDD can be transformed into an IA by removing its $\bot$-labeled node, and by recursively removing all edges pointing to no node, and all non-leaf nodes without outgoing edges. The graph obtained then becomes an IA if each $\top$-labeled edge is replaced by a $[1, 1]$-labeled edge, and each $\bot$-labeled edge is replaced by a $[0, 0]$-labeled edge. □

Proof of Proposition 6. If we use the procedure described in the proof of Prop. 5 on an FBDD, the resulting IA is focusing. Indeed, as each variable can only be encountered once on each path, there is no risk that an interval conflicts with another. □

Lemma 19 (Sizes of BDDs and IAs)**.** *For any BDD (resp. any FBDD) ϕ, and ψ its corresponding IA (resp. FIA), $|\psi|_{\mathtt{IA}}$ is linear in $|\phi|_{\mathtt{BDD}}$.*

Proof. This is straightforward: as the size of a BDD is only its number of edges, its number of variables is bounded by its number of nodes, and the number of intervals necessary to represent a Boolean is 2, we have $|\psi|_{\mathtt{IA}} \leq 2|\phi|_{\mathtt{BDD}}$. □

Proof of Proposition 13. Let us apply Algorithm 1 on an IA ϕ.

For a given node N:

- the operation of l. 4 suppresses N if it is stammering
- the operation of l. 10 ensures that N has no more unreachable outgoing edges
- the operation of l. 15, ensures that N has no more contiguous outgoing edges
- the operation of l. 18 suppresses N if it is undecisive
- the operation of l. 24 suppresses all nodes that are isomorphic to N

and the algorithm stops when no operation has to be applied, so obviously the resulting IA is reduced. Moreover, it is easy to verify that each operation leaves the semantics of ϕ unchanged. Finally, every operation removes stricly more edges

Algorithm 1 Reduction algorithm. At any time during process, if an edge has no source or destination, it is suppressed; so are non-leaf nodes without outgoing edges and non-root nodes without incoming edges.

1: **repeat**
2: number the nodes of ϕ in such a way that if $N_i \in \mathrm{Ch}(N_j)$ then $i < j$
3: **for** i from 1 to the number of nodes in $\Gamma(\phi)$ **do**
4: **if** N_i is stammering **then**
5: **for all** $(E_{\mathrm{in}}, E_{\mathrm{out}}) \in \mathrm{In}(N_i) \times \mathrm{Out}(N_i)$ **do**
6: add an edge from $\mathrm{Src}(E_{\mathrm{in}})$ to $\mathrm{Dest}(E_{\mathrm{out}})$ labelled by $\mathrm{Itv}(E_{\mathrm{in}}) \cap \mathrm{Itv}(E_{\mathrm{out}})$
7: suppress N_i
8: **else**
9: **for all** $E \in \mathrm{Out}(N_i)$ **do**
10: **if** E is unreachable **then**
11: suppress E
12: **else**
13: mark E
14: **for all** $E' \in \mathrm{Out}(N)$ such that E' is not marked **do**
15: **if** E and E' are contiguous **then**
16: label E with $\mathrm{Itv}(E) \cup \mathrm{Itv}(E')$
17: suppress E'
18: **if** N_i is undecisive **then**
19: **for all** $E_{\mathrm{in}} \in \mathrm{In}(N_i)$ **do**
20: redirect E_{in} to the child of N_i
21: suppress N_i
22: **else**
23: **for** j from 1 to the number of nodes in $\Gamma(\phi)$ **do**
24: **if** N_i and N_j are isomorphic **then**
25: **for all** $E_{\mathrm{in}} \in \mathrm{In}(N_i)$ **do**
26: redirect E_{in} to N_j
27: suppress N_i
28: **until** ϕ has not changed during process

or nodes than it creates[8]; this proves that (i) the algorithm eventually stops (once the IA is empty, it does not change anymore), and (ii) the size of the resulting IA is lower than $|\phi|$ (the only case where the size does not change is when the input IA is already reduced).

The computation for each node is obviously polynomial[9] and the traversal loop (l. 3) treats each node once; as a result, what is inside of the repeating loop (from l. 2 to l. 27) is processed in polytime.

As the reducibility properties are not mutually independant, the traversal must be repeated while it has modified something in ϕ (l. 28). This does not change the polynomiality: since a traversal lowers the size of ϕ (except of course for the last one), the traversal loop will not be repeated more than $|\phi|$ times.

Note that there obviously exists more efficient methods to reduce an IA, but the only point here is to show that this operation is polytime. □

Definition 20 (IA focusing w.r.t. a variable)**.** *An IA ϕ is said to be* focusing w.r.t. *y, with $y \in \mathrm{Var}(\phi)$ iff every edge E in ϕ s.t. $\mathrm{Var}(E) = y$ is focusing.*

Lemma 21. *Algorithm 1 maintains the property of focusing w.r.t. a given variable.*

Proof. Let ϕ be an IA that is focusing w.r.t. $y \in \mathrm{Var}(\phi)$. Let us suppose that we are at step i in the algorithm, with $\mathrm{Var}(N_i) = y$.

- "stammering" operation: let $E \in \mathrm{Out}(N_i)$ and E' be an edge on a path from $\mathrm{Dest}(E)$ to the sink such that $\mathrm{Var}(E') = y$. E' is focusing, so $\mathrm{Itv}(E') \subseteq \mathrm{Itv}(E)$. Thus $\mathrm{Itv}(E') \subseteq \mathrm{Itv}(E) \cap \mathrm{Itv}(E_{\mathrm{in}})$ for any $E_{\mathrm{in}} \in \mathrm{In}(N_i)$. Hence E' is still focusing after the "stammering" operation.
- "unreachable" operation: suppressing edges does not have any influence on the focusingness of other edges in the graph.
- "contiguous" operation: the two contiguous edges points to the same node, so every descendant edge E associated with y is such that $\mathrm{Itv}(E)$ is included in the labelling interval of either one of the contiguous edge; hence it is included in their union.
- "undecisive" operation: every descendant edge of the child of N_i is also a descendant edge of N_i, so this operation does not compromise their focusingness.
- since every outgoing edge of every node isomorphic to N_i is focusing, redirecting all the parent edges to N_i is harmless.

□

Lemma 22 (IAs focusing w.r.t. all their variables)**.** *A reduced IA ϕ that is focusing w.r.t. every variable in $\mathrm{Var}(\phi)$ is focusing.*

Proof. Let E be an edge in ϕ. Either $\mathrm{Var}(E) = y \neq \veebar$, in which case E is focusing, as ϕ is focusing w.r.t. y; or $\mathrm{Var}(E) = \veebar$, in which case E is also focusing, as every edge E' in ϕ such that $\mathrm{Var}(E') = \veebar$ is labelled by $\mathbb{R}$ (since ϕ is reduced). Hence ϕ is focusing. □

Proof of Proposition 14. Let ϕ be a FIA. ϕ is *a fortiori* focusing w.r.t. all of its variables. Lemma 21 states that the IA ϕ' obtained by applying the reduction algorithm defined in the proof of Prop. 13 is also focusing w.r.t. all of its variables. Since ϕ' is reduced, we use Lemma 22 to infer that ϕ' is focusing. Thus our reduction algorithm maintains the focusing property, hence the result. □

Definition 23 (Mesh of a variable in an IA)**.** *Let ϕ be an IA and $x \in \mathrm{Var}(\phi)$. A* mesh *of x in ϕ is a partition $\mathcal{M} = \{M_1, \ldots, M_n\}$ of $\mathbb{R}$ such that for any edge E in ϕ verifying $\mathrm{Var}(E) = x$, for all $1 \leq i \leq n$, $(M_i \cap \mathrm{Itv}(E) \neq \emptyset) \Rightarrow (M_i \subseteq \mathrm{Itv}(E))$.*

Lemma 24 (Obtaining a mesh)**.** *We can build in quasi-linear time a mesh $\mathcal{M} = \{M_1, \ldots, M_n\}$ of a variable in any IA, which moreover verifies*

$$\forall M_i \in \mathcal{M}, (M_i \cap \mathrm{Dom}(x) \neq \emptyset \Rightarrow M_i \subseteq \mathrm{Dom}(x)) \quad .$$

Proof. Let ϕ be an IA and $x \in \mathrm{Var}(\phi)$. We only have to recover the (finite, lower and upper) bounds of all intervals associated to x, that is to say the disjoint intervals constituting $\mathrm{Dom}(x)$ and those labelling x-edges; this process is linear (simple traversal of the graph). Let $\mathcal{B} = \{b_1, \ldots, b_k\}$ be the obtained set of bounds, sorted in ascending order (quasi-linear process). Then

$$\mathcal{M} = \Big\{]-\infty, b_1[, \{b_1\},]b_1, b_2[, \{b_2\}, \ldots,]b_{i-1}, b_i[, \{b_i\},]b_i, b_{i+1}[, \ldots, \{b_k\},]b_k, +\infty[\Big\}$$

is a mesh of x in ϕ. Indeed:

- $\mathcal{M}$ is obviously a partition of $\mathbb{R}$ (all sets are disjoint and their union is $\mathbb{R}$) ;
- let E be an edge in ϕ s.t. $\mathrm{Var}(E) = x$, and i an integer s.t. $M_i \cap \mathrm{Itv}(E) \neq \emptyset$. By construction of $\mathcal{M}$, M_i is either a singleton, or an interval opened on both sides. In the first case, it is straightforward that $M_i \subseteq \mathrm{Itv}(E)$. In the scond case, it is impossible for M_i to contain any bound of $\mathrm{Itv}(E)$, by construction; thus $M_i \subseteq \mathrm{Itv}(E)$.

Let $M_i \in \mathcal{M}$ s.t. $M_i \cap \mathrm{Dom}(x) \neq \emptyset$. If M_i is a singleton, it comes immediately that $M_i \subseteq \mathrm{Dom}(x)$; if not, M_i cannot contain any domain interval bound anyway, by construction of $\mathcal{M}$. Hence $M_i \subseteq \mathrm{Dom}(x)$ □

Lemma 25 (Conditioning on a mesh element)**.** *Let ϕ be an IA whose interpretation function is* I*, $x \in \mathrm{Var}(\phi)$, $\mathcal{M}$ a mesh of x in ϕ, and $\vec{x}$ a $\{x\}$-assignment. Let us denote M the element of the mesh such that $\vec{x}(x) \in M$: for any $\{x\}$-assignment $\vec{m}$ such that $\vec{m}(x) \in M$, we have $\mathrm{I}_{|\vec{x}} = \mathrm{I}_{|\vec{m}}$.*

Proof. Let $Z = \mathrm{Var}(\phi) \setminus \{x\}$, and $\vec{z}$ a Z-assignment.

[8] The only operation that creates anything is the stammering one, and recall that either $\mathrm{In}(N_i)$ or $\mathrm{Out}(N_i)$ contains only one element.

[9] The only difficulty is about checking whether two disjoint edges are contiguous. This can simply be done by verifying whether $(I_1 \cup I_2) \cap \mathrm{Dom}(x) = I_{\min} \cap \mathrm{Dom}(x)$, where $I_{\min}$ is the narrowest interval covering $(I_1 \cup I_2) \cap \mathrm{Dom}(x)$. We can show that if the property is not true for Imin, it cannot be true for any I.

($\Rightarrow$) Suppose that $\mathrm{I}_{|\vec{x}}(\vec{z}) = \top$, then $\mathrm{I}(\vec{x} \,.\, \vec{z}) = \top$. Consequently, there exists a path p in ϕ that is compatible with $\vec{x}$ and $\vec{z}$. Let $\vec{m}$ be any $\{x\}$-assignment such that $\vec{m}(x) \in M$. p is compatible with $\vec{m}$, because for any edge E along p verifying $\mathrm{Var}(E) = x$, we know that $M \cap \mathrm{Itv}(E) \neq \emptyset$ (since $\vec{x}(x) \in M$ and $\vec{x}(x) \in \mathrm{Itv}(E)$) and that consequently, by definition of $\mathcal{M}$, $M \subseteq \mathrm{Itv}(E)$. Hence, $\mathrm{I}(\vec{m} \,.\, \vec{z}) = \top$, so $\mathrm{I}_{|\vec{m}}(\vec{z}) = \top$.

($\Leftarrow$) Suppose that $\mathrm{I}_{|\vec{x}}(\vec{z}) = \bot$, then $\mathrm{I}(\vec{x} \,.\, \vec{z}) = \bot$, therefore any path in ϕ that is compatible with $\vec{z}$ is not compatible with $\vec{x}$. Let $\vec{m}$ be any $\{x\}$-assignment such that $\vec{m}(x) \in M$. Suppose that there exists a path p compatible with $\vec{m} \,.\, \vec{z}$: any edge E along p s.t. $\mathrm{Var}(E) = x$ then verifies $\vec{m}(x) \in \mathrm{Itv}(E)$. Hence, by definition of $\mathcal{M}$, $M \subseteq \mathrm{Itv}(E)$; since we chosed M such that $\vec{x}(x) \in M$, we get that $\vec{x}(x) \in \mathrm{Itv}(E)$, which is absurd, as p is not compatible with $\vec{x}$. We infer that $\mathrm{I}(\vec{m} \,.\, \vec{z}) = \bot$ and that consequently $\mathrm{I}_{|\vec{m}}(\vec{z}) = \bot$.

□

Lemma 26 (*Forgetting/ensuring*)**.** *Let ϕ be an IA whose interpretation function is* I, *$x \in \mathrm{Var}(\phi)$, and $\mathcal{M} = \{M_1, \ldots, M_n\}$ a mesh of x in ϕ. Let $(\vec{m}_1, \ldots, \vec{m}_n)$ be a sequence of $\{x\}$-assignments such that for any $1 \leq i \leq n$, $\vec{m}_i(x) \in M_i$. We show that*

- $\mathrm{forget}(\mathrm{I}, \{x\}) = \bigvee_{i=1}^{n} \mathrm{I}_{|\vec{m}_i}$;
- $\mathrm{ensure}(\mathrm{I}, \{x\}) = \bigwedge_{i=1}^{n} \mathrm{I}_{|\vec{m}_i}$.

Proof. Let $Z = \mathrm{Var}(\phi) \setminus \{x\}$, and $\vec{z}$ a Z-assignment.

- Let us prove that $\mathrm{forget}(\mathrm{I}, \{x\}) = \bigvee_{i=1}^{n} \mathrm{I}_{|\vec{m}_i}$:

($\Rightarrow$) Suppose that $\mathrm{forget}(\mathrm{I}, \{x\})(\vec{z}) = \top$. This means there exists a $\{x\}$-assignment $\vec{x}$ such that $\mathrm{I}(\vec{z} \,.\, \vec{x}) = \top$, *i.e.* $\mathrm{I}_{|\vec{x}}(\vec{z}) = \top$. Let i be the integer verifying $\vec{x}(x) \in M_i$ (it exists because $\mathcal{M}$ is a partition of $\mathbb{R}$). Lemma 25 states that $\mathrm{I}_{|\vec{m}_i}(\vec{z}) = \top$. It is hence obvious that $(\bigvee_{i=1}^{n} \mathrm{I}_{|\vec{m}_i})(\vec{z}) = \top$.

($\Leftarrow$) Suppose that $\mathrm{forget}(\mathrm{I}, \{x\})(\vec{z}) = \bot$. Then any $\{x\}$-assignment $\vec{x}$ verifies $\mathrm{I}(\vec{z} \,.\, \vec{x}) = \bot$, and therefore $\mathrm{I}_{|\vec{x}}(\vec{z}) = \bot$. Consequently $(\bigvee_{i=1}^{n} \mathrm{I}_{|\vec{m}_i})(\vec{z}) = \bot$.

- Let us prove that $\mathrm{ensure}(\mathrm{I}, \{x\}) = \bigwedge_{i=1}^{n} \mathrm{I}_{|\vec{m}_i}$:

($\Rightarrow$) Suppose that $\mathrm{ensure}(\mathrm{I}, \{x\})(\vec{z}) = \top$. Then any $\{x\}$-assignment $\vec{x}$ verifies $\mathrm{I}(\vec{z} \,.\, \vec{x}) = \top$, and therefore $\mathrm{I}_{|\vec{x}}(\vec{z}) = \top$. Consequently $(\bigwedge_{i=1}^{n} \mathrm{I}_{|\vec{m}_i})(\vec{z}) = \top$.

($\Leftarrow$) Suppose that $\mathrm{ensure}(\mathrm{I}, \{x\})(\vec{z}) = \bot$. This means there exists a $\{x\}$-assignment $\vec{x}$ such that $\mathrm{I}(\vec{z} \,.\, \vec{x}) = \bot$, *i.e.* $\mathrm{I}_{|\vec{x}}(\vec{z}) = \bot$. Let i be the integer verifying $\vec{x}(x) \in M_i$ (it exists because $\mathcal{M}$ is a partition of $\mathbb{R}$). Lemma 25 states that $\mathrm{I}_{|\vec{m}_i}(\vec{z}) = \bot$. It is hence obvious that $(\bigvee_{i=1}^{n} \mathrm{I}_{|\vec{m}_i})(\vec{z}) = \bot$.

□

Definition 27 (Restriction on a variable)**.** *For any IA ϕ, for any $x \in \mathrm{Var}(\phi)$, for any closed interval $A \in \mathbb{R}$, the restriction of ϕ to $x \in A$, denoted $\phi_{|x \in A}$ is the IA that has the same graph Γ as ϕ, except that for each edge E such that $\mathrm{Var}(E) = x$, $\mathrm{Itv}_{\phi_{|x \in A}}(E) = \mathrm{Itv}_{\phi}(E) \cap A$.*

Note that the order in which restriction operations are performed does not matter: $(\phi_{|x_i \in A})_{|x_j \in B} = (\phi_{|x_i \in B})_{|x_j \in A}$. We shall denote $\phi_{|x_1 \in I_1, \ldots, x_k \in I_k}$ the restriction of ϕ to $x_1 \in I_1, \ldots, x_k \in I_k$.

Lemma 28 (Properties of restriction)**.**

- *For any IA ϕ, for any $x \in \mathrm{Var}(\phi)$, for any closed interval A, $I(\phi_{|x \in A}) = \mathrm{I}(\phi) \wedge f_{x,A}$.*
- *For any IA ϕ, $\mathrm{I}(\phi_{|x_1 \in A_1, \ldots, x_k \in A_k}) = \mathrm{I}(\phi) \wedge f_{x_1,A_1} \wedge \cdots \wedge f_{x_k,A_k}$.*
- *If ϕ is a FIA, then $\phi_{|x_1 \in I_1, \ldots, x_k \in I_k}$ is also a FIA.*

Proof. • Suppose that $\vec{y} \models \phi_{|x \in A}$; then there exists a path in ϕ such that for any edge E in the path, $\vec{y}(\mathrm{Var}(E)) \in \mathrm{Itv}(E)$ whenever $\mathrm{Var}(E) \neq x$, and $\vec{y}(x) \in \mathrm{Itv}(E) \cap A$. Therefore $\vec{y} \models \phi$ and $\vec{y}(x) \in A$: $\vec{y} \models \mathrm{I}(\phi) \wedge f_{x,A}$.
Reciprocally, suppose that $\vec{y} \models \mathrm{I}(\phi) \wedge f_{x,A}$; then $\vec{y} \models \mathrm{I}(\phi)$, and $\vec{y} \models f_{x,A}$.
So, there exists in ϕ a path p from the root to the sink such that for any E on p, $\vec{y}(\mathrm{Var}(E)) \in \mathrm{Itv}(E)$; moreover $\vec{y}(x) \in A$. Hence for any E on p, $\vec{y}(\mathrm{Var}(E)) \in \mathrm{Itv}(E)$ whenever $\mathrm{Var}(E) \neq x$, and $\vec{y}(\mathrm{Var}(E)) \in \mathrm{Itv}(E) \cap A$ whenever $\mathrm{Var}(E) = x$.
Because ϕ and $\phi_{|x \in A}$ have the same edges, and by definition of the intervals labelling the edges of $\phi_{|x \in A}$, this yields that there exists in ϕ a path p from the root to the sink such that for any E on p, $\vec{y}(\mathrm{Var}(E)) \in \mathrm{Itv}(E)$: $\vec{y} \models \mathrm{I}(\phi_{|x \in A})$.
So, $\mathrm{I}(\phi_{|x \in A}) = \mathrm{I}(\phi) \wedge f_{x,A}$.

- $\mathrm{I}(\phi_{|x_1 \in A_1, \ldots, x_k \in A_k}) = \mathrm{I}(\phi) \wedge f_{x_1 \in A_1} \cdots \wedge f_{x_k \in A_k}$ holds because the order in which restriction operations are performed does not matter.
- The last item holds because $A \subseteq B$ implies $(A \cap C) \subseteq (B \cap C)$: if ϕ is focusing, then it is also the case for $\phi_{|x_i \in A_i}$. Repeating the operation for $1 \leq i \leq k$, we get that $\phi_{|x_1 \in A_1, \ldots, x_k \in A_k}$ is focusing.

□

Proof of the `IA` *part of Proposition 18.*

CO, VA, EQ. Any BDD can be translated into an IA in polytime (Prop. 5). Let ϕ be a BDD, and ψ its corresponding IA. If `IA` satisfied **CO**, we would have an algorithm polynomial in $|\psi|$, and thus in the size of ϕ (Lemma 19), to decide whether ϕ is consistent; however, `BDD` does not support **CO** unless $\mathsf{P} = \mathsf{NP}$. The same goes for **VA** and **EQ**.

ME. If **ME** were polynomial, we would be able to decide in polytime whether an IA is inconsistent, as an inconsistent IA has no model; yet **CO** is hard.

MX, CX. If **MX** (resp. **CX**) were polynomial, we had a polytime algorithm for deciding whether an IA is consistent or not, whereas `IA` does not support **CO**.

FO. Similarly, given any IA ϕ, ϕ is consistent iff $\mathrm{forget}(\mathrm{I}(\phi), \mathrm{Var}(\phi)) \equiv \top$. The only reduced IAs without any variable are the empty and the sink-only automata, and testing the emptyness of an IA is done is constant time. If `IA` were satisfying **FO**, we would have a polytime algorithm for deciding the consistency of any IA.

EN. Given any IA ϕ, ϕ is valid iff ensure$(\mathrm{I}(\phi), \mathrm{Var}(\phi)) \equiv \top$. Again, the only reduced IAs without any variable are the empty and the sink-only automata. If `IA` were satisfying **EN**, we would have a polytime algorithm for deciding the validity of any IA.

$\wedge$tC. Thanks to Lemma 28, we only have to syntactically condition ϕ, by replacing the label of each edge E such that $\mathrm{Var}(E) = y_i \in \{y_1, \ldots, y_k\}$ by $\mathrm{Itv}(E) \cap A_i$.

CD, SCD. Let ϕ be an IA, $Y \subseteq \mathrm{Var}(\phi)$ and $\vec{y}$ a Y-assignment. We denote $\phi_{|\vec{y}}$ the IA obtained by replacing, for each node N such as $\mathrm{Var}(N) \in Y$, its label by $\veebar$ and the label of each $E \in \mathrm{Out}(N)$ by $\mathbb{R}$ if $\vec{y}(\mathrm{Var}(E)) \in \mathrm{Itv}(E)$ and by $\emptyset$ otherwise.

By definition of the conditioning, $\vec{z} \in \mathrm{Dom}(\mathrm{Var}(\phi) \setminus \{Y\})$ is a model of $\mathrm{I}(\phi)_{|\vec{y}}$ iff $\vec{y} \,.\, \vec{z}$ is a model of ϕ.

Suppose that $\vec{y} \,.\, \vec{z}$ is a model of ϕ. Then there is in ϕ a path p such that $\vec{z} \,.\, \vec{y} \models p$. By construction, a copy $p_{|\vec{y}}$ of p exists in $\phi_{|\vec{y}}$ and $\vec{z} \models p_{|\vec{y}}$: $\vec{z}$ is a model of $\phi_{|\vec{y}}$.

Suppose that $\vec{y} \,.\, \vec{z}$ is a not model of ϕ: for any path p in ϕ, there is an edge E on this path such that $\vec{y} \,.\, \vec{z}(\mathrm{Var}(E)) \notin \mathrm{Itv}(E)$. Recall that the paths in $\phi_{|\vec{y}}$ are the same than those in ϕ and let $p_{|\vec{y}}$ be the one corresponding to p. If $\mathrm{Var}(E) \in Y$, $\vec{y} \,.\, \vec{z}(\mathrm{Var}(E)) \notin \mathrm{Itv}(E)$ has led to label the corresponding edge in $\phi_{|\vec{y}}$ by $\emptyset$: $\vec{z}$ cannot be compatible with this path. If $\mathrm{Var}(E) \in Y$, $\vec{y} \,.\, \vec{z}(\mathrm{Var}(E)) \notin \mathrm{Itv}(E)$ means that $\vec{z}(\mathrm{Var}(E)) \notin \mathrm{Itv}(E)$: because these edges remain unchanged, $\vec{z}$ cannot be compatible with this path. Therefore $\vec{z}$ is not compatible with any path in $\phi_{|\vec{y}}$: $\vec{z}$ is not a model of $\phi_{|\vec{y}}$.

Hence $\mathrm{I}(\phi_{|\vec{y}}) = \mathrm{I}(\phi)_{|\vec{y}}$.

Since the construction of $\phi_{|\vec{y}}$ is done in polytime, `IA` satisfies **CD** (and consequently **SCD**).

MC. First condition the IA by the X-assignment that is to be checked, say $\vec{x}$. Then reduce: we get either the empty IA (then the assigmement is not a model) or sink-only IA (then $\vec{x}$ is a model).

$\vee$C, $\vee$BC. To make the disjunction of k IAs $\phi_1, \ldots, \phi_k$, simply define a new node, say N, labelled by $\veebar$. Then, for each ϕ_i, add an edge from N to $\mathrm{Root}(\phi_i)$ labelled by $\mathbb{R}$. Fuse the sink nodes of the ϕ_i into a single one.

$\wedge$C, $\wedge$BC. To make the conjunction of k IAs $\phi_1, \ldots, \phi_k$, replace the sink of ϕ_i by the root of ϕ_{i+1}, for all $1 \leq i \leq k-1$. The root of the new IA is the one of ϕ_1, its sink the one of ϕ_k.

SFO. Let ϕ be an IA whose interpretation function is I, and $x \in Z = \mathrm{Var}(\phi)$ the variable to forget. Lemma 24 states that it is possible to build in time quasi-linear in $|\phi|$ a mesh $\mathcal{M} = \{M_1, \ldots, M_n\}$ of x in ϕ. Now, thanks to Lemma 26, we know that $\mathrm{forget}(\mathrm{I}, \{x\}) = \bigvee_{i=1}^{n} \mathrm{I}_{|\vec{m}_i}$; to obtain an IA whose interpretation function is equal to forget$(\mathrm{I}, \{x\})$, it is sufficient to make n simple conditionings and $n-1$ disjunctions. These two operations are polytime (see **SCD** and **$\vee$C** above), and n is linear in $|\phi|$, thus **SFO** is feasible in time polynomial in $|\phi|$.

SEN. Let ϕ be an IA whose interpretation function is I, and $x \in Z = \mathrm{Var}(\phi)$ the variable to ensure. Lemma 24 states that it is possible to build in time quasi-linear in $|\phi|$ a mesh $\mathcal{M} = \{M_1, \ldots, M_n\}$ of x in ϕ. Now, thanks to Lemma 26, we know that $\mathrm{ensure}(\mathrm{I}, \{x\}) = \bigwedge_{i=1}^{n} \mathrm{I}_{|\vec{m}_i}$; to obtain an IA whose interpretation function is equal to ensure$(\mathrm{I}, \{x\})$, it is sufficient to make n simple conditionings and $n-1$ conjunctions. These two operations are polytime (see **SCD** and **$\wedge$C** above), and n is linear in $|\phi|$, thus **SEN** is feasible in time polynomial in $|\phi|$.

□

Lemma 29 (Correspondence between terms, clauses and `FIA`)**.** *Any term (resp. clause) in propositional logic can be expressed in the form of a FIA in polytime.*

Proof. It is straightforward to convert any term (resp. any clause) into an equivalent FBDD in polytime; now Prop. 6 states that any FBDD can be turned into an FIA in polytime. □

Lemma 30 (Correspondence between `DNF` and `FIA`)**.** *Any formula in the* `DNF` *language can be expressed in the form of a FIA in polytime.*

Proof. Lemma 29 states that any term can be turned into a FIA in polytime, and `FIA` supports **$\vee$C**. □

Proof of the `FIA` *part of Proposition 18.*

$\vee$C, $\vee$BC. The same proof as for the IAs obviously hold. If the ϕ_i are focusing, the resulting IA is also focusing.

VA, EQ. Any DNF can be turned into a FIA in polytime (Lemma 30). `DNF` does not satisfy **VA** and **EQ** unless $\mathsf{P} = \mathsf{NP}$. If `FIA` satisfied **VA** (resp. **EQ**), we would have a polytime algorithm for deciding whether a DNF is valid (resp. two DNFs are equivalent).

CO. Let ϕ be a FIA. We can reduce ϕ in polytime (Prop. 14). Now, the only reduced FIA that is inconsistent is the empty automaton. Indeed, suppose that ϕ has at least one variable-edge E (a reduced IA cannot contain only $\veebar$nodes); as $\mathrm{Itv}(E) \cap \mathrm{Dom}(\mathrm{Var}(E)) \neq \emptyset$, there is at least a value ω in $\mathrm{Itv}(E)$ that is coherent with the domain of its variable. As E is focusing, ω is also coherent with the preceding edges in ϕ. Since it is the case for all edges, ϕ cannot be inconsistent.

CX. The following polynomial algorithm (each edge is encountered once) computes the context of y in ϕ:

```
reduce φ
let C := ∅
mark the sink of φ
for all node N in φ, ordered from the sink to the root do
  if N is marked then
    for all E ∈ In(N) do
      if Var(E) = y then
        add Itv(E) to C
      else
        mark Src(E)
```

11: **if** the root of ϕ is marked **then**
12: $\quad \mathcal{C} := \mathrm{Dom}(y)$
13: **return** $\mathcal{C} \cap \mathrm{Dom}(y)$

The idea is to find the y-frontier of the sink (*e.g.* the set of the y-labelled nodes N such that there exists a path from a child of N to the sink not mentioning y), by pulling up a mark meaning that no y-labelled node has been encountered. If a mark reaches the root, there is at least one path from the root to the sink on which there is no y-labelled node, so the context of y in ϕ is $\mathrm{Dom}(y)$ (because ϕ is reduced, so the path is trivially satisfiable, in the same way that a non-empty reduced FIA is satisfiable). If not, the context of y is the union $\mathcal{C}$ of the intervals labelling edges by which the y-frontier access the sink, intersected with the domain of y (this intersection being polytime w.r.t. the number of intervals in $\mathcal{C}$ and $\mathrm{Dom}(y)$, and thus w.r.t. $|\phi|$, which bounds both of them).

MX. Let ϕ be a FIA. If it is consistent (which can be verified in polytime), let $\vec{x} \in \mathrm{Dom}(\mathrm{Var}(\phi)$. First, we reduce ϕ, which can be done in polytime (Prop. 14). Starting from the sink, we chose a path to the root. For each edge E of this path, we chose a value $\omega \in \mathrm{Itv}(E) \cap \mathrm{Dom}(\mathrm{Var}(E))$ (as ϕ is reduced, we know that this set is not empty; the intersection can be done in time polynomial w.r.t. the number of intervals representing $\mathrm{Dom}(\mathrm{Var}(E))$), and assign it to $\vec{x}$: $\vec{x}(\mathrm{Var}(E)) := \omega$, except if we already encountered this variable (we know that the value we chosed before is also compatible with this edge, as ϕ is focusing) or if $\mathrm{Var}(E) = \mathbb{Y}$. When the root is reached, $\vec{x}$ is a model of ϕ (for every unencountered variable, $\vec{x}$ was set to a domain-compatible value at the beginning). This procedure is done in time polynomial in $|\phi|$.

MC. Inferred from the fact that **MC** holds for `IA`.

$\wedge$BC, $\wedge$C. Thanks to Prop. 6, any OBDD can be turned into an equivalent FIA in polytime. If `FIA` supported $\wedge$**BC**, we would have a polytime (Lemma 19) algorithm to decide whether the conjunction of two OBDDs (the variable orderings being possibly different in each OBDD) is consistent, forasmuch as `FIA` supports **CO**; yet, this problem is NP-complete, as shown in Lemma 8.14 of [13]. Therefore `FIA` does not support $\wedge$**BC**, and *a fortiori* does not support $\wedge$**C**. Syntaxe de variable

$\wedge$tC. The same proof as for the IAs holds, since the operation described does not compromise the focusing property, thanks to the last item of Lemma 28.

SEN, EN. Let ϕ_1 and ϕ_2 two FIAs. Let $Z = \mathrm{Var}(\phi_1) \cup \mathrm{Var}(\phi_2)$, and a variable $x \notin Z$ of domain $\mathbb{R}$. We build the automaton ψ by merging the sinks of ϕ_1 and ϕ_2, and adding a node which will be the root of ψ, labelled by x, with one outgoing edge labelled by $\mathbb{R}_-$ and pointing to the root of ϕ_1, and a second outgoing edge labelled by $\mathbb{R}_+$ and pointing to the root of ϕ_2. ψ is obviously focusing, since ϕ_1 and ϕ_2 are, and x is mentioned in one node only. We will prove that $\mathrm{ensure}(\mathrm{I}(\psi), \{x\} = \mathrm{I}(\phi_1) \wedge \mathrm{I}(\phi_2)$, *i.e.* for any Z-assignment $\vec{z}$, $(\mathrm{I}(\phi_1) \wedge \mathrm{I}(\phi_2))(\vec{z}) = \top \Leftrightarrow \forall \vec{x} \in \mathrm{Dom}(\{x\}), \mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \top$.

($\Rightarrow$) Let $\vec{z}$ be a Z-assignment verifying $(\mathrm{I}(\phi_1) \wedge \mathrm{I}(\phi_2))(\vec{z}) = \top$. Let $\vec{x}$ be a $\{x\}$-assignment: either $\vec{x}(x) \in \mathbb{R}_-$, in which case $\mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \top$, as $\mathrm{I}(\phi_1)(\vec{z}) = \top$; or $\vec{x}(x) \in \mathbb{R}_+$, in which case $\mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \top$ too, because this time $\mathrm{I}(\phi_2)(\vec{z}) = \top$.

($\Leftarrow$) Let $\vec{z}$ be a Z-assignment verifying $(\mathrm{I}(\phi_1) \wedge \mathrm{I}(\phi_2))(\vec{z}) = \bot$. Either $\mathrm{I}(\phi_1)(\vec{z}) = \bot$, or $\mathrm{I}(\phi_2)(\vec{z}) = \bot$.In the first case, let us take $\vec{x}$ such that $\vec{x}(x) = -1$: there exists no path in ψ that is compatible with $\vec{x} \,.\, \vec{z}$, since the value of x forces to chose the edge leading to ϕ_1, in which no path is compatible with $\vec{z}$. Symmetrically, in the second case, taking $\vec{x}$ such that $\vec{x}(x) = 1$, there exists no path in ψ that is compatible with $\vec{x} \,.\, \vec{z}$. Hence, there always exists a $\vec{x}$ such that $\mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \bot$.

We thus proved that $\mathrm{ensure}(\mathrm{I}(\psi), \{x\}) = \mathrm{I}(\phi_1) \wedge \mathrm{I}(\phi_2)$. It is possible to build ψ in time linear in the size of ϕ_1 and ϕ_2. Therefore, if `FIA` supported **SEN**, we had a polytime algorithm allowing to build a FIA equivalent to the conjunction of two FIAs. Yet it is impossible, unless $\mathsf{P} = \mathsf{NP}$ (see $\wedge$**BC**). Hence `FIA` does not support **SEN**, and *a fortiori* **EN**.

CD, SCD. Let ϕ be a FIA, $Y \subseteq \mathrm{Var}(\phi)$ and $\vec{y}$ a Y-assignment. Let $\phi_{|\vec{y}}$ be the IA defined in the proof of **CD** on IAs, and $\phi'_{|\vec{y}}$ the IA obtained by applying the reduction operation to $\phi_{|\vec{y}}$. We show that $\phi'_{|\vec{y}}$ is focusing.

Indeed, in $\phi_{|\vec{y}}$ the only edges that have been modified are those whose corresponding edge in ϕ is associated with a variable in Y. In $\phi_{|\vec{y}}$, they are all associated with $\mathbb{Y}$.

As for the other edges in $\phi_{|\vec{y}}$, since they *all* remain unchanged, they are still focusing. We infer from Lemma 21 that $\phi'_{|\vec{y}}$ is focusing w.r.t. all variables in $\mathrm{Var}(\phi) \setminus Y = \mathrm{Var}(\phi'_{|\vec{y}})$. Then Lemma 22 shows that $\phi'_{|\vec{y}}$ is focusing. Because $\phi'_{|\vec{y}} \equiv \phi_{|\vec{y}}$, $\mathrm{I}(\phi_{|\vec{y}}) = \mathrm{I}(\phi)_{|\vec{y}}$ and $\phi'_{|\vec{y}}$ is obtained in polytime, `FIA` supports **CD** (and hence **SCD**).

ME. Let ϕ be a FIA. We check (in polytime) whether ϕ is consistent ; if it is not the case, the empty set is returned. Otherwise, we build a decision tree representing a set of boxes, whose union is equal to $\mathrm{Mod}(\phi)$. For each variable $x_i \in \mathrm{Var}(\phi) = \{x_1, \ldots, x_n\}$, let us build (in time quasi-linear w.r.t. $|\phi|$) a mesh $\mathcal{M}_i = \{M_1^i, \ldots, M_{n_i}^i\}$ of x_i in ϕ, in the way described in the proof of Lemma 24). For each $\mathcal{M}_i$, we consider a sequence $(\vec{m}_1^i, \ldots, \vec{m}_{n_i}^i)$ of $\{x_i\}$-assignments such that $\vec{m}_k^i(x_i) \in M_k^i$. Then, we create a tree T, initially containing only one node, labelled by the empty assignment. We complete the tree thanks to the following process:

1: **for** i from 1 to $|\mathrm{Var}(\phi)|$ **do**
2: $\quad$ let $\mathcal{F}$ be the set of T's leaves.
3: $\quad$ **for all** F in $\mathcal{F}$ **do**
4: $\quad\quad$ let $\vec{z}$ be the $\{x_1, \ldots, x_i\}$-assignment labelling F.
5: $\quad\quad$ **for** j from 1 to n_i **do**
6: $\quad\quad\quad$ **if** $\mathrm{I}(\phi)_{|\vec{z} \,.\, \vec{m}_j^i}$ is consistent **then**
7: $\quad\quad\quad\quad$ add a child to F, labelled by $\vec{z} \,.\, \vec{m}_j^i$.
8: $\quad\quad$ remove F from $\mathcal{F}$.
9: let $\mathcal{B} := \emptyset$
10: **for all** F leaf in T **do**
11: $\quad$ let $\vec{z}$ be the $\mathrm{Var}(\phi)$-assignment labelling F.
12: $\quad$ for each i, let j_i be the integer $(1 \leq j_i \leq n_i)$ such that $\vec{z}(x_i) = \vec{m}_{j_i}^i$ (we know j_i exists, by construction).
13: $\quad$ add to $\mathcal{B}$ the box defined by $M_{j_1}^1 \times \cdots \times M_{j_n}^n$.

Thanks to Lemma 25, we know that conditioning x_i to $\vec{m}_j^i$ gives the same result as to any value in M_j^i; that is why at the end, we return the set of boxes corresponding to the set of assignments we obtained. Since the meshes are partitions of $\mathbb{R}$, we tested every possible value for each variable, so every model of ϕ is in at least one box of $\mathcal{B}$. Since each element of a mesh is either completely included in or completely excluded from the domain of the corresponding variable, every box in $\mathcal{B}$ is included in $\mathrm{Dom}(\mathrm{Var}(\phi))$.

At the end of the algorithm, all the leaves of T are at the same level (*i.e.* the paths of T are of equal length); indeed, for each node, at least one of the tests of l. 6 must pass (as the current FIA is consistent). It implies that at each incrementation of i, $|\mathcal{F}| \leq |\mathcal{B}|$. Moreover, n_i is bounded by $4|\phi|$ for each i (see proof of Lemma 24). Hence, the test of l. 6 is not done more than $4n.|\mathcal{B}|.|\phi|$ in the whole algorithm.

Now, this test is made in time polynomial w.r.t. $|\phi|$, as `FIA` supports **CO** and **SCD**. Hence, the global algorithm is polytime w.r.t. $|\mathcal{B}|$ and $|\phi|$ (as $n \leq \phi$).

SFO. Let ψ be any reduced FIA such that $\mathrm{Var}(\psi) \neq \emptyset$, and let $x \in \mathrm{Var}(\psi)$. We denote $\psi^{\downarrow x}$ the FIA obtained by changing every x-labelled node in ψ by an $\veebar$-labelled node and every outgoing edge of such nodes by a $\mathbb{R}$-labelled edge. We will show that $\mathrm{I}(\psi^{\downarrow x}) = \mathrm{forget}(\mathrm{I}(\psi), \{x\})$, by induction on the number of nodes in ψ.

For $n \in \mathbb{N}$, let $\mathcal{P}(n)$ be the following proposition: "For any reduced FIA ψ such that $\mathrm{Var}(\psi) \neq \emptyset$ and ψ contains n nodes, and any $x \in \mathrm{Var}(\psi)$, $\mathrm{I}(\psi^{\downarrow x}) = \mathrm{forget}(\mathrm{I}(\psi), \{x\})$".

(I) $\mathcal{P}(0)$ and $\mathcal{P}(1)$ are obviously true: the forgetting of any variable in the empty automaton (resp. the sink-only automaton) always returns $\bot$ (resp. $\top$), and ψ is exactly the same as $\psi^{\downarrow x}$ in both cases.

(II) Let $n \geq 2$. Let us suppose that $\mathcal{P}(k)$ is true for all $k < n$. We need to infer that $\mathcal{P}(n)$ is also true. Let ψ be a reduced FIA such that $\mathrm{Var}(\psi) \neq \emptyset$ and ψ contains n nodes, and $x \in \mathrm{Var}(\psi)$. Let $\mathrm{Out}(\mathrm{Root}(\psi)) = \{E_1, \ldots, E_m\}$, and for all $1 \leq i \leq m$, let us denote ψ_i the subgraph rooted at $\mathrm{Dest}(E_i)$. Obviously, each ψ_i contains strictly less than n nodes, so for all $1 \leq i \leq m$,

$$\mathrm{I}(\psi_i^{\downarrow x}) = \mathrm{forget}(\psi_i, \{x\}) \quad . \tag{1}$$

It is also clear that the FIA obtained by replacing in ψ each ψ_i by $\psi_i^{\downarrow x}$, and if $\mathrm{Var}(\mathrm{Root}(\psi)) = x$, modifying $\mathrm{Root}(\psi)$ such that $\mathrm{Var}(\mathrm{Root}(\psi)) = \veebar$ and $\forall E \in \mathrm{Out}(\mathrm{Root}(\psi)), \mathrm{Itv}(E) = \mathbb{R}$, corresponds exactly to $\psi^{\downarrow x}$. We will now show that $\mathrm{I}(\psi^{\downarrow x}) = \mathrm{forget}(\mathrm{I}(\psi), \{x\})$. Let $Z = \mathrm{Var}\,\psi \setminus \{x\}$. We have to prove that

$$\forall \vec{z} \in \mathrm{Dom}(Z), \mathrm{I}(\psi^{\downarrow x})(\vec{z}) = \top \Leftrightarrow \exists \vec{x} \in \mathrm{Dom}(\{x\}), \mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \top \quad .$$

Let $\vec{z}$ be any Z-assignment:

($\Rightarrow$) if $\mathrm{I}(\psi^{\downarrow x})(\vec{z}) = \top$, there is a path in the graph that is compatible with $\vec{z}$: let $E_i^{\downarrow x}$ be the first edge on this path, and $\psi_i^{\downarrow x}$ the subgraph to the root of which it points. We have $\mathrm{I}(\psi_i^{\downarrow x})(\vec{z}) = \top$, by definition of the semantics of an IA. Therefore, thanks to Eq. (1), there exists an $\{x\}$-assignment $\vec{x}$ such that $\mathrm{I}(\psi_i)(\vec{x} \,.\, \vec{z}) = \top$. Let E_i be the edge in ψ coming from $\mathrm{Root}(\psi)$ and pointing to the root of ψ_i.

- if $\mathrm{Var}(E_i) \neq x$, by construction of $\psi^{\downarrow x}$, E_i and $E_i^{\downarrow x}$ have the same variable and labelling interval. Since $E_i^{\downarrow x}$ is compatible with $\vec{z}$, E_i also is, so $\mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \top$.
- if $\mathrm{Var}(E_i) = x$, we can harmlessly suppose that $\vec{x}(x) \in \mathrm{Itv}(E_i)$. Indeed, $\mathrm{I}(\psi_i)(\vec{x} \,.\, \vec{z}) = \top$, so there is a path in ψ_i that is compatible with $\vec{x}$: either there is an x-node along this path, in which case $\vec{x}(x) \in \mathrm{Itv}(E_i)$, as ψ is focusing; or there is not, in which case any value in $\mathrm{Dom}(x)$ can be chosen (and $\mathrm{Dom}(x) \cap \mathrm{Itv}(E_i) \neq \emptyset$, because ψ is reduced). Therefore $\mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \top$.

($\Leftarrow$) if $\mathrm{I}(\psi^{\downarrow x})(\vec{z}) = \bot$, let us consider any $E_i^{\downarrow x} \in \mathrm{Out}(\mathrm{Root}(\psi^{\downarrow x}))$, $\psi_i^{\downarrow x}$ the subgraph to the root of which it points, and E_i the corresponding edge in ψ (*i.e.* the one which points to the root of ψ_i):

- if either $\mathrm{Var}(E_i^{\downarrow x}) = \veebar$ or $\vec{z}(\mathrm{Var}(E_i^{\downarrow x})) \in \mathrm{Itv}(E_i^{\downarrow x})$, we know that $\mathrm{I}(\psi_i^{\downarrow x})(\vec{z}) = \bot$. Therefore, for each $\{x\}$-assignment $\vec{x}$, $\mathrm{I}(\psi_i)(\vec{x} \,.\, \vec{z}) = \bot$ (thanks to Eq. (1)), so E_i belongs to no path in ψ that is compatible with $\vec{x} \,.\, \vec{z}$.
- in the other cases, we have (i) $\mathrm{Var}(E_i^{\downarrow x}) \neq \veebar$ and (ii) $\vec{z}(\mathrm{Var}(E_i^{\downarrow x})) \notin \mathrm{Itv}(E_i^{\downarrow x})$. By construction of $\psi^{\downarrow x}$, (i) implies that E_i and $E_i^{\downarrow x}$ have the same variable and labelling interval. We infer from this and from (ii) that $\vec{z}(\mathrm{Var}(E_i)) \notin \mathrm{Itv}(E_i)$, so we cannot find an $\{x\}$-assignment $\vec{x}$ such that E_i belongs to a path in ψ that is compatible with $\vec{x} \,.\, \vec{z}$.

We deduce from this that there is no $\{x\}$-assignment $\vec{x}$ such that $\mathrm{I}(\psi)(\vec{x} \,.\, \vec{z}) = \top$.

The two points proves that $\mathrm{I}(\psi^{\downarrow x}) = \mathrm{forget}(\mathrm{I}(\psi), \{x\})$; therefore $\mathcal{P}(n)$ is true.

(III) Since both the basis (I) and the inductive step (II) have been proven, we showed by induction that $\mathcal{P}(n)$ holds for all $n \in \mathbb{N}$.

Since for any reduced FIA ψ, building its corresponding $\psi^{\downarrow x}$ only requires to encounter each node and each edge once, we can obtain, in time linear in the size of ψ, a FIA whose interpretation function is equal to $\mathrm{forget}(\mathrm{I}(\psi), \{x\})$. As reduction can be done in polytime on FIAs, this proves that `FIA` supports **SFO**.

FO. As the single forgetting operation can be done in linear time on *reduced* FIAs, the general forgetting operation is polytime on reduced FIAs. As reduction can be done in polytime on FIAs, this proves that `FIA` supports **FO**.

□

B Problems

The classical `ring` problem is described in [6]. The `obsmem` and `robot` problems can be found in the following technical report: [12].

We detail here the two versions of the `drone` problem.

B.1 Discrete Drone Problem

This problem deals with a competition drone (Micro Air Vehicle Conference Competition, see for example

`http://www.mav07.org`) having to achieve different goals on a number of zones.

There are three different kinds of goal:

- identify the target of a given zone, by doing "eight"-shapes above it
- localize the target in a given zone, by scanning it
- drop a marble on the target of a given zone

Each zone contains at most one goal (no target has to be both identified and localized). There is a special "home" zone where the drone takes off and lands; it cannot land anywhere else without losing the competition.

B.1.1 Data

The following data define an instance of the problem:

- An integer $nbZones$, the total number of zones;
- An integer $nbZonesId$, the total number of zones containing a target to identify;
- An integer $nbZonesLoc$, the total number of zones containing a target to localize;
- An integer $nbZonesDrop$, the total number of zones containing a target to touch;
- An integer $givenTime$, the number of minutes given to complete the mission;
- An integer $nbMarblesMax$, the maximal number of marbles that the drone can carry;
- An integer $idDuration$, the number of minutes needed to identify a target;
- An integer $locDuration$, the number of minutes needed to localize a target;
- An integer $dropDuration$, the number of minutes needed to touch a target;
- An integer $toffDuration$, the number of minutes needed to take off;
- An integer $landDuration$, the number of minutes needed to land;
- A sequence of integers $(zoneId_n)_{1 \leq n \leq nbZonesId}$, all different and between 0 and $nbZones - 1$, representing the zones containing a target to identify;
- A sequence of integers $(zoneLoc_n)_{1 \leq n \leq nbZonesLoc}$, all different and between 0 and $nbZones - 1$, representing the zones containing a target to localize;
- A sequence of integers $(zoneDrop_n)_{1 \leq n \leq nbZonesDrop}$, all different and between 0 and $nbZones - 1$, representing the zones containing a target to touch;
- A table of integers $(gotoDuration_n)_{1 \leq i \leq nbZones, 1 \leq j \leq nbZones}$, indicating the time (in minutes) necessary for the drone to go from zone i to zone j.

Since each zone contains at most one target, it is obvious that $nbZonesId + nbZonesLoc + nbZonesDrop \leq nbZones$, and that one integer cannot belong to different sequences.

B.1.2 State Variables

State variables are the following:

- A Boolean *flying* indicating whether the drone is flying;
- An integer $zone$ representing the zone it is in;
- An integer $remTime$, the number of remaining minutes;
- An integer $nbMarbles$, the number of remaining marbles;
- A sequence of Booleans $(goalAch_n)_{0 \leq n < nbZones}$, indicating for each zone whether its corresponding goal has been achieved.

B.1.3 Decision Variables

Decision variables are the following:

- A Boolean $TOFF$, true iff the drone takes off;
- A Boolean $LAND$, true iff the drone lands;
- A Boolean $EIGHT$, true iff the drone identifies a target (making an "eight" above);
- A Boolean $SCAN$, true iff the drone localizes a target (scanning the zone);
- A Boolean $DROP$, true iff the drone drops a marble;
- A Boolean $GOTO$, true iff the drone goes from a zone to another;
- An integer $zoneGOTO$, representing the zone to which heads the drone if $GOTO$ is true.

B.1.4 Preconditions

The set $P(S, D)$ contains constraints deciding which decisions are possible to be made, according to the state.

- The following constraint ($\oplus$ being the "xor" operator)

$$TOFF \oplus LAND \oplus EIGHT \oplus SCAN \oplus DROP \oplus GOTO$$

  forbids that more than one decision is made at the same time.

- Given time:

$$remTime \leq givenTime$$

  There cannot remain more time than the given time.

- Maximal number of marbles:

$$nbMarbles \leq nbMarblesMax$$

  There cannot remain more marbles than the maximum.

- Precondition to the takeoff:

$$TOFF \rightarrow ((zone = home) \wedge \neg flying \wedge (remTime \geq toffDuration))$$

  It must be landed at home, and have enough time to take off.

- Precondition to the landing:

$$LAND \rightarrow ((zone = home) \wedge flying \wedge (remTime \geq landDuration))$$

  It must be flying at home, and have enough time to land.

- Precondition to the identification:

$$EIGHT \rightarrow (flying \wedge (remTime \geq idDuration))$$

  It must be flying and have enough time.

$$EIGHT \rightarrow (zone \in \{zoneId_1, \ldots, zoneId_{nbZonesId}\})$$

  It must be on a zone containing a target to identify.

- Precondition to the localization:

$$SCAN \rightarrow (flying \wedge (remTime \geq locDuration))$$

  It must be flying and have enough time.

$$SCAN \rightarrow (zone \in \{zoneLoc_1, \ldots, zoneLoc_{nbZonesLoc}\})$$

  It must be on a zone containing a target to localize.

- Precondition to the dropping:

$$DROP \rightarrow (\mathit{flying} \wedge (\mathit{remTime} \geq \mathit{dropDuration}) \wedge (\mathit{nbMarbles} > 0))$$

It must be flying, have enough time, and have enough marbles.

$$DROP \rightarrow \left(\mathit{zone} \in \{\mathit{zoneDrop}_1, \ldots, \mathit{zoneDrop}_{\mathit{nbZonesDrop}}\}\right)$$

It must be on a zone containing a target to touch.

- Precondition to the moving

$$GOTO \rightarrow (\mathit{flying} \wedge \left(\mathit{remTime} \geq \mathit{gotoDuration}_{\mathit{zone},\mathit{zoneGOTO}}\right))$$

It must be flying and have enough time to go to the specified zone.

B.1.5 Effects

The set $E(S, D, S')$ contains constraints indicating the resulting state, according to the previous state and the decision made. First, the following constraints describe how the state changes when a given decision is made.

- Effects of a takeoff:

$$TOFF \rightarrow \left(\mathit{flying}' \wedge \left(\mathit{remTime}' = (\mathit{remTime} - \mathit{toffDuration})\right)\right)$$

The drone is flying, and the duration of a takeoff has been removed from the remaining time.

- Effects of a landing:

$$LAND \rightarrow \left(\neg \mathit{flying}' \wedge \left(\mathit{remTime}' = (\mathit{remTime} - \mathit{landDuration})\right)\right)$$

The drone is landed, and the duration of a landing has been removed from the remaining time.

- Effects of an identification:

$$EIGHT \rightarrow (\mathit{remTime}' = (\mathit{remTime} - \mathit{idDuration}))$$

The duration of an identification has been removed from the remaining time.

$$EIGHT \rightarrow \mathit{goalAch}'_{\mathit{zone}}$$

The goal corresponding to this zone has been achieved.

- Effects of an localization:

$$SCAN \rightarrow (\mathit{remTime}' = (\mathit{remTime} - \mathit{locDuration}))$$

The duration of a localization has been removed from the remaining time.

$$SCAN \rightarrow \mathit{goalAch}'_{\mathit{zone}}$$

The goal corresponding to this zone has been achieved.

- Effects of a dropping:

$$DROP \rightarrow \left(\left(\mathit{nbMarbles}' = (\mathit{nbMarbles} - 1)\right) \wedge \left(\mathit{remTime}' = (\mathit{remTime} - \mathit{dropDuration})\right)\right)$$

The drone has lost a marble, and the duration of a dropping has been removed from the remaining time.

$$DROP \rightarrow \mathit{goalAch}'_{\mathit{zone}}$$

The goal corresponding to this zone has been achieved.

- Effects of a move:

$$GOTO \rightarrow \left(\left(\mathit{zone}' = \mathit{zoneGOTO}\right) \wedge \left(\mathit{remTime}' = \left(\mathit{remTime} - \mathit{gotoDuration}_{\mathit{zone},\mathit{zoneGOTO}}\right)\right)\right)$$

The drone is in the specified zone, and the duration of the journey has been removed from the remaining time.

B.1.6 Conditions of state change

These constraints are also in $E(S, D, S')$, but they define all decisions modifying a given state variable.

- Decisions modifying the flying state:

$$\neg \left(\mathit{flying}' \leftrightarrow \mathit{flying}\right) \rightarrow (\mathit{TOFF} \vee \mathit{LAND});$$

The flying state can only change when the drone takes off or lands.

- Decisions modifying the zone:

$$\neg \left(\mathit{zone}' = \mathit{zone}\right) \rightarrow \mathit{GOTO};$$

The zone can only change when the drone moves.

- Decisions modifying the number of marbles:

$$\neg \left(\mathit{nbMarbles}' = \mathit{nbMarbles}\right) \rightarrow \mathit{DROP};$$

The number of marbles canonly change when the drone drops one.

- Decisions modifying the goals achievement state:

$$\begin{aligned}&\neg \left(\mathit{goalAch}'_k \leftrightarrow \mathit{goalAch}_k\right) \rightarrow \\ &\quad ((\mathit{EIGHT} \wedge (k \in \{\mathit{zoneId}_1, \ldots, \mathit{zoneId}_{\mathit{nbZonesId}}\})) \\ &\quad \vee (\mathit{SCAN} \wedge (k \in \{\mathit{zoneLoc}_1, \ldots, \mathit{zoneLoc}_{\mathit{nbZonesLoc}}\})) \\ &\vee (\mathit{DROP} \wedge \left(k \in \{\mathit{zoneDrop}_1, \ldots, \mathit{zoneDrop}_{\mathit{nbZonesDrop}}\}\right)))\end{aligned}$$

A goal can only be achieved if the corresponding decision has been made and the drone is in the corresponding zone.dante.

- Goals associated to targetless zones:

$$\begin{aligned}\mathit{zone} \notin (&\{\mathit{zoneId}_1, \ldots, \mathit{zoneId}_{\mathit{nbZonesId}}\} \\ &\cup \{\mathit{zoneLoc}_1, \ldots, \mathit{zoneLoc}_{\mathit{nbZonesLoc}}\} \\ &\cup \{\mathit{zoneDrop}_1, \ldots, \mathit{zoneDrop}_{\mathit{nbZonesTou}}\}) \\ &\rightarrow \left(\mathit{goalAch}_{\mathit{zone}} \wedge \mathit{goalAch}'_{\mathit{zone}}\right)\end{aligned}$$

The goal is considered as achieved from the start in the zones containing no target.

Goal of the mission

The goal of the mission is given by this constraint:

$$\neg \mathit{flying}' \wedge (\mathit{zone}' = \mathit{home}) \wedge \bigwedge_{k=0}^{\mathit{nbZones}-1} \mathit{goalAch}'_k$$

At the end, the drone must be landed at home, and having achieved all zone goals.

B.2 Continuous Drone Problem

The continuous version of this problem is the same as the discrete version, except for the "remaining time" state variable, that is a real variable; the corresponding data (initial given time, action durations, journey durations) are also real numbers.