Handling RealPaver's Output by Interval Automata Compilation

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

[1] ONERA Toulouse, 2 av. Edouard Belin, 31400 Toulouse, France
{aniveau, pralet, verfaillie}@onera.fr
[2] IRIT-CNRS, Université Paul Sabatier, Toulouse, France
fargier@irit.fr

Abstract. Interval-based solvers are useful for various applications, but their output is not always succinct nor easy to handle. We propose a *target compilation language* designed to represent the solution set of a constraint network such that the compiled form is compact and allows to process a number of common operations in polytime. We also show a way to compile the solution set into our language by using the trace of an interval-based solver, RealPaver.

Introduction

When there is a need to solve a numerical problem, it may be interesting to use an interval-based constraint solver. Users can easily model their problems as sets of constraints on real variables; they are guaranteed that the output of the solver will contain all solutions, or the optimum if they have specified an objective function.

One of the drawbacks of interval-based solvers is that their output is often verbose, and not always handleable. In order to exploit the solution set, and adapt it to its needs, the user may want to keep only a certain kind of solution (*e.g.*, solutions for which a given variable is in a given interval), to know which values of a variable are compatible with at least one solution, to propagate the changes, etc.

In this perspective, it could be interesting to use *knowledge compilation* [DM02], which consists in representing the solution set in some *target compilation language*, such that the compiled form is as compact as possible and allow the desired operations to be processed in polytime.

Efficient target languages have been proposed in the past for enumerated domain variables (*e.g.* OBDDs [Bry86], finite-state automata [Vem92], DNNFs [DM02], etc.); however, we need here to represent assignments of both continuous and enumerated variables.

The purpose of this paper is to provide a target language which represents as compactly as possible the solution set of a constraint network with continuous variables as well as discrete ones, and allows in polytime a number of important operations. We will focus on the particular structure of *interval automata*.

1 Interval Automata

1.1 Structure and Semantics

Definition 1 (Interval automaton) *An* interval automaton *(IA) is a couple $\langle X, \Gamma \rangle$, 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 leave (the* sink*), whose non-leaf nodes are labelled by a variable of X, and whose edges are labelled by a closed interval from* $\mathbb{R}$.

This definition allows an interval automaton 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 provides an example of interval automaton.

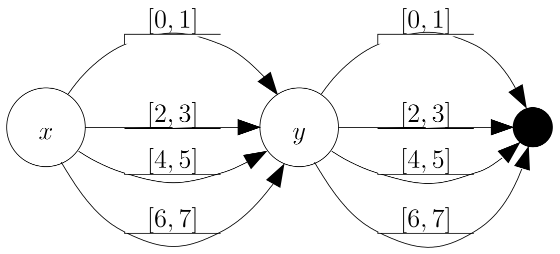

Fig. 1. An example of interval automaton. Its model set (see Definition 2), represented as a union of boxes (which is the format of RealPaver's output), is $[0,1] \times [0,1] \cup [0,1] \times [2,3] \cup [0,1] \times [4,5] \cup [0,1] \times [6,7] \cup [2,3] \times [0,1] \cup [2,3] \times [2,3] \cup [2,3] \times [4,5] \cup [2,3] \times [6,7] \cup [4,5] \times [0,1] \cup [4,5] \times [2,3] \cup [4,5] \times [4,5] \cup [4,5] \times [6,7] \cup [6,7] \times [0,1] \cup [6,7] \times [2,3] \cup [6,7] \times [4,5] \cup [6,7] \times [6,7]$.

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 ($\mathrm{Dom}(x) = [1, 7] \cup [23.4, 28]$). 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 $\overrightarrow{y}$ denotes an Y*-assignment* of variables from Y, *i.e.* $\overrightarrow{y} \in \mathrm{Dom}(Y)$. $\overrightarrow{y}(y_i)$ denotes the value assigned to y_i in $\overrightarrow{y}$. For a set $B \subseteq \mathrm{Dom}(Y)$ of Y-assignments, we denote $B_{|y_i}$ the *projection* of B on variable y_i, defined as $B_{|y_i} = \{\omega \in \mathrm{Dom}(y_i) \mid \exists \overrightarrow{y} \in B, \overrightarrow{y}(y_i) = \omega\}$

Let $\phi = \langle X, \Gamma \rangle$ be an interval automaton, N a node and E an edge in Γ. We denote:

– $\mathcal{N}_\phi$ (resp. $\mathcal{E}_\phi$) the set of nodes (resp. edges) of Γ;
– $\mathrm{Root}(\phi)$ the root of Γ and $\mathrm{Sink}(\phi)$ its sink;
– $|\phi|$ the *size* of Γ, *i.e.* its number of edges;
– $\mathrm{Out}(N)$ (resp. $\mathrm{In}(N)$) the set of outgoing (resp. incoming) edges of N;
– $\mathrm{Var}(N)$ the labelling variable of N (we decide to denote $\mathrm{Var}(\mathrm{Sink}(\phi)) = \emptyset$);
– $\mathrm{Ch}(N)$ (resp. $\mathrm{Pa}(N)$) the set of children (resp. parents) of N;
– $\mathrm{Src}(E)$ the node from which E comes and $\mathrm{Dest}(E)$ the node to which E points;
– $\mathrm{Itv}(E)$ the labelling interval of E;
– $\mathrm{Var}(E)$ the variable of E, defined as $\mathrm{Var}(E) = \mathrm{Var}(\mathrm{Src}(E))$.

An interval automaton can be seen as a compact representation of some set of variable assignments.

Definition 2 (Semantics of an interval automaton) *Every interval automaton ϕ on X (*i.e. *with* $\text{Var}(\phi) = X$*) represents a function from* $\text{Dom}(X)$ *to* $\{\top, \bot\}$*. It is called its* interpretation function $\text{I}(\phi)$*, defined as follows: for any X-assignment $\overrightarrow{x}$,* $\text{I}(\phi)(\overrightarrow{x}) = \top$ *if and only if there exists a path from the root to the sink of ϕ such that for each edge E on this path,* $\overrightarrow{x}(\text{Var}(E)) \in \text{Itv}(E)$*.*

The model set *of ϕ is the subset of* $\text{Dom}(X)$ *defined by* $\text{Mod}(\phi) = \{\overrightarrow{x} \in \text{Dom}(X) \mid \text{I}(\phi)(\overrightarrow{x}) = \top\}$*. Its elements are the* models *or* solutions *of ϕ. ϕ is said to be* equivalent *to another IA ψ (denoted $\phi \equiv \psi$) if and only if* $\text{Mod}(\phi) = \text{Mod}(\psi)$*.*

Let us notice that the interpretation function of the empty automaton always returns $\bot$, since an empty IA contains no path from the root to the sink; its model set is thus empty. 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. Therefore, its model set is $\text{Dom}(X)$.

As it can be seen on Figure 1, the size of the extended representation (*i.e.* union of boxes) of an IA's model set can be exponentially larger than the size of the automaton. This is notably due to the fact that IAs can be *reduced* by suppressing redundancies, in the manner of BDDs.

1.2 Relation with BDDs

Interval automata are, in many respects, similar to BDDs; this is why we will now rapidly recall this notion.

Introduced by Bryant in [Bry86], *binary decision diagrams* (BDDs)[3] 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 $\bot$.

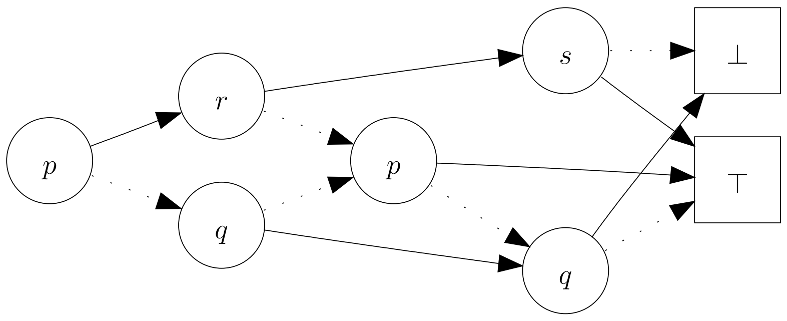

Fig. 2. An example of BDD. It represents (not in the most succinct way...) the boolean formula $(p \wedge \neg r) \vee (p \wedge r \wedge s) \vee (\neg p \wedge \neg q)$.

The interpretation function of BDDs is similar to the one of IAs: for a given assignment of the boolean variables, the function's value is $\top$ if and only if there exists a path from the root to the

[3] In fact, the data structure presented in [Bry86] is a restriction of the BDD language, that has been later named the FBDD language (free BDD), for which each variable can appear at most one time on a path from the root to a sink.

$\top$-labelled leaf such that the given assignment is coherent with each edge along the path (each edge representing a possible value of the variable labelling its source node). Since each non-leaf node cannot have only one outgoing edge, each assignment is coherent with at least one path from the root to a sink, so there is an equivalent definition of the interpretation function: its value is $\top$ if and only if there exists no path from the root to the $\bot$-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, the remaining graph has the same structure as an IA. Hence, it is easy to translate a BDD into an IA:

Lemma 3 (Correspondence between IAs and BDD) *Any sentence in the* BDD *language can be expressed in the form of an equivalent IA, in time polynomial in the sentence's size.*

Proof. As boolean variables can be represented by real variables with domain $\{0,1\}$, a BDD sentence can be transformed into an IA by removing its "false" 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 $\top$-labelled edge is replaced by a $[1,1]$-labelled edge, and each $\bot$-labelled edge is replaced by a $[0,0]$-labelled edge.

This *polynomial translatability* will help prove further propositions.

The truth table of a boolean formula can be exponentially larger than a BDD representing this formula. Indeed, the graph format allows to factorize identical subfonctions, by means of the reduction operation. It has two parts: suppression of one-child nodes (their incoming edges being redirected to their child) and merging of identical subgraphs. The same kind of process can be applied to interval automata.

1.3 Reduction

Like a BDD, an interval automaton can be reduced in size without changing its semantics, by merging redundant nodes and non-disjoint edges. The reduction operations introduced thereafter are based on the notions of isomorphic, stammering and undecisive nodes, and of contiguous and unreachable edges, which are the reducible elements of an IA. Some of these notions are just generalizations of definitions introduced in the context of BDDs [Bry86], while others are specific to interval automata. Each example is illustrated in Figure 3.

Definition 4 (Isomorphic nodes) *Two 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.

Definition 5 (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 undesirable, since the information they bring could harmlessly be deported to their parents.

Definition 6 (Undecisive node) *A node N of an IA ϕ is* undecisive *if and only if* $|\operatorname{Out}(N)| = 1$ *and* $E \in \operatorname{Out}(N)$ *is such that* $\operatorname{Dom}(\operatorname{Var}(E)) \subseteq \operatorname{Itv}(E)$*.*

An undecisive node does not restrain the solutions corresponding to the paths it is in; it is "automatically" crossed.

Definition 7 (Contiguous edges) *Two edges E_1, E_2 of an IA ϕ are* contiguous *if and only if*

- $\operatorname{Src}(E_1) = \operatorname{Src}(E_2)$*;*
- $\operatorname{Dest}(E_1) = \operatorname{Dest}(E_2)$*;*
- $\operatorname{Itv}(E_1) \cap \operatorname{Itv}(E_2) \neq \emptyset$*.*

Contiguous edges both come from the same node, both point to the same node and are not disjoint: they could be replaced by a single edge.

Definition 8 (Unreachable edge) *An edge E of an IA ϕ is* unreachable *if and only if* $\operatorname{Itv}(E) \cap \operatorname{Dom}(\operatorname{Var}(E)) = \emptyset$

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

Definition 9 (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.*

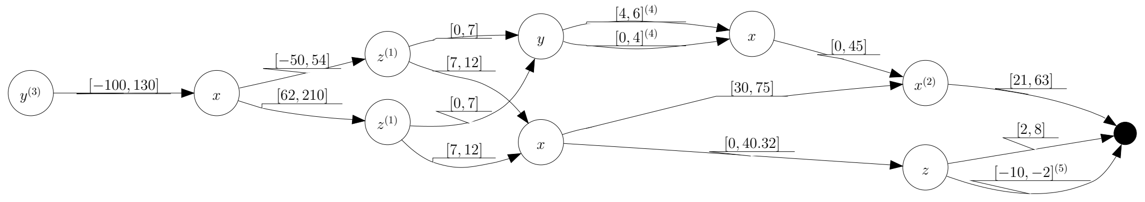

Fig. 3. An example of what can be reduced in an interval automaton. Variable domains are as follows: $\operatorname{Dom}(x) = [0, 100]$, $\operatorname{Dom}(y) = [0, 100]$ and $\operatorname{Dom}(z) = \{0, 3, 7, 10\}$. The two nodes marked $^{(1)}$ are isomorphic; the node $^{(2)}$ is stammering; the node $^{(3)}$ is undecisive; the edges marked $^{(4)}$ are contiguous; and the edge $^{(5)}$ is unreachable.

In the following, we can consider only reduced IAs since reduction is polynomial; we prove this after a lemma allowing to make *traversals* of IAs from the root to the sink or conversely.

Lemma 10 (Numbering of an IA) *There exists a polytime algorithm that numbers the nodes of any IA ϕ in such a way that if $N_i \in \operatorname{Ch}(N_j)$, then $i < j$.*

Proposition 11 (Reduction of an IA) *Any IA ϕ can be turned into an equivalent reduced IA in polytime.*

Proof. The following polytime algorithm takes as input an IA ϕ and modifies it in such a way that when it stops, ϕ is reduced but its interpretation function has not changed. 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 $\phi$ in such a way that if $N_i \in \mathrm{Ch}(N_j)$ then $i < j$
3:   **for** $i$ from 1 to $|\mathcal{N}_\phi|$ **do**
4:     **if** $N_i$ is stammering **then**
5:       **for all** $E_{\text{in}} \in \mathrm{In}(N_i)$ **do**
6:         **for all** $E_{\text{out}} \in \mathrm{Out}(N_i)$ **do**
7:           add an edge going from $\mathrm{Src}(E_{\text{in}})$ to $\mathrm{Dest}(E_{\text{out}})$ labelled by $\mathrm{Itv}(E_{\text{in}}) \cap \mathrm{Itv}(E_{\text{out}})$
8:       suppress $N_i$
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** $E_{\text{in}} \in \mathrm{In}(N_i)$ **do**
20:        redirect $E_{\text{in}}$ to the child of $N_i$
21:      suppress $N_i$
22:    **for** $j$ from 1 to $|\mathcal{N}_\phi|$ **do**
23:      **if** $N_i$ and $N_j$ are isomorphic **then**
24:        **for all** $E_{\text{in}} \in \mathrm{In}(N_i)$ **do**
25:          redirect $E_{\text{in}}$ to $N_j$
26:        suppress $N_i$
27: **until** $\phi$ has not changed during process
```

The computation for each node is obviously[4] polynomial and the traversal loop (l. 3) treats each node once; as a result, what is inside of the first loop (from l. 2 to l. 26) 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. 27). This does not change the polynomiality: as the traversal loop can only remove edges and nodes, it will not be repeated more than $|\phi|$ times.

There obviously exists more efficient methods to reduce an IA, but the only point here is to show hat this operation is polytime.

Figure 4 shows the result of applying this algorithm on the IA of Figure 3.

2 Operations on IA

2.1 Definitions

To be handleable, our target compilation language should support a number of interesting queries and transformations, which we define here.

[4] Computation of stammering nodes is polynomial because either $\mathrm{In}(N_i)$ or $\mathrm{Out}(N_i)$ contains only one element.

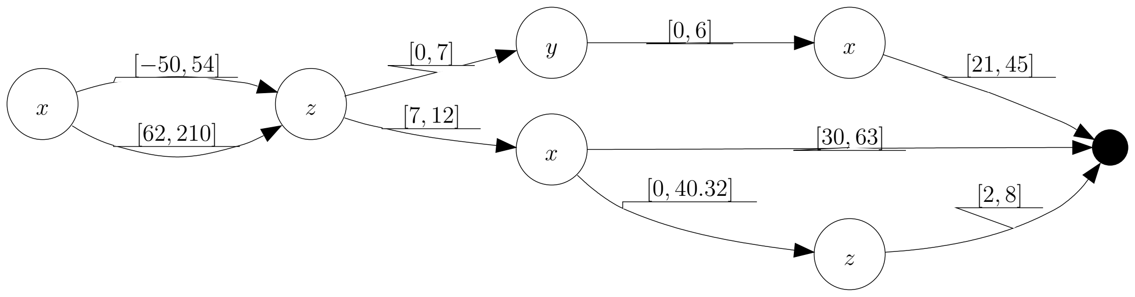

Fig. 4. The reduced IA equivalent to the IA of Figure 3.

First of all, we may need to focus on a certain part of the solution set, and ignore what is outside of it. This corresponds to the conditioning of an interval automaton.

Definition 12 (Conditioning) *Let ϕ be an interval automaton on X, y a variable from X and γ a closed interval. An interval automaton ψ is a* conditioning *of ϕ by γ on y if and only if for all X-assignment $\overrightarrow{x}$, $\mathrm{I}(\psi)(\overrightarrow{x}) = \mathrm{I}(\phi)(\overrightarrow{x})$ if $\overrightarrow{x}(y) \in \gamma$, and else $\mathrm{I}(\psi)(\overrightarrow{x}) = \bot$.*

Figure 5 provides an example of conditioning.

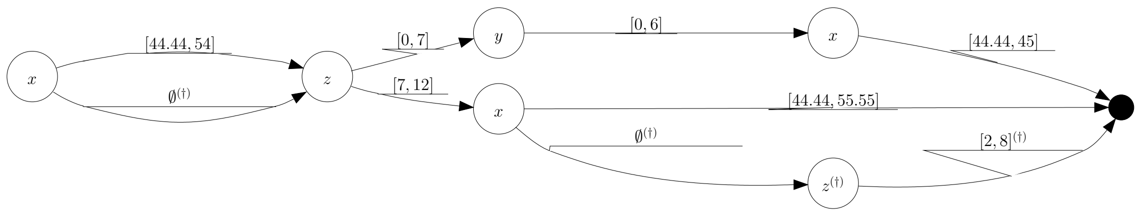

Fig. 5. A conditioning of the IA of Figure 4 by $[44.44, 55.55]$ on x. It has been obtained by the syntactic process described in the proof of Proposition 16, and has not been reduced; the elements that would be removed by a reduction are marked with a $^{(\dagger)}$.

It would be useful to know whether or not the solution set is empty, and, given a variable, what values are compatible with at least one solution.

Definition 13 (Consistency, context) *An interval automaton ϕ is said to be* consistent *if and only if $\mathrm{Mod}(\phi) \neq \emptyset$.*

A value $\omega \in \mathbb{R}$ is said to be consistent *for a variable $y \in X$ in an interval automaton ϕ on X, if and only if there exists an X-assignment $\overrightarrow{x}$ in $\mathrm{Mod}(\phi)$ such that $\overrightarrow{x}(y) = \omega$. The set of all consistent values for y in ϕ is called the* context *of y in ϕ, denoted $\mathrm{Ctxt}(\phi, y)$.*

Sometimes, we only need to know what is the highest or the lowest value that a variable can take in the solution set. We call these values the context bounds of the variable.

Definition 14 (Context bounds) *When* $\mathrm{Ctxt}(\phi, y) \neq \emptyset$*, we call* lower bound *(resp.* upper bound*)of* y *in* ϕ*, and denote* $\mathrm{Low}(\phi, y)$ *(resp.* $\mathrm{Upp}(\phi, y)$*), the lower (resp. upper) bound of the context of* y *in* ϕ*,* i.e. $\mathrm{Low}(\phi, y) = \min \mathrm{Ctxt}(\phi, y)$ *and* $\mathrm{Upp}(\phi, y) = \max \mathrm{Ctxt}(\phi, y)$.

After a conditioning, we may want to eliminate from the automaton the values that are obviously inconsistent, *i.e.* lower than the lower bound or higher than the higher bound of their corresponding variable. In other words, we want our interval automaton to be bound-consistent.

Definition 15 (Bound-consistency) *An edge* E *in an interval automaton* ϕ *is said to be* bound-consistent *if and only if the bounds of* $\mathrm{Itv}(E)$ *are consistent.*

A bound-consistent *interval automaton (BCIA) is an IA whose edges are all bound-consistent.*

2.2 Complexity

Now that we know on which operations we will focus, let us decide whether interval automata are handleable, *i.e.* support these operations in polytime.

Proposition 16 (Conditioning of an IA) *There exists a polytime algorithm able to build, for any IA* ϕ*, any variable* $y \in \mathrm{Var}(\phi)$ *and any closed interval* γ*, an IA* ψ *that is a conditioning of* ϕ *by* γ *on* y*.*

Proof. We only have to syntactically condition ϕ, by replacing the label of each edge E such that $\mathrm{Var}(E) = y$ by $\mathrm{Itv}(E) \cap \gamma$.

Obviously, it is not difficult to condition an interval automaton; unfortunately, it will not be the same for the other operations. To prove it, we will use Lemma 3; indeed, deciding whether an IA is consistent cannot be easier than deciding whether a BDD is.

Proposition 17 (Consistency of an IA) *Unless* $\mathsf{P} = \mathsf{NP}$*, there exists no polytime algorithm able to check whether any IA* ϕ *is consistent.*

Proof. If such an algorithm existed, we could check in time polynomial whether a BDD sentence is consistent, by translating it into an IA (Lemma 3); yet it is impossible (unless $\mathsf{P} = \mathsf{NP}$), as shown in [DM02].

This implies that extracting the context or the context bounds is not polynomial.

Proposition 18 (Context extraction for an IA) *Unless* $\mathsf{P} = \mathsf{NP}$*, there exists no polytime algorithm able, for any IA* ϕ *and any variable* $y \in \mathrm{Var}(\phi)$*, to return the context of* y *in* ϕ*.*

Proof. If it were the case, Proposition 17 would be wrong, as an IA ϕ is consistent if and only if for each variable $y \in \mathrm{Var}(\phi)$, the context of y in ϕ is not empty.

Proposition 19 (Bounds extraction for an IA) *Unless* $\mathsf{P} = \mathsf{NP}$*, there exists no polytime algorithm able, for any IA* ϕ *and any variable* $y \in \mathrm{Var}(\phi)$*, either to return the bounds of* y *in* ϕ *or to stop without returning anything if* ϕ *is inconsistent.*

Proof. If it were the case, Proposition 17 would be false.

Finally, we deduce from the latter proposition that it is impossible to obtain bound-consistency on any IA in polytime:

Proposition 20 (Obtain bound-consistency on an IA) *Unless* $\mathsf{P} = \mathsf{NP}$*, there exists no polytime algorithm able, for any IA* ϕ*, to return an equivalent IA* ψ *which is bound-consistent.*

Proof. Extracting the lower (resp. upper) bound of a variable y in a BCIA is polytime: it is the minimum (resp. the maximum) of all the lower (resp. upper) bounds of the intervals that label y-edges. Consequently, if it were possible to turn any IA into a BCIA in polytime, Proposition 19 would be false.

The raw interval automata language is therefore insufficient for our application. We need to impose a restrictive property on this language, in order to make it handleable.

3 Towards a Handleable Class of Interval Automata : FIAs

Let us define a category of interval automata, the focusing ones, that will support more interesting queries and transformations.

3.1 Definition

One of the reasons why these operations are hard to apply to interval automata is that they have no hierarchical structure: on a given path, the intervals concerning the same variable can get bigger after having shrunk, and conversely. We will therefore consider focusing IAs, *i.e.* IAs in which intervals can only shrink from the root to the sink.

Definition 21 (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 Figure 6.

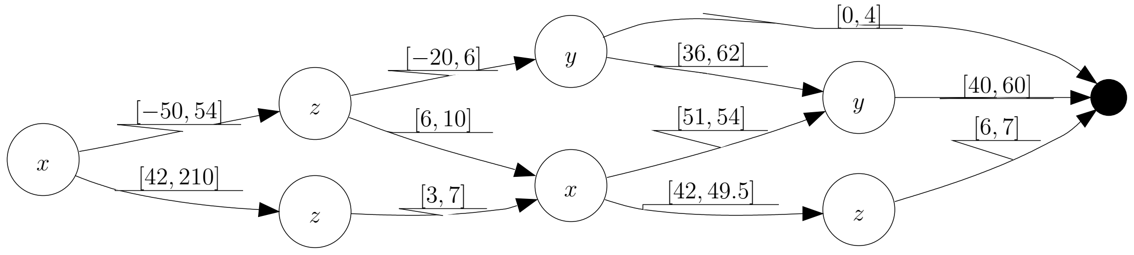

Fig. 6. An example of focusing interval automaton. Variable domains are as follows: $\mathrm{Dom}(x) = [0, 100]$, $\mathrm{Dom}(y) = [0, 100]$ and $\mathrm{Dom}(z) = \{0, 3, 7, 10\}$.

The first result we get is that FIAs are not harder to reduce than IAs:

Proposition 22 (Reduction of FIAs) *There exists a polytime algorithm that transforms any FIA* ϕ *into an equivalent reduced FIA* ψ*.*

Proof. The algorithm proposed in Proposition 11, applied on a FIA, maintains the focusing property. Indeed, no operation is made that enlarges any interval[5].

3.2 Operations

We show here that the focusing restriction make IAs handleable. First, the conditioning remains polynomial:

Proposition 23 (Conditioning on FIAs) *There exists a polytime algorithm able to build, for any FIA ϕ, any variable $y \in \mathrm{Var}(\phi)$ and any closed interval γ, a FIA ψ that is a conditioning of ϕ by γ on y.*

Proof. The algorithm sketched in the proof of Proposition 16, applied on a FIA, obviously maintains the focusing property (indeed, if $\gamma_1 \subseteq \gamma_2$, then $\gamma_1 \cap \gamma \quad \subseteq \quad \gamma_2 \cap \gamma$).

Moreover, this time, checking consistency can be done in polytime:

Proposition 24 (Consistency of a FIA) *There exists a polytime algorithm able to check whether any FIA ϕ is consistent.*

Proof. We can reduce ϕ in polytime (prop. 22). Now, the only reduced FIA that is inconsistent is the empty automaton. Indeed, suppose that ϕ has at least one edge E; 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.

Context extraction and consequently bounds extraction are also polytime.

Proposition 25 (Context extraction for a FIA) *There exists a polytime algorithm able, for any FIA ϕ and any variable $y \in \mathrm{Var}(\phi)$, to return the context of y in ϕ.*

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

```
1:  reduce $\phi$
2:  let $C := \emptyset$
3:  mark the sink of $\phi$
4:  **for all** $N \in \mathcal{N}_\phi$, ordered from the sink to the root **do**
5:    **if** $N$ is marked **then**
6:      **for all** $E \in \mathrm{In}(N)$ **do**
7:        **if** $\mathrm{Var}(E) = y$ **then**
8:          $C := C \cup \mathrm{Itv}(E)$
9:        **else**
10:         mark $\mathrm{Src}(E)$
11: **if** the root of $\phi$ is marked **then**
12:   $C := \mathrm{Dom}(y)$
13: **return** $C$
```

[5] The reduction of contiguous edges does *literally* enlarge intervals, but only with values proven not to be out of focus (if $\gamma_1 \subseteq \gamma$ and $\gamma_2 \subseteq \gamma$, then $\gamma_1 \cup \gamma_2 \subseteq \gamma$)...

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 and non-empty, and therefore consistent). If not, the context of y is the union of the intervals labelling edges by which the y-frontier access the sink.

Corollary 26 (Bounds extraction for a FIA) *There exists a polytime algorithm able, for any FIA ϕ and any variable $y \in \mathrm{Var}(\phi)$, either to return the bounds of y in ϕ or to stop without returning anything if ϕ is inconsistent.*

Finally, we show that we can easily obtain bound-consistency on a FIA.

Proposition 27 (Obtain bound-consistency on a FIA) *There exists a polytime algorithm able, for any FIA ϕ, to return an equivalent FIA ψ which is bound-consistent.*

Proof. Inferred from Proposition 23 and Corollary 26; indeed, it is sufficient to condition ϕ by $[\mathrm{Low}(\phi, y), \mathrm{Upp}(\phi, y)]$, for all $y \in \mathrm{Var}(\phi)$.

4 Building FIAs from the Trace of a Constraint Solver

We saw that FIA is an interesting target language, since it allows useful operations in polytime. We now focus on how to build such automata.

4.1 Algorithms

We will adopt a process similar to [HD05]: using the trace of an algorithm to build a compiled form. Here, we will use the interval-based solver RealPaver [GB06] to create an interval automaton representing an approximation of the solution set of a constraint network.

Algorithme 1 `RealPaver(`$\mathcal{C}$`, `B`, `ϵ`)`: returns an union U of boxes not larger than ϵ, such that U is included in the box B, and the solution set of the CSP $\mathcal{C}$ is included in U.

```
1:  $B^P$ := Prune($\mathcal{C}, B$)
2:  if $B^P = \emptyset$ then
3:      return $\emptyset$
4:  if $B^P$ is not included in the solution set of $\mathcal{C}$ then
5:      if $B^P$ is not more precise than $\epsilon$ then
6:          choose the variable $y$ to split on
7:          $S$ := Split($\mathcal{C}, B^P, y$)
8:          $U := \emptyset$
9:          for all $B^S \in S$ do
10:             $U := U \cup$ RealPaver($\mathcal{C}, B^S, \epsilon$)
11:         return $U$
12: return $B_P$
```

Algorithm 1 is a simplified version of RealPaver's generic branch-and-prune algorithm. It uses two internal functions that we do not detail: The function `Prune` reduces the given box by removing values that are proven not to be in the solution set; it returns the reduced box. The function `Split`, applied on a given box and a given variable, splits the box w.r.t. the variable and returns a set of smaller boxes that cover the initial box.

Algorithm 2 is the "twin" of Algorithm 1, but instead of returning a union of boxes, it builds a corresponding interval automaton. The conversion is not that obvious: there is a problem, due to the fact that Algorithm 1 is not forced to split along every variable. Thus, if our algorithm creates IA nodes only when RealPaver splits, some variables may not appear in the resulting IA. This is annoying, because if RealPaver prunes w.r.t. these variables, this will not be taken into account in the IA, and its model set will be bigger than RealPaver's output. For example, if we solve the problem $5 \leq x \leq 6$ in RealPaver, with an initial box $x \in [0, 10]$, the box will be pruned, but there will be no splitting; therefore, the resulting box will be $[5, 6]$, but the IA's model set will be $[0, 10]$.

This is why it is necessary to create nodes corresponding to the pruning of a box. However, creating them at each step would be wasteful: he information brought by such nodes is useful only if the algorithm does not split w.r.t. their labelling variable afterwards. One solution is to maintain a set L of "unspecified variables" (*i.e.* w.r.t. which the algorithm has pruned, but not yet split), in order to create according nodes at the end.

In order to be blindingly obvious, additions to Algorithm 1 are framed in Algorithm 2. Note that this algorithm only builds a *graph* $\mathit{\Gamma}$; to be a proper interval automaton, the latter must be associated with the set X of variables whose domains define the initial box.

Algorithme 2 `IA-Builder(`$\mathcal{C}$`, `B`, `ϵ`, `L`)`: returns an interval automaton that represents the set calculated by `RealPaver(`$\mathcal{C}$`, `B`, `ϵ`)`. L is the set of variables that must be "specified" at the end.

```
1:  $B^P :=$ Prune$(\mathcal{C}, B)$
2:  | add to $L$ the variables $y$ such that $B^P_{|y} \neq B_{|y}$ |
3:  if $B^P = \emptyset$ then
4:      return the empty automaton
5:  if $B^P$ is not included in the solution set of $\mathcal{C}$ then
6:      if $B^P$ is not more precise than $\epsilon$ then
7:          choose the variable $y$ to split on
8:          | $L := L \setminus \{y\}$ |
9:          $S :=$ Split$(\mathcal{C}, B^P, y)$
10:         | create a node $N$ labelled by $y$ |
11:         for all $B^S \in S$ do
12:             let $\psi :=$ IA $-$ Builder$(\mathcal{C}, B^S, \epsilon, L)$
13:             | if $\psi$ is not empty then
14:             |     add to $N$ an outgoing edge labelled by $B^S_{|y}$ and pointing to the root of $\psi$ |
15:         | if $N$ has no outgoing edge then
16:         |     return the empty automaton |
17:         return the automaton rooted at $N$
18: | let $\phi$ be the sink-only interval automaton
19: | for all $y \in L$ do
20: |     create a $y$-labelled node $N$ with one outgoing edge labelled by $B^P_{|y}$ and pointing to Root$(\phi)$
21: |     replace $\phi$ by the IA rooted at $N$ |
22: return $\phi$
```

Let us illustrate Algorithm 2 on an example. Let $\mathcal{C}$ be the constraint problem having two variables, x and y, with $\mathrm{Dom}(x) = [0, 1]$ and $\mathrm{Dom}(y) = [0, 2]$, and the following set of constraints:

$$\begin{cases} y \geq 2\sqrt{\max(0, 0.25 - x^2)} \\ y \leq 2 - 2\sqrt{\max(0, 0.25 - x^2)} \\ y \geq 2\sqrt{\max(0, 0.25 - (x-1)^2)} \\ y \leq 2 - 2\sqrt{\max(0, 0.25 - (x-1)^2)} \end{cases} \quad .$$

A graphical view of the solution set of $\mathcal{C}$ is proposed in Figure 4.1. The union of boxes returned by RealPaver for this problem with $\epsilon = 1$ is

$$[0.5, 0.5669872981077808] \times [1.5, 2] \cup [0.5, 1] \times [1, 1.5] \cup [0.4330127018922192, 0.5] \times [1.5, 2] \cup \\ [0, 0.5] \times [1, 1.5] \cup [0.5, 1] \times [0.5, 1] \cup [0.5, 0.5669872981077808] \times [0, 0.5] \cup \\ [0, 0.5] \times [0.5, 1] \cup [0.4330127018922192, 0.5] \times [0, 0.5] \quad .$$

Figure 4.1 shows the corresponding IA obtained with Algorithm 2.

4.2 Properties of the Resulting IAs

Structure The resulting interval automata will always satisfy the focusing property. Indeed, both splitting and pruning nodes can only restrict the current box: no labels are created, that are larger than an ancestor edge's label.

This algorithm has not been tested yet, but we can imagine that resulting FIAs will have a particular structure: a binary[6] tree from the leaves of which are hung chains of nodes (most often of length zero) going to the sink.

Consistency Note that if we obtain the empty automaton, it means that the problem is inconsistent, since RealPaver does not leave solutions. However, the consistency of the automaton does not imply the consistency of the problem: indeed, RealPaver can return boxes even if the problem is inconsistent, and our resulting IAs are only translations of RealPaver output. Getting a consistent IA only means that RealPaver has not proven the problem to be inconsistent.

We nonetheless lose some information by compiling into IAs, since RealPaver's boxes can be of two types: inner and outer ones. An inner box is guaranteed to be included in the solution set, and is thus a proof of consistency. Yet, interval automata do not allow to represent different kinds of sets (an idea could be to make two sinks, an "inner" and an "outer" one...).

Considerations about Enumerated Variables As for enumerated variables, using the trace of RealPaver does not *a priori* takes advantage of the IA structure. Indeed, it does not accept "intervals of integers": given a problem having only one integer variable k with $\mathrm{Dom}(k) = \{0, 1, 2, 3\}$ and no constraint, RealPaver will return the union of boxes $[0, 0] \cup [1, 1] \cup [2, 2] \cup [3, 3]$, as all variables are considered as real ones. The sink-only interval automaton would suffice to represent this solution set; unfortunately, if we build the IA corresponding to the trace of RealPaver, as the intervals are disjoint, reduction will be null and void.

[6] The tree is binary if RealPaver splits boxes in two parts, but other modes are possible.

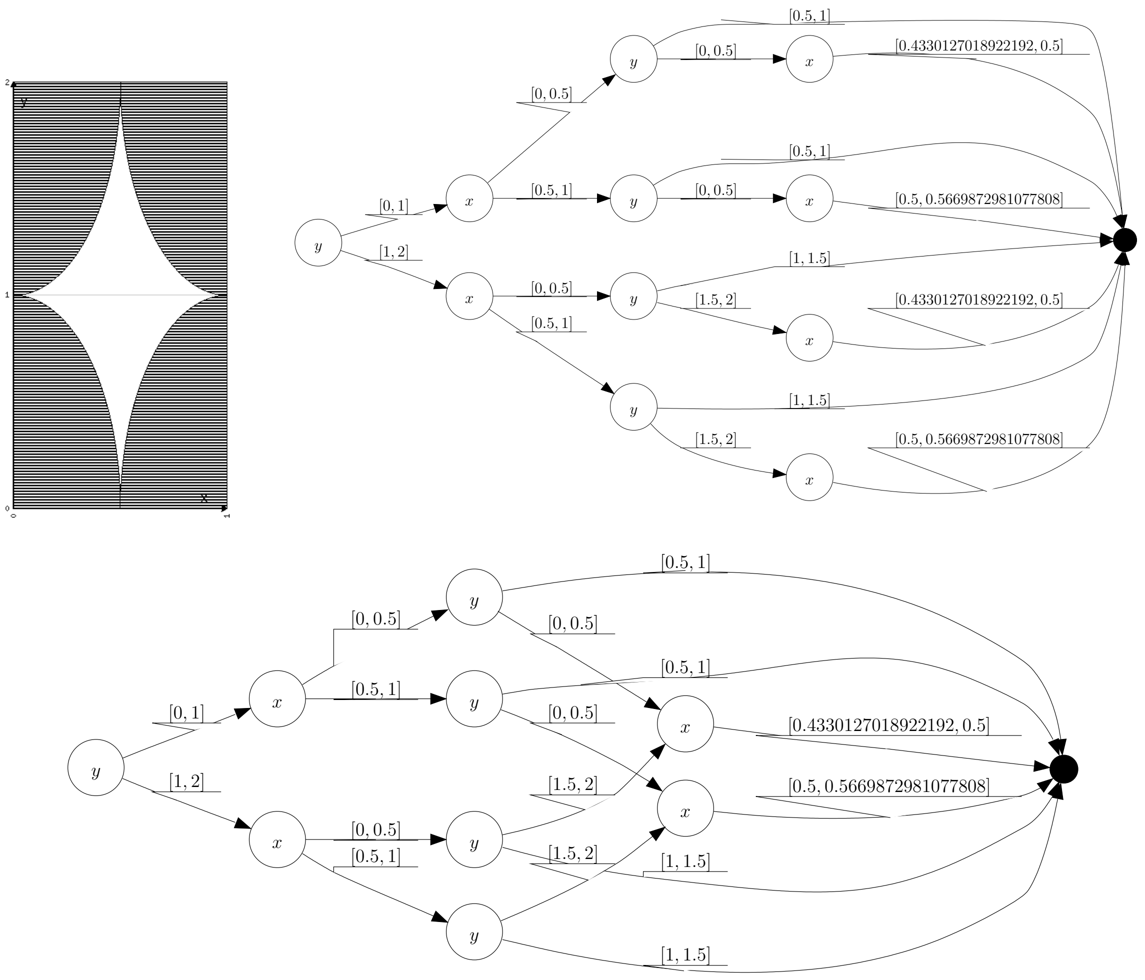

Fig. 7. This figure illustrates Algorithm 2. Top-left picture is a graphical representation of problem $\mathcal{C}$ defined in the text: the solution set is the non-hatched area. The top-right IA is the one obtained by following Algorithm 2. The IA at bottom is the same, after reduction.

Conclusion

The FIA structure saves memory space thanks to reduction, and supports various operations that one may need when handling the solution set of some mixed constraint network. Moreover, we showed a convenient way to build FIAs, by using the trace of the interval-based solver RealPaver.

Future work will have to decide whether our propositions are efficient once implemented, and will address various questions, *e.g.* which heuristics of RealPaver give the most succinct FIAs? Does our algorithm creates enough isomorphisms for the compiled form to be efficiently reducible? Indeed, before reduction, the size of computed FIAs is likely to be exponential in the number of variables of the problem; if reduction is not efficient, it is useless that operations be polytime. Moreover, interval-based solvers can be very slow when paving large continuum of solutions, so it is possible that compiling an interval automaton takes infinite time.

Some ways to solve these issues could be to find another means to build FIAs (for example, as a conjunction of constraints...), to use approximate compilation, which consists in keeping only certain solutions to improve the succinctness of the compiled form [OP06], to stray from the compilation framework by iteratively refining IAs online according to the user's requests, or to search for a different, more appropriate target language.

References

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *Institute of Electrical and Electronics Engineers (IEEE) Transactions on Computers*, 35(8):677–691, 1986.

[DM02] Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

[GB06] Laurent Granvilliers and Frédéric Benhamou. Algorithm 852: RealPaver: an Interval Solver Using Constraint Satisfaction Techniques. *ACM Trans. Math. Softw.*, 32(1):138–156, 2006.

[HD05] Jinbo Huang and Adnan Darwiche. DPLL with a Trace: From SAT to Knowledge Compilation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 156–162, 2005.

[OP06] Barry O'Sullivan and Gregory M. Provan. Approximate Compilation for Embedded Model-Based Reasoning. In *Association for the Avancement of Artificial Intelligence Conference (AAAI)*, 2006.

[Vem92] Nageshwara Rao Vempaty. Solving Constraint Satisfaction Problems Using Finite State Automata. In *Association for the Avancement of Artificial Intelligence Conference (AAAI)*, pages 453–458, 1992.