ECAI 2010
H. Coelho et al. (Eds.)
IOS Press, 2010

doi:10.3233/978-1-60750-606-5-459

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 [6, 5, 14, 8]. 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 [16], DNNFs [7], 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.

Most of the proofs are omitted for the sake of brevity. An extended version including the proofs can be found at `ftp://ftp.irit.fr/IRIT/RPDMP/PapersFargier/ECAI10-NFPV.pdf`.

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.

[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.

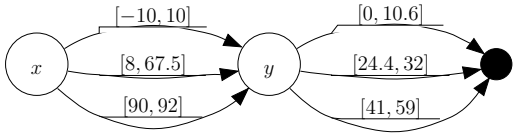

Figure 1. An example of interval automaton. Its model set (see Definition 2), represented as a union of boxes, is $[-10, 10] \times [0, 10.6] \cup [-10, 10] \times [24.4, 32] \cup [-10, 10] \times [41, 59] \cup [8, 67.5] \times [0, 10.6] \cup [8, 67.5] \times [24.4, 32] \cup [8, 67.5] \times [41, 59]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

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

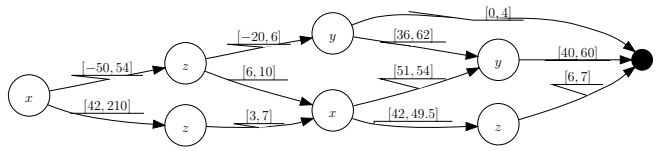

Figure 2. An example of focusing interval automaton. Variable domains are as follows: $\mathrm{Dom}(x) = [0, 100]$, $\mathrm{Dom}(y) = [0, 100]$ and $\mathrm{Dom}(z) = \{0, 3, 7, 10\}$.

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 [16, 1], SLDDs [19] 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 [15].

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 [7], but keep the essence of the decomposability property: they are linkless [11] — 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).

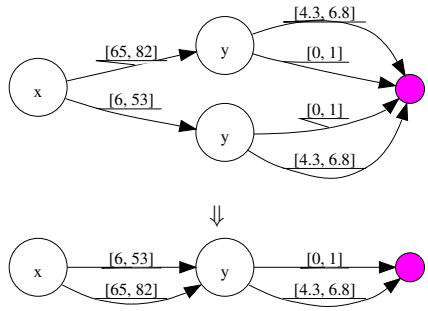

Figure 3. Merging of isomorphic nodes.

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).

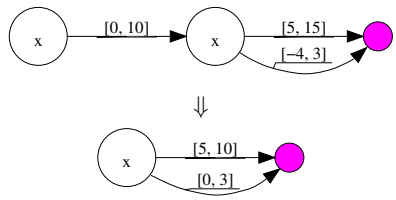

Figure 4. Merging of stammering nodes.

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).

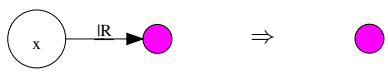

Figure 5. Elimination of undecisive nodes.

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 $[0,8]$.

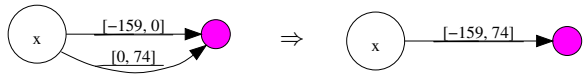

Figure 6. Merging of contiguous edges.

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).

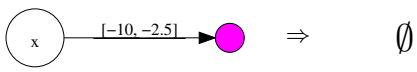

Figure 7. Elimination of unreachable edges (here $\mathrm{Dom}(x) = \mathbb{R}_+$).

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 [7].

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	$\vee$C	$\wedge$BC	$\wedge$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 [10], using the trace of a search algorithm as a convenient way to transform a CSP into an FIA [12]. 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 [9] 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 the extended version 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 [13]. 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 [18]), 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 [17]), 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] A. Darwiche, 'Decomposable Negation Normal Form', *Journal of the ACM*, **48**(4), 608–647, (2001).

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

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

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

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

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

[12] 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).

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

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

[15] 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).

[16] N. R. Vempaty, 'Solving Constraint Satisfaction Problems Using Finite State Automata', in *AAAI*, pp. 453–458, (1992).

[17] Alberto Venturini and Gregory Provan, 'Incremental Algorithms for Approximate Compilation', in *AAAI*, pp. 1495–1499, (2008).

[18] Toby Walsh, 'SAT v CSP', in *CP*, pp. 441–456, (2000).

[19] Nic Wilson, 'Decision Diagrams for the Computation of Semiring Valuations', in *IJCAI*, pp. 331–336, (2005).