An abstract domain for trees with numeric relations *

Matthieu Journault[1], Antoine Miné[1,2], Abdelraouf Ouadjaout[1]

[1] Sorbonne Université, CNRS,
Laboratoire d'Informatique de Paris 6, LIP6,
F-75005 Paris, France
[2] Institut universitaire de France
(matthieu.journault‖antoine.mine‖abdelraouf.ouadjaout)@lip6.fr

Abstract. We present an abstract domain able to infer invariants on programs manipulating trees. Trees considered in the article are defined over a finite alphabet and can contain unbounded numeric values at their leaves. Our domain can infer the possible shapes of the tree values of each variable and find numeric relations between: the values at the leaves as well as the size and depth of the tree values of different variables. The abstract domain is described as a product of (1) a symbolic domain based on a tree automata representation and (2) a numerical domain lifted, for the occasion, to describe numerical maps with potentially infinite and heterogeneous definition set. In addition to abstract set operations and widening we define concrete and abstract transformers on these environments. We present possible applications, such as the ability to describe memory zones, or track symbolic equalities between program variables. We implemented our domain in a static analysis platform and present preliminary results analyzing a tree-manipulating toy-language.

1 Introduction

The abstract interpretation framework [5] enables the development of sound static analyzers by inferring and proving invariants on reachable states of programs. Invariants in the scope of abstract interpretation are elements of a lattice called an abstract domain. Most domains focus on numeric or pointer variables. By contrast, we propose an abstract domain for variables whose values are tree data-structures. Tree values appear natively in some languages (such as OCaml) and applications (such as the DOM in web programming) or can be encoded through pointer manipulations (as in C). Trees can abstract terms in logic programming. A tree domain can also be useful to collect symbolic expressions appearing in a program.

* This work is supported by the European Research Council under Consolidator Grant Agreement 681393 – MOPSA.

```c
typedef struct node
{
  int data;
  struct node* next;
} node;

node* append(node* head, int data)
{
  if (head==NULL) {
    return (create(data, NULL));
  } else {
    node *cursor=head;
    while(cursor->next != NULL)
      cursor=cursor->next;
    node* new_node=create(data,NULL);
    cursor->next=new_node;
    return head;
  }
}
```

Program 1: Append to list in C

```c
float golden_ratio(int n) {
  int i = 0;
  float r = 1;
  while (i < n) {
    r = 1 + 1 / r;
    i += 1;
  }
  return r;
}
```

Program 2: Golden ratio in C

```
let rec f x n =
  match n with
  | 0 -> []
  | _ -> (x+1)::(x-1)::(f x (n-1))

let () =
  (*Assume x:int and n:int>=0*)
  let t = f x n in
  match t with
  | [] -> ()
  | p :: q when p > x -> ()
  | _ -> assert false
```

Program 3: List type in OCaml

Used memory zones. Prog. 1 describes an `append` function defined in the C language, this function adds an integer at the end of a linked list. The infinite set of unbounded terms of the form `*(*( ...*(head + 4) ...+ 4) + 4)` represents memory zones that are used by the `append` function. Our analyzer is able to infer and represent such sets of terms. This provides the information that Prog. 1 does not use any of the `data` field of the linked list. Such a function would be fairly commonly called in a real-life project. In a classical top-down static analysis by abstract interpretation, function calls are inlined at each call site. A way to improve scalability is to design modular analyzers able to reuse previous analysis results (as emphasized in [7]). In order to be able to successfully reuse function body analysis, input states must be unified. Moreover the cost of performing the analysis of the body of functions grows with the number of variables that need to be tracked. A common way to deal with both problems is to use framing on the inputs of the functions (as in separation logic [25]). This improves (1) precision: as we know that they are not modified by the function call, (2) body analysis efficiency: as the input state is reduced and finally (3) modularity: as constraints on the usage of the first analysis are relaxed by the removal of constraints.

Symbolic relations. Prog. 2 is a C function computing an approximation of the golden ration (as it is the limit of the sequence $r_0 = 1, r_{n+1} = 1+\frac{1}{r_n}$). As classical numerical domains can not represent such numerical relations, methods were proposed to track symbolic equality between expressions (see [23]). However such methods can not handle the unbounded iteration of Prog. 2. The set of reachable states at the end of Prog. 2 can be expressed by $\mathtt{r} = 1+1/(1+1/\ldots 1\ldots)$ with depth `n`. Please note that to infer such results we need to express numerical relations between the size of trees and the numeric variables from the program.

Numerical environment. Consider now the OCaml Prog. 3, we want to prove that the `assert false` expression is never reached. This program builds a list of size $2 * \mathtt{n}$ with alternating values $\mathtt{x}+1$ and $\mathtt{x}-1$. The assertion states that the head of the list is $\mathtt{x}+1$. After the definition of `t` there are two types of reachable states. (1) Those that have not gone through the loop ($\mathtt{t} \mapsto \mathtt{[]}, \mathtt{x} \mapsto \mathbb{Z}, \mathtt{n} \mapsto 0$), and (2) those that have gone through at least one iteration of the loop: ($\mathtt{t} \mapsto \mathtt{[}a_1\mathtt{;}a_2\mathtt{;}\ a_3\mathtt{;}\ \mathtt{...]}, x \mapsto \alpha, \mathtt{n} > 0, a_1 \mapsto \alpha+1, a_2 \mapsto \alpha-1, a_3 \mapsto \alpha+1$), where $\alpha \in \mathbb{Z}$. Therefore we need to be able to keep numerical relations between the parametric and unbounded number of numeric values appearing in `t` and numeric variables from the program. Classical numeric domains do not provide out-of-the-box abstractions for sets of partially defined numerical functions, therefore we define such an abstraction. As an example of analysis result, the memory representation obtained by our analysis for `t` describes the set of trees of the form: `Cons(`a`, Cons(`b`, Cons(`a`, ..., Nil) ...))` where $a = \mathtt{x}+1$ and $b = \mathtt{x}-1$. Therefore we are able to prove that the `assert false` expression is never reached.

Contributions. The main contributions of the article are threefold: (1) The extension of results on tree automata to the abstract interpretation framework by definition of a widening operator, in order to represent the set of tree shapes that a variable can contain. (2) The definition of a numerical domain built upon classical abstract domains able to represent sets of partial numerical maps with heterogeneous and unbounded definition sets. This is necessary to represent the numeric values at the leaves of a set of trees, as trees are unbounded and can contain a different number of leaves. (3) The definition of a novel abstraction for trees that can contain numerical values at their leaves. This last domain combines the abstractions (1) and (2). Moreover it is relational as it can express relations between numerical values found in trees and in the rest of the program, and relations between trees. Finally all results were implemented in an existing framework and experimented on a toy-language.

Limitations. At this point, analyses can only be performed on the toy language presented thereinafter, not on real life code, therefore we do not present any benchmark results, even though examples of analysis results will be put forth. Indeed Prog. 1, 2 and 3 were precisely analyzed once encoded into our toy-language (see Prog 4 and Prog 5).

Outline. We start, in Sec. 2, by presenting the concrete semantic we want to abstract. In Sec. 3 we build a first abstraction which forgets numerical values and focuses on abstracting tree shapes. Sec. 4 presents a novel numerical abstract domain required for the definition of the abstract domain of Sec. 5, which aims at precisely representing numerical constraints between trees and program variables. In Sec. 6 we provide remarks on the implementation and results of the analyzer. Finally Sec. 7 mentions related works while Sec. 8 concludes.

Notations. Classical Galois connections (see [5]) are denoted $(A, \subseteq_A) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (B, \subseteq_B)$. When no best abstraction can be defined, we use the *representation* framework (as defined by Bourdoncle in [3], also known as concretization only framework), representations are denoted by $(A, \subseteq_A) \overset{\gamma}{\leftarrow} (B, \subseteq_B)$. $A \nrightarrow B$ denotes the set of partial maps from A to B, and $\lambda_{|A} x.f(x) \in B$ denotes the map in $A \to B$ that associates $f(x)$ to x. Finally when $f \in A \to C$ and $g \in B \to C$, with $A \cap B = \emptyset$, $f \uplus g$ is the function defined on $A \cup B$, that associates $f(x)$ (resp. $g(x)$) to x whenever $x \in A$ (resp. $x \in B$).

2 Syntax and concrete semantics

Definition 1. *An* alphabet $\mathcal{F}$ *is a finite set, a* ranked alphabet *is a pair* $\mathcal{R} = (\mathcal{F}, a)$ *where* $\mathcal{F}$ *is an alphabet and* $a \in \mathcal{F} \to \mathbb{N}$*. For* $f \in \mathcal{F}$*, we call* arity *of* f *the value* $a(f)$*. We assume that* $\mathbb{Z}$ *and* $\mathcal{F}$ *are disjoint and we define the set of* natural terms *over* $\mathcal{R}$ *(denoted* $T_{\mathbb{Z}}(\mathcal{R})$*) to be the smallest set defined by:*
- $\mathbb{Z} \subseteq T_{\mathbb{Z}}(\mathcal{R})$
- $\forall p \geq 0,\ f \in \mathcal{F}, t_1, \ \ldots, t_p \in T_{\mathbb{Z}}(\mathcal{R}),\ a(f) = p \Rightarrow f(t_1, \ldots, t_p) \in T_{\mathbb{Z}}(\mathcal{R})$

Moreover when $\mathcal{R}$ *contains at least one symbol of arity* 0*, we define* terms *over* $\mathcal{R}$ *(denoted* $T(\mathcal{R})$*) to be the smallest set defined by:*
- $\forall p \geq 0,\ f \in \mathcal{F}, t_1, \ \ldots, t_p \in T(\mathcal{R}),\ a(f) = p \Rightarrow f(t_1, \ldots, t_p) \in T(\mathcal{R})$

In the following, $\mathcal{F}_n$ *denotes the subset of* $\mathcal{F}$ *of arity* n*. Moreover given a term* $t \in T(\mathcal{R})$ *we denote* $f = \mathbf{head}(t) \in \mathcal{F}$ *and* $\mathbf{sons}(t)$ *a possibly empty tuple* $(t_1, \ldots, t_n)$ *of elements of* $T(\mathcal{R})$ *such that* $t = f(t_1, \ldots, t_n)$*.*

Remark 1. Numerical leaves are defined to contain integers, however this could be modified to rationals, real numbers or floats. We are parametric in the type of numeric values, as they are delegated to an underlying numerical domain.

Example 1. Consider the ranked alphabet $\mathcal{R} = \{\texttt{*}(1), \texttt{\&}(1), \texttt{+}(2), \texttt{x}(0)\}$, $u(n)$ means that symbol u has arity n. Then $\texttt{\&x} \in T(\mathcal{R})$, but $\texttt{*(\&x+4)} \in T_{\mathbb{Z}}(\mathcal{R})$, and $\texttt{*(\&x+4)} \notin T(\mathcal{R})$. Using this alphabet we can model C pointer arithmetic.

Example 2. $U = \{+(x, y) \mid x \leq y\}$ and $V = \{+(x, +(z, y)) \mid x \leq y \wedge z \leq y\}$ are two sets of natural terms over $\mathcal{R} = \{+(2)\}$ which we use as running examples.

Syntax of the language and concrete operations. We assume already defined a small imperative language and extend it (in Fig. 1) with statements, tree expressions (*tree-expr*) which are expressions that are evaluated to trees, and simple symbol expressions (*sym-expr*) which enable the manipulation of symbols. We add the ability to build a tree which contains only a numerical leaf: $\texttt{make\_integer}(e)$, the ability to read the i-th son of a tree t: $\texttt{get\_son}(t, i)$, ... Fig. 2 defines concrete operations over the set $\wp(T_{\mathbb{Z}}(\mathcal{R}))$. Fig. 2 assumes given a set of program numerical variables $\mathcal{V}$, a set of numerical expressions (over $\mathcal{V}$) denoted *expr*, a set of statements *stmt*, a notion of numerical environment $E \in \mathfrak{E} = \mathcal{V} \to \mathbb{Z}$, a set of tree program variables $\mathcal{T}$, a notion of tree environment

$$
\begin{array}{rcl}
\textit{tree-expr} & \triangleq & \mid \texttt{make\_symbolic}(\mathcal{F}, \\
& & \quad \textit{tree-expr}, \ldots, \textit{tree-expr}) \\
& & \mid \texttt{make\_integer}(\textit{expr}) \\
& & \mid \texttt{get\_son}(\textit{tree-expr}, \textit{expr}) \\
\textit{stmt} & \triangleq & \ldots \\
& & \mid \mathcal{T} \texttt{ = } \textit{tree-expr}
\end{array}
\qquad
\begin{array}{rcl}
\textit{sym-expr} & \triangleq & \mid \texttt{get\_sym\_head}(\textit{tree-expr}) \\
\textit{expr} & \triangleq & \ldots \\
& & \mid \texttt{get\_num\_head}(\textit{tree-expr}) \\
& & \mid \texttt{is\_symbol}(\textit{tree-expr}) \\
& & \mid \textit{sym-expr} \texttt{ == } \mathcal{F}
\end{array}
$$

Fig. 1: Syntax extension of the language

$$
\begin{aligned}
&\mathbb{E}[\![\texttt{make\_symbolic}(s \in \mathcal{F}_m, T_1, \ldots, T_m)]\!](E,F) = \{s(t_1, \ldots, t_m) \mid \forall i,\ t_i \in \mathbb{E}[\![T_i]\!](E,F)\} \\
&\mathbb{E}[\![\texttt{make\_integer}(e \in \textit{expr})]\!](E,F) = \mathbb{E}[\![e]\!](E,F) \\
&\mathbb{E}[\![\texttt{is\_symbol}(T)]\!](E,F) = \{\texttt{true} \mid \exists t \in \mathbb{E}[\![T]\!](E,F), \exists f \in \mathcal{R},\ t = f(\ldots)\} \\
&\qquad\qquad \cup \{\texttt{false} \mid \exists t \in \mathbb{E}[\![T]\!](E,F), t \in \mathbb{Z}\} \\
&\mathbb{E}[\![\texttt{get\_son}(T,e)]\!](E,F) = \{t \mid \exists i \in \mathbb{E}[\![e]\!](E,F),\ t' \in \mathbb{E}[\![T]\!](E,F), f \in \mathcal{F}_{m>i}, \\
&\qquad\qquad t' = f(t_0, \ldots, t_{m-1}) \wedge t_i = t\} \\
&\mathbb{E}[\![\texttt{get\_num\_head}(T)]\!](E,F) = \{i \in \mathbb{Z} \mid \exists t \in \mathbb{E}[\![T]\!](E,F),\ t = i\} \\
&\mathbb{E}[\![\texttt{get\_sym\_head}(T)]\!](E,F) = \{s \in \mathcal{R} \mid \exists t \in \mathbb{E}[\![T]\!](E,F),\ t = s(\ldots)\}
\end{aligned}
$$

Fig. 2: Concrete operations on natural terms

```
int i;
int n;
tree y;
assume(n >= 0);
i = 0;
y = make_symbolic("p",{});
while (i < n) {
  y = make_symbolic("*",
           {make_symbolic("+",
                 {y,
                  make_integer(4)
                 })
           });
  i = i+1;
 }
```

Program 4: `*(p+4)` iterated

```
int n; int i; int x; int rep;
tree t;
assume(n>=0);
i = 0;
t = make_symbolic("Nil",{});
while (i < n) {
  t = make_symbolic("Cons",
        {make_integer(x-1), t});
  t = make_symbolic("Cons",
        {make_integer(x+1), t});
  i = i + 1;
 };
if (get_sym_head(t) != "Nil") {
    rep = get_num_head(get_son(t,0));
    assert(rep > x);
 }
```

Program 5: List manipulation

$F \in \mathfrak{F} = \mathcal{T} \to \wp(T_{\mathbb{Z}}(\mathcal{R}))$, $D = E \times F$ is our concrete domain. Finally we assume already partially defined on numerical expressions an evaluation function $\mathbb{E}[\![e \in \textit{expr}]\!](E \in \mathcal{V} \to \mathbb{Z}, F \in \mathcal{T} \to \wp(T_{\mathbb{Z}}(\mathcal{R}))) \in \wp(\mathbb{Z})$. Using this operator we are able to define Prog. 4 which computes the memory zones used by `append` from Prog 1, and Prog. 5 that simulates the behavior of Prog. 3.

3 Natural term abstraction by tree automata

In this section we start by defining a value abstraction for tree sets (in Sec. 3.1), which is then lifted to an environment abstraction (in Sec. 3.2).

3.1 Value abstraction

As a first abstraction for natural terms, we put aside numerical values and define an abstraction able to describe sets of tree shapes. Tree automata enable the description of set of terms built upon a finite ranked alphabet. The ranked alphabet of the language we want to analyze is extend with the $\square$ symbol to denote potential positions of numerical values.

Definition 2 (Finite tree automata). *A* finite tree automaton *(FTA) over a ranked alphabet $\mathcal{R}$ is a tuple $(Q, \mathcal{R}, Q_f, \delta)$, where Q is a (finite) set of states, $Q_f \subseteq Q$ is the set of final states, and $\delta \in \wp(\bigcup_{n\in\mathbb{N}} \mathcal{F}_n \times Q^n \times Q)$ is the set of transitions. We define $\overline{\delta} : (\bigcup_{n\in\mathbb{N}} \mathcal{F}_n \times Q^n) \to \wp(Q)$ by: $\overline{\delta}(f, \overrightarrow{q}) = \{q' \mid (f, \overrightarrow{q}, q') \in \delta\}$. When $\overline{\delta}$ is such that, $\forall n \in \mathbb{N},\ f \in \mathcal{F}_n,\ \overrightarrow{q} \in Q^n,\ |\overline{\delta}(f, \overrightarrow{q})| = 1$, we say that the automaton is complete and deterministic (CDFTA). We then abuse notations and denote by $\delta(f, \overrightarrow{q})$ the unique element in the set $\overline{\delta}(f, \overrightarrow{q})$.*

Definition 3 (Reachability). *Given a FTA $\mathcal{A} = (Q, \mathcal{R}, Q_f, \delta)$ we define, a reachability function* $\textsc{reach}_{\mathcal{A}} : T(\mathcal{R}) \to \wp(Q)$

$$\textsc{reach}_{\mathcal{A}}(t) = \textbf{let } t_1, \ldots, t_n = \textbf{sons}(t) \textbf{ in} \bigcup_{(q_1,\ldots,q_n)\in(\textsc{reach}_{\mathcal{A}}(t_1),\ldots,\textsc{reach}_{\mathcal{A}}(t_n))} \overline{\delta}(\textbf{head}(t), (q_1, \ldots, q_n))$$

If $\textbf{sons}(t)$ *is the empty tuple (which is the case when t is a constant a), the union is made over a unique element (which is the empty tuple), which then boils down to: $\overline{\delta}(a, ())$. If* $\textbf{sons}(t)$ *is not the empty tuple and for some i,* $\textsc{Reach}_{\mathcal{A}}(t_i)$ *is empty, then* $\textsc{Reach}_{\mathcal{A}}(t)$ *is also empty*

Example 3. Consider the ranked alphabet $\mathcal{R} = \{f(2), a(0)\}$, and the automaton $\mathcal{A} = (\{u, v\}, \mathcal{R}, \{v\}, \{a() \to u, f(v, v) \to v, f(u, u) \to u, f(u, u) \to v\})$. Then $\textsc{reach}_{\mathcal{A}}(a) = \{u\}$, $\textsc{reach}_{\mathcal{A}}(f(a, a)) = \{u, v\}$, $\textsc{reach}_{\mathcal{A}}(f(f(a, a), a)) = \{u, v\}$.

Definition 4 (Acceptance). *Given a FTA $\mathcal{A} = (Q, \mathcal{R}, Q_f, \delta)$, a term t, we say that t is* accepted *by the automaton if* $\textsc{reach}_{\mathcal{A}}(t) \cap Q_f \neq \emptyset$. *$\mathcal{L}(\mathcal{A})$ denotes the set of terms accepted by automaton $\mathcal{A}$.*

Example 4. With the definition of Ex. 3, $\mathcal{L}(\mathcal{A})$ is the set of terms over $\mathcal{R}$ that contain at least one f.

Definition 5 (Tree regular languages). *A set of terms $\mathcal{T}$ over a ranked alphabet $\mathcal{R}$ is called* tree regular *if there exists a FTA $\mathcal{A}$ over $\mathcal{R}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{T}$. The set of such languages is denoted $TReg(\mathcal{R})$.*

Remark 2. As for regular languages, for all $\mathcal{A} \in \text{FTA}$ there exists $\mathcal{A}' \in \text{CDFTA}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$, moreover $\mathcal{A}'$ is computable (see [4]).

Example 5.
- As proved in Ex. 4 the set of all terms over $\{f(2), a(0)\}$ that contain at least one f is tree regular.
- Consider now the ranked alphabet $\{a(1), b(1), \epsilon(0)\}$ and the set of terms $\mathcal{T} = \{\epsilon, a(b(\epsilon)), a(a(b(b(\epsilon)))), \dots\}$. We can prove (in a similar way as for $a^n b^n$ in regular languages) that $\mathcal{T}$ is not tree regular.
- On every ranked alphabet $\mathcal{R}$: every finite language, the empty language and $T(\mathcal{R})$ are tree regular.

Proposition 1. *$(TReg(\mathcal{R}), \subseteq, \cap, \cup, .^c, \emptyset, T(\mathcal{R}))$ is a complemented lattice with infinite height, moreover it is not complete. $\subseteq, \cap, \cup$ and complementation $(.^c)$ are computable operations on tree automata [4].*

We denote by $\mathcal{R}^\square$ the ranked alphabet $\mathcal{R}$ after adding the symbol $\square$ of arity 0 (we assume that $\square \notin \mathcal{R}$). Given a natural term t, we define $t^\square$ to be the term obtained by replacing every integer with the $\square$ symbol.

Proposition 2. *$(\wp(T_{\mathbb{Z}}(\mathcal{R})), \subseteq) \xleftarrow{\gamma} (TReg(\mathcal{R}^\square), \subseteq)$ where $\gamma(\mathcal{A}) = \{t \mid t^\square \in \mathcal{L}(\mathcal{A})\}$ is a representation. Moreover with such a γ definition, $\cup$, $\cap$ soundly represent the union and the intersection.*

Remark 3. We only have a representation and not a Galois connection as language $\mathcal{T}$ of Ex. 5 does not have a best tree regular over approximation.

Example 6. Let $\mathcal{R} = \{+(2)\}$ and $\mathcal{A} = (\{0,1\}, \mathcal{R}^\square, \{0,1\}, \{(\square() \to 0, +(0,0) \to 1, +(0,1) \to 1)\})$. Examples of terms recognized by $\mathcal{A}$ are shown on Fig. 3. Natural terms from our running example U and V (defined in Ex. 2) are also contained in $\gamma(\mathcal{A})$. Moreover as we do not provide numerical constraints: $1 + (3 + 4)$, 23, $1 + (2 + (3 + 4))$ are also elements in $\gamma(\mathcal{A})$.

Due to the infinite height of the lattice, a widening operator is required. In the following, we assume given a constant $w \in \mathbb{N}$, this constant will be used to stabilize increasing chains, the greater the constant, the more precise our widening operator will be.

Definition 6. *Let $\mathcal{A} = (Q, \mathcal{R}, Q_f, \delta) \in \text{FTA}$, and $\sim$ be an equivalence relation on Q, such that $p \sim q \wedge p \in Q_f \Rightarrow q \in Q_f$. We define $\mathcal{A}/\sim = (Q/\sim, \mathcal{R}, Q_f/\sim, \bigcup_{(f,q_1,\dots,q_n,q)\in\delta}\{(f, q_1^\sim, \dots, q_n^\sim, q^\sim)\})$ where $q^\sim$ is the equivalence class of q in $\sim$.*

Proposition 3. *For every $\mathcal{A} \in \text{FTA}$ and every $\sim$ equivalence relation on its states, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}/\sim)$.*

Therefore following the idea from [9] and in [11], we define a widening operation by quotienting states of automata by an equivalence relation of finite index. We define by induction a special sequence of equivalence relations on states of tree automata: $\sim_1 = \{Q_f, Q \setminus Q_f\}$ and $\sim_{k+1}$ is $\sim_k$ where we split equivalence classes not satisfying the following condition: $\forall f \in \mathcal{F}_n,\ \forall p_1, \dots, p_n \in$

$Q,\ \forall q_1,\ldots,q_n \in Q, (\bigwedge_{i=1}^{n} p_i \sim_k q_i) \Rightarrow \delta(f,p_1,\ldots,p_n) \sim_k \delta(f,q_1,\ldots,q_n)$ and $\forall q \in Q_f,\ q^{\sim_k} \subseteq Q_f$. This sequence of equivalence relations is the Myhill-Nerode sequence (see [4]). This sequence is of length at most the number of states of the automaton (before stabilization). Let $\phi(w) = \max\{i \leq |Q| \mid \text{index of } \sim_i \leq w\}$ (given an integer w, ϕ yields the index of the most precise of the equivalence relationships in the Myhill-Nerode sequence, that contains at most w equivalence classes) and $[\mathcal{A}]_w = \mathcal{A}/\sim_{\phi(w)}$. $[\mathcal{A}]_w$ is therefore a FTA with at most w states such that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}([\mathcal{A}]_w)$. As for regular languages, for every CDFTA a equivalent minimal CDFTA (in the sense of the number of states, and unique modulo state renaming) can be obtained by quotienting the automaton by $\sim_{|Q|}$. Therefore we define a widening operator on CDFTAs, which is then lifted to tree regular languages.

Definition 7 (Widening operator $\triangledown$). $\mathcal{A} \triangledown \mathcal{A}' = [\mathcal{A} \cup \mathcal{A}']_w$.

Proposition 4. *This widening is sound and stabilizes infinite sequences.*

Remark 4. Consider the two following complete and deterministic tree automata: $\mathcal{A} = (\{a,b,h\}, \{+(2)\}, \{a\}, \{\Box() \to b, +(b,b) \to a\})$ and $\mathcal{B} = (\{a,b,c,h\}, \{+(2)\}, \{a\}, \{\Box() \to b, +(b,b) \to c, +(b,c) \to a\})$ (unmentioned transitions go to h). $\mathcal{A}$ (resp. $\mathcal{B}$) recognizes the tree $+(\Box,\Box)$ (resp. $+(\Box,+(\Box,\Box))$), it over-approximates U (resp. V) from our running example. $\mathcal{A} \cup \mathcal{B}$ is recognized by the following complete and deterministic tree automaton: $\mathcal{C} = (\{a,b,c,h\}, \{+(2)\}, \{a,c\}, \{\Box() \to b, +(b,b) \to c, +(b,c) \to a\})$. If we want to widen $\mathcal{A}$ and $\mathcal{B}$ with parameter 3, the following equivalence relation is computed: $\{\{h\},\{b\},\{a,c\}\}$. Merging equivalent states produces $(\{a,b,h\}, \{+(2)\}, \{a\}, \{\Box() \to b, +(b,b) \to a, +(b,a) \to a\})$, which contains a loop and over-approximates the union.

3.2 Environment abstraction

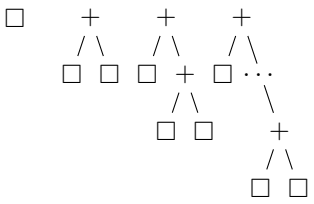

Fig. 3: Example of accepted trees from Ex. 6

Now that we are given an abstraction for natural term sets, let us show how this is lifted to a notion of abstract natural term environments mapping variables to natural terms. Given a set of natural term variables $\mathcal{T}$, consider $\mathfrak{F}^\sharp = (\mathcal{T} \to \mathrm{TReg}(\mathcal{R}^\Box)) \cup \{\bot\}$ and the set operators defined by the point-wise lifting of operators on $\mathrm{TReg}(\mathcal{R}^\Box)$. We also lift the concretization function $\wp(T_\mathbb{Z}(\mathcal{R})) \leftarrow \mathrm{TReg}(\mathcal{R}^\Box)$ to $\mathfrak{F} \leftarrow \mathfrak{F}^\sharp$. We assume given an abstract numerical environment $E^\sharp$ and an abstract evaluator $\mathbb{E}[\![e]\!]^\sharp$. Abstract transformers $[\![\texttt{make\_symbolic}]\!]^\sharp$, $[\![\texttt{is\_symbol}]\!]^\sharp$, $[\![\texttt{get\_son}(e)]\!]^\sharp$, $[\![\texttt{get\_sym\_head}]\!]^\sharp$ and $[\![\texttt{get\_num\_head}]\!]^\sharp$ are simple tree automata operations. For concision Fig. 4 only provides definitions of two of these operators. Please note that these definitions require all states of the automata to be reachable. An example of use of the `is_symbol` operator can be found in Ex. 7. Other abstract operators are similar.

Example 7. Consider the tree automaton $\mathcal{A}$ of Ex. 6, (Fig. 3), with $F^\sharp = (x \mapsto \mathcal{A})$: $[\![\texttt{get\_sym\_head}(x)]\!]^\sharp(E^\sharp, F^\sharp) = \{+\}$ and $[\![\texttt{get\_num\_head}(x)]\!]^\sharp(E^\sharp, F^\sharp) = \top$

$$\mathbb{E}^\sharp[\![\texttt{make\_integer}(e \in \mathit{expr})]\!](E^\sharp, F^\sharp) = \langle\{a\}, \mathcal{R}, \{a\}, \{\Box() \to a\}\rangle$$

$$\mathbb{E}^\sharp[\![\texttt{get\_son}(T, e \in \mathit{expr})]\!](E^\sharp, F^\sharp) =$$

$$\bigcup_{\substack{(Q,\mathcal{R},Q_f,\delta)\in\mathbb{E}^\sharp[\![T]\!](E^\sharp,F^\sharp)\\ i\in\mathbb{E}^\sharp[\![e]\!](E^\sharp)\cap\{0,\ldots,m-1\}}} (Q, \mathcal{R}, \{q \in Q \mid \exists p \in Q_f,\ \exists s(p_0, \ldots, p_{m-1}) \to p \in \delta \wedge p_i = q\}, \delta)$$

Fig. 4: Abstract operators

4 Numerical abstractions

As emphasized in the introductory example, we rely on numerical domains to introduce constraints on numerical variables found in trees. In a classical numeric abstraction (e.g. intervals [6], octagons [22], polyhedra [8], ...), each abstract element represents a set of maps $\mathcal{V} \to \mathbb{R}$ for a fixed, finite set of variables $\mathcal{V}$. In contrast, our numeric variables are leaves of a possibly infinite set of trees of unbounded size. Hence before starting the presentation of the numerical abstraction for natural terms, we show how to extend in a generic way an abstract element in two steps. Firstly we want to be able to represent a set of maps, where each map is defined over a (possibly different) finite subset of an infinite set of variables (this is done in Sec. 4.1). Secondly, we use summarization variables to relax the finiteness constraint, so as to represent sets of maps over heterogenous maps over infinitely many variables (done in Sec. 4.2).

4.1 Heterogeneous support

We define $\mathfrak{M} \triangleq \wp(\mathcal{V} \nrightarrow \mathbb{R})$, the set of partial maps from $\mathcal{V}$, to $\mathbb{R}$. $\mathfrak{M}$ is ordered by the inclusion relation $\subseteq$. In the following $\mathbf{def}(f)$ denotes the definition set of f. We assume defined a representation $(\wp(\mathcal{S} \to \mathbb{R}), \subseteq) \xleftarrow{\gamma_0^{\mathcal{S}}} (N_{\mathcal{S}}, \sqsubseteq_0^{\mathcal{S}})$, for every finite set $\mathcal{S} \subseteq \mathcal{V}$ (such as octagons in $|S|$ dimensions). $N_{\mathcal{S}}$ comes with the usual abstract set operator $\sqcap_0^{\mathcal{S}}$, $\sqcup_0^{\mathcal{S}}$. Moreover if $x \in \mathcal{S}$, $y \notin \mathcal{S}$, $\mathcal{S}'$ is another finite set and $N^\sharp \in N_{\mathcal{S}}$ then $N^\sharp[x \mapsto y] \in N_{\mathcal{S}\cup\{y\}\setminus\{x\}}$ is the abstract element obtained by renaming x into y, $N^\sharp_{|\mathcal{S}'} \in N_{\mathcal{S}'}$ is obtained by existentially quantifying dimensions associated to elements in $\mathcal{S}$ and not in $\mathcal{S}'$ and adding unconstrained dimensions for elements in $\mathcal{S}'$ and not in $\mathcal{S}$. From now on we assume that this last operator is exact (as for intervals, octagons, polyhedra over $\mathbb{R}$). However results from this section can be extended to numerical domains that are able, given $N^\sharp \in N_{\mathcal{S}}$, $N^{\sharp\prime} \in N_{\mathcal{S}'}$, to check if $\gamma_0^{\mathcal{S}}(N^\sharp) \subseteq \gamma_0^{\mathcal{S}'}(N^{\sharp\prime})_{|\mathcal{S}}$. The precision of the extension defined in this subsection would then depend upon the precision of this test in the underlying domain. Finally $[\![.]\!]_0^{\mathcal{S}}$ (resp. $[\![.]\!]_0^{\sharp,\mathcal{S}}$) refers to the classical concrete (resp. abstract) semantic of operators on sets of numerical maps (resp. abstract elements). A classical method for the abstraction of heterogeneous maps is the use of a partitioning of the concrete element according to the definition set of its represented maps. However partitioning induces an increase in numerical operation cost (exponential in the number of variable) which we would like to avoid.

Therefore in order to abstract sets of maps with heterogeneous definition sets, we start by abstracting the potential definition set. We choose a simple lower-bound/upper-bound abstraction (l and u in the following definition). Moreover we need to abstract the potential mappings given a definition set: this is done using a classical numerical domain. Contrary to partitioning, we will use only one numerical abstract element, defined on the upper-bound u, to represent all environments (instead of one abstract element by definition set). We also add a $\top$ element, used in the case where the upper bound u is infinite.

Definition 8 (Numerical abstraction). *Let us define the following set:* $\mathfrak{M}^\sharp \triangleq \{\langle N^\sharp, l, u\rangle \mid l, u \in \wp(V) \wedge l$ *and* u *are finite* $\wedge l \subseteq u \wedge N^\sharp \in N_u \wedge N^\sharp \neq \bot_0^u\} \cup \{\top, \bot\}$. *An element of* $\mathfrak{M}^\sharp$ *is therefore: either* $\top$, $\bot$ *or a triple* $\langle N^\sharp, l, u\rangle$ *where* l *and* u *are finite sets of variables such that* $N^\sharp$ *is defined over* u.

Definition 9 (Concretization function). *Abstract elements from* $\mathfrak{M}^\sharp$ *are mapped to* $\mathfrak{M}$ *thanks to the following concretization function:* $\gamma(\bot) = \emptyset$, $\gamma(\top) = \mathfrak{M}$ *and* $\gamma(\langle N^\sharp, l, u\rangle) = \{\rho \in \mathcal{S} \to \mathbb{Z} \mid l \subseteq \mathcal{S} \subseteq u \wedge \rho \in \gamma_0^{\mathcal{S}}(N^\sharp)_{|\mathcal{S}})\}$

Example 8. As an example consider $\gamma(\langle\{x = y, x \leq 3, z = 0\}, \{x\}, \{x, y, z\}\rangle) = \{(x \mapsto a) \mid a \leq 3\} \cup \{(x \mapsto a, y \mapsto a) \mid a \leq 3\} \cup \{(x \mapsto a, z \mapsto 0) \mid a \leq 3\} \cup \{(x \mapsto a, y \mapsto a, z \mapsto 0) \mid a \leq 3\}$. As intended, the resulting set of maps contains maps with different definition sets.

Definition 10 (Order). *On* $\mathfrak{M}^\sharp$ *we define the following comparison operator:* $\langle N^\sharp, l, u\rangle \sqsubseteq \langle N^{\sharp\prime}, l', u'\rangle \Leftrightarrow l' \subseteq l \subseteq u \subseteq u' \wedge N^\sharp \sqsubseteq_0^u N^{\sharp\prime}_{|u}$, *this comparison is trivially extended to* $\top$ *(resp.* $\bot$*) as being the biggest (resp. smallest) element in* $\mathfrak{M}^\sharp$. *In the following* $\mathfrak{M}^\sharp_{\mathfrak{p}}$ *denotes the subset of* $\mathfrak{M}^\sharp$ *where* $u = \mathfrak{p}$ *extended with* $\top$ *and* $\bot$.

Proposition 5. γ *is monotonic for* $\sqsubseteq$.

Fig. 5 provides the definition of the concrete and abstract semantics of the classical numerical statements, `Assume` and `Assign` (denoted $x \leftarrow e$). We denote $\mathbf{vars}(e)$ the set of variables appearing in e. We recall that $[\![\texttt{Assume}(c)]\!]_0^{\mathcal{S}}(E \in \wp(\mathcal{S} \to \mathbb{R})) = \{f \in E \mid \texttt{true} \in \mathbb{E}[\![c]\!](f)\}$ and $[\![x \leftarrow e]\!]_0^{\mathcal{S}}(E \in \wp(\mathcal{S} \to \mathbb{R})) = \{f[x \mapsto e'] \mid f \in E \wedge e' \in \mathbb{E}[\![e]\!](f)\}$. In order to ease the lifting of these classical operators we define $[\![\texttt{stmt}]\!]_0(\mathcal{M} \in \mathfrak{M}) \triangleq \cup_{\mathcal{S} \text{ finite} \subseteq \mathcal{V}} [\![\texttt{stmt}]\!]_0^{\mathcal{S}}(\mathcal{M} \cap (\mathcal{S} \to \mathbb{R}))$, for every statement `stmt`. Moreover we assume the existence of the following abstract operators: $[\![\texttt{Assume}(c)]\!]_0^{\sharp,u}(N^\sharp)$ and $[\![x \leftarrow e]\!]_0^{\sharp,u} N^\sharp$ abstracting soundly their respective concrete transformers. Note that the concrete semantic of $\texttt{Assume}(c)$ (resp. $x \leftarrow e$) enforces that maps are defined at least on the variables appearing in c (resp. in e and on x). Abstract operators from Fig. 5 are sound with respect to γ and their concrete operators.

We now need to define $\sqcup$ that abstracts the classic set operator $\cup$. We can not directly apply the corresponding abstract operator on the numerical component of the abstractions as they might have different definition sets. A first naive solution would be to extend their respective definition set and to perform the abstract operation on the resulting elements: $N^\sharp_{|u \cup u'} \sqcup_0^{u \cup u'} N^{\sharp\prime}_{|u \cup u'}$. However consider

$$[\![\texttt{Assume}(c)]\!](\mathcal{M}) = [\![\texttt{Assume}(c)]\!]_0(\{f \mid f \in \mathcal{M} \wedge \mathbf{vars}(c) \subseteq \mathbf{def}(f)\})$$
$$[\![\texttt{Assume}(c)]\!]^\sharp(\langle N^\sharp, l, u\rangle) = \langle [\![\texttt{Assume}(c)]\!]_0^{\sharp,u}(N^\sharp), l \cup \mathbf{vars}(c), u\rangle$$
$$[\![x \leftarrow e]\!](\mathcal{M}) = [\![x \leftarrow e]\!]_0(\{f \mid f \in \mathcal{M} \wedge \mathbf{vars}(e) \cup \{x\} \subseteq \mathbf{def}(f)\})$$
$$[\![x \leftarrow e]\!]^\sharp(\langle N^\sharp, l, u\rangle) = \langle [\![x \leftarrow e]\!]_0^{\sharp,u}(N^\sharp), l \cup \mathbf{vars}(e) \cup \{x\}, u\rangle$$

Fig. 5: Concrete and abstract semantic of usual numerical operators

Algorithm 1: strengthening operator

Input : $X^\sharp$, C: a set of constraints, $U^\sharp \in N_u$: a soundness threshold on environment u, $V^\sharp \in N_v$: a soundness threshold on environment v
Output: $Z^\sharp$ an abstract element over-approximating $U^\sharp$ on u and $V^\sharp$ on v

```
Z♯ ← X♯;
foreach c ∈ C do
    T♯ ← [[Assume(c)]]_0^{♯,u∪v}(Z♯);
    if U♯ ⊑_0^u T♯_|u ∧ V♯ ⊑_0^v T♯_|v then
        Z♯ ← T♯;
    end
return Z♯;
```

$M = \langle\{x = y\}(= U^\sharp), \{x, y\}, \{x, y\}\rangle$ and $N = \langle\{x = z\}(= V^\sharp), \{x, z\}, \{x, z\}\rangle$, where the underlying domain is the octagon domain where elements are represented as a set of linear constraints (e.g. $\{x = y\}$). We have $U^\sharp_{|\{x,y,z\}} = \{x = y\}$ and $V^\sharp_{|\{x,y,z\}} = \{x = z\}$, hence $U^\sharp_{|\{x,y,z\}} \sqcup_0^{\{x,y,z\}} V^\sharp_{|\{x,y,z\}} = \top$. Consider now the abstract element in $\mathfrak{M}^\sharp$: $R = \langle\{x = y, x = z\}(= W^\sharp), \{x\}, \{x, y, z\}\rangle$. The concretization of R over-approximates the union of the concretization of M and N, and its numerical component is more precise than $\top$. We note that the numerical constraints appearing in $W^\sharp$ could be found in $U^\sharp$ or $V^\sharp$, therefore in order to remove the aforementioned imprecision we define a refined abstract union operator, denoted as $\bowtie$, that uses constraints found in the inputs in order to refine its result. This is done using the **strenghtening** operator of Algo. 1 which adds constraints from C that do not make the projection of $X^\sharp$ to u (resp. v) lower than the threshold $U^\sharp$ (resp. $V^\sharp$). We assume that, given an abstract element $U^\sharp$, we can extract a finite set of constraints satisfied by $U^\sharp$, those are denoted $\mathbf{constraints}(U^\sharp)$ (the more constraints can be extracted, the more precise the result will be). For example if the numerical domain is the interval domain, constraints have the form $\pm x \geq a$. If the numerical domain is the octagon domain the **constraints** operator yields all the linear relations among variables that define the octagon.

Definition 11 ($\bowtie$ operator). *Let $U^\sharp \in N_u$, $V^\sharp \in N_v$ be two numerical environments, let $X^\sharp \in N_{u\cup v}$, let C be a sequence of numerical constraints over*

$u \cup v$, let $\mathfrak{c} = u \cap v$ we define:

$$U^\sharp \boxplus V^\sharp = \textbf{let } X^\sharp = (U^\sharp_{|\mathfrak{c}} \sqcup_0^{\mathfrak{c}} V^\sharp_{|\mathfrak{c}})_{|u\cup v} \textbf{ in}$$
$$\textbf{let } C = \textbf{constraints}(U^\sharp) \cup \textbf{constraints}(V^\sharp) \textbf{ in}$$
$$\textbf{strengthening}(X^\sharp, C, U^\sharp, V^\sharp)$$

Remark 5. – The precision of $\boxplus$ depends upon the order of iteration over constraints $c \in C$ in Algo. 1. Our implementation currently iterates in the order in which constraints are returned from the abstract domains. More clever heuristics will be considered in future work.
– $U^\sharp \boxplus V^\sharp$ starts by performing the join over the domain $\mathfrak{c}$, the result is then strengthened. Other **strenghtening**$(X^\sharp, U^\sharp \in N_u, V^\sharp \in N_v)$ operator could be defined, however in order to ensure soundness of $\boxplus$, it must satisfy the following constraints: $U^\sharp \sqsubseteq_0^u$ **strenghtening**$(X^\sharp, U^\sharp, V^\sharp)$ and $V^\sharp \sqsubseteq_0^v$ **strenghtening**$(X^\sharp, U^\sharp, V^\sharp)$.

Example 9. Let us now consider the example introduced thereinbefore $U^\sharp \boxplus V^\sharp = \{x = y, y = z\} \in N_{\{x,y,z\}}$. Indeed using the notations of Def. 11: $Z^\sharp \triangleq X^\sharp = \top \in N_{\{x,y,z\}}$, $C = \{x = y, y = z\}$, moreover $[\![\texttt{Assume}(x = y)]\!]_0^{\sharp,u\cup v}(\top) = \{x = y\} (\triangleq T^\sharp)$, $U^\sharp \sqsubseteq_0^{\{x,y\}} \{x = y\} = T^\sharp_{|\{x,y\}}$ and $V^\sharp \sqsubseteq_0^{\{x,z\}} \top = T^\sharp_{|\{x,z\}}$. Therefore constraint $x = y$ is added to $Z^\sharp$. At the next loop iteration: $[\![\texttt{Assume}(x = z)]\!]_0^{\sharp,u\cup v}(\{x = y\}) = \{x = y, x = z\} (\triangleq T^\sharp)$, $U^\sharp \sqsubseteq_0^{\{x,y\}} \{x = y\} = T^\sharp_{|\{x,y\}}$ and $V^\sharp \sqsubseteq_0^{\{x,z\}} \{x = z\} = T^\sharp_{|\{x,z\}}$. Therefore constraint $x = z$ is added to $Z^\sharp$.

Proposition 6 (Soundness of $\boxplus$). *let $U^\sharp \in N_u$ and $V^\sharp \in N_v$, then $\gamma_0^u(U^\sharp) \subseteq (\gamma_0^{u\cup v}(U^\sharp \boxplus V^\sharp))_{|u}$ and $\gamma_0^v(V^\sharp) \subseteq (\gamma_0^{u\cup v}(U^\sharp \boxplus V^\sharp))_{|v}$*

Definition 12 (Union abstract operators). *We define the following abstract set operator: $\langle N^\sharp, l, u\rangle \sqcup \langle N^{\sharp\prime}, l', u'\rangle \triangleq \langle N^\sharp \boxplus N^{\sharp\prime}, l \cap l', u \cup u'\rangle$. This operator soundly abstracts the union. Moreover in order to ensure the stabilization of infinitely increasing chains in $\mathfrak{M}^\sharp$ we define the following widening operator:*

$$\langle N^\sharp, l, u\rangle \triangledown \langle N^{\sharp\prime}, l', u'\rangle = \begin{cases} \langle N^\sharp \triangledown_0^u N^{\sharp\prime}_{|u}, l, u\rangle & \text{when } l \subseteq l' \wedge u' \subseteq u \\ \langle N^\sharp \boxplus N^{\sharp\prime}, l', u\rangle & \text{when } l' \subset l \wedge u' \subseteq u \\ \top & \text{otherwise} \end{cases}$$

Remark 6. This widening operator over-approximates to $\top$ whenever the upper-bound on the definition set is growing. This yields a huge loss of information however this numerical domain is designed as a tool domain used by a higher level abstraction in charge of stabilizing the environment before applying the widening, so that this case will not be used in practice.

Subsequent tree abstractions require the definition of the following operators:
– $\langle N^\sharp, l, u\rangle_{|-x} \triangleq \langle N^\sharp_{|u\setminus\{x\}}, l \setminus \{x\}, u \setminus \{x\}\rangle$ and $\langle N^\sharp, l, u\rangle_{|+x} \triangleq \langle N^\sharp_{|u\cup\{x\}}, l \cup \{x\}, u \cup \{x\}\rangle$ which respectively removes (adds) a variable to the numerical environment.
– $\langle N^\sharp, l, u\rangle_{|\mathcal{S}}$ is computed by adding variables in $\mathcal{S}$ and not in u and removing variables in u that are not in $\mathcal{S}$.

4.2 Representation of maps over potentially unbounded sets

In this subsection we focus on the problem of defining abstract numerical environments on potentially infinite environments. A classical method we use here is variable summarization (see [13]). This is based on the folding of several concrete objects (a potentially infinite number) to an abstract element which summarizes all concrete objects. The folding is encoded in a function f mapping summarized variables to the set of concrete variables they abstract. Given an abstract numerical environment $N^\sharp$ and a mapping from summary variables: $\mathcal{V}'$ to sets of concrete variables $f \in \mathcal{V}' \to \wp(\mathcal{V})$ where $f(v_1) \cap f(v_2) \neq \emptyset \Rightarrow v_1 = v_2$, we define the collapsing of a partial map $\rho \in \mathcal{V} \rightharpoonup \mathbb{Z}$ under a summarizing function f:

$$\downarrow_f (\rho) = \{\rho' \in \mathcal{V}' \rightharpoonup \mathbb{Z} \mid \forall v' \in \mathcal{V}',\ (f(v') \cap \mathbf{def}(\rho) = \emptyset \wedge \rho'(v') = \mathbf{undefined}) \\ \vee (\exists v \in \mathcal{V},\ v \in f(v') \cap \mathbf{def}(\rho) \wedge \rho'(v') = \rho(v))\}$$

Example 10. Consider $\mathcal{V}' = \{x, y, z, t\}$ and $\mathcal{V} = \{a, b, c, d, g, h\}$, the environment $\rho = (a \mapsto 0, b \mapsto 1, c \mapsto 2, d \mapsto 3)$ and finally the summarizing function $f = (x \mapsto \{a\}, y \mapsto \{b, c\}, z \mapsto \{d\}, t \mapsto \{g\})$. Collapsing environment ρ under f yields the set of environments: $(x \mapsto 0, y \mapsto 1, z \mapsto 3)$ and $(x \mapsto 0, y \mapsto 2, z \mapsto 3)$.

Given a summarizing function f we can now define an extension of the concretization function γ of the previous subsection in the following manner:

$$\gamma[f](N^\sharp) = \{\rho \in \mathcal{V} \rightharpoonup \mathbb{Z} \mid \downarrow_f (\rho) \subseteq \gamma(N^\sharp)\}$$

Example 11. Going back to Ex. 10 and considering the numerical abstract element: $N^\sharp = \langle\{x \leq y\}, \{x\}, \{x, y\}\rangle$, we have: $\gamma(N^\sharp) = \{(x \mapsto \alpha) \mid \alpha \in \mathbb{Z}\} \cup \{(x \mapsto \alpha, y \mapsto \beta) \mid \alpha \leq \beta\}$. We have: $m \in \gamma[f](N^\sharp) \Leftrightarrow \downarrow_f (m) \subseteq \gamma(N^\sharp) \Rightarrow \{x\} \subseteq \mathbf{def}(\downarrow_f (m)) \subseteq \{x, y\}$. Therefore if we assume m defined on d then $f(z) \cap \mathbf{def}(m) \neq \emptyset$ hence there would be an element in $\downarrow_f (m)$ defined on z. Hence m is not defined on d, similarly for g. Moreover $\{x\} \subseteq \mathbf{def}(\downarrow_f (m))$ implies that m is defined on a. Finally: defining $S = \{(a \mapsto \alpha) \mid \alpha \in \mathbb{Z}\} \cup \{(a \mapsto \alpha, b \mapsto \beta) \mid \alpha \leq \beta\} \cup \{(a \mapsto \alpha, c \mapsto \beta) \mid \alpha \leq \beta\} \cup \{(a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma) \mid \alpha \leq \beta \wedge \alpha \leq \gamma\}$. We have: $\gamma[f](N^\sharp) = S \cup (\bigcup_{f \in S} \{f \uplus (h \mapsto \delta) \mid \delta \in \mathbb{Z}\})$.

The abstract domains we will define in the following sections will employ this summarization framework. The manipulation of summarized variables requires the definition of a $\mathbf{fold}(E, x, \mathcal{S})$ (resp. $\mathbf{expand}(E, x, \mathcal{S})$) operator yielding a new environment where x is used as a summary variable for $\mathcal{S}$ (resp. where a summary variable x is desummarized into a set of variables $\mathcal{S}$). Let $\mathcal{S}$ and $\mathcal{S}'$ be two finite sets of elements such that $\mathcal{S}' \cap \mathcal{S} \subseteq \{x\}$, we define: $\mathbf{expand}_0(N^\sharp, x, \mathcal{S}'') = \bigsqcap_{v \in \mathcal{S}''} N^\sharp[x \mapsto v]_{|(\mathcal{S} \setminus \{x\}) \cup \mathcal{S}''}$ and $\mathbf{fold}_0(N^\sharp, x, \mathcal{S}'') = \bigsqcup_{v \in \mathcal{S}''} N^\sharp[v \mapsto x]_{|(\mathcal{S} \setminus \mathcal{S}'') \cup \{x\}}$ (which generalize the one introduced in [13]). These operations are lifted as operators on elements of $\mathfrak{M}^\sharp$:

$$\mathbf{expand}(\langle N^\sharp, l, u\rangle, x, \mathcal{S}) \triangleq \langle \mathbf{expand}_0(N^\sharp, x, \mathcal{S}), l \setminus \{x\}, (u \setminus \{x\}) \cup \mathcal{S}\rangle$$

$$\mathbf{fold}(\langle N^\sharp, l, u\rangle, x, \mathcal{S}) \triangleq \langle \mathbf{fold}_0(N^\sharp, x, \mathcal{S}), \begin{cases} (l \setminus \mathcal{S}) \cup \{x\} & \text{if } \mathcal{S} \subseteq l \\ (l \setminus \mathcal{S}) & \text{otherwise} \end{cases}, (u \setminus \mathcal{S}) \cup \{x\}\rangle$$

5 Natural term abstraction by numerical constraints

We are now able to represent sets of maps with heterogeneous supports and to lift their concretization (modulo a summarization function) to sets of maps with infinite and heterogeneous supports. Given a tree shape (in the sense of Sec. 3), we can associate a numeric variable to each numeric leaf, and use a numeric abstract element to represent the possible values of these leaves. We will name the variable of each leaf as the path from the root to the leaf, i.e., $\mathcal{V}$ is a set of words in $\{0, ..., n-1\}$ where n is the maximum arity of the considered ranked alphabet. In order to avoid confusion such paths will be denoted $\wr 0, 1, 1 \wr$ for the word $(0, 1, 1)$. A summarized variable then represents a set of such paths. We will abstract such sets as regular expressions. Using the summarization extended to heterogeneous supports presented in the previous section, it will be possible to represent, using a single numeric abstract element, a set of contraints over the numeric leaves of an infinite set of unbounded trees of arbitrary shape.

5.1 Hole positions and numerical constraints

The presentation of our computable abstraction able to represent numerical values in trees is broken down (for presentation purposes) into two consecutive abstractions. The first one is not computable, as natural terms are abstracted as partial environments over tree paths to numerical values. This abstraction looses most of the tree shapes but focuses on their numerical environment. A second abstraction will show how partial environments over paths are abstracted into numerical abstract elements defined over a regular expression environment.

In the following, when $\mathcal{R}$ is a ranked alphabet of maximum arity n, we call *words* sequences of integers, $w = (w_0, \ldots, w_{p-1}) \in \{0, \ldots, (n-1)\}^p$ will be called a word of length p (denoted $|w|$), w_i denotes the i-th integer of the sequence, $\overline{w} = (w_1, \ldots, w_{p-1})$ is the tail of word w, $\mathcal{W}(\mathcal{R}) = \{0, \ldots, (n-1)\}^\star$ is the set of all words over $\{0, \ldots, n-1\}$ of arbitrary size.

Definition 13 (Position in a term). *Given a natural term t and a word w we inductively define the subterm of t at position w (denoted $t_{|w}$) to be:*

$$t_{|w} = \begin{cases} (t_{w_0})_{|\overline{w}} & \text{when } |w| > 0 \wedge t = f(t_0, \ldots, t_{p-1}) \text{ with } w_0 < p \\ t & \text{when } |w| = 0 \\ \textbf{undefined} & \text{otherwise} \end{cases}$$

Moreover we denote by $\textbf{numeric}(t) = \{w \in \mathbb{N}^\star \mid t_{|w} \in \mathbb{Z}\}$.

Definition 14 (Positioning lattice with exact numerical constraints). *We define $\mathcal{C}(\mathcal{R}) \stackrel{\Delta}{=} \wp(\mathcal{W}(\mathcal{R}) \nrightarrow \mathbb{Z})$, an element of $\mathcal{C}(\mathcal{R})$ is therefore a set of partial maps that are acceptable bindings of positions to integers.*

Proposition 7 (Galois connection with natural terms). *When t is a natural term, $t_\mathbb{Z}$ is the partial map: $\lambda_{|\textbf{numeric}(t)} w.t_w$. We have the following Galois connection: $(\wp(T_\mathbb{Z}(\mathcal{R})), \subseteq) \underset{\alpha_{\mathcal{C}(\mathcal{R})}}{\overset{\gamma_{\mathcal{C}(\mathcal{R})}}{\leftrightarrows}} (\mathcal{C}(\mathcal{R}), \subseteq)$, with:*

$$\gamma_{\mathcal{C}(\mathcal{R})}(\Gamma) = \{t \in T_\mathbb{Z}(\mathcal{R}) \mid t_\mathbb{Z} \in \Gamma\} \quad \alpha_{\mathcal{C}(\mathcal{R})}(\mathcal{T}) = \{t_\mathbb{Z} \mid t \in \mathcal{T}\}$$

Example 12. Consider our running example (introduced in Ex. 2), $V = \{+(x, +(z, y)) \mid x \leq y \wedge z \leq y\}$, we have $\alpha_{\mathcal{C}(\mathcal{R})}(V) = \{\wr 0 \wr \mapsto \alpha, \wr 1, 0 \wr \mapsto \gamma, \wr 1, 1 \wr \mapsto \beta \mid \alpha \leq \beta \wedge \gamma \leq \beta\}$. The concretization of which is exactly V.

Example 13. Consider however the ranked alphabet $\{f(2), g(2), a(0)\}$, and the tree a. Its abstraction contains only the empty map, the concretization of which is the set of all terms that do not contain any numerical value. For example: $f(g(a, a), a), g(a, a), \ldots$. This emphasizes that we loose information on:

- the labels in the natural terms: we only have the path from the root of the term to leaves with numerical labels, not the actual symbols along the path.
- the shape of the natural terms: we do not keep any information on subterms that do not contain numerical values.

Now that we have abstracted away the shape of the terms, we are left with numerical environments with potentially infinite dimensions (that are words over the alphabet $\{0, \ldots, n-1\}$) and different definition sets. Therefore following the idea of Sec. 4 we want to define a summarization for sets of words over the alphabet $\{0, \ldots, n-1\}$. A summarization of such a language can be expressed as a partition into sub-languages. The set of regular languages over the alphabet $\{0, \ldots, n-1\}$ is a subset of the set of languages over this alphabet, that is closed under common set operations. Hence given a set $\{r_1, \ldots, r_m\}$ of regular expressions (with respective recognized language $\{L_1, \ldots, L_m\}$), we summarize all words in L_i inside a common variable r_i and therefore $\uparrow \{r_1, \ldots, r_m\}$ denotes the summarization function: $\lambda r_i.L_i$. In the following, Reg_n denotes the set of regular expressions over the alphabet $A_n = \{0, \ldots, n-1\}$. As for tree regular expressions, $(\mathrm{Reg}_n, \subset, \cap, \cup, .^c, \emptyset, A_n^\star)$ is a (non complete) complemented lattice of infinite height, upon which we can define a widening operator $\triangledown$ (see [10]) in a similar manner as for tree regular expressions (this widening is also parameterized by an integer constant). We recall moreover that operators $\subset, \cap, \cup$ and complementation $(.^c)$ are computable, and that every finite set of words is regular. Moreover we have the following representation: $(A_n^\star, \sqsubseteq) \xleftarrow{\gamma_{\mathrm{Reg}_n} = Id} (\mathrm{Reg}_n, \sqsubseteq)$. Finally in order to disambiguate regular expressions from integers we will typeset them within $\lfloor . \rfloor$ in a bold font as in: $\lfloor \mathbf{0} + \mathbf{0}.\mathbf{1}^\star \rfloor$.

Example 14. Using notations from Sec. 4.2, $\mathcal{V}' = \mathrm{Reg}_n$ and $\mathcal{V} = \mathcal{W}(\mathcal{R})$. Consider our running example (introduced in Ex. 2), natural terms from $V = \{+(x, +(z, y)) \mid x \leq y \wedge z \leq y\}$ contain three paths to numerical values: $\wr 0 \wr$, $\wr 1, 0 \wr$ and $\wr 1, 1 \wr$. Numerical constraints on $\wr 0 \wr$ and $\wr 1, 0 \wr$ are similar, therefore the two paths are summarized into one regular expression: $\lfloor \mathbf{0} + \mathbf{1}.\mathbf{0} \rfloor$, $\wr 1, 1 \wr$ is left alone in its regular expression: $\lfloor \mathbf{1}.\mathbf{1} \rfloor$. The two constraints $x \leq y \wedge z \leq y$ can now be expressed as one: $\lfloor \mathbf{0} + \mathbf{1}.\mathbf{0} \rfloor \leq \lfloor \mathbf{1}.\mathbf{1} \rfloor$.

In Ex. 14, we saw that tree paths with similar numerical constraints can be summarized in one regular expression. However, for precision purposes, we do not want to summarize all tree paths into one regular expression. Hence, we will keep several disjoint regular expressions, which we call a subpartitioning.

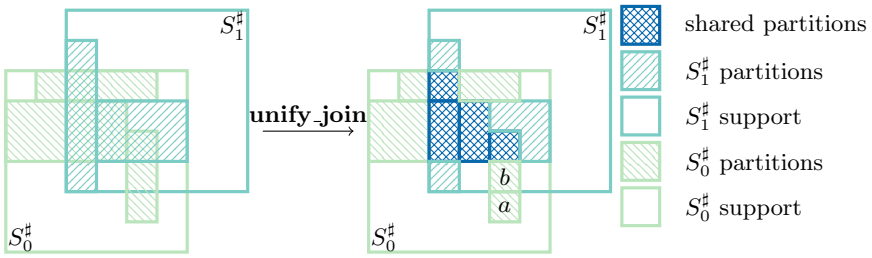

Fig. 6: Unification operator

Definition 15 (Subpartitioning). *Given a regular expression s, a subpartitioning of s is a set $\{s_1, \ldots, s_n\}$ of regular expressions such that $\forall i \neq j,\ s_i \cap s_j = \emptyset$ and $\bigcup_{i=1}^{n} s_i \subseteq s$. We note $P(s)$ the set of all subpartitioning of s. Moreover if $S = \{s_1, \ldots, s_n\}$ is a set of regular expressions, $[S]_\emptyset = S \setminus \{\emptyset\}$.*

Remark 7. Contrary to a partitioning of s, we do not require that the set of partitions covers s. Indeed when a set of tree paths is unconstrained we can just remove it from the partitioning, therefore no dimension in the numerical abstract environment will be allocated for this path.

Definition 16 (Positioning lattice with numerical abstraction). *Given a ranked alphabet $\mathcal{R}$, where the maximum arity of symbols is n, we define $\mathcal{C}^\sharp(\mathcal{R}) = \{\langle s, \mathfrak{p}, R^\sharp\rangle \mid s \in Reg_n, \mathfrak{p} \in P(s), R^\sharp \in \mathfrak{M}_\mathfrak{p}^\sharp\}$. Therefore $\mathcal{C}^\sharp(\mathcal{R})$ are triples containing:*

- *s: (called support) a regular expression coding for positions at which numerical values can be located.*
- *$\mathfrak{p}$: a subpartitioning of s. Elements of the same partition are subject to the same numerical constraints. Note that these partitions are regular.*
- *$R^\sharp$: an abstract numeric element where a dimension is associated to each partition, this dimension plays the role of a summary dimension.*

Remark 8. In the following, numerical abstract elements described in the form $\{c\}$, where c is a set of constraints, refer to $\langle c, \mathbf{vars}(c), \mathbf{vars}(c)\rangle \in \mathfrak{M}^\sharp$.

Unification. The previous definition shows that two elements $U^\sharp = \langle s, \mathfrak{p}, R^\sharp\rangle$ and $V^\sharp = \langle s', \mathfrak{p}', R^{\sharp\prime}\rangle$ can have different subpartitionings ($\mathfrak{p}$ and $\mathfrak{p}'$). However the partitions in $\mathfrak{p}$ and in $\mathfrak{p}'$ might overlap, thus giving constraints to similar tree paths. Therefore in order to define the classical operators: $\sqsubseteq, \sqcup$ and $\triangledown$, we need to unify the two abstract elements ($U^\sharp$ and $V^\sharp$) so that given a tree path and the partition in which it is contained in $U^\sharp$, it is contained in the same partition in $V^\sharp$. This will enable us to rely on abstract operators on the numerical domain. In order to perform unification, we rely on the **expand** and **fold** operators. Indeed consider our running example, $U^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1}\rfloor, \{\lfloor \mathbf{0}\rfloor, \lfloor \mathbf{1}\rfloor\}, \{\lfloor \mathbf{0}\rfloor \leq \lfloor \mathbf{1}\rfloor\}\rangle$ and $V^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1}.(\mathbf{0}+\mathbf{1})\rfloor, \{\lfloor \mathbf{0}+\mathbf{1.0}\rfloor, \lfloor \mathbf{1.1}\rfloor\}, \{\lfloor \mathbf{0}+\mathbf{1.0}\rfloor \leq \lfloor \mathbf{1.1}\rfloor\}\rangle$. We see that

Algorithm 2: unify_join operator

Input : $\langle s, \{p_1, \ldots, p_n\}, R^\sharp\rangle, \langle s', \{p'_1, \ldots, p'_m\}, R^{\sharp\prime}\rangle$ two abstract elements
Output: two unified abstract elements

1 $(\underline{c_{i,j}})_{i\leq n, j\leq m} \leftarrow p_i \cap p'_j$;
2 $(\underline{p_i})_{i\leq n} \leftarrow p_i \cap s'^c$;
3 $(\underline{p'_j})_{j\leq m} \leftarrow p'_j \cap s^c$;
4 $(\underline{q_i})_{i\leq n} \leftarrow p_i \cap s' \cap (\cup_{j\leq m}\underline{c_{i,j}})^c$;
5 $(\underline{q'_j})_{j\leq m} \leftarrow p'_j \cap s \cap (\cup_{i\leq n}\underline{c_{i,j}})^c$;
6 $\underline{R^\sharp} \leftarrow R^\sharp$;
7 $\underline{R^{\sharp\prime}} \leftarrow R^{\sharp\prime}$;
8 **for** $i = 1$ **to** n **do**
9 $\quad R^\sharp \leftarrow \textbf{expand}(R^\sharp, p_i, [\{c_{i,j}\}_{j\leq m} \cup \{\underline{p_i}\} \cup \{\underline{q_i}\}]_\emptyset)$;
10 **for** $j = 1$ **to** m **do**
11 $\quad R^{\sharp\prime} \leftarrow \textbf{expand}(R^{\sharp\prime}, p'_j, [\{c_{i,j}\}_{i\leq n} \cup \{\underline{p'_j}\} \cup \{\underline{q'_j}\}]_\emptyset)$;
12 **return** $\langle s, \bigcup_{i\leq n, j\leq m}[\{\underline{q_i}, \underline{p_i}, \underline{c_{i,j}}\}]_\emptyset, \underline{R^\sharp}\rangle, \langle s', \bigcup_{i\leq n, j\leq m}[\{\underline{q'_i}, \underline{p'_j}, \underline{c_{i,j}}\}]_\emptyset, \underline{R^{\sharp\prime}}\rangle$;

constraints on tree path $\wr 0 \wr$ is given: in $U^\sharp$ by partition $\lfloor \mathbf{0} \rfloor$ and in $V^\sharp$ by partition $\lfloor \mathbf{0} + \mathbf{1.0} \rfloor$. However we can split the partition $\lfloor \mathbf{0} + \mathbf{1.0} \rfloor$ into two partitions: $\lfloor \mathbf{0} \rfloor$ and $\lfloor \mathbf{1.0} \rfloor$, and expand variable $\lfloor \mathbf{0}+\mathbf{1.0} \rfloor$ into the two variables $\lfloor \mathbf{0} \rfloor$ and $\lfloor \mathbf{1.0} \rfloor$ in the numeric component: $\textbf{expand}(\{\lfloor \mathbf{0}+\mathbf{1.0} \rfloor \leq \lfloor \mathbf{1.1} \rfloor\}, \lfloor \mathbf{0}+\mathbf{1.0} \rfloor, \{\lfloor \mathbf{0} \rfloor, \lfloor \mathbf{1.0} \rfloor\}) = \{\lfloor \mathbf{0} \rfloor \leq \lfloor \mathbf{1.1} \rfloor, \lfloor \mathbf{1.0} \rfloor \leq \lfloor \mathbf{1.1} \rfloor\}$. Once $U^\sharp$ and $V^\sharp$ are unified we can rely on the numerical join to soundly abstract the union. Note that splitting partitions is more precise than merging them. Indeed, consider the example where: in $U^\sharp$ we have $\lfloor \mathbf{0} \rfloor \geq 0$ and $\lfloor \mathbf{1} \rfloor \leq 0$ and in $V^\sharp$ we have $\lfloor \mathbf{0} + \mathbf{1} \rfloor = 0$. Splitting partition in $V^\sharp$ yields: $\lfloor \mathbf{0} \rfloor = 0, \lfloor \mathbf{1} \rfloor = 0$, after joining we get $\lfloor \mathbf{0} \rfloor \geq 0, \lfloor \mathbf{1} \rfloor \leq 0$. Whereas merging partitions in $U^\sharp$ yields $\lfloor \mathbf{0} + \mathbf{1} \rfloor$ unconstrained, after joining we also get that $\lfloor \mathbf{0}+\mathbf{1} \rfloor$ is unconstrained. However unifying by splitting or merging partitions in both abstract elements might result in an over-approximation of the initial elements. This does not pose a threat to the soundness of the join operator, but it does for the inclusion test. Unifying by splitting partitions induces an increase in the number of partitions which we want to avoid when trying to stabilize abstract elements in the widening. Hence, we define three unification operators:

- An operator **unify_join** that splits partitions from $U^\sharp$ and $V^\sharp$, this operator might induce an over-approximation for both $U^\sharp$ and $V^\sharp$ and is used in the join operation. This operator is presented in Algo. 2, and illustrated in Fig. 6.
- An operator **unify_subset** that does not modify $V^\sharp$ (in order to avoid over-approximated it), we only split and merge (using the **fold** operator) partitions from $U^\sharp$ as, if the over-approximated $U^\sharp$ is smaller than $V^\sharp$, then so is the original $U^\sharp$.
- An operator **unify_widen** that unifies $U^\sharp$ and $V^\sharp$ by only merging partitions so that the number of partitions does not increase. This operator is used in the widening definition.

Operators **unify_subset** and **unify_widen** are very similar to **unify_join**.

Definition 17 (Comparison $\sqsubseteq_{\mathcal{C}^\sharp(\mathcal{R})}$). *Using* **unify_subset** *we define a relation on* $\mathcal{C}^\sharp(\mathcal{R})$*:* $\sqsubseteq_{\mathcal{C}^\sharp(\mathcal{R})} = \{(U^\sharp, V^\sharp) \mid (\langle s, \mathfrak{p}, N^\sharp\rangle, \langle s', \mathfrak{p}', N^{\sharp\prime}\rangle) = \mathbf{unify\_subset}(U^\sharp, V^\sharp) \Rightarrow s \subseteq s' \wedge \forall b \in \mathfrak{p}', (b \subseteq s^c \vee \exists! a \in \mathfrak{p},\, b \cap s = a) \wedge N^\sharp \sqsubseteq N^{\sharp\prime}[\phi]\}$ *where* ϕ *is the renaming from* $\mathfrak{p}'$ *into* $\mathfrak{p}$ *that renames* b *to* a *when such an* a *exists.*

Example 15. Going back to our running example: $U^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1}\rfloor, \{\lfloor \mathbf{0}\rfloor, \lfloor \mathbf{1}\rfloor\}, \{\lfloor \mathbf{0}\rfloor \leq \lfloor \mathbf{1}\rfloor\} (= A^\sharp)\rangle$ and $V^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1}.(\mathbf{0}+\mathbf{1})\rfloor, \{\lfloor \mathbf{0}+\mathbf{1}.\mathbf{0}\rfloor, \lfloor \mathbf{1}.\mathbf{1}\rfloor\}, \{\lfloor \mathbf{0}+\mathbf{1}.\mathbf{0}\rfloor \leq \lfloor \mathbf{1}.\mathbf{1}\rfloor\}\rangle$. We have $s \not\subseteq s'$ hence $U^\sharp \not\sqsubseteq V^\sharp$. However if we now consider $W^\sharp$: $\langle \lfloor (\epsilon+\mathbf{1}).(\mathbf{0}+\mathbf{1})\rfloor, \{\lfloor (\epsilon+\mathbf{1}).\mathbf{0}\rfloor, \lfloor (\epsilon+\mathbf{1}).\mathbf{1}\rfloor\}, \{\lfloor (\epsilon+\mathbf{1}).\mathbf{0}\rfloor \leq \lfloor (\epsilon+\mathbf{1}).\mathbf{1}\rfloor\} (= B^\sharp)\rangle$. $W^\sharp$ is already unified with $U^\sharp$, we have $s \subseteq s'$ and $\phi : (\lfloor (\epsilon+\mathbf{1}).\mathbf{0}\rfloor \mapsto \mathbf{0}, \lfloor (\epsilon+\mathbf{1}).\mathbf{1}\rfloor \mapsto \lfloor \mathbf{1}\rfloor)$. Moreover $A^\sharp \sqsubseteq B^\sharp[\phi] = \{\lfloor \mathbf{0}\rfloor \leq \lfloor \mathbf{1}\rfloor\}$. Hence $U^\sharp \sqsubseteq W^\sharp$.

Proposition 8. *We have:* $(\mathcal{C}(\mathcal{R}), \sqsubseteq_{\mathcal{C}(\mathcal{R})}) \xleftarrow{\gamma_1} (\mathcal{C}^\sharp(\mathcal{R}), \sqsubseteq_{\mathcal{C}^\sharp(\mathcal{R})})$*, where:* $\gamma_1(\langle s, \mathfrak{p}, R^\sharp\rangle) = \{f \mid \mathbf{def}(f) \subseteq \gamma_{Reg_n}(s) \wedge f \in \gamma[\uparrow \mathfrak{p}](R^\sharp)\}$*. By composition we get:* $(\wp(T_{\mathbb{Z}}(\mathcal{R})), \subseteq) \xleftarrow{\gamma_2} (\mathcal{C}^\sharp(\mathcal{R}), \sqsubseteq_{\mathcal{C}^\sharp\mathcal{R}})$*, with* $\gamma_2 = \gamma_{\mathcal{C}(\mathcal{R})} \circ \gamma_1$*.*

Example 16. Going back to our running example: $V^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1}.(\mathbf{0}+\mathbf{1})\rfloor, \{\lfloor \mathbf{0}+\mathbf{1}.\mathbf{0}\rfloor, \lfloor \mathbf{1}.\mathbf{1}\rfloor\}, \{\lfloor \mathbf{0}+\mathbf{1}.\mathbf{0}\rfloor \leq \lfloor \mathbf{1}.\mathbf{1}\rfloor\}\rangle$. We have: $\uparrow \mathfrak{p} = (\lfloor \mathbf{0}+\mathbf{1}.\mathbf{0}\rfloor \mapsto \{\wr 0\wr, \wr 1, 0\wr\}, \lfloor \mathbf{1}\rfloor \mapsto \wr 1\wr)$. Hence, $\gamma_1(V^\sharp) = \{(\wr 0\wr \mapsto \alpha, \wr 1\wr \mapsto \beta) \mid \alpha \leq \beta\} \cup \{(\wr 1, 0\wr \mapsto \alpha, \wr 1\wr \mapsto \beta) \mid \alpha \leq \beta\} \cup \{(\wr 0\wr \mapsto \alpha, \wr 1, 0\wr \mapsto \gamma, \wr 1\wr \mapsto \beta) \mid \alpha \leq \beta \wedge \gamma \leq \beta\}$. The product with tree automata refines this result so that only the last set is left.

We now define the $\sqcup$ operator that relies on the **unify_join** operator of Algo. 2. Once elements are unified we can distinguish three kinds of partitions: (1) Partitions found in both abstract elements (e.g.  in Fig. 6). (2) Partitions found in only one of the two, which do not overlap over the support of the other abstract element (denoted u^o), these are outer-partitions. Information on such partitions can be soundly kept when joining two abstract elements (e.g. partition a in Fig. 6). (3) Partitions found in only one of the two, which overlap over the support of the other abstract element, these are inner-partitions. Information on such partitions can not be soundly kept when joining two abstract elements. (e.g. partition b in Fig. 6). Therefore in the following definition of the join operator, we compute (once elements are unified) the common partitions and both outer-partitions and merge them to form the resulting subpartitioning.

Definition 18 (Union abstract operator). *Given* $U^\sharp, V^\sharp \in \mathcal{C}^\sharp(\mathcal{R})$*, if* $(\langle s, \mathfrak{p}, R^\sharp\rangle, \langle s', \mathfrak{p}', R^{\sharp\prime}\rangle) = \mathbf{unify\_join}(U^\sharp, V^\sharp)$*, let* $\mathfrak{c}$ *be* $\mathfrak{p} \cup \mathfrak{p}'$*, let* u^o *(*$U^\sharp$ *outer-partition) be* $\{e \in \mathfrak{p} \mid e \subseteq s'^c\}$*, let* v^o *(*$V^\sharp$ *outer-partition) be* $\{e \in \mathfrak{p}' \mid e \subseteq s^c\}$*, we then define:*

$$U^\sharp \sqcup_{\mathcal{C}^\sharp(\mathcal{R})} V^\sharp = \langle s \cup s', \mathfrak{c} \cup u^o \cup v^o, R^\sharp_{|\mathfrak{c} \cup u^o} \sqcup R^{\sharp\prime}_{|\mathfrak{c} \cup v^o}\rangle$$

Proposition 9. *We have:* $\gamma_1(U^\sharp) \cup \gamma_1(V^\sharp) \subseteq \gamma_1(U^\sharp \sqcup_{\mathcal{C}^\sharp(\mathcal{R})} V^\sharp)$*.*

Example 17. Consider the two following abstract elements (this is the particular case of our running example where all numerical values are equal): $V^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1}.(\mathbf{0}+\mathbf{1})\rfloor (= s), \{\lfloor \mathbf{0}+\mathbf{1}.\mathbf{0}\rfloor (= a), \lfloor \mathbf{1}.\mathbf{1}\rfloor (= b), \{a = b\}\}\rangle$, and $U^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1}\rfloor (=$

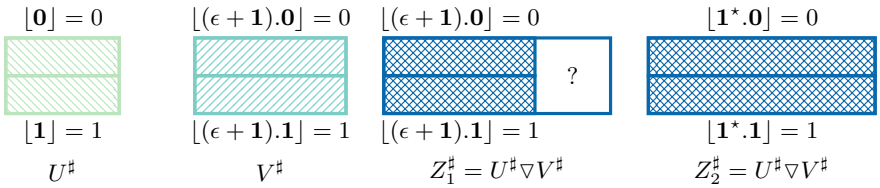

Fig. 7: Widening illustration

$s'), \{\lfloor \mathbf{0} \rfloor (= c), \lfloor \mathbf{1} \rfloor (= d)\}, \{c = d\}\rangle$. Intuitively $U^\sharp$ could encode the term $(x + x)$ and $V^\sharp$ the term $(x + (x + x))$. The unification of those two elements is: $V_1^\sharp = \langle s, \{c, b, \lfloor \mathbf{1}.\mathbf{0} \rfloor (= e)\}, R^\sharp\rangle$ where $R^\sharp = \langle \{c = b, e = b\}, \{b\}, \{c, b, e\}\rangle$ and $U_1^\sharp = U^\sharp$, moreover the common environment ($\mathfrak{c}$ in previous definition) is: $\{c\}$, $V^\sharp$ outer-partitioning is $\{e, f\}$, $U^\sharp$ outer-partitioning is $\{d\}$. Hence: the numerical component resulting of the join is: $\langle \{c = d\}, \{c, d\}, \{c, d\}\rangle \sqcup \langle \{c = b, e = b\}, \{b\}, \{c, b, e\}\rangle$ which is: $\langle \{c = b, e = b, c = d\}, \emptyset, \{c, d, e, b\}\rangle$. We see here that using a naive numerical join operator, we would not have been able to get such a precise result (the numerical join would have yielded $\top$).

unify_widen $\mathcal{C}^\sharp(\mathcal{R})$ contains infinite increasing chains, therefore, we need to provide a widening operator. As for the other operators, widening is computed on unified abstract elements. A **unify_widen** operator is defined: it produces $U^\sharp$ and $V^\sharp$, over-approximations of its inputs with the same number of partitions. Moreover it ensures that each partition of $U^\sharp$ intersects exactly one partition of $V^\sharp$. This can be obtained by iterative merging partitions that overlap in both arguments until the abstract elements have the exact same partitions. Therefore from the result of **unify_widen** we can extract a list of pairs (a, b) where a is a partition from $U^\sharp$, b is a partition from $V^\sharp$ and $a \cap b \neq \emptyset$. This defines a bijection from partitions of $U^\sharp$ onto partitions of $V^\sharp$.

compose. In order to ensure stabilization we first need to stabilize the supports on which abstract elements are defined. This is easily done using the automaton widening ($s_1 \triangledown s_2$ in Algo. 3). Fig. 7 illustrates the following simple example: $U^\sharp$ is an abstract element with support $\lfloor \mathbf{0} + \mathbf{1} \rfloor$, two partitions $u = \lfloor \mathbf{0} \rfloor$ and $u' = \lfloor \mathbf{1} \rfloor$, and numerical constraints $u' = 1$ and $u = 0$. $V^\sharp$ is an abstract element with support $\lfloor (\epsilon + \mathbf{1}).(\mathbf{0} + \mathbf{1}) \rfloor$, two partitions $v = \lfloor (\epsilon + \mathbf{1}).\mathbf{0} \rfloor$ and $v' = \lfloor (\epsilon + \mathbf{1}).\mathbf{1} \rfloor$ with the numerical constraints that $v = 0$ and $v' = 1$. Supports are unstable, therefore we start by widening them, which yields a new support: $\lfloor \mathbf{1}^\star.(\mathbf{0} + \mathbf{1}) \rfloor$. The unification of $U^\sharp$ and $V^\sharp$ leaves subpartitionings unchanged and yields the bijection $(u \mapsto v, u' \mapsto v')$. Given this information we now need to provide a new subpartitioning for the result of the widening. We see in this example that we could soundly use the subpartitioning from $V^\sharp$, this would produce the abstract element $Z_1^\sharp$ depicted in Fig. 7. However due to the widening of the support, paths of the form $\wr 1, 1, 1, 0 \wr$ are in the support of the result but are

Algorithm 3: widening operator

Input : $U^\sharp, V^\sharp$ two abstract elements

$(\langle s_1, \mathfrak{p}_1, R_1^\sharp\rangle, \langle s_2, \mathfrak{p}_2, R_2^\sharp\rangle) \leftarrow$ **unify_widen**$(U^\sharp, V^\sharp)$;
$s \leftarrow s_1 \triangledown s_2$;
$r \leftarrow s \setminus (s_1 \cup s_2)$;
foreach $a \in \mathfrak{p}_1$ **do**
  $b \leftarrow$ the unique element from $\mathfrak{p}_2$ such that $b \cap a \neq \emptyset$;
  $p \leftarrow$ **compose**(a, b, s_1, s_2, r);
  $\mathfrak{p} \leftarrow \{p\} \cup \mathfrak{p}$;
  $R_1^{\sharp\star} \leftarrow R_1^{\sharp\star}[a \mapsto p]$;
  $R_2^{\sharp\star} \leftarrow R_1^{\sharp\star}[b \mapsto p]$;
  $r \leftarrow r \setminus p$;
if $\mathfrak{p} = \mathfrak{p}_1$ **then**
  return $\langle s, \mathfrak{p}, R_1^{\sharp\star} \triangledown R_2^{\sharp\star}\rangle$;
else
  return $\langle s, \mathfrak{p}, R_1^{\sharp\star} \sqcup R_2^{\sharp\star}\rangle$;

left unconstrained as they are not in any of the partitions. Therefore we need to use the opportunity of the extension of the support to place constraints on the newly added paths. In order to do so we would like to force the extension of the existing partitions from $U^\sharp$ and $V^\sharp$ into the new support. Therefore we need to define a **compose** operator that produces a sound new partition, given: (1) a pair a, b of partitions (such as the one produced by **unify_widen**), (2) the support s_1 (resp s_2) in which a (resp. b) lives and (3) a space to occupy r. The following criteria must be verified by the resulting partition p in order to be sound and to terminate: $p \cap s_1 = a$, $p \cap s_2 = b$ and $p \setminus (s_1 \cup s_2) \subseteq r$. A variety of **compose** operators could be defined, we chose: **compose**$(a, b, s_1, s_2, r) = a \cup (b \cap (s_2 \setminus s_1)) \cup ((a \triangledown (a \cup b)) \cap r)$. The idea is the following: we keep a (as it is always sound thanks to the definition of the **unify_widen** operator), we keep the part from b that satisfies the soundness condition, and we extend into the space left to occupy according to the automata widening of a and $a \cup b$. In our example, considering the pair (u, v), this would translate as: $a = \mathbf{0}$, $b \cap (s_2 \setminus s_1) = \lfloor \mathbf{1}.\mathbf{0} \rfloor$ and $(a \triangledown (a \cup b)) \cap r = \lfloor \mathbf{0} \rfloor \triangledown \lfloor (\epsilon + 1).\mathbf{0} \rfloor \cap \lfloor \mathbf{1}^{\geq 2}(\mathbf{0} + \mathbf{1}) \rfloor = \lfloor \mathbf{1}^{\geq 2}.\mathbf{0} \rfloor$. We get the new partition: $\lfloor \mathbf{1}^\star.\mathbf{0} \rfloor$. Doing the same with the pair (v, v') yields $\lfloor \mathbf{1}^\star.\mathbf{1} \rfloor$. Finally we get the abstract element $Z_2^\sharp$ from Fig. 7, which is more precise than $Z_1^\sharp$.

Definition 19 (Widening). *Algo. 3 provides the definition of a widening operator using the* **unify_widen** *operator and parameterized by a* **compose** *function.*

Widening stabilization. Our abstraction contains three components: (1) a support that describes the set of paths (2) a subpartitioning of this support and (3) a numerical component giving constraints on partitions in the subpartitioning. We show how the widening operator stabilizes all three components.

– Regular expression widening is used on supports when widening is called. Therefore ensuring support stabilization.

- Once supports are stable (this means $s_2 \subseteq s_1$), we have $p = a$ for every pair (a, b) of partitions. Meaning that once shapes stabilize, the only modifications allowed on the subpartitionings are those made by the **unify_widen** operator. Each partition resulting from the operator is the union of input partitions, hence the subpartitioning will stabilize.
- Once subpartitionings are stable ($\mathfrak{p}_1 = \mathfrak{p}$ in Algo. 3) numerical widening is applied on the numerical component in order to ensure stabilization.

Example 18 (Numerical example). Consider the simple example where: $\mathcal{R} = \{f(2)\}$, $U^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1} \rfloor, \{\lfloor \mathbf{0} \rfloor, \lfloor \mathbf{1} \rfloor\}, \{\lfloor \mathbf{1} \rfloor = \lfloor \mathbf{0} \rfloor\}\rangle$ and $V^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1} \rfloor, \{\lfloor \mathbf{0} \rfloor, \lfloor \mathbf{1} \rfloor\}, \{\lfloor \mathbf{1} \rfloor \geq \lfloor \mathbf{0} \rfloor, \lfloor \mathbf{1} \rfloor \leq \lfloor \mathbf{0} \rfloor + 1\}\rangle$. $U^\sharp$ and $V^\sharp$ have the same shape, therefore widening will be performed on the numerical component of the abstraction, therefore: $U^\sharp \triangledown V^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1} \rfloor, \{\lfloor \mathbf{0} \rfloor, \lfloor \mathbf{1} \rfloor\}, \{\lfloor \mathbf{1} \rfloor \geq \lfloor \mathbf{0} \rfloor\}\rangle$

Reducing dimensionality and improving precision. As emphasized by the previous examples, definitions and illustrations, the numerical component of an abstract state is used as a container for constraints on regular expressions, every node in a regular expression must then satisfy all numerical constraints on the underlying regular expression. Therefore when two nodes of a tree satisfy the same constraints, they should be stored in the same partition so as to reduce the dimension of the numerical domain (thus improving efficiency). Moreover the widening operator provided in Fig.3 relies (for precision) on the fact that partitions are built by similarity of constraints, therefore partition merging, when it does not result in an over-approximation, also leads to a precision gain. The unification operator defined in Fig. 2 tends to split partitions whereas the widening operator defined in Algo. 3 tends to merge them. In order to reduce dimensionality, we would like to define a **reduce** : $\mathcal{C}^\sharp(\mathcal{R}) \to \mathcal{C}^\sharp(\mathcal{R})$ operator, that folds variables with similar constraints into one. Please note that $\forall S \cap S' \subseteq \{x\}$, $x \in S$ and $R^\sharp \in N_S$, we have that $R^\sharp \sqsubseteq_{N_S} \mathbf{expand}(\mathbf{fold}(R^\sharp, x, S'), x, S')$. This means that when variables are folded into one, expanding them afterwards would yield a bigger abstract element. For example, consider the octagon $R^\sharp = \{x \geq 2, y \geq 2, x = y\}$ then $\mathbf{fold}(R^\sharp, z, \{x, y\}) = \{z \geq 2\} (\triangleq R^{\sharp\prime})$ and $\mathbf{expand}(R^{\sharp\prime}, z, \{x, y\}) = \{x \geq 2, y \geq 2\}$. However if we consider $R^\sharp = \{x \geq 2, y \geq 2\}$ then $\mathbf{fold}(\mathbf{expand}(R^\sharp, z, \{x, y\}), z, \{x, y\}) = R^\sharp$. Therefore if we assume given a score function $\mathbf{score}(R^\sharp, x, S')$ ranging in $[0, 1]$ such that $\mathbf{score}(R^\sharp, x, S') = 1 \Leftrightarrow R^\sharp = \mathbf{expand}(\mathbf{fold}(R^\sharp, x, S'), x, S')$, we are able to define a generic **reduce** operator parameterized by a value α. This **reduce** operator merges partitions until no more set of partitions has a high enough score according to the **score** function. Finding a good **score** function is a work in progress. As a first approximation we used the following trivial one: $\mathbf{score}_0(R^\sharp, S) = 1$ when $\mathbf{expand}(\mathbf{fold}(R^\sharp, x, S), x, S) = R^\sharp$ and 0 otherwise. This $\mathbf{score}_0$ guarantees there is no loss of precision, but can miss opportunities for simplification.

Example 19. Consider the following example: $U^\sharp = \langle \lfloor \mathbf{0}+\mathbf{1} \rfloor, \{\lfloor \mathbf{0} \rfloor, \lfloor \mathbf{1} \rfloor\}, \{\lfloor \mathbf{0} \rfloor = 0, \lfloor \mathbf{1} \rfloor = 0\}\rangle$. Relations on $\lfloor \mathbf{0} \rfloor$ and $\lfloor \mathbf{1} \rfloor$ can be expressed in one relation using the summarizing variable $\lfloor \mathbf{0}+\mathbf{1} \rfloor$. This yields: $\mathbf{reduce}(U^\sharp) = \langle \lfloor \mathbf{0}+\mathbf{1} \rfloor, \{\lfloor \mathbf{0} +$

$\mathbf{1}\rfloor\}, \{\lfloor \mathbf{0}+\mathbf{1}\rfloor = 0\}\rangle$. Note that $\mathbf{expand}(\{\lfloor \mathbf{0}+\mathbf{1}\rfloor = 0\}, \lfloor \mathbf{0}+\mathbf{1}\rfloor, \{\lfloor \mathbf{1}\rfloor, \lfloor \mathbf{0}\rfloor\}) = \{\lfloor \mathbf{0}\rfloor = 0, \lfloor \mathbf{1}\rfloor = 0\}$. Therefore no information is lost.

Abstract semantic of operators. As for tree automata, abstract semantic of operators defined in Sec. 2 can be defined as simple transformations on regular automata. Indeed the `make_symbolic`($s \in \mathcal{R}$) (resp. `get_son`) operator, amounts to adding (resp. removing) an integer letter to: (1) the partitions in the subpartitioning and (2) the support. `make_integer`($e \in$ *expr*) amounts to building an abstract element with support $\lfloor \epsilon \rfloor$ and a subpartitioning containing only $\{\lfloor \epsilon \rfloor\}$, on which we put the constraint that it is equal to e. `is_symbol` needs only split the support and each partition, in the two language $L = \{\epsilon\}$ and $A_n^\star \setminus L$. Indeed in order to restrict to terms having only an integer as root, the support must be reduced to ϵ. The `get_sym_head` operator always yields the whole ranked alphabet (as this was abstracted away and will be refined by the automaton abstraction). Finally for `get_num_head`: (1) if the empty path $\wr\int$ is in the support we produce the set of integers satisfying the numerical constraints on the partition containing ϵ, and $\top$ in case no such partition could be found, and (2) otherwise we know that no numerical value is produced.

5.2 Product of tree automata and numerical constraints

The abstraction by tree automata defined in Sec. 3 and the abstraction by numerical constraints on tree paths defined in Sec. 5.1 provide non comparable information on the set of terms they abstract. Indeed the former describes precisely the shape of the term but can not express numerical constraints whereas the latter abstracts away most of the shape and focuses on numerical constraints. To benefit from both kinds of information, we use a reduced product between the two domains. Both abstractions in the product contain information on potential integer positions. The position of the $\square$ symbol in the tree automaton abstraction and the support in the numerical constraints abstractions both yield this information. We remove the support component from the product as the information can be retrieved from the tree abstraction. The definitions of the abstract operators in Sec. 5.1 require the support to be a regular language. We show in this subsection how to retrieve the support of a tree automaton with holes and that it is regular.

Given a FTA$(Q, \mathcal{R}, Q_f, \delta)$ over a ranked alphabet $\mathcal{R}$ with maximum arity n. We assume that every node in Q is reachable. Consider the following system over variables v_p for $p \in Q$ with values in the set of languages over the alphabet A_n (. designates the classical concatenation operator lifted to languages) :

$$\{v_p = \bigcup_{(s,(q_1,\ldots,q_m),q)\in\delta \mid q_i = p} v_q.\{i\} \cup \begin{cases} \{\epsilon\} & \text{if } p \in Q_f \\ \emptyset & \text{otherwise} \end{cases} \mid p \in Q\}$$

Every language $\{i\}$ for $i \in \mathbb{N}$ is regular and does not contain ϵ, moreover $\emptyset$ and $\{\epsilon\}$ are regular languages. By application of Arden's rule (see [18]) and Gauss elimination we can compute the unique solution of this system, moreover

every v_p is regular. Variable v_p is defined so that: $w \in v_p$ if and only if there exists a tree t recognized by the automaton such that $p \in \text{REACH}(t_{|w})$. If $\square \in \mathcal{R}$ we have that the regular language: $\cup_{(\square,(),p)\in\delta} v_p$ represents exactly the potential positions of integers in trees accepted by the tree automaton.

Height and size. The product is enriched with a simple height and size abstraction: numerical variables (encoding heights and sizes) are added to the numerical component of the abstraction.

5.3 Environment abstraction

In the previous section, we designed abstractions for sets of trees. However in order to be able to tackle the examples from the introductory section (Sec. 1) we need to design an abstraction able to represent maps from a set of variables to natural terms. In Sec. 3 we have shown how to lift abstractions on natural terms to abstractions of environments over a given finite set of finite term variables $\mathcal{T}$. We apply the same mechanism here to lift the product presented in Sec. 5.2. However lifting the product would result in abstract environments being maps from natural term variables to abstractions containing a numerical environment. In order to be able to express numerical relations between two sets of natural terms or even between numerical program variables and numerical values of natural terms we factor away the numerical environment so that it is shared by all natural term abstractions in the term environment and by the program variables in the numerical environment. Therefore the final abstraction is a pair $(m, R^\sharp)$ where: (1) m is a map from $\mathcal{T}$ to an abstract element that is a product of the automaton abstraction and the hole positioning abstraction. Moreover as all the numerical constraints are stored in a common numerical environment the product abstraction amounts to a pair $(\mathcal{A}, \mathfrak{p})$ where $\mathcal{A}$ is an element of the automaton abstraction and $\mathfrak{p}$ is a partitioning of its support. (2) $R^\sharp$ is an element of $\mathfrak{M}^\sharp$ binding in the same numerical element: numerical program variables and all partitions found in the mapping m.

6 Implementation and example

6.1 Implementation

The analyzer was implemented in OCaml ($\sim$ 5000 loc) in the novel and still in development Mopsa framework (see [21]). Mopsa enables a modular development of static analyzers defined by abstract interpretation. An analyzer is built by choosing abstract domains, and combining them according to the user specification. Mopsa comes with pre-existing iterators and domains (e.g. interprocedural analysis, loop iterators, numerical domains, ...), and new ones can be added (e.g. tree abstract domain). A key feature of Mopsa is the ability of an abstract domain to use the abstract knowledge it maintains to transform dynamically expressions into other expressions that can be manipulated

more easily by further domains, providing a flexible way to combine relational domains. For instance, assume that a domain abstracts arrays by associating a scalar variable a_0, a_1, ..., to each element $a[0]$, $a[1]$, ..., of an array a, and delegating the abstraction of the array contents to a numeric domain for scalars. It can then evaluate $\mathbb{E}^\sharp[\![2 * a[i] + i]\!](i \mapsto [0,1])$ into the disjunction $(2 * a_0 + i, i \mapsto [0,0]) \vee (2 * a_1 + i, i \mapsto [1,1])$, indicating that $2 * a[i] + i$ is equivalent to $2 * a_0 + i$ in the sub-environment where $i = 0$ and to $2 * a_1 + i$ in the sub-environment where $i = 1$. Each term of the disjunction contains an array-free expression that can be handled by the scalar domain in the corresponding sub-environment. In the abstract, expressions can be evaluated by induction on the syntax into symbolic expressions to retain the full power of relational domains and disjunctive reasoning (see [21] for more details). We exploit this feature in our implementation to combine our tree abstractions. We implemented (in the Mopsa framework) libraries for regular and tree regular languages that offer the usual lattice interface enriched with a widening operator. These libraries can be reused for the definition of other abstract domains. The overall complexity of the analysis is driven by the complexity of the lattice operations in the regular and tree regular libraries. These are exponential in the number of states of the considered automata, which is bounded by the widening parameter.

6.2 Examples of analysis

Numerical variables of the form $\mathtt{t}.x$, where $\mathtt{t}$ is a natural term variable, represent a variable allocated for tree $\mathtt{t}$. For example: $\mathtt{t}.r$ where r is a regular expression is the variable allocated for partition r in tree $\mathtt{t}$.

C introductory example. Let us consider the introductory example Prog. 4. The loop invariant inferred with our analysis is the following abstract element: $U^\sharp = (\mathtt{y} \mapsto (\mathcal{A}, \{\lfloor \mathbf{0}.(\mathbf{0}.\mathbf{0})^\star.\mathbf{1} \rfloor (= r)\}), R^\sharp)$, with $\mathcal{A} = \langle \{a,b,c,d\}, \{*(1), +(2), \square(0), (p,0)\}, \{c\}, \{*(d) \to c, +(c,a) \to d, \square() \to a, p \to c\}\rangle$, and $R^\sharp$ satisfies the constraints: $\{\mathtt{i} \geq 0, \mathtt{i} \leq \mathtt{n}, \mathtt{y}.r = 4\}$. This describes precisely the set of terms of the form: p, $*(p+4)$, $*(*(p+4)+4)$, ... As mentioned in Sec. 6.1 evaluations of tree expressions yield pairs containing an expression and an abstract environment. Tree expressions are pairs $(\mathcal{A}, \mathfrak{p})$, partitions in $\mathfrak{p}$ are bound by the adjoined environment. Let us now present the result of the evaluation of the `make_integer(4)` expression in the abstract environment $U^\sharp$. Here we get the expression $(\mathcal{A}', \{\lfloor \epsilon \rfloor\})$ (where $\mathcal{A}'$ recognizes only $\square$) in the environment: $(\mathtt{y} \mapsto (\mathcal{A}, \{r\}), R^{\sharp\prime})$ where $R^{\sharp\prime} = R^\sharp \cup \{\lfloor \epsilon \rfloor = 4\}$. This emphasizes how the environment is used to give constraints on the adjoined expression. This transports numerical relations from the leafs of the expression up to the assigned variable $\mathtt{t}$.

OCaml introductory example. Let us now consider the introductory example Prog. 5. The inferred loop invariant is the following ($r = \lfloor (\mathbf{1}.\mathbf{1})^\star.\mathbf{0} \rfloor$ and $r' = \lfloor (\mathbf{1}.\mathbf{1})^\star.\mathbf{1}.\mathbf{0} \rfloor$): $(\mathtt{t} \mapsto (\mathcal{A}, \{r, r'\}), R^\sharp)$ and $R^\sharp$ satisfies the constraints: $\{\mathtt{t}.r' = \mathtt{x} - 1$, $\mathtt{t}.r = \mathtt{t}.r' + 2, i \geq 0, i \leq \mathtt{n}\}$ and $\mathcal{A} = (\{a,b,c,d\}, \{\mathtt{Cons}(2), \mathtt{Nil}(0), \square(0)\}, \{a\}, \{\mathtt{Cons}(c,a) \to d, \mathtt{Cons}(c,d) \to a, \mathtt{Nil} \to a, \square \to c\})$. Please note that at the end

of the `while` loops the two numerical environments that need to be joined are not defined over the same set of variables (in the environments that have not gone through the loop, variables $\mathtt{t}.r'$ and $\mathtt{t}.r$ are not present). However thanks to the $\boxplus$ operator, we do not have to loose the numerical relations between these variables and `x`. Hence we are able to prove that the assertion holds.

The analyzer was able to successfully analyze and infer the expected invariants for both examples.

7 Related works

Previous works on sets of trees abstractions [20] were able to recognize larger classes of tree languages than tree automata. However we focused here on the abstraction of trees labeled with numerical values, therefore the work closest to ours would be [12]. Indeed it defines tree automata where leaves can be elements of a lattice (for example an interval). They are therefore able to represent sets of natural terms, but can not express numerical relations between the leaves of trees. Moreover they rely on a partitioning of the leaf lattice for tree automata operations. In [1] (and [2]) tree automata and regular automata are used for the model checking of programs manipulating C pointers and structures. Other uses have been made of tree automata in verification: shape analysis of C programs as in [15], computation of an over-approximation of terms computable by attackers of cryptographic protocols as in [24]. Widening regular languages by the computation of an equivalence relation of bounded index is also done in [9] and in [11]. As mentioned, variable summarization is often used to represent unbounded memory locations as in [17] or [14]. Moreover numerical abstract domains able to handle optional variables have been defined such as [19]. Finally termination analyses have been proposed for the analysis of programs manipulating tree structures (AVL, red-black trees) see [16].

8 Conclusion

In this article we presented a relational abstract environment for sets of trees over a finite algebra, with numerically labeled leaves. We emphasized the potential applications of being able to describe such trees: description of reachable memory zones, tracking symbolic equalities between program variables, description of tree like structures. In order to improve the precision of the analysis while not blowing up its cost we defined a novel abstraction for sets of maps with heterogeneous supports. This numeric abstraction is able to represent optional dimensions in numerical domains without losing relations with optional variables. All domains presented in the article were implemented as a library in the Mopsa framework.

References

1. Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomás Vojnar. Abstract regular tree model checking of complex dynamic data structures. In *Proc. of*

SAS, volume 4134 of *Lecture Notes in Computer Science*, pages 52–70. Springer, 2006.
2. Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In Rajeev Alur and Doron A. Peled, editors, *Proc. of CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.
3. François Bourdoncle. *Sémantiques des Langages Impératifs d'Ordre Supérieur et Interprétation Abstraite*. PhD thesis, Ecole polytechnique, 1992.
4. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007. release October, 12th 2007.
5. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL*, pages 238–252. ACM, 1977.
6. Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of generalized type unions. In *Language Design for Reliable Software*, pages 77–94, 1977.
7. Patrick Cousot and Radhia Cousot. Modular static program analysis. In *Proc. of CC ETAPS*, volume 2304 of *Lecture Notes in Computer Science*, pages 159–178. Springer, 2002.
8. Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. of POPL*, pages 84–96. ACM Press, 1978.
9. Jérôme Feret. Abstract interpretation-based static analysis of mobile ambients. In *Proc. of SAS*, number 2126 in LNCS. Springer-Verlag, 2001. © Springer-Verlag.
10. Tristan Le Gall. *Abstract lattices for the verification of systèmes with stacks and queues.* PhD thesis, University of Rennes 1, France, 2008.
11. Tristan Le Gall, Bertrand Jeannet, and Thierry Jéron. Verification of communication protocols using abstract interpretation of FIFO queues. In *Proc. of AMAST*, volume 4019 of *Lecture Notes in Computer Science*, pages 204–219. Springer, 2006.
12. Thomas Genet, Tristan Le Gall, Axel Legay, and Valérie Murat. Tree regular model checking for lattice-based automata. *CoRR*, abs/1203.1495, 2012.
13. Denis Gopan, Frank DiMaio, Nurit Dor, Thomas W. Reps, and Shmuel Sagiv. Numeric domains with summarized dimensions. In *Proc. of TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 512–529. Springer, 2004.
14. Denis Gopan, Thomas W. Reps, and Shmuel Sagiv. A framework for numeric analysis of array operations. In *Proc. of POPL*, pages 338–350. ACM, 2005.
15. Peter Habermehl, Lukás Holík, Adam Rogalewicz, Jirí Simácek, and Tomás Vojnar. Forest automata for verification of heap manipulation. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proc. of CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 424–440. Springer, 2011.
16. Peter Habermehl, Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Proving termination of tree manipulating programs. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *Proc. of ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2007.
17. Nicolas Halbwachs and Mathias Péron. Discovering properties about arrays in simple programs. In *Proc. of PLDI*, pages 339–348. ACM, 2008.
18. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
19. Jiangchao Liu and Xavier Rival. Abstraction of optional numerical values. In *Proc. of APLAS*, volume 9458 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2015.

20. Laurent Mauborgne. *Representation of Sets of Trees for Abstract Interpretation.* PhD thesis, Ecole polytechnique, 1999.
21. A. Miné, A. Ouadjaout, and M. Journault. Design of a Modular Platform for Static Analysis. In *Proc. of (TAPAS)*, Lecture Notes in Computer Science (LNCS), page 4, 28 Aug. 2018.
22. Antoine Miné. The octagon abstract domain. In *Proc. of WCRE*, page 310. IEEE Computer Society, 2001.
23. Antoine Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *Proc. of VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2006.
24. David Monniaux. Abstracting cryptographic protocols with tree automata. In *Proc. of SAS*, number 1694 in Lecture Notes in Computer Science, pages 149–163. Springer Verlag, 1999.
25. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of 17th IEEE (LICS 2002*, pages 55–74. IEEE Computer Society, 2002.