Abstraction of Optional Numerical Values

Jiangchao Liu and Xavier Rival

INRIA, ENS, CNRS, PSL*, Paris, France,
jliu@di.ens.fr, rival@di.ens.fr

Abstract. We propose a technique to describe properties of numerical stores with optional values, that is, where some variables may have no value. Properties of interest include numerical equalities and inequalities. Our approach lifts common linear inequality based numerical abstract domains into abstract domains describing stores with optional values. This abstraction can be used in order to analyze languages with some form of *option* scalar type. It can also be applied to the construction of abstract domains to describe complex memory properties that introduce symbolic variables, e.g., in order to summarize unbounded sets of program variables, and where these symbolic variables may be undefined, as in some array or shape analyses. We describe the general form of abstract states, and propose sound and automatic static analysis algorithms. We evaluate our construction in the case of an array abstract domain.

1 Introduction

The abstraction of sets of stores is a common problem in static analysis. At a high level, it boils down to identifying a set of predicates over functions that map variables into values. In particular, when the set of variables $\mathbb{X}$ is of fixed and finite size N and the values are scalars (typically, machine integers or floating point numbers), concrete stores are functions of the form $\sigma : \mathbb{X} \to \mathbb{V}$ (where $\mathbb{V}$ stands for the set of values), and can equivalently be described by finite vectors of scalar values $\sigma \in \mathbb{V}^N$. Then, an abstract state describes a set of finite scalar vectors. Typical sorts of abstract states consist of conjunctions of equality constraints [11] or inequality constraints [3,5,13] over variables.

Optional values. Many programming languages feature *possibly empty* memory locations. For instance, OCaml and Scala have an `option` type. This type can be defined by `type 'a option = None | Some of 'a`, which means a value of type `int option` may either be an integer, or undefined, represented by `None`. Similarly, spreadsheet environments feature empty cells as well as an `empty` type. When each variable either contains a scalar or no value, then the set of stores is $\mathcal{P}(\mathbb{X} \to \{\ominus\} \uplus \mathbb{V})$ where $\ominus$ stands for "no value". The conventional abstract domains mentioned above fail to describe such sets, as they require each dimension in the abstract domain to correspond to one concrete memory location, i.e., concrete stores should be of the form $\overrightarrow{\sigma} \in \mathbb{V}^N$. Therefore, they would need to be extended with support for *empty* values in order to deal with optional values.

Packing memory locations. Another situation where support for optional values would be needed occurs when designing abstract domains for complex data-structures. Indeed, a common technique *packs* sets of memory locations together, so that a single abstract constraint describes the values stored in a group of memory locations. For instance, array analyses often rely on *array partitioning* techniques, that divide an array into groups of cells. Then, the values of the cells of a group are described by a single abstract dimension. Many array analyses do not allow empty groups or treat them in a specific way [8,9,4]. In this case, *summarizing dimensions* [8] can be used in the abstract domain, so as to describe groups of several concrete locations. However, other analyses such as [12] allow empty groups. Then, considering an abstract state, a given abstract dimension may describe an empty set of values, even though the abstract state itself describes a non empty set of stores. A similar situation arises in some shape analyses [18]. Again, the aforementioned numerical domains do not support such an abstraction relation.

Abstraction of stores in presence of possibly empty locations. In the previous two paragraphs, we have shown several cases where abstractions for stores with optional numerical values are needed. A first approach relies on disjunctions so as to partition a set of stores S into several sets $S_0, \ldots, S_p$, such that, each S_i corresponds to a fixed set of defined variables. However, when k variables may be empty or not, this would lead to an exponential factor of complexity. Another solution consists in adding a flag $\mathtt{f_x}$ for each variable $\mathtt{x}$, such that $\mathtt{f_x} = 1$ if $\mathtt{x}$ is defined and $\mathtt{f_x} = 0$ otherwise [16]. While this technique nicely describes relations of the form "$\mathtt{x}$ is defined if and only if $\mathtt{y}$ is defined", it is less adapted to infer that a variable is undefined from a set of constraints that show no value can be found for $\mathtt{x}$ thus it is undefined. The latter situation is common in array analyses like [12] and where the emptiness of a group of cells may follow from the numerical constraints over the values of these cells. To alleviate this issue, [6,12] deploy one relational abstract value per possibly empty zone, which is overly costly and limits the relations that can be expressed.

In this paper, we take a radically different approach, where constraints over a variable $\mathtt{x}$ may prove that no value is admissible for $\mathtt{x}$, hence it is undefined. Yet, the concretizations of the existing numerical abstract domains do not cope with one dimension describing the empty set while the others are still defined. Therefore, we let a variable $\mathtt{x}$ that may be undefined be described by a group of *avatars* $\mathtt{x}^0, \ldots, \mathtt{x}^k$, and assume that $\mathtt{x}$ can be defined if and only if all its avatars may be defined to a common value. For instance, constraints $\mathtt{x}^0 < 10 \wedge \mathtt{x}^1 > 20$ cannot be satisfied with a value assignment that maps $\mathtt{x}^0$ and $\mathtt{x}^1$ to the same value, hence this pair of constraints describes states where $\mathtt{x}$ is necessarily undefined. This principle can be applied to any numerical abstract domain where abstract values are finite conjunctions of constraints (the vast majority of numerical abstract domains are of that form). We propose an *abstract domain functor* for linear inequalities abstractions, called the *Maya functor* [1]. We present the following contributions:

– we define a concrete model for optional values (Section 3);

[1] Mayas are among the civilizations believed to have independently invented number "zero".

```
1. int option y; int x;
2. if(y == Some v_y)
3.       assume(v_y ≤ x && v_y ≥ 20);
4. if(x ≤ 10)
5.       assert(y == None);
```

Fig. 1. A routine involving optional variables

- we set up a general functor lifting a numerical abstract domain without optional dimensions into a domain with optional dimensions (Section 4);
- we define sound transfer functions and lattice operators for the automatic analysis of programs with optional variables, with linear inequalities (Section 5);
- we handle possibly empty summary dimensions (Section 6);
- we evaluate an implementation of the Maya functor (Section 7).

2 Overview

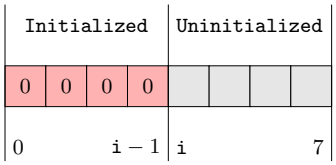
In this section, we demonstrate the principle of our abstraction, with a numeric analysis on a program involving optional variables and a basic array analysis applied to an initialization routine.

Abstraction of optional variables. We first consider the code fragment shown in Figure 1. It is written in a C-like language extended with optional variables. An optional variable (as **int option** y in line 1) could be either no value (represented by **None**) or one value (represented by **Some** $\mathtt{v}_y$ where $\mathtt{v}_y$ is an integer). The statements in line 2 and 3 constrain $\mathtt{v}_y$ to be between 20 and the value of x ($20 \leq \mathtt{v}_y \leq \mathtt{x}$) if optional variable y stores an integer $\mathtt{v}_y$. The test in line 4 constrains variable x to be smaller or equal than 10 ($\mathtt{x} \leq 10$). This implies that, if y stores an integer $\mathtt{v}_y$, then $\mathtt{v}_y$ is greater or equal than 20 and smaller or equal than 10 ($20 \leq \mathtt{v}_y \leq 10$). There exists no such integer, thus, y may only store **None**, and the assertion in line 5 never fails. To prove this assertion by static analysis, we first need to represent all numerical properties using a numerical abstract domain. All the numeric constraints in this example are of the form $\pm\mathtt{x} \pm \mathtt{y} \leq c$ and can thus be described in the octagons abstract domain [13]. However, an octagon describes either the empty set of stores, or a set containing at least one store, that maps each variable, including y, into a value. Thus, this abstract domain cannot express that y stores **None**, while the other variables hold a value. Siegel et *al* [16] add a flag variable $\mathtt{f}_\mathtt{y}$ to indicate whether y stores one value or no value. However, solving the problem using this approach requires to precisely capture the property that $20 \leq \mathtt{v}_y \leq \mathtt{x}$ when $\mathtt{f}_\mathtt{y} = 1$ and $\mathtt{y} = \mathbf{None}$ when $\mathtt{f}_\mathtt{y} = 0$. This property cannot be expressed in a single octagon. Hence, the approach of [16] would require a stronger, more ad hoc abstract domain (most likely, using a disjunction of octagons).

Abstraction of possibly empty sets of values. The key idea of our method is to represent a single variable using several instances called *avatars*, carrying differ-

```
int i = 0; int a[8];
while(i < 8){
  ⋆ (0 ≤ Idx_0 ≤ i − 1 ∧ i ≤ Idx_1 ≤ 7)
    a[i] = 0;
i = i + 1;
}
```

(a) Array initialization routine and invariant



(b) Shape of intermediate states

Fig. 2. An array initialization example

ent constraints. This way, we ensure both that (1) the abstract domain describes stores which map each variable to one value and (2) we can express either that $\mathtt{y}$ must be empty (when it is not possible to find a value all its avatars can be mapped to) or that it may store some value v (when all constraints are satisfied when all the avatars of $\mathtt{y}$ are mapped to v). To implement this idea using the octagons abstract domain, we simply distinguish, for a variable $\mathtt{y}$ that may have an empty set of values, two sets of constraints: the constraints of the form $\mathtt{y} \pm \mathtt{x} \leq c$ (resp., $-\mathtt{y} \pm \mathtt{x} \leq c$) are carried out by its *upper-bound avatar* $\mathtt{y}^+$ (resp., *lower-bound avatar* $\mathtt{y}^-$). This means, that a (non-bottom) octagon containing constraints $\mathtt{y}^+ \leq 0$ and $1 \leq \mathtt{y}^-$ expresses $\mathtt{y}$ is necessarily mapped to no value, as there is no way to satisfy the constraints over its two avatars, while mapping them to a common value. Applying this method to the example in Figure 1, we associate two avatars $\mathtt{y}^-$ and $\mathtt{y}^+$ to the optional variable $\mathtt{y}$ (non-optional variable $\mathtt{x}$ is not associated with distinct avatars). Three numeric relations ($20 \leq \mathtt{y}^- \ \wedge \ \mathtt{y}^+ \leq \mathtt{x} \ \wedge \ \mathtt{x} \leq 10$) are observed before the assertion in line 5. According to the constraints, $\mathtt{y}^-$ may take any value greater than 20, whereas $\mathtt{y}^+$ may take any value smaller than 10, so the concretization of this abstract state contains no state that maps both avatars of $\mathtt{y}$ to a common value. Thus $\mathtt{y}$ stands for no value, which proves the assertion in line 5. Unlike [16], the bi-avatar approach allows constraints $20 \leq \mathtt{y}$ and $\mathtt{y} \leq 10$ to co-exist in a single abstract element, that does not describe the empty set of states. The computation of abstract post-conditions is quite standard, except that it needs to always associate upper and lower constraints to the right avatar.

Packing array cells. Figure 2(a) shows a C code segment of array initialization. We consider an array analysis inspired by [12], which proceeds by forward abstract interpretation [3] (note that the main emphasis of this section is not the array analysis itself, but the abstraction of optional values). A store observed after 4 iterations is shown in Figure 2(b). We note the array can be divided into two sets of cells, namely initialized cells and uninitialized cells. As in [12], we consider an abstraction of the array, that partitions it into two groups of cells G_0, G_1 (where all cells in G_0 are initialized to zero and cells in G_1 may hold any value), and we let two summary variables $\underline{\mathrm{Idx}}_0, \underline{\mathrm{Idx}}_1$ over-approximate the sets of indexes corresponding to the cells of each group. Before the loop starts, $\underline{\mathrm{Idx}}_0$ stands for $\emptyset$. In Figure 2(b), $\underline{\mathrm{Idx}}_0$ stands for set $\{0, 1, 2, 3\}$. Before the loop execution, group G_0 is

empty. Actually, [12] will introduce it only during the first iteration of the loop. At this point, the analysis infers that $\underline{\mathrm{Idx}}_0 = \{0\}$ (group G_0 has a single element at this point); moreover, it computes that $1 \leq \underline{\mathrm{Idx}}_1 \leq 7$. During the loop execution, we observe the following constraints over group indexes form a loop invariant:

$$0 \leq \underline{\mathrm{Idx}}_0 \leq \mathtt{i} - 1 \wedge \mathtt{i} \leq \underline{\mathrm{Idx}}_1 \leq 7$$

After the loop exit, $8 \leq \mathtt{i}$, therefore the analysis will return $8 \leq \underline{\mathrm{Idx}}_1 \leq 7$. Obviously no value satisfies this constraint. This actually means that group G_1 is empty at this stage, thus, the analysis proves the whole array is initialized to 0. The representation of the numerical properties over group indexes suffers from the same issue as for the optional values, in the analysis of the program of Figure 1: a non bottom octagon element cannot express $8 \leq \underline{\mathrm{Idx}}_1 \leq 7$. The solution used in [12] describes each group with a separate octagon. In this layout, an empty group is naturally described by a bottom octagon attached to its $\underline{\mathrm{Idx}}_i$ variable. Yet, this prevents the analysis from inferring constraints across distinct groups.

Using our method, the analysis could describe symbolic variables associated to the groups (that is, in our example, $\underline{\mathrm{Idx}}_0, \underline{\mathrm{Idx}}_1$) by a pair of avatars (while program variables (like i) are not associated to distinct avatars in octagons). It computes the following invariants:

before the loop	$\mathtt{i} = 0 \wedge 0 \leq \underline{\mathrm{Idx}}_1^- \wedge \underline{\mathrm{Idx}}_1^+ \leq 7$
end of the 1st iter	$\mathtt{i} = 1 \wedge 0 \leq \underline{\mathrm{Idx}}_0^- \wedge \underline{\mathrm{Idx}}_0^+ \leq 0 \wedge 1 \leq \underline{\mathrm{Idx}}_1^- \wedge \underline{\mathrm{Idx}}_1^+ \leq 7$
loop invariant	$0 \leq \mathtt{i} \leq 7 \wedge 0 \leq \underline{\mathrm{Idx}}_0^- \wedge \underline{\mathrm{Idx}}_0^+ \leq \mathtt{i} - 1 \wedge \mathtt{i} \leq \underline{\mathrm{Idx}}_1^- \wedge \underline{\mathrm{Idx}}_1^+ \leq 7$
loop exit	$8 \leq \mathtt{i} \wedge 0 \leq \underline{\mathrm{Idx}}_0^- \wedge \underline{\mathrm{Idx}}_0^+ \leq \mathtt{i} - 1 \wedge \mathtt{i} \leq \underline{\mathrm{Idx}}_1^- \wedge \underline{\mathrm{Idx}}_1^+ \leq 7$

When the loop terminates, we observe that the abstract state contains constraints $8 \leq \mathtt{i}$, $\mathtt{i} \leq \underline{\mathrm{Idx}}_1^-$ and $\underline{\mathrm{Idx}}_1^+ \leq 7$. Since $\underline{\mathrm{Idx}}_1^-$ and $\underline{\mathrm{Idx}}_1^+$ cannot be concretized to a common value, $\underline{\mathrm{Idx}}_1$ describes an empty set of values at this point, thus group G_1 is empty. In other words, all cells of the array are initialized at this point.

3 A language with optional values and its semantics

Before we can formalize the abstraction relation of our domain functor, we need to specify a concrete semantics. To do that, we describe a basic imperative language where some variables have an optional value. It models both languages with an option type as well as the operations shape and array analyses with empty groups require their base domain to provide. The semantics of this language will serve as a basis to state the soundness properties of the transfer functions defined in the functor for the abstraction of optional scalar values.

Syntax. The syntax is shown in Figure 3. We distinguish the variables that may be empty, called the *optional* variables from the *standard* variables, that must store one value. We let $\mathbb{X}$ denote the set of standard variables, and we write $\mathbb{Y}$ for the set of optional variables (we assume $\mathbb{X} \cap \mathbb{Y} = \emptyset$). These two sets are assumed to be fixed throughout the paper. We also let $\mathbb{V}$ stand for the set of values. Values and variables

$$
\begin{array}{lll}
\mathbb{X}: & \text{standard (non empty) variables} & (\mathtt{x} \in \mathbb{X}) \\
\mathbb{Y}: & \text{optional variables (may be empty)} & (\mathtt{y} \in \mathbb{Y}) \\
\mathbb{V}: & \text{values} & (\mathtt{c} \in \mathbb{V})
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{t} & ::= & \mathtt{x} \mid \mathtt{y} & \oplus ::= + \mid - \mid * \mid \div \qquad \otimes ::= < \mid \leq \mid == \mid != \\
\mathtt{ex} & ::= & \mathtt{t} \mid \mathtt{c} \mid \mathtt{ex} \oplus \mathtt{ex} & \text{scalar expressions} \\
\mathtt{cond} & ::= & \mathtt{TRUE} \mid \mathtt{FALSE} \mid \mathtt{ex} \otimes \mathtt{ex} \mid \mathbf{is\_empty}(\mathtt{y}) & \text{condition tests} \\
\mathtt{s} & ::= & \mathbf{skip} \mid \mathtt{t} = \mathtt{ex} \mid \mathbf{assume}(\mathtt{cond}) \mid \mathbf{assert}(\mathtt{cond}) & \text{basic statements} \\
 & \mid & \mathtt{s}; \mathtt{s} \mid \mathbf{if}(\mathtt{cond})\{\mathtt{s}\}\mathbf{else}\{\mathtt{s}\} \mid \mathbf{while}(\mathtt{cond})\{\mathtt{s}\} & \text{compound statements}
\end{array}
$$

Fig. 3. A language with optional values: syntax

Evaluation of Expressions: $[\![\mathtt{ex}]\!] : \mathbb{S} \to \mathbb{V} \uplus \{\ominus\}$

$$
[\![\mathtt{t}]\!](\sigma) = \sigma(\mathtt{t}) \qquad [\![\mathtt{c}]\!](\sigma) = \mathtt{c}
$$

$$
[\![\mathtt{ex}_0 \oplus \mathtt{ex}_1]\!](\sigma) = \begin{cases} [\![\mathtt{ex}_0]\!](\sigma) \oplus [\![\mathtt{ex}_1]\!](\sigma) & \forall i \in \{0,1\},\ [\![\mathtt{ex}_i]\!](\sigma) \in \mathbb{V} \\ \ominus & \exists i \in \{0,1\},\ [\![\mathtt{ex}_i]\!](\sigma) = \ominus \end{cases}
$$

Condition tests: $[\![\mathtt{cond}]\!] : \mathcal{P}(\mathbb{S}) \to \mathcal{P}(\mathbb{S})$

$$
[\![\mathtt{TRUE}]\!](\mathtt{S}) = \mathtt{S} \qquad [\![\mathtt{FALSE}]\!](\mathtt{S}) = \emptyset \qquad [\![\mathbf{is\_empty}(\mathtt{y})]\!](\mathtt{S}) = \{\sigma \in \mathtt{S} \mid \sigma(\mathtt{y}) = \ominus\}
$$

$$
[\![\mathtt{ex}_0 \otimes \mathtt{ex}_1]\!](\mathtt{S}) = \{\sigma \in \mathtt{S} \mid [\![\mathtt{ex}_0]\!](\sigma) \otimes ([\![\mathtt{ex}_1]\!](\sigma) = \mathtt{TRUE}) \vee ([\![\mathtt{ex}_0]\!](\sigma) = \ominus) \vee ([\![\mathtt{ex}_1]\!](\sigma) = \ominus)\}
$$

Main statements: $[\![\mathtt{s}]\!] : \mathcal{P}(\mathbb{S}) \to \mathcal{P}(\mathbb{S})$

$$
\begin{array}{rl}
[\![\mathbf{skip}]\!](\mathtt{S}) = \mathtt{S} & [\![\mathtt{s}_0; \mathtt{s}_1]\!](\mathtt{S}) = [\![\mathtt{s}_1]\!] \circ [\![\mathtt{s}_0]\!](\mathtt{S}) \\
[\![\mathtt{t} = \mathtt{ex}]\!](\mathtt{S}) = \{\sigma[\mathtt{t} \mapsto [\![\mathtt{ex}]\!](\sigma)] \mid \sigma \in \mathtt{S}, [\![\mathtt{ex}]\!](\sigma) \in \mathbb{V} \vee \mathtt{t} \in \mathbb{Y}\} & \\
[\![\mathbf{if}(\mathtt{cond})\{\mathtt{s}_0\}\mathbf{else}\{\mathtt{s}_1\}]\!](\mathtt{S}) = [\![\mathtt{s}_0]\!] \circ [\![\mathtt{cond}]\!](\mathtt{S}) \cup [\![\mathtt{s}_1]\!] \circ [\![\mathtt{cond} == \mathtt{FALSE}]\!](\mathtt{S}) & \\
[\![\mathbf{while}(\mathtt{cond})\{\mathtt{s}\}]\!](\mathtt{S}) = [\![\mathtt{cond} == \mathtt{FALSE}]\!](\mathbf{lfp}\ \lambda \mathtt{S}' \cdot \mathtt{S} \cup [\![\mathtt{s}]\!]([\![\mathtt{cond}]\!](\mathtt{S}'))) &
\end{array}
$$

Fig. 4. A language with optional values: concrete semantics

all have scalar type (integer or floating point). Finally, we let $\ominus \notin \mathbb{V}$ denote the absence of value. Conditions include usual arithmetic tests and the emptiness test of an optional variable. Statements include the usual skip statement (that does nothing), assignments, sequences, condition tests and loops.

Memory states. A *concrete memory* (or *store*) σ maps each standard variable into a value and each optional variable to either a scalar value or to the $\ominus$ placeholder, meaning that this variable is not defined. Therefore the set of stores is:

$$
\mathbb{S} \overset{\text{def.}}{::=} (\mathbb{X} \to \mathbb{V}) \uplus (\mathbb{Y} \to (\mathbb{V} \cup \{\ominus\}))
$$

Semantics. The concrete semantics is formally defined in Figure 4 (**assume** and **assert** statements are classical and omitted). While the overall structure of this semantics is standard, a few points should be noticed. The semantics $[\![\mathtt{ex}]\!]$ of expression $\mathtt{ex}$ evaluates it in a given store, and produces either a value or the no

value $\ominus$ element. It produces $\ominus$ whenever it reads an empty optional variable: all operators are $\ominus$-strict, i.e., they return $\ominus$ whenever one of their arguments is equal to $\ominus$, thus $\ominus$ always propagates. The semantics $[\![\texttt{cond}]\!]$ of condition `cond` filters out the stores in which `cond` does not evaluate to `TRUE`, thus, it will also include stores where the evaluation encounters $\ominus$.

The semantics $[\![\texttt{s}]\!]$ of statement `s` takes a set of input stores and returns the corresponding set of output stores, following an angelic denotational semantics [2] (non terminating behaviors are not represented —this choice simplifies the presentation, while it does not change anything to the core points of the paper). Note that the semantics of an assignment $\texttt{x} = \texttt{ex}$ where $\texttt{x} \in \mathbb{X}$ will produce no output store when `ex` evaluates to $\ominus$. Intuitively, we consider only executions where the empty value is never assigned to a standard variable.

Example 1. We consider the program below, where $\mathbb{X} = \{\texttt{x}\}$ and $\mathbb{Y} = \{\texttt{y}, \texttt{z}\}$:

$$
\begin{array}{l}
\textbf{if}(\texttt{x} \leq \texttt{y})\{ \\
\quad \textbf{if}(\texttt{y} \leq 6)\{ \\
\qquad ① \ \texttt{z} = \texttt{y} + 2; \\
\qquad ② \ \ldots;
\end{array}
$$

Assuming that all variables may take any value (including $\ominus$ for optional variables) at the beginning of the execution:

- at point ①, we can observe exactly the stores such that $\sigma(\texttt{x}) \leq \sigma(\texttt{y}) \leq 6$, and the stores defined by $\sigma(\texttt{y}) = \ominus$;
- at point ②, we can observe exactly the stores such that $\sigma(\texttt{x}) \leq \sigma(\texttt{y}) \leq 6 \wedge \sigma(\texttt{z}) = \sigma(\texttt{y}) + 2$ and the stores where $\sigma(\texttt{y}) = \ominus$ or $\sigma(\texttt{z}) = \ominus$.

4 Abstraction in presence of optional numerical values

In this section, we assume a numerical domain $\mathbb{N}^\sharp$ is fixed, where abstract values correspond to conjunctions of constraints. For instance, linear equalities [11], intervals [3], octagons [13] and polyhedra [5] fit into this category. An abstract value $N^\sharp \in \mathbb{N}^\sharp$ describes constraints over a finite set of "abstract variables" that we refer to as *dimensions* (so as to distinguish them from the "concrete" —standard or optional— variables). Dimensions range over a countable set $\mathbb{D}$, and we write $\mathbf{Dim}(N^\sharp)$ for the dimensions of abstract value $N^\sharp$ ($\mathbf{Dim}(N^\sharp) \subseteq \mathbb{D}$). We let $\gamma_{\mathbb{N}^\sharp} : \mathbb{N}^\sharp \to \mathcal{P}(\mathbb{D} \to \mathbb{V})$ denote its concretization function.

Abstract states. An abstract state of the *Maya abstract domain* over $\mathbb{N}^\sharp$ is defined by an abstract value $N^\sharp \in \mathbb{N}^\sharp$ describing constraints over a set of dimensions defined as follows:

- each standard variable `x` corresponds to exactly one dimension, also noted `x`;
- each optional variable `y` corresponds to a finite set of *avatar dimensions* (for clarity, we always mark avatars with superscripts such as: $\texttt{y}^-, \texttt{y}^+, \texttt{y}^0, \ldots$).

Therefore, we attach a function $\mathcal{A} : \mathbb{Y} \to \mathcal{P}(\mathbb{D})$ which describes the mapping of optional variables into their set of avatars to numerical abstract value $N^\sharp$.

Definition 1 (Abstract state). *An abstract state of the* Maya abstract domain *over* $\mathbb{N}^\sharp$ *is a pair* $M^\sharp = (N^\sharp, \mathcal{A})$ *such that:*

$$\mathbf{Dim}(N^\sharp) = \left(\biguplus\{\mathcal{A}(\mathtt{y}) \mid \mathtt{y} \in \mathbb{Y}\}\right) \uplus \mathbb{X}$$

We let $\mathbb{M}^\sharp$ *denote the set of such states.*

Note that the above definition implicitly asserts that distinct variables are represented by disjoint sets of dimensions.

Example 2. In this example, we assume $\mathbb{N}^\sharp$ is the octagon domain, and that $\mathbb{X} = \{\mathtt{x}\}, \mathbb{Y} = \{\mathtt{y}\}$. Furthermore, as shown in Section 2, we let each optional variable be described by two avatars. Thus, $\mathbb{D} = \{\mathtt{x}, \mathtt{y}^-, \mathtt{y}^+\}$. Moreover, an example abstract state is $M^\sharp = (N^\sharp, \mathcal{A})$, with:

$$N^\sharp = \left\{0 \leq \mathtt{x} \wedge \mathtt{x} \leq 10 \wedge 5 \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq \mathtt{x}\right\} \qquad \mathcal{A} : \mathtt{y} \longmapsto \{\mathtt{y}^-, \mathtt{y}^+\}$$

Concretization. To express the meaning of an abstract state $M^\sharp = (N^\sharp, \mathcal{A})$, we use a valuation ν, that maps all dimensions to a value, as an intermediate step towards the concrete stores. Then, we retain only the concrete stores, that can be obtained by collapsing *all avatars of each optional variable* to a unique value. This second step is described by a pair of *consistency predicates*, which state when a store σ is compatible with ν:

Definition 2 (Concretization). *Given abstract state* $M^\sharp = (N^\sharp, \mathcal{A})$*, we define the following consistency predicates:*

$$\begin{array}{lcl} P_{\mathbb{X}}(\sigma, M^\sharp, \nu) & \overset{def.}{\iff} & \forall \mathtt{x} \in \mathbb{X},\ \sigma(\mathtt{x}) = \nu(\mathtt{x}) \\ P_{\mathbb{Y}}(\sigma, M^\sharp, \nu) & \overset{def.}{\iff} & \forall \mathtt{y} \in \mathbb{Y},\ (\forall d \in \mathcal{A}(\mathtt{y}),\ \nu(d) = \sigma(\mathtt{y})) \vee \sigma(\mathtt{y}) = \ominus \end{array}$$

Then, the concretization of $M^\sharp = (N^\sharp, \mathcal{A})$ *is defined by:*

$$\gamma_{\mathbb{M}^\sharp}(M^\sharp) \overset{def.}{::=} \left\{\sigma \in \mathbb{S} \mid \exists \nu \in \gamma_{\mathbb{N}^\sharp}(N^\sharp),\ P_{\mathbb{X}}(\sigma, M^\sharp, \nu) \wedge P_{\mathbb{Y}}(\sigma, M^\sharp, \nu)\right\}$$

Intuitively, consistency predicate $P_{\mathbb{X}}$ asserts that the valuation and the concrete store agree on the mapping of the standard variables, whereas consistency predicate $P_{\mathbb{Y}}$ asserts that the valuation assigns all avatars of each optional variable to the value of that variable in the store.

Example 3. We consider the abstract state shown in Example 2. Its concretization consists of:

- the stores defined by $\sigma(\mathtt{x}) \in [5, 10], \sigma(\mathtt{y}) \in [5, \sigma(\mathtt{x})]$ (the valuation is then fully defined by the store since no variable stores $\ominus$);
- the stores defined by $\sigma(\mathtt{x}) \in [0, 10], \sigma(\mathtt{y}) = \ominus$ (a possible valuation is defined by $\nu(\mathtt{x}) = \sigma(\mathtt{x}), \nu(\mathtt{y}^-) = 15, \nu(\mathtt{y}^+) = \nu(\mathtt{x})$).

This example shows how our domain can distribute the constraints on an optional variable $\mathtt{y}$ over several dimensions, so as to express the fact that $\mathtt{y}$ must store $\ominus$.

Remark 1. In this example, we also observe that, given $\sigma \in \gamma_{\mathbb{M}^\sharp}(M^\sharp)$, and if σ' is such that, for all standard variable $\mathtt{x}$, $\sigma'(\mathtt{x}) = \sigma(\mathtt{x})$, and for all optional variable $\mathtt{y}$, either $\sigma'(\mathtt{y}) = \sigma(\mathtt{y})$ or $\sigma'(\mathtt{y}) = \ominus$, then $\sigma' \in \gamma_{\mathbb{M}^\sharp}(M^\sharp)$. In other words, our functor cannot express that an optional variable *must not* store $\ominus$. In the context of array analyses such as [12], this is not a limitation: that analysis can already express that a group *cannot* be empty (using size constraints). However, our abstraction also allows to derive emptiness of a group via constraints over multiple avatars of variables denoting its contents or indexes, which [12] does in a rather *ad hoc* manner, at the expense of relations between groups.

Choice of avatar dimensions. The definition of abstract elements assumes nothing about the number of avatar dimensions, and about the way the constraints over an optional variable are distributed over its avatars. However, in practice, the way avatar dimensions are managed has a great impact on the efficiency and precision of the analysis. It is the role of the transfer functions and abstract lattice operations to implement an efficient strategy to manage these dimensions. In particular at certain stages, new avatars have to be introduced so as to avoid a loss of precision.

Example 4. We discuss possible abstract invariants for the program shown in Example 1, starting with the set of all stores as a pre-condition, described by abstract state $\top$. After test $\mathtt{x} \leq \mathtt{y}$, the analysis should compute an abstraction of the stores where, either $\mathtt{y}$ is mapped only to $\ominus$ or where the numerical constraint is satisfied. Using the octagon abstract domain, and a single avatar $\mathtt{y}^0$ for $\mathtt{y}$, this boils down to abstract state $(\mathtt{x} - \mathtt{y}^0 \leq 0, \mathtt{y} \mapsto \{\mathtt{y}^0\})$. After the second test, we get the set of stores observed at point ①, that is such that, either $\sigma(\mathtt{x}) \leq \sigma(\mathtt{y}) \leq 6$ or $\sigma(\mathtt{y}) = \ominus$. Note that this set of stores cannot be described exactly with octagons using a single avatar. Indeed, this set contains stores such that $\sigma(\mathtt{x}) > 6$ (when $\sigma(\mathtt{y}) = \ominus$). Thus, using a single avatar to describe constraints over $\mathtt{y}$ would force the analysis to drop either constraint $\mathtt{y} \leq 6$ or constraint $\mathtt{x} \leq \mathtt{y}$. Keeping both constraints would unsoundly assert $\mathtt{x} \leq 6$. Thus, adding a second avatar for $\mathtt{y}$ at this point is necessary in order to maintain maximal precision. In particular, the abstract state below describes exactly the stores that can be observed at point ①:

$$(\mathtt{x} \leq \mathtt{y}^0 \wedge \mathtt{y}^1 \leq 6, \mathtt{y} \mapsto \{\mathtt{y}^0, \mathtt{y}^1\})$$

The above example demonstrates the need to introduce enough avatars so that all constraints on optional variables can be maintained, without "over-constraining" standard variables (which would result in an unsound analysis). Intuitively, each avatar should not carry too much information: the base numerical domain cannot express emptiness of a specific avatar; instead, only the conjunction of all avatars of an optional variable $\mathtt{y}$ may express that $\mathtt{y}$ is empty. We formalize this as a *sufficient condition*, that we call the *independence* property, and that should be maintained by all abstract operators in the Maya domain. This property states that dropping the constraints over an avatar dimension $\mathtt{y}^0$ associated to variable $\mathtt{y}$ should have no impact on the variables other than $\mathtt{y}$. To maintain this property, transfer functions and abstract operators may either pay the cost of adding new avatar dimensions or will have to drop constraints that cannot be represented

without adding more avatars. To formalize the independence property, and given abstract value $N^\sharp \in \mathbb{N}^\sharp$ and dimension d, we note $\mathbf{drop}(N^\sharp, d)$ for the abstract value obtained by removing from $N^\sharp$ all the constraints that involve d (this operation is well defined since we assumed elements of abstract domain $\mathbb{N}^\sharp$ correspond to the finite conjunctions of all the constraints of a certain form). Moreover, if ν is a valuation, we write $\nu_{|\neg d}$ for the restriction of ν to $\mathbb{D} \setminus \{d\}$.

Definition 3 (Independence property). *Let $M^\sharp = (N^\sharp, \mathcal{A})$ be an abstract state. We say $M^\sharp$ satisfies the independence property if and only if*

$$\forall \mathtt{y} \in \mathbb{Y},\ \forall d \in \mathcal{A}(\mathtt{y}),\ \{\nu_{|\neg d} \mid \nu \in \gamma_{\mathbb{N}^\sharp}(N^\sharp)\} = \{\nu_{|\neg d} \mid \nu \in \gamma_{\mathbb{N}^\sharp}(\mathbf{drop}(N^\sharp, d))\}$$

Example 5. The abstract state given at the end of Example 4 satisfies the independence property, using two avatars, that respectively carry the lower and upper bound constraints over $\mathtt{y}$. Section 5 generalizes this approach to lift any domain based on linear inequalities.

Example 6. Intuitively, the independence property is likely to break when an avatar dimension carries several constraints, the conjunction of which may be unsatisfiable. Therefore, an alternate technique to achieve it consists in using one avatar per constraint over each optional variable. As an example, we consider the set of concrete states defined by $\mathbb{X} = \{\mathtt{x}\}$ and $\mathbb{Y} = \{\mathtt{y}, \mathtt{z}\}$ and where the optional variables are either undefined or satisfy the following conditions: $\mathtt{x} \leq \mathtt{y} \wedge \mathtt{y} \leq 2\mathtt{x} \wedge \mathtt{y} = \mathtt{z} + 2$. Then, assuming $\mathbb{N}^\sharp$ is the polyhedra abstract domain, this *multi-avatar* strategy will construct the following abstract state:

$$\begin{array}{l} N^\sharp = \left\{\mathtt{x} \leq \mathtt{y}^0 \wedge \mathtt{y}^1 \leq 2\mathtt{x} \wedge \mathtt{y}^2 \leq \mathtt{z}^0 + 2 \wedge \mathtt{z}^1 + 2 \leq \mathtt{y}^3\right\} \\ \mathcal{A} \ :\ \mathtt{y} \mapsto \{\mathtt{y}^0, \mathtt{y}^1, \mathtt{y}^2, \mathtt{y}^3\}, \mathtt{z} \mapsto \{\mathtt{z}^0, \mathtt{z}^1\}, \end{array}$$

This strategy is general (it can be applied to, e.g., linear equalities [11]) but costly.

5 Application to numerical domains based on linear inequalities

We now propose a strategy to manage avatar dimensions and design abstract operations under the hypothesis that base abstract domain $\mathbb{N}^\sharp$ expresses linear inequality constraints (which includes intervals, octagons, polyhedra, and their variants).

5.1 The bi-avatar strategy

Numerical constraints in the base domain are all of the form $a_0\mathtt{d}_0 + \ldots + a_n\mathtt{d}_n \leq c$ (where $a_0, \ldots, a_n, c$ are constants), thus a constraint involving $\mathtt{d}_i$ (i.e., where $a_i \neq 0$) is either specifying an upper bound for $\mathtt{d}_i$ (if $a_i > 0$) or a lower bound (if $a_i < 0$). The bi-avatar strategy treats those two sets of constraints separately, using two avatar dimensions per optional variable, as shown in Section 2:

Definition 4 (The bi-avatar strategy). *Abstract state $M^\sharp = (N^\sharp, \mathcal{A})$ follows the* bi-avatar strategy *if and only if $\mathcal{A}$ maps each optional variable $\mathtt{y}$ to a pair of dimensions $\{\mathtt{y}^-, \mathtt{y}^+\}$, and is such that each "upper" avatar $\mathtt{y}^+$ (resp., "lower" avatar $\mathtt{y}^-$) carries only "upper bound constraints" (resp., "lower bound constraints").*

In other words, the bi-avatar strategy fully determines $\mathcal{A}$. In order to implement this strategy, we need to ensure that all abstract operators preserve $\mathcal{A}$, and the property of lower and upper avatars. We define such abstract operations in the next subsections. Interestingly, whenever an abstract state satisfies this strategy, and if we drop all constraints over an (upper or lower) avatar of $\mathtt{y}$, the concretization restricted to the dimensions other than that avatar do not change. This entails:

Theorem 1 (Independence property). *All abstract values following the bi-avatar strategy satisfy the independence property (Definition 3).*

To express the emptiness of an optional variable, we simply need to let its avatars carry a pair of constraints that would be unsatisfiable, if carried by a unique dimension, such as $1 \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq 0$.

Example 7. Let $\mathbb{X} = \{\mathtt{x}\}, \mathbb{Y} = \{\mathtt{y}\}$, and let $\mathcal{A}$ specify the avatars defined by the bi-avatar strategy. Then, the following numerical abstract values specify the sets of concrete states below:

abstract numerical state $N^\sharp$	concretization $\gamma_{\mathbb{M}^\sharp}(N^\sharp, \mathcal{A})$
$1 \leq \mathtt{x} \wedge \mathtt{x} \leq 1 \wedge \mathtt{x} \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq \mathtt{x}$	$\{\mathtt{x} \mapsto 1, \mathtt{y} \mapsto 1\}, \{\mathtt{x} \mapsto 1, \mathtt{y} \mapsto \ominus\}$
$1 \leq \mathtt{x} \wedge \mathtt{x} \leq 1 \wedge \mathtt{x} \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq \mathtt{x} - 1$	$\{\mathtt{x} \mapsto 1, \mathtt{y} \mapsto \ominus\}$

Preservation. The abstract operators described in the remainder of this section either discard constraints violating the bi-avatar strategy (such as assignment, in Section 5.4), or never apply operations of $\mathbb{N}^\sharp$ that would cause them to bound a $\mathtt{y}^+$ (resp., $\mathtt{y}^-$) avatar below (resp., above). This implies straightforwardly that, in the resulting domain, all abstract elements with a non empty concretization follow the bi-avatar strategy (all $\mathtt{y}^-$ dimensions are not bounded by above and all $\mathtt{y}^+$ dimensions are not bounded by below). The only abstract operation that may input an abstract state with non empty concretization and return an abstract state with empty concretization is the abstract condition test $\mathbf{test}_{\mathbb{N}^\sharp}[.]$ (used in Section 5.2) and requires an output check that no constraint violates the bi-avatar strategy.

Expressiveness. Under the bi-avatar strategy, we can compare the expressiveness of Maya domain $\mathbb{M}^\sharp$ with that of its base domain $\mathbb{N}^\sharp$: if a set of stores S with no optional variable containing $\ominus$ can be described exactly by $N^\sharp \in \mathbb{N}^\sharp$, we can still describe S in $\mathbb{M}^\sharp$, up-to the change of any set of optional variable to $\ominus$. Indeed, if we let $\mathbb{S}_{\text{def}} = (\mathbb{X} \uplus \mathbb{Y}) \to \mathbb{V}$, we have:

Theorem 2. *If $\mathcal{A}$ follows the bi-avatar strategy, then:*

$$\forall N_0^\sharp \in \mathbb{N}^\sharp,\ \mathbf{Dim}(N_0^\sharp) = \mathbb{X} \uplus \mathbb{Y} \Longrightarrow \left(\exists N_1^\sharp \in \mathbb{M}^\sharp,\ \gamma_{\mathbb{N}^\sharp}(N_0^\sharp) = \gamma_{\mathbb{M}^\sharp}(N_1^\sharp, \mathcal{A}) \cap \mathbb{S}_{\text{def}}\right)$$

5.2 Condition test

The concrete semantics of a condition test `cond` filters out stores for which `cond` does not evaluate to `TRUE`. We assume $\mathbb{N}^\sharp$ provides a sound abstract function $\mathbf{test}_{\mathbb{N}^\sharp}[\mathtt{cond}] : \mathbb{N}^\sharp \to \mathbb{N}^\sharp$, and build an abstract operator $\mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{cond}] : M^\sharp \to M^\sharp$.

Optional variable emptiness test. To evaluate condition $\mathbf{test}_{\mathbb{M}^\sharp}[\mathbf{is\_empty}(\mathtt{y})]$, and filter out stores that do not map $\mathtt{y}$ into $\ominus$, we can simply add two constraints on $\mathtt{y}^-$ and $\mathtt{y}^+$ that would be unsatisfiable, if added on a same dimension, such as $1 \leq \mathtt{y}^-$ and $\mathtt{y}^+ \leq 0$. This can be done using $\mathbf{test}_{\mathbb{N}^\sharp}[.]$.

Numerical tests. We consider only conditions that are linear inequalities, as non-linear conditions are often handled by linearization techniques [14], and a linear equality is equivalent to a pair of inequalities.

Intuitively, $\mathbf{test}_{\mathbb{M}^\sharp}[.]$ should simply add a linear constraint to some abstract state $M^\sharp$ (with some approximation, as this constraint is in general not representable exactly in $\mathbb{N}^\sharp$). Given condition test $a_0\mathtt{x}_0 + \ldots + a_n\mathtt{x}_n + b_0\mathtt{y}_0 + \ldots + b_m\mathtt{y}_m \leq c$ (where $\mathtt{x}_i \in \mathbb{X}$ and $\mathtt{y}_i \in \mathbb{Y}$), we can produce another constraint that involves only standard variables and avatar dimensions by replacing $\mathtt{y}_i$ either by $\mathtt{y}_i^-$ or by $\mathtt{y}_i^+$ depending on the sign of b_i. This constraint is compatible with the bi-avatar strategy (Section 5.1), hence it can be represented precisely in the numerical domain, even if it indirectly entails emptiness of some optional variables (in other words, not using the bi-avatar property would cause a severe precision loss here). Thus, numerical condition test can be applied to this constraint. In turn, the absence of constraints violating the bi-avatar strategy needs to be verified on the output of $\mathbf{test}_{\mathbb{N}^\sharp}[.]$. Moreover, this constraint is equivalent to the initial constraint up-to the $\gamma_{\mathbb{M}^\sharp}$ concretization function. Thus, this principle defines a sound abstract transfer function for condition tests.

Theorem 3 (Soundness of condition test). *The abstract transfer function* $\mathbf{test}_{\mathbb{M}^\sharp}[.]$ *is* sound *in the sense that, for all linear inequality constraint* `cond` *and for all abstract state* $M^\sharp$ *satisfying the bi-avatar strategy:*

$$[\![\mathtt{cond}]\!](\gamma_{\mathbb{M}^\sharp}(M^\sharp)) \subseteq \gamma_{\mathbb{M}^\sharp}(\mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{cond}](M^\sharp))$$

Example 8. In this example, we assume that $\mathbb{N}^\sharp$ is the octagon domain, and that $\mathbb{X} = \{\mathtt{x}\}$, and $\mathbb{Y} = \{\mathtt{y}\}$ (thus, $\mathcal{A} : \mathtt{y} \mapsto \{\mathtt{y}^-, \mathtt{y}^+\}$). We consider an abstract pre-condition $M^\sharp = (N_0^\sharp, \mathcal{A})$, where $N_0^\sharp = (5 \leq \mathtt{x} \wedge \mathtt{x} \leq 5)$, and a condition test $\mathtt{y} - \mathtt{x} \leq 3$. Abstract test $\mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{y} - \mathtt{x} \leq 3](M^\sharp)$ first substitutes $\mathtt{y}^+$ for $\mathtt{y}$ in $(\mathtt{y} - \mathtt{x} \leq 3)$, which generates condition $\mathtt{y}^+ - \mathtt{x} \leq 3$. Then, it computes $\mathbf{test}_{\mathbb{N}^\sharp}[\mathtt{y}^+ - \mathtt{x} \leq 3](N_0^\sharp)$. Thus, we obtain the abstract post-condition $(N_1^\sharp, \mathcal{A})$ where $N_1^\sharp = (5 \leq \mathtt{x} \wedge \mathtt{x} \leq 5 \wedge \mathtt{y}^+ - \mathtt{x} \leq 3)$.

5.3 Verifying the satisfaction of a constraint

To verify assertions, we need an operator $\mathbf{sat}_{\mathbb{M}^\sharp}[\mathtt{cond}] : \mathbb{M}^\sharp \to \{\mathtt{TRUE}, \mathtt{FALSE}\}$ such that, if $\sigma \in \gamma_{\mathbb{M}^\sharp}(M^\sharp)$ and $\mathbf{sat}_{\mathbb{M}^\sharp}[\mathtt{cond}](M^\sharp) = \mathtt{TRUE}$, then $[\![\mathtt{cond}]\!](\sigma) = \mathtt{TRUE}$. The case of numerical assertions is very similar to the case of numeric tests.

To test whether $\mathtt{y}$ can store only $\ominus$ in any store described by $(N^\sharp, \mathcal{A})$, we simply need to check whether constraint $\mathtt{y}^- = \mathtt{y}^+$ is unsatisfiable. This suggests $\mathbf{sat}_{\mathbb{M}^\sharp}[\mathbf{is\_empty}(\mathtt{y})](N^\sharp, \mathcal{A}) = \mathbf{is\_bot}_{\mathbb{N}^\sharp}(\mathbf{test}_{\mathbb{N}^\sharp}[\mathtt{y}^- = \mathtt{y}^+](N^\sharp))$, where $\mathbf{is\_bot}_{\mathbb{N}^\sharp} : \mathbb{N}^\sharp \to \{\mathtt{TRUE}, \mathtt{FALSE}\}$ is a sound emptiness test (if $\mathbf{is\_bot}_{\mathbb{N}^\sharp}(N^\sharp) = \mathtt{TRUE}$, then $\gamma_{\mathbb{N}^\sharp}(N^\sharp) = \emptyset$).

5.4 Assignment

We now describe a transfer function $\mathbf{assign}_{\mathbb{M}^\sharp}$ that over-approximates the effect of an assignment. We consider assignments with a linear right hand side expression (non linear assignment can be implemented using linearization [14]).

Emptiness test. If the left-hand side $\mathtt{x}$ is a standard variable and optional variable $\mathtt{y}$ appears in the right hand side, the concrete semantics produces no output state when $\mathtt{y}$ takes no value. Therefore, given abstract pre-condition $M^\sharp$ and optional variable $\mathtt{y}$ appearing in the right hand side, if $\mathbf{sat}_{\mathbb{M}^\sharp}[\mathbf{is\_empty}(\mathtt{y})](M^\sharp)$ (Section 5.3), $\mathbf{assign}_{\mathbb{M}^\sharp}$ can safely return $\bot$. The computation of the abstract assignment starts with this check for all optional variables in the right hand side.

Numerical assignment. We first consider basic assignment $\mathtt{y} = \mathtt{y} + \mathtt{z}$, where $\mathbb{Y} = \{\mathtt{y}, \mathtt{z}\}$, in order to give some intuition. If $M^\sharp = (N^\sharp, \mathcal{A})$ is an abstract pre-condition and $\sigma \in \gamma_{\mathbb{M}^\sharp}(M^\sharp)$ is such that $\sigma(\mathtt{y}) \neq \ominus$ and $\sigma(\mathtt{z}) \neq \ominus$, there exists a valuation $\nu \in \gamma_{\mathbb{N}^\sharp}(N^\sharp)$ such that $\nu(\mathtt{y}^-) = \nu(\mathtt{y}^+) = \sigma(\mathtt{y})$ and the same for $\mathtt{z}$. After the assignment evaluates, we obtain a store σ' such that $\sigma'(\mathtt{y}) = \sigma(\mathtt{y}) + \sigma(\mathtt{z})$ (and is unchanged for all other variables). Therefore, we need to make sure that the abstract post-condition will describe a valuation ν' such that $\nu'(\mathtt{y}^-) = \nu(\mathtt{y}^+) = \sigma(\mathtt{y}) + \sigma(\mathtt{z})$. We can achieve that by performing a pair of assignments to $\mathtt{y}^-, \mathtt{y}^+$ using *any* combination of avatars to represent $\mathtt{y}, \mathtt{z}$ in the right hand side. For instance, the following choices are sound:

$$\begin{cases} \mathtt{y}^- = \mathtt{y}^- + \mathtt{z}^-; \\ \mathtt{y}^+ = \mathtt{y}^+ + \mathtt{z}^+; \end{cases} \qquad \begin{cases} \mathtt{y}^- = \mathtt{y}^- + \mathtt{z}^+; \\ \mathtt{y}^+ = \mathtt{y}^+ + \mathtt{z}^-; \end{cases}$$

Yet, not all choices are of optimal precision. To show this, we assume that the pre-condition bounds both $\mathtt{y}$ and $\mathtt{z}$ from the above, for example with octagon $N^\sharp = \{\mathtt{y}^+ \leq 0 \wedge \mathtt{z}^+ \leq 0\}$. Then, only the left choice will produce a precise upper bound on $\mathtt{y}^+$. However, this approach may also produce constraints that violate the bi-avatar strategy, such as $\mathtt{y}^+ - \mathtt{z}^+ \leq 0$, where $\mathtt{z}^+$ gets assigned a lower bound. Such a lower bound can be removed by adding a temporary dimension $\mathtt{t}$, assuming that it is positive (using $\mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{t} \geq 0]$), and performing assignment $\mathtt{z}^+ = \mathtt{z}^+ - \mathtt{t}$. To conclude, the analysis of assignment $\mathtt{y} = \sum_{i=0}^{n} a_i \mathtt{x}_i + \sum_{i=0}^{m} b_i \mathtt{y}_i + c$ proceeds as follows:

1. $\mathbf{assign}_{\mathbb{M}^\sharp}$ performs *in parallel* [10] the two assignments $\mathtt{y}^- = \mathtt{ex}^- \,||\, \mathtt{y}^+ = \mathtt{ex}^+$, where $\mathtt{ex}^-, \mathtt{ex}^+$ are obtained from the assignment right hand by substituting $\mathtt{y}_i$ with $\mathtt{y}_i^-$ or $\mathtt{y}_i^+$ depending on the sign of the b_is (see below);
2. then it forces the removal of constraints violating the bi-avatar property, using the aforementioned method.

Expression $\mathtt{ex}^+$ is defined as $\sum_{i=0}^{n} a_i \mathtt{x}_i + \sum_{i=0}^{m} b_i \mathtt{y}_i^{\epsilon_i} + c$ where avatar signs are determined as follows ($\mathtt{ex}^-$ uses the opposite avatar dimensions as $\mathtt{ex}^+$):

- if the assignment is not invertible ($\mathtt{y}$ does not appear in the right hand side), then ϵ_i is the sign of b_i;
- if the assignment is invertible and $\mathtt{y}$ is $\mathtt{y}_0$, then ϵ_i is the sign of the product $b_0 b_i$.

Finally, an assignment with a standard variable $\mathtt{x}$ as a left hand side can be handled in a similar manner (after the emptiness test described earlier): it will boil down to the introduction of a temporary dimension $\mathtt{x}'$, the analysis of two assignments $\mathtt{x} = \mathtt{ex}^+$ and $\mathtt{x}' = \mathtt{ex}^-$ with the above notations, the application of $\mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{x} = \mathtt{x}']$, and finally the removal of $\mathtt{x}'$. By contrast, doing a single assignment would possibly cause relations between $\mathtt{x}$ and avatars be discarded.

The resulting abstract operator is sound in the following sense:

Theorem 4 (Soundness). *If* $\mathtt{t} \in \mathbb{X} \uplus \mathbb{Y}$ *and* $\mathtt{ex}$ *is a linear expression, then:*

$$\forall M^\sharp \in \mathbb{M}^\sharp,\ [\![\mathtt{t} = \mathtt{ex}]\!](\gamma_{\mathbb{M}^\sharp}(M^\sharp)) \subseteq \gamma_{\mathbb{M}^\sharp}(\mathbf{assign}_{\mathbb{M}^\sharp}(\mathtt{t}, \mathtt{ex}, M^\sharp))$$

Example 9. We assume $\mathbb{X} = \{\mathtt{x}\}$, $\mathbb{Y} = \{\mathtt{y}, \mathtt{z}\}$ and consider the abstract pre-condition defined by octagon $N^\sharp = \{0 \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq 10 \wedge 0 \leq \mathtt{z}^- \wedge \mathtt{z}^+ \leq 1 + \mathtt{x}\}$.

- non invertible assignment $\mathtt{y} = 1 - \mathtt{z}$ boils down to parallel assignments $\mathtt{y}^+ = 1 - \mathtt{z}^- \,||\, \mathtt{y}^- = 1 - \mathtt{z}^+$ in Octagons [13] and produces numerical post-condition $\{-\mathtt{x} \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq 1 \wedge 0 \leq \mathtt{z}^- \wedge \mathtt{z}^+ \leq 1 + \mathtt{x}\}$;
- invertible assignment $\mathtt{y} = \mathtt{y} + \mathtt{z}$ boils down to parallel assignments $\mathtt{y}^+ = \mathtt{y}^+ + \mathtt{z}^+ \,||\, \mathtt{y}^- = \mathtt{y}^- + \mathtt{z}^-$, and produces numerical post-condition $\{0 \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq 11 + \mathtt{x} \wedge 0 \leq \mathtt{z}^- \wedge \mathtt{z}^+ \leq 1 + \mathtt{x}\}$.

5.5 Inclusion checking, join and widening

To analyze condition tests and loops, we also need abstract operations for join, widening and inclusion test. Using the bi-avatar strategy, these operations can be implemented in a straightforward manner, using the operations of the underlying domain, since avatars are the same for all abstract values. We write $\mathcal{A}$ for the set of avatars defined by the bi-avatar strategy in $\mathbb{X} \uplus \mathbb{Y}$. We let $\mathbf{is\_le}_{\mathbb{N}^\sharp}$, $\mathbf{join}_{\mathbb{N}^\sharp}$, $\mathbf{widen}_{\mathbb{N}^\sharp}$ denote the abstract inclusion check, abstract join and abstract widening of abstract domain $\mathbb{N}^\sharp$, satisfying the following soundness conditions:

$$\begin{array}{l}
\forall N_0^\sharp, N_1^\sharp \in \mathbb{N}^\sharp,\ \mathbf{is_le}_{\mathbb{N}^\sharp}(N_0^\sharp, N_1^\sharp) = \mathtt{TRUE} \Longrightarrow \gamma_{\mathbb{N}^\sharp}(N_0^\sharp) \subseteq \gamma_{\mathbb{N}^\sharp}(N_1^\sharp) \\
\forall N_0^\sharp, N_1^\sharp \in \mathbb{N}^\sharp,\ \gamma_{\mathbb{N}^\sharp}(N_0^\sharp) \cup \gamma_{\mathbb{N}^\sharp}(N_1^\sharp) \subseteq \gamma_{\mathbb{N}^\sharp}(\mathbf{join}_{\mathbb{N}^\sharp}(N_0^\sharp, N_1^\sharp)) \\
\forall N_0^\sharp, N_1^\sharp \in \mathbb{N}^\sharp,\ \gamma_{\mathbb{N}^\sharp}(N_0^\sharp) \cup \gamma_{\mathbb{N}^\sharp}(N_1^\sharp) \subseteq \gamma_{\mathbb{N}^\sharp}(\mathbf{widen}_{\mathbb{N}^\sharp}(N_0^\sharp, N_1^\sharp))
\end{array}$$

Furthermore, we assume that $\mathbf{widen}_{\mathbb{N}^\sharp}$ ensures convergence of any sequence of abstract iterates [3].

Definition 5 (Inclusion checking, join and widening). *We let the operators over* $\mathbb{M}^\sharp$ *be defined by:*

$$\begin{array}{rl}
\mathbf{is_le}_{\mathbb{M}^\sharp}((N_0^\sharp, \mathcal{A}), (N_1^\sharp, \mathcal{A})) &= \mathbf{is_le}_{\mathbb{N}^\sharp}(N_0^\sharp, N_1^\sharp) \\
\mathbf{join}_{\mathbb{M}^\sharp}((N_0^\sharp, \mathcal{A}), (N_1^\sharp, \mathcal{A})) &= (\mathbf{join}_{\mathbb{N}^\sharp}(N_0^\sharp, N_1^\sharp), \mathcal{A}) \\
\mathbf{widen}_{\mathbb{M}^\sharp}((N_0^\sharp, \mathcal{A}), (N_1^\sharp, \mathcal{A})) &= (\mathbf{widen}_{\mathbb{N}^\sharp}(N_0^\sharp, N_1^\sharp), \mathcal{A})
\end{array}$$

$$\begin{array}{rl}
[\![\mathbf{skip}]\!]^\sharp(M^\sharp) = M^\sharp & [\![\mathtt{z} = \mathtt{ex}]\!]^\sharp(M^\sharp) = \mathbf{assign}_{\mathbb{M}^\sharp}(M^\sharp, \mathtt{z}, \mathtt{ex}) \\
[\![\mathtt{s}_0; \mathtt{s}_1]\!]^\sharp(M^\sharp) = [\![\mathtt{s}_1]\!]^\sharp \circ [\![\mathtt{s}_0]\!]^\sharp(M^\sharp) & [\![\mathbf{assume}(\mathtt{cond})]\!]^\sharp(M^\sharp) = \mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{cond}](M^\sharp) \\
[\![\mathbf{if}(\mathtt{cond})\{\mathtt{s}_0\}\mathbf{else}\{\mathtt{s}_1\}]\!]^\sharp(M^\sharp) = \mathbf{join}_{\mathbb{M}^\sharp}([\![\mathtt{s}_0]\!]^\sharp \circ \mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{cond}](M^\sharp), & \\
\qquad [\![\mathtt{s}_1]\!]^\sharp \circ \mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{cond} == \mathtt{FALSE}](M^\sharp)) & \\
[\![\mathbf{while}(\mathtt{cond})\{\mathtt{s}\}]\!]^\sharp(M^\sharp) = \mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{cond} == \mathtt{FALSE}](\mathbf{lfp}^\#_{M^\sharp} F^\#) & \\
\qquad \text{where} \quad F^\# : M^\sharp \mapsto [\![\mathtt{s}]\!]^\sharp(\mathbf{test}_{\mathbb{M}^\sharp}[\mathtt{cond}](M^\sharp)) &
\end{array}$$

Fig. 5. Abstract semantics

These operators trivially inherit the properties of the operators of $\mathbb{N}^\sharp$:

Theorem 5. *Operations* $\mathbf{is\_le}_{\mathbb{M}^\sharp}$, $\mathbf{join}_{\mathbb{M}^\sharp}$ *and* $\mathbf{widen}_{\mathbb{M}^\sharp}$ *satisfy soundness condition of the same form as their underlying counterpart. In particular:*

$$\forall N_0^\sharp, N_1^\sharp \in \mathbb{N}^\sharp,\ \gamma_{\mathbb{N}^\sharp}(N_0^\sharp) \cup \gamma_{\mathbb{N}^\sharp}(N_1^\sharp) \subseteq \gamma_{\mathbb{N}^\sharp}(\mathbf{join}_{\mathbb{N}^\sharp}(N_0^\sharp, N_1^\sharp))$$

Moreover, $\mathbf{widen}_{\mathbb{M}^\sharp}$ *also ensures termination.*

5.6 Analysis

We now propose a static analysis for the language of Section 3. We define the abstract semantics of programs in Figure 5. It uses the abstract operators defined in the previous subsections and an abstract least fixpoint operator $\mathbf{lfp}^\#$, which performs abstract iterations with widening $\mathbf{widen}_{\mathbb{M}^\sharp}$ until convergence can be checked using abstract inclusion test $\mathbf{is\_le}_{\mathbb{M}^\sharp}$ [3]. Operator $\mathbf{lfp}^\#$ ensures that, when $F : \mathcal{P}(\mathbb{S}) \to \mathcal{P}(\mathbb{S})$ is continuous and $F^\# : \mathbb{M}^\sharp \to \mathbb{M}^\sharp$ satisfies $F \circ \gamma_{\mathbb{M}^\sharp} \subseteq \gamma_{\mathbb{M}^\sharp} \circ F^\#$, then $\mathbf{lfp}_{\gamma_{\mathbb{M}^\sharp}(M^\sharp)} \subseteq \gamma_{\mathbb{M}^\sharp}(\mathbf{lfp}^\#_{M^\sharp} F^\#)$. The analysis of statement $\mathbf{assert}(\mathtt{cond})$ (not shown in the figure) simply reports failure to prove the assertion $\mathtt{cond}$ if $\mathbf{sat}_{\mathbb{M}^\sharp}[\mathtt{cond}]$ does not return $\mathtt{TRUE}$.

The abstract semantics $[\![\mathtt{s}]\!]^\sharp : \mathbb{M}^\sharp \to \mathbb{M}^\sharp$ takes an abstract pre-condition and returns an abstract post-condition. We can prove by induction over the syntax of programs that this abstract semantics is sound:

Theorem 6 (Soundness). *Given a program* $\mathtt{s}$ *and an abstract pre-condition* $M^\sharp$*, the post-condition derived by the analysis is sound:*

$$[\![\mathtt{s}]\!](\gamma_{\mathbb{M}^\sharp}(M^\sharp)) \subseteq \gamma_{\mathbb{M}^\sharp}([\![\mathtt{s}]\!]^\sharp(M^\sharp))$$

6 Possibly empty summary variables

While Sections 3 to 5 studied the abstraction of stores where optional variables may contain either *one* value or $\ominus$, the array analysis shown in Section 2 makes use of *summary dimensions*, which may take no value, one value, or many values. We extend the analysis shown in the previous sections to handle such cases. The construction of Gopan et *al* [8] handles non empty summary dimensions (the main

feature of this abstraction is to perform weak updates when writing into a summary dimension). We apply the same technique to the Maya functor, and call the resulting abstract domain functor Maya+, that lifts an abstraction of numerical vectors into an abstraction of sets of vectors of *sets* of numerical values. As the extension is fairly straightforward, we present only its novel characteristics.

Concrete states and abstraction. First, we extend the language of Section 3. We now let $\mathbb{Y}$ denote summary dimensions, that may take zero, one or several values. The concrete states are now defined by:

$$\sigma_+ \in \mathbb{S}_+ \stackrel{\text{def.}}{::=} (\mathbb{X} \to \mathbb{V}) \uplus (\mathbb{Y} \to \mathcal{P}(\mathbb{V}))$$

An abstract state $M^\sharp_+$ in the resulting Maya+ domain is a tuple composed of a numeric abstract value $N^\sharp$ and an avatar mapping function $\mathcal{A}$, as in the Maya domain. However, the concretization is different: Maya+ assumes a concretization function γ_+ of a numeric domain with summarized dimensions, which returns sets of valuations ν_+ which map each avatar dimensions into a set of values.

Definition 6 (Concretization). *Given an abstract state $M^\sharp_+ = (N^\sharp, \mathcal{A})$, we define the following consistency predicates:*

$$\begin{array}{lcl} P_{\mathbb{X}}(\sigma_+, M^\sharp_+, \nu_+) & \stackrel{def.}{\iff} & \forall \mathtt{x} \in \mathbb{X},\ \sigma_+(\mathtt{x}) = \nu_+(\mathtt{x}) \\ P_{\mathbb{Y}}(\sigma_+, M^\sharp_+, \nu_+) & \stackrel{def.}{\iff} & \forall \mathtt{y} \in \mathbb{Y},\ \sigma_+(\mathtt{y}) \subseteq \bigcap_{\mathtt{d} \in \mathcal{A}(\mathtt{y})} \nu_+(\mathtt{d}) \end{array}$$

Then, the concretization of $M^\sharp_+ = (N^\sharp, \mathcal{A})$ is defined by:

$$\gamma_{\mathbb{M}^\sharp_+}(M^\sharp_+) \stackrel{def.}{::=} \left\{ \sigma_+ \in \mathbb{S} \mid \exists \nu_+ \in \gamma_+(N^\sharp),\ P_{\mathbb{X}}(\sigma_+, M^\sharp_+, \nu_+) \wedge P_{\mathbb{Y}}(\sigma_+, M^\sharp_+, \nu_+) \right\}$$

Example 10 (Concretization of Maya+). We assume $\mathbb{X} = \{\mathtt{x}\}$, $\mathbb{Y} = \{\mathtt{y}\}$, and consider the abstract element $(3 \leq \mathtt{x} \wedge \mathtt{x} \leq 4 \wedge 0 \leq \mathtt{y}^- \wedge \mathtt{y}^+ \leq \mathtt{x} - 3, \mathcal{A})$ where $\mathcal{A}$ follows the bi-avatar strategy. These constraints define valid elements of both Maya and Maya+ domains. However, the concretizations of this abstract element in both domains are different as shown below:

$$\begin{array}{lllll} Maya: & ① & \mathtt{x} \mapsto 3 & \mathtt{y} \mapsto 0 & \\ & ② & \mathtt{x} \mapsto 3 & \mathtt{y} \mapsto \ominus & \\ & ③ & \mathtt{x} \mapsto 4 & \mathtt{y} \mapsto 1 & \\ & ④ & \mathtt{x} \mapsto 4 & \mathtt{y} \mapsto 0 & \\ & ⑤ & \mathtt{x} \mapsto 4 & \mathtt{y} \mapsto \ominus & \end{array} \qquad \begin{array}{lllll} Maya+: & ① & \mathtt{x} \mapsto 3 & \mathtt{y} \mapsto \{0\} & \\ & ② & \mathtt{x} \mapsto 3 & \mathtt{y} \mapsto \emptyset & \\ & ③ & \mathtt{x} \mapsto 4 & \mathtt{y} \mapsto \{1\} & \\ & ④ & \mathtt{x} \mapsto 4 & \mathtt{y} \mapsto \{0\} & \\ & ⑤ & \mathtt{x} \mapsto 4 & \mathtt{y} \mapsto \emptyset & \\ & ⑥ & \mathtt{x} \mapsto 4 & \mathtt{y} \mapsto \{0,1\} & \end{array}$$

Concrete Semantics. The extension of the concrete semantics is mostly straightforward. We describe its salient aspects below.

– The semantics of arithmetic expressions $[\![\mathtt{ex}]\!] : \mathbb{S}_+ \to \mathcal{P}(\mathbb{V})$ evaluates each expression into a set of values.

$$[\![\mathtt{c}]\!](\sigma_+) \qquad \forall \mathtt{x} \in \mathbb{X},\ [\![\mathtt{x}]\!](\sigma_+) = \{\sigma_+(\mathtt{x})\} \qquad \forall \mathtt{y} \in \mathbb{Y},\ [\![\mathtt{y}]\!](\sigma_+) = \sigma_+(\mathtt{y})$$

$$[\![\mathtt{ex}_0 \oplus \mathtt{ex}_1]\!](\sigma_+) = \begin{cases} \emptyset, \text{ if } \exists i,\ [\![\mathtt{ex}_i]\!](\sigma_+) = \emptyset \\ \{\mathtt{c}_0 \oplus \mathtt{c}_1 \mid \forall i,\ \mathtt{c}_i \in [\![\mathtt{ex}_i]\!](\sigma_+)\}, \text{ otherwise} \end{cases}$$

– Since the operands of a logical operator are sets of values, the evaluations of logical expressions may also return a set of booleans, thus we can define two semantics for conditions, that filter states where the condition must (resp., may) evaluate to true.
(1) Given $\mathtt{S}_+ \subseteq \mathbb{S}_+$, the strong condition semantics $[\![\mathtt{cond}]\!]_{\mathrm{s}}(\mathtt{S}_+) : \mathcal{P}(\mathbb{S}_+) \to \mathcal{P}(\mathbb{S}_+)$ narrows $\mathtt{S}_+$ to stores that always make `cond` evaluate to `TRUE`:

$$[\![\mathtt{ex}_0 \otimes \mathtt{ex}_1]\!]_{\mathrm{s}}(\mathtt{S}_+) = \{\sigma_+ \in \mathtt{S}_+ \mid (\forall \mathtt{c}_i \in [\![\mathtt{ex}_i]\!](\sigma_+), i \in \{0,1\}\ \mathtt{c}_0 \otimes \mathtt{c}_1 = \mathtt{TRUE}) \\ \vee [\![\mathtt{ex}_0]\!](\sigma_+) = \emptyset \vee [\![\mathtt{ex}_1]\!](\sigma_+) = \emptyset\}$$

(2) the weak condition semantics $[\![\mathtt{cond}]\!]_{\mathrm{w}}(\mathtt{S}_+)$ narrows $\mathtt{S}_+$ to stores that may make `cond` evaluate to `TRUE`:

$$[\![\mathtt{ex}_0 \otimes \mathtt{ex}_1]\!]_{\mathrm{w}}(\mathtt{S}_+) = \{\sigma_+ \in \mathtt{S}_+ \mid (\exists \mathtt{c}_i \in [\![\mathtt{ex}_i]\!](\sigma_+), i \in \{0,1\}\ \mathtt{c}_0 \otimes \mathtt{c}_1 = \mathtt{TRUE}) \\ \vee [\![\mathtt{ex}_0]\!](\sigma_+) = \emptyset \vee [\![\mathtt{ex}_1]\!](\sigma_+) = \emptyset\}$$

– In assignment statement $[\![\mathtt{x} = \mathtt{ex}]\!] : \mathcal{P}(\mathbb{S}_+) \to \mathcal{P}(\mathbb{S}_+)$, the evaluation of the right hand side produces a set which may have several elements; in that case, we leave the choice of the new value non-deterministic. Optional variables are now *summaries*. Thus an assignment $\mathtt{y} = \mathtt{ex}$ to an optional variable results in a *weak update*.

$$\text{if } \mathtt{x} \in \mathbb{X},\ [\![\mathtt{x} = \mathtt{ex}]\!](\mathtt{S}_+) = \{\sigma_+[\mathtt{x} \mapsto \mathtt{c}] \mid \mathtt{c} \in [\![\mathtt{ex}]\!](\sigma_+)\} \\ \text{if } \mathtt{y} \in \mathbb{Y},\ [\![\mathtt{y} = \mathtt{ex}]\!](\mathtt{S}_+) = \{\sigma_+[\mathtt{y} \mapsto \sigma_+(\mathtt{y}) \cup [\![\mathtt{ex}]\!](\sigma_+)] \mid \sigma_+ \in \mathtt{S}_+\}$$

– We apply the strong semantics of test in the semantics of **assume** statements and the weak one in semantics of **if** and **while** statements.

$$\begin{aligned} [\![\mathbf{assume}((\mathtt{cond})_s)]\!](\mathtt{S}_+) &= [\![\mathtt{cond}]\!]_{\mathrm{s}}(\mathtt{S}_+) \\ [\![\mathbf{if}(\mathtt{cond})\{\mathtt{s}_0\}\mathbf{else}\{\mathtt{s}_1\}]\!](\mathtt{S}_+) &= [\![\mathtt{s}_0]\!] \circ [\![\mathtt{cond}]\!]_{\mathrm{w}}(\mathtt{S}_+) \\ &\quad \cup [\![\mathtt{s}_1]\!] \circ [\![\mathtt{cond} == \mathtt{FALSE}]\!]_{\mathrm{w}}(\mathtt{S}_+) \\ [\![\mathbf{while}(\mathtt{cond})\{\mathtt{s}\}]\!](\mathtt{S}_+) &= [\![\mathtt{cond} == \mathtt{FALSE}]\!]_{\mathrm{w}}(\mathtt{S}''_+) \\ &\quad \text{where } \mathtt{S}''_+ = \mathbf{lfp}\ \lambda \mathtt{S}'_+ \cdot \mathtt{S}_+ \cup [\![\mathtt{s}]\!]([\![\mathtt{cond}]\!]_{\mathrm{w}}(\mathtt{S}'_+)) \end{aligned}$$

Analysis. The abstract interpretation of this semantics is straightforward, as it simply combines the analysis in Section 5 and the classical technique for manipulating summarized dimensions [8]. It is worth noting that, while the abstract condition test described in Section 5.2 precisely over-approximates the strong semantics of tests, another abstract transfer function needs to be defined for the weak semantics. When applied to a condition that involves summaries, that function checks whether the condition *cannot* be satisfied (by applying $\mathbf{sat}_{\mathbb{M}^\sharp}[.]$ with the opposite condition), and returns $\bot$ if that is the case; otherwise, it leaves the abstract state unchanged.

Theorem 7 (Soundness). *We let $[\![.]\!]^\#$ represent the abstract semantics. Given a program* `s` *and an abstract pre-condition $M^\sharp_+$, the post-condition derived by the analysis is sound:*

$$[\![\mathtt{s}]\!](\gamma_{\mathbb{M}^\sharp_+}(M^\sharp_+)) \subseteq \gamma_{\mathbb{M}^\sharp_+}([\![\mathtt{s}]\!]^\#(M^\sharp_+))$$

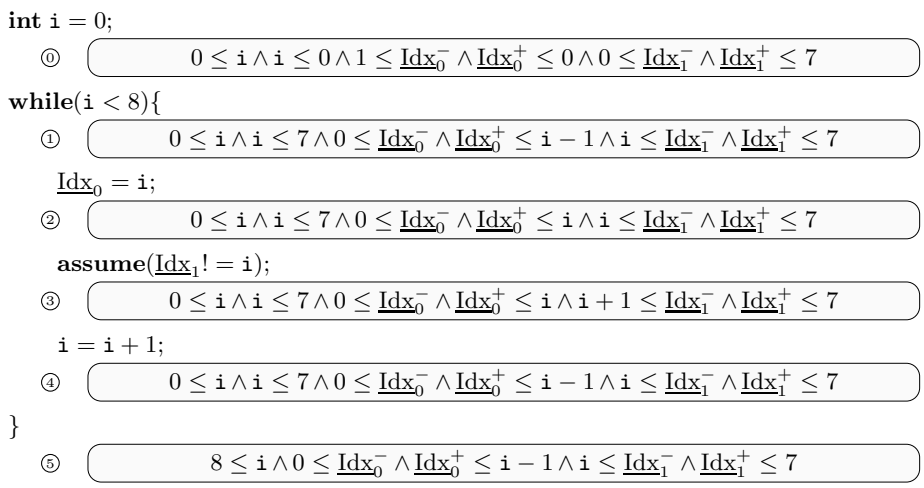


Fig. 6. Analysis of the array initialization example: invariants over group indexes

7 Implementation and examples

We have implemented abstract domain functors Maya and Maya+ with the bi-avatar strategy (so that they can be applied to numerical abstract domains representing linear inequalities), as well as the analysis of the language of Figure 3. To assess its precision, we have encoded into this language the computations over array indexes related to possibly empty groups encountered in [12] for a few basic array analyses. This approach allows to assess the optional value analysis, outside of the array analysis. We discuss in details the analysis of the array initialization example shown in Section 2. In this analysis $\mathbb{Y} = \{\underline{\mathrm{Idx}}_0, \underline{\mathrm{Idx}}_1\}$, ($\underline{\mathrm{Idx}}_i$ over-approximates the set of indexes of cells in group G_i), and $\mathcal{A}$ is defined according to the bi-avatar strategy ($\mathcal{A}(\underline{\mathrm{Idx}}_i) = \{\underline{\mathrm{Idx}}_i^-, \underline{\mathrm{Idx}}_i^+\}$ —note these are all *summary* dimensions, since a group of cells may span several indexes). The resulting invariants are shown in Figure 6. At point ⓪, group G_1 contains all the elements of the array (uninitialized elements) and G_0 is empty (initialized elements). The weak update $\underline{\mathrm{Idx}}_0 = \mathtt{i}$ and statement **assume**($\underline{\mathrm{Idx}}_1 ! = \mathtt{i}$) stem from the assignment $\mathtt{a}[\mathtt{i}] = 0$ in the array program (Figure 2(a)). They are analyzed by $\mathbf{assign}_{\mathbb{M}^\sharp}$ and $\mathbf{test}_{\mathbb{M}^\sharp}[.]$, and effectively extend group G_0 and shrink group G_1 by one cell. The loop exit invariant shown at point ⑤ defines stores where $\underline{\mathrm{Idx}}_1$ is mapped to no value, which indeed means that the group of uninitialized cells is empty.

The analysis was run on a few similar programs encoding the steps that [12] needs to achieve to verify array programs, and the results are shown in Figure 7. The columns show numbers of lines of codes, standard variables, summary optional variables, runtime, total numbers of assertions and numbers of verified assertions. Test case "array-init" is what we show in Fig 6. Test cases "array-random-access", "array-traverse" and "array-compare" simulate the array analysis on programs of

Program	LOCs	#Standard	#Optional	Time (ms)	#Assertions	#Verified
array-init	9	1	2	4.7	1	1
array-random-access	30	3	6	36.5	3	3
array-traverse	6	1	1	6.6	1	1
array-compare	10	3	2	14.1	1	1

Fig. 7. Analysis results

corresponding algorithm. The analyses are performed with Polyhedra as underlying domain. Runtimes are comparable to those observed in [12] for the numerical domain part. All invariants needed for the verification of array constraints are also verified. Last, the invariants produced express relations between groups, even when those could be empty.

8 Related works

Numerical abstract domains [11,3,13,5,1,7] describe constraints over sets of vectors, where each dimension is mapped to *one* value. Our work aims at extending such domains so as to abstract vectors of possibly empty sets of scalars.

Abstractions based on summary dimensions [8,17] extend basic numerical domains to abstract vectors of non empty sets, so that one dimension may describe an unbounded family of variables. Summaries are also used in shape analysis [15], with a similar semantics. Empty summaries can be dealt with using disjunctions.

Siegel and Simon [16] abstract dynamic stores, where the set of memory cells is dynamic, and also utilize summary dimensions. In this work, a summary variable may also denote an empty set of values. To abstract precisely which dimension may be empty, a flag is associated to each summary variable, and it is true if and only if the variable is defined to at least one value. This approach allows to express relations between the emptiness of distinct variables. However, it does not allow to infer that a variable is undefined from conflicting constraints over its value (as needed in, e.g., [12]). This approach is thus orthogonal to ours, and both techniques could actually be combined. Another technique [6,12] uses a conjunction of numerical abstract elements $N_0^\sharp, \ldots, N_p^\sharp$ such that a group of variables that should either all be empty or all be defined are constrained together in a same $N_i^\sharp$. While this approach tracks emptiness precisely and without disjunctions, it is fairly *ad hoc* and expresses no relational constraints across groups.

Last, we note that other works on numerical abstract domains use several dimensions in the abstract domain so as to constrain a single variable. For instance, the implementation of octagons on top of DBMs lets a variable $\mathtt{x}$ be described in a DBM by dimensions $\mathtt{x}^+ = \mathtt{x}$ and $\mathtt{x}^- = -\mathtt{x}$ (so that $\mathtt{x} = \frac{1}{2}(\mathtt{x}^+ - \mathtt{x}^-)$) [13].

9 Conclusion

We have proposed the Maya functor to lift numerical abstract domains into abstractions for sets of stores where some variables may be undefined, and a functor Maya+ that performs the same task in presence of possibly empty summary dimensions. We have fully described the design of abstract operations using a "bi-avatar" strategy, that allows to cope with abstract domains based on linear inequalities. Our construction can be applied either to analyze languages that allow optional values, or as a back-end for static analyses that rely on groups of locations to describe complex memories (such as array and shape analyses). Future work should focus on additional strategies, for instance, based on the multi-avatar strategy (Example 6), to accommodate other kinds of numerical abstract domains. Moreover, it will also be interesting to integrate our functors in array or shape analyses.

References

1. L. Chen, J. Liu, A. Miné, D. Kapur, and J. Wang. An abstract domain to infer octagonal constraints with absolute value. In *SAS*, 2014.
2. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *ENTS*, 1997.
3. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.
4. P. Cousot, R. Cousot, and F. Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. In *POPL*, 2011.
5. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, 1978.
6. A. Cox, B.-Y. E. Chang, and S. Sankaranarayanan. QUIC graphs: relational invariant generation for containers. In *VMCAI*, 2015.
7. K. Ghorbal, F. Ivancic, G. Balakrishnan, N. Maeda, and A. Gupta. Donut domains: Efficient non-convex domains for abstract interpretation. In *VMCAI*, 2012.
8. D. Gopan, F. DiMaio, N. Dor, T. Reps, and M. Sagiv. Numeric domains with summarized dimension. In *TACAS*, 2004.
9. N. Halbwachs and M. Péron. Discovering properties about arrays in simple programs. In *PLDI*, 2008.
10. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, 2009.
11. M. Karr. Affine relationships among the variables of a program. *Acta Informatica*, 1976.
12. J. Liu and X. Rival. Abstraction of arrays based on non contiguous partitions. In *VMCAI*, 2015.
13. A. Miné. The octagon abstract domain. In *HOSC*, 2006.
14. A. Miné. Relational domains for the detection of floating point run-time errors. In *ESOP*, 2004.
15. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *POPL*, 1999.
16. H. Siegel, B. Mihaila, and A. Simon. The undefined domain: precise relational information for entities that do not exist. In *APLAS*, 2013.
17. H. Siegel and A. Simon. Summarized dimensions revisited. *NSAD*, 2012.
18. H. Siegel and A. Simon. Fesa: Fold and expand-based shape analysis. In *CC*, 2013.