Invariant Synthesis for Programs Manipulating Lists with Unbounded Data

Ahmed Bouajjani, Cezara Dragoi, Constantin Enea, Ahmed Rezine, Mihaela Sighireanu

Invariant Synthesis for Programs Manipulating Lists with Unbounded Data ⋆

A. Bouajjani[1], C. Drăgoi[1], C. Enea[1], A. Rezine[2], and M. Sighireanu[1]
[1] LIAFA, University of Paris Diderot and CNRS, 75205 Paris 13, France,
{abou,cezarad,cenea,sighirea}@liafa.jussieu.fr
[2] Uppsala University, Sweden, rahmed@it.uu.se

Abstract. We address the issue of automatic invariant synthesis for sequential programs manipulating singly-linked lists carrying data over infinite data domains. We define for that a framework based on abstract interpretation which combines a specific finite-range abstraction on the shape of the heap with an abstract domain on sequences of data, considered as a parameter of the approach. We instantiate our framework by introducing different abstractions on data sequences allowing to reason about various aspects such as their sizes, the sums or the multisets of their elements, or relations on their data at different (linearly ordered or successive) positions. To express the latter relations we define a new domain whose elements correspond to an expressive class of first order universally quantified formulas. We have implemented our techniques in an efficient prototype tool and we have shown that our approach is powerful enough to generate non-trivial invariants for a significant class of programs.

1 Introduction

Invariant synthesis is an essential ingredient in various program verification and analysis methodologies. In this paper, we address this issue for sequential programs manipulating singly-linked lists carrying data over infinite data domains such as integers or reals. Specifications of such programs typically involve constraints on various aspects such as the sizes of the lists, the multisets of their elements, as well as relations between data at their different positions, e.g., ordering constraints or even more complex arithmetical constraints on consecutive elements, or combining relations between the sizes, the sum of all elements, etc., of different lists.

Consider for instance the procedure `Dispatch3` given in Figure 1(b). It puts all the cells of the input list which have data larger than 3 to the list `grt`, and it puts all the other ones to the list `less`. Naturally, the specification of this procedure (at line 12) includes (1) the property expressed by the universally-quantified first-order formula

$$\forall y.\ \texttt{grt} \xrightarrow{*} y \Rightarrow \texttt{data}(y) \geq 3 \quad \wedge \quad \forall y.\ \texttt{less} \xrightarrow{*} y \Rightarrow \texttt{data}(y) < 3 \qquad \text{(A)}$$

which say that all elements of `grt` (resp. `less`) have data larger (resp. smaller) than 3, and (2) the preservation property corresponding to the fact that the multiset of the input list is equal to the union of the multisets of the two output lists. This property is expressed by the equality

⋆ A full version is available at http://www.liafa.jussieu.fr/∼cezarad/inv.pdf

$$\texttt{ms\_init} = \texttt{ms}(\texttt{grt}) \cup \texttt{ms}(\texttt{less}) \tag{B}$$

where the variable `ms_init` represents the multiset of the elements of the input list, and $\texttt{ms}(\texttt{grt})$ (resp. $\texttt{ms}(\texttt{less})$) denotes the multiset of the elements of `grt` (resp. `less`).

```
   procedure Fibonacci(list* head)
1: { list *x=head;
2:   int m1=1;
3:   int m2=0;
4:   while (x != NULL)
5:   { x->data=m1+m2;
6:     m1=m2;
7:     m2=x->data;
8:     x=x->next;
9:   }
10:}
```

(a)

```
   procedure Dispatch3(list* head)
1: { list *tmp=null, grt=null, less=null;
2:   while (head != null)
3:   { tmp=head->next;
4:     if (head->data >= 3) {
5:       head->next=grt;
6:       grt=head; }
7:     else {
8:       head->next=less;
9:       less=head; }
10:    head=tmp;
11:  }
12:}
```

(b)

Fig. 1. Procedures `Fibonacci` and `Dispatch3`.

The specification of sorting algorithms is similar since it includes an ordering constraint on the output list that is easily expressible using a universally quantified first-order formula, and a preservation constraint saying that the input and output lists have the same elements that is expressible using multiset constraints.

Moreover, an interesting property of the procedure `Dispatch3` above is that the sum of all the elements in the list `grt` is larger than 3 times the size of that list. This is expressible by the inequality

$$\texttt{grt} \xrightarrow{+} \texttt{null} \wedge \sum_{\texttt{grt} \xrightarrow{*} y} \texttt{data}(y) - 3 \times \texttt{len}(\texttt{grt} \xrightarrow{+} \texttt{null}) \geq 0 \tag{C}$$

Consider now the procedure `Fibonacci` given in Figure 1(a). It takes a list as an input and initializes its elements following the Fibonacci sequence. The natural specification for the procedure (at line 10) is expressed by the universally-quantified formula

$$\forall y_1, y_2, y_3.\, \texttt{head} \xrightarrow{*} y_1 \rightarrow y_2 \rightarrow y3 \Rightarrow \texttt{data}(y_3) = \texttt{data}(y_2) + \texttt{data}(y_1) \tag{D}$$

which corresponds precisely to the definition of the Fibonacci sequence. Moreover, an interesting property of the Fibonacci sequence $\{f_i\}_{i\geq 1}$ is that $\sum_{i=0}^{i=n} f_i = 2f_n + f_{n-1} - 1$. This can be expressed (again at line 10) by the following constraint

$$\sum_{(\texttt{head} \xrightarrow{*} y)} \texttt{data}(y) = 2 \times \texttt{m2} + \texttt{m1} - 1 \tag{E}$$

The automatic synthesis of invariants like those shown above is a challenging problem since it requires combining in a nontrivial way different analysis techniques. This paper introduces a uniform framework based on abstract interpretation for tackling this problem. We define a generic abstract domain $\mathcal{A}_{\mathbb{HS}}$ for reasoning about dynamic lists with unbounded data which includes an abstraction on the shape of the heap and which is parametrized by some abstract domain on finite sequences of data (a data words abstract domain, $\mathbb{DW}$-domain for short). The latter is intended to abstract the sequences

of data in the lists by capturing relevant aspects such as their sizes, the sums or the multisets of their elements, or some class of constraints on their data at different (linearly ordered or successive) positions.

We instantiate our framework by defining new $\mathbb{DW}$-domains corresponding to the aspects mentioned above. In particular, we define new abstract domains for reasoning about the multisets of elements of lists, and about the sums of the elements of integer lists. Moreover, we introduce a $\mathbb{DW}$-domain where objects are composed of first-order formulas such that their (quantified) universal part is of the form $\forall \mathbf{y}.\ (P \Rightarrow U)$, where $\mathbf{y}$ is a vector of variables on the positions in the word, P is a constraint on the positions (seen as integers) associated with the $\mathbf{y}$'s, and U is a constraint on the data values at these positions, and possibly also on the positions when data are of numerical type. Then, we assume that our $\mathbb{DW}$-domain on first-order properties is parametrized by some abstract data domain, and we consider that U is defined as an object in that abstract domain. For the sake of simplicity of the presentation, we consider in the rest of the paper that the data are always of type integer (and therefore it is possible to take as abstract data domains the standard octagons or polyhedra abstract domains for instance). Our approach can in fact be applied to any other data domain. As for the syntax of the constraint P, we assume that we are given a finite set of fixed patterns (or templates) such as, for instance, order constraints or difference constraints.

Then, an object in the domain $\mathcal{A}_{\mathbb{HS}}$ is a finite collection of pairs $(\widetilde{G}, \widetilde{W})$ such that (1) $\widetilde{G}$ is a graph (where each node has an out-degree of at most 1) representing the set of all the garbage-free heap graphs that can be obtained by inserting sequences of non-shared nodes (nodes with in-degree 1) between any pair of nodes in $\widetilde{G}$ (thus edges in $\widetilde{G}$ represents list segments without sharing), and (2) $\widetilde{W}$ is an abstract object in the considered $\mathbb{DW}$-domain constraining the sequences of data attached to each edge in $\widetilde{G}$. So, all the shared nodes in the concrete heaps are present in $\widetilde{G}$, but $\widetilde{G}$ may have nodes which are not shared. Non-shared nodes which are not pointed by program variables are called simple nodes. We assume that objects in our abstract domain have graphs with a bounded number of simple nodes, for some given bound k that is also a parameter of the domain. This assumption implies that the number of such graphs is finite (since for a given program with lists it is well known that the number of shared nodes is bounded).

We define sound abstract transformers for the statements in the class of programs we consider. Due to the bound on the number of simple nodes, and since heap transformations may add simple nodes, we use a normalization operation that shrinks paths of simple nodes into a single edge. This operation is accompanied with an operation that generalizes the known relations on the data attached to the eliminated simple nodes in order to produce a constraint (in the $\mathbb{DW}$-domain) on the data word associated with the edge resulting from the normalization. This step is actually quite delicate and special care has to be taken in order to keep preciseness. In particular, this is the crucial step that allows to generate universally quantified properties from a number of relations between a finite (bounded) number of relations on the data attached to linearly ordered or successive simple nodes (depending on the allowed patterns for constraining the positions in the universal formulas). We have defined sufficient conditions on the sets of allowed patterns under which we show that we obtain best abstract transformers.

We have implemented (in C) a prototype tool CINV based on our approach, and we have carried out several experiments (more than 30 examples) on list manipulating programs (including for instance sorting algorithms such as insertion sort, and the two examples in Figure 1). The tool is powerful enough to synthesize nontrivial invariants such as all those mentioned above in this section. All the examples we have considered have been carried out in less that 1 sec, which is, we believe, quite encouraging.

2 Modeling and reasoning about programs with singly-linked lists

We consider a class of strongly typed imperative programs manipulating dynamic singly linked lists. We suppose that all manipulated lists have the same type, i.e., reference to a record called `list` including one reference field `next` and one data field `data` of integer type. While the generalization to records with several data fields is straightforward, the presence of a single reference field is important for this work. The programs we consider do not contain procedure calls or concurrency constructs.

Program syntax Programs are defined on a set of data variables *DVar* of type $\mathbb{Z}$ and a set of pointer variables *PVar* of type `list` (which includes the constant `null`). Data variables can be used in *data terms* built using operations over $\mathbb{Z}$ and in boolean conditions built using predicates over $\mathbb{Z}$. Pointers can be used in assignments corresponding to heap manipulation like memory allocation/deallocation (`new`/`free`), selector field updates (`p->next=...`, `p->data=...`), and pointer assignments (`p=...`). Boolean conditions on pointers are built using predicates (`p==q` and `p==null`) testing for equality and definedness of pointer variables. No arithmetics is allowed on pointers. We allow sequential composition (`;`), conditionals (`if-then-else`), and iterations (`while`). The full syntax is given in Figure 2.

$p, q \in PVar$ pointer variables — P predicate over $\mathbb{Z}$
$d \in DVar$ data variable — O operator over $\mathbb{Z}$

$$
\begin{aligned}
pt &::= \texttt{null} \mid p \mid p \rightarrow \texttt{next} \\
dt &::= d \mid p \rightarrow data \mid O(dt_1, \cdots, dt_n) \\
cond &::= p == q \mid p == \texttt{null} \mid P(dt_1, \cdots, dt_n) \mid \\
&\quad\ !cond \mid cond \wedge cond \\
asgnStmt &::= p = \texttt{new} \mid \texttt{free}(p) \mid p = pt \mid \\
&\quad\ p \rightarrow \texttt{next} = pt \mid p \rightarrow \texttt{data} = dt \mid d = dt \\
ifStmt &::= \texttt{if}\ cond\ \texttt{then}\ \{stmt\}^*\ [\texttt{else}\ \{stmt\}^*] \\
whileStmt &::= \texttt{while}\ cond\ \texttt{do}\ \{stmt\}^* \\
stmt &::= whileStmt \mid ifStmt \mid asgnStmt \\
program &::= \{stmt\}^*
\end{aligned}
$$

Fig. 2. Syntax for programs with singly linked lists.

For simplicity, we consider that all programs are precompiled as follows. Each pointer assignment of the form p=`new`, p=q or p=$q \rightarrow$ `next` is immediately preceded by an assignment of the form p=`null`. A pointer assignment of the form p=$p \rightarrow$ `next`

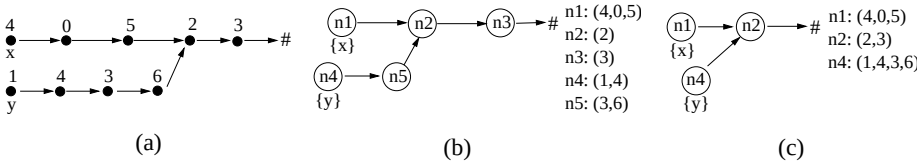
is turned into q=p, p=null, p=$q \rightarrow$ next, possibly introducing a fresh variable q. Each pointer assignment of the form $p \rightarrow$ next=q is immediately preceded by $p \rightarrow$ next=null.

Program semantics A configuration of a program is given by a configuration for the program heap and a valuation of data variables. Heaps can be represented naturally by a directed graph. Each object of type list is represented by a node. The constant null is represented by a distinguished node $\sharp$. The pointer field next is represented by edges. The nodes are labeled by the values of the data field data and by the program pointer variables which are pointing to the corresponding objects. Every node has exactly one successor, except for the node representing null. For example, the graph in Figure 3(a) represents a heap containing two lists $[4,0,5,2,3]$ and $[1,4,3,6,2,3]$ which share their two last cells. Two of the nodes are labeled by the pointer variables x and y. A node which is labeled by a pointer variable or which has at least two predecessors is called a *cut point*. Otherwise, it is called a *simple node*.

In this work, we use an equivalent representation for heaps obtained as follows. Let G be a graph as above. It can be encoded (1) by a graph H containing at least all the cut points in G such that two nodes are connected by an edge if there exists a path between them in G, and (2) by a function that associates to each node n in H a word over $\mathbb{Z}$ which represents the data values from G of the path starting in n and ending before the successor of n in H. For example, Figure 3(b) and Figure 3(c) give possible encodings for the graph in Figure 3(a).

Definition 1. *A* heap graph over *PVar* and *DVar is a tuple* $H = (N,S,V,L,D)$ *where:*
– N *is a finite set of nodes which contains a distinguished node* $\sharp$*,*
– $S : N \rightharpoonup N$ *is a successor partial function s.t. only* $S(\sharp)$ *is undefined,*
– $V : PVar \rightarrow N$ *is a function associating nodes to pointer variables s.t.* $V(\texttt{null}) = \sharp$*,*
– $L : N \rightharpoonup \mathbb{Z}^+$ *is a partial function associating nodes to non-empty words over* $\mathbb{Z}$ *s.t. only* $L(\sharp)$ *is undefined, and*
– $D : DVar \rightarrow \mathbb{Z}$ *is a valuation for the data variables.*

A heap graph H is called a *k-heap graph* if the number of simple nodes is at most k. Figure 3(b) pictures a 2-heap graph where $n3$ and $n5$ are the simple nodes.



(a) (b) (c)

Fig. 3. A representation for program heaps

In the following, we consider only heap graphs without garbage, i.e. all the nodes are reachable from nodes labeled by pointer variables. We define a postcondition operator, denoted $\texttt{post}(St,H)$, for any statement St and any k-heap graph H. In general $\texttt{post}(St,H)$ is not a k-heap graph; it may contain more than k simple nodes.

Some interesting cases in the definition of $\texttt{post}(St,H)$ are given in Figure 4. For the statement $p :=$ new we add to the heap graph a node n whose successor is $\sharp$ and

which is labeled by a singleton data word obtained using the procedure $\texttt{sglt}(L,n)$. The latter updates $L(n)$ by a random word of length 1..

If p points to the node representing `null` then $\texttt{post}(\texttt{free}(p),H)$ returns H_{err} which is a special sink heap configuration. (The same happens for other `null` dereferences like $p = q$, $p \rightarrow \texttt{next} = q$, and $q = p \rightarrow \texttt{next}$ when p points to $\sharp$.) Otherwise, $\texttt{post}(\texttt{free}(p),H)$ modifies H by removing the list element pointed by p. Remember that the node n in H pointed by p may represent more list elements at once. Thus, we start by calling the procedure $\texttt{Uncover}(H,p)$ described in the right of Figure 4. The goal of this procedure is to obtain a heap graph $H' = (N',S',V',L',D')$ representing the same heap as H in which the word associated to the node pointed by p is of length 1. It uses two procedures on words over $\mathbb{Z}$, `split` and `isSglt`. For any $n \in N$ such that the word $L(n)$ is of size at least 2, $\texttt{split}(L,n,m)$ returns a function L' obtained from L by assigning to n a singleton word containing the first symbol in $L(n)$ and by setting $L(m)$ to be the word containing all symbols in $L(n)$ except the first one. Also, $\texttt{isSglt}(L,n)$ returns 1 if the length of $L(n)$ is 1 and 0, otherwise. Then, $\texttt{post}(\texttt{free}(p),H)$ applies `RemNode` which removes from the current heap graph the node n pointed by p (for any predecessor m of n, $S(m)$ is set to $\sharp$), and `RemGrb` which models the garbage collector. For the latter, `post` calls a projection operator $\texttt{proj}(L,n)$ that removes n from the domain of L.

$$\frac{n \notin N \text{ is a fresh node}}{\texttt{post}(p := \texttt{new}, H) = (N \cup \{n\}, S[n \mapsto \sharp], V[p \mapsto n], \texttt{sglt}(L,n), D)}$$

$$\frac{V(p) \neq \sharp \quad (N',S',V',L',D') = \texttt{Uncover}(H,p)}{\texttt{post}(\texttt{free}(p),H) = \texttt{RemGrb}\,(\texttt{RemNode}\,((N',S',V',L',D'),p))}$$

$$\frac{V(p) \neq \sharp}{\texttt{post}(p.data = dt, H) = \texttt{UpdateData}(H,p,dt)}$$

```
procedure Uncover(H,p) {
  n = V(p);
  if(isSglt(L,n)) {
    N' = N; L' = L; S' = S; }
  else {
    N' = N ∪ {m}; // m ∉ N
    L' = split(L,n,m);
    S' = S[n ↦ m, m ↦ S(n)]; }
  return (N',S',V,L',D); }
```

Fig. 4. The postcondition operator `post`.

The definition of `post` for $p.data := dt$ calls the procedure $\texttt{UpdateData}(H,p,dt)$ which, for any input heap graph $H = (N,S,V,L,D)$ such that $V(p) = n$, returns the heap graph $(N,S,V,\texttt{updateFirst}((L,D),n,dt,V))$.

The procedure $\texttt{updateFirst}((L,D),n,\texttt{dt},V)$ substitutes the first symbol in the word associated to n by the value of `dt`. The mapping V is used to substitute in `dt` any term `p->data` by the first symbol of the word associated to $V(\texttt{p})$. The same procedure `updateFirst` is used for assignments of the form `d=dt`, where `d` is a data variable.

The postcondition operator `post` can be extended to obtain a postcondition operator on k-heap graphs, denoted $\texttt{post}_k$, by

$$\texttt{post}_k(St,H) = \texttt{Normalize}_k(\texttt{post}(St,H)),$$

where $\texttt{Normalize}_k$ takes as input a heap graph H and, if H is not a k-heap graph then it returns a 0-heap graph (a heap graph with no simple nodes) which represents the same

heap as H. Suppose that $V_1,\ldots,V_t$ are all the (disjoint) paths in H of length greater than 1 between a cut-point and a predecessor of some cut-point. $\texttt{Normalize}_k$ calls the procedure $\texttt{concat}(L,V_1,\ldots,V_t)$ which modifies the function L. For any $1 \leq i \leq t$, if $V_i = n_0 \ldots n_j$ then the domain of $L' = \texttt{concat}(L,V_1,\ldots,V_t)$ does not contain $n_1,\ldots,n_j$ and $L'(n_0)$ is the concatenation of $L(n_0),\ldots,L(n_j)$. The procedure $\texttt{Normalize}_k$ removes from the graph all the nodes removed from the domain of L. A collecting semantics can be defined as usual by extending $\texttt{post}_k$ to sets of k-heap graphs.

3 Abstract domain for program configurations

The elements of this abstract domain are finite sets of graphs with constraints on the data words attached to their nodes and on the program data variables. Constraints are expressed using abstract domains on words capturing various aspects such as the size of the words, the multiset of their elements, or other properties that relate data at different positions. For example, Figure 5(a) pictures a possible heap configuration at line 2 of the procedure `Dispatch3` from Figure 1(b). Different abstractions of it are defined using the same graph together with the constraints from Figure 5(b), Figure 5(c), and Figure 5(d). In the following, because all the data updates affect only the first symbol of the words, we refer separately to the head of a word (i.e. its first symbol) and its tail (i.e., the suffix that starts with the second symbol). The constraints from Figure 5(b) characterize the sum of the symbols of a word. They use variables representing words (which have the same name as the nodes of the graph) and terms interpreted as integers: $\texttt{hd}(n)$ denotes the first symbol in the word represented by n, $\texttt{len}(n)$ its length, and $\texttt{sum\_tl}(n)$ the sum of the symbols in its tail. The integer variable $\texttt{init\_sum}$ denotes the sum of the integers contained in the initial list. The constraints from Figure 5(c) characterize the multiset containing the symbols of a word. They use variables to represent words and terms interpreted as multisets: $\texttt{ms\_hd}(n)$ denotes the multiset containing the first symbol and $\texttt{ms\_tl}(n)$ denotes the multiset containing all symbols from the tail. The multiset variable $\texttt{init\_ms}$ denotes the multiset containing the integers from the initial list. Finally, we can define an abstraction using constraints expressed by universally quantified first-order formulas. In Figure 5(d), y is a variable interpreted as a position in some word, $y \in \texttt{tl}(n2)$ means that y belongs to the tail of the word denoted by $n2$, and $n2[y]$ is a term interpreted as the data at the position y of $n2$.

In the following, an abstract domain $\mathcal{A}$ is a tuple $(L,\sqsubseteq,\sqcap,\sqcup,\top,\bot)$, where $(L,\sqsubseteq)$ is a lattice whose greatest lower bound (meet) operator is $\sqcap$ and lowest greater bound (join) operator is $\sqcup$. The top element is denoted by $\top$ and the bottom element by $\bot$. The domain $\mathcal{A}$ represents a domain of concrete elements $\mathcal{C}$ by a Galois connection, that is, a pair of monotone functions $(\alpha : \mathcal{C} \to \mathcal{A}, \gamma : \mathcal{A} \to \mathcal{C})$ such that $\alpha(C) \sqsubseteq A$ iff $C \subseteq \gamma(A)$. As usual in the abstract interpretation framework [6], ∇ represents the widening operator.

3.1 Data words abstract domains

An abstract domain representing words is called a *data words abstract domain* ($\mathbb{DW}$-domain, for short). Let *DWVars* be a set of variables called data word variables and $\mathcal{C}(DWVars, DVar)$ be the lattice of sets of pairs (L,D) with $L : DWVars \to \mathbb{Z}^+$ and $D : DVar \to \mathbb{Z}$.

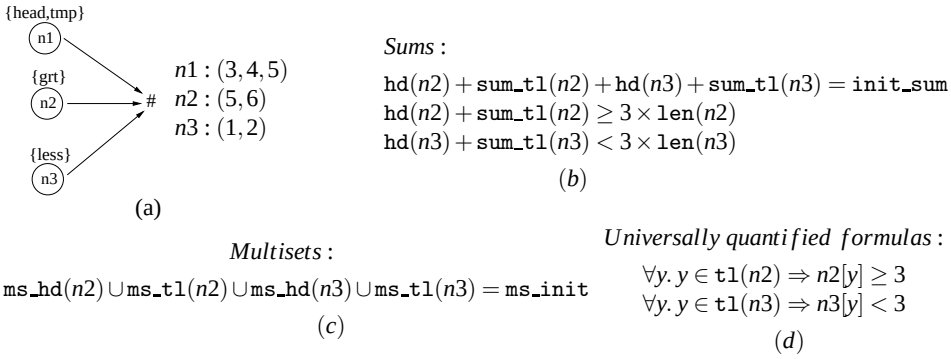


Fig. 5. Different abstractions for some configuration of `Dispatch3`

Definition 2. *An abstract domain* $\mathcal{A}_{\mathbb{W}} = \left(L^{\mathbb{W}}, \sqsubseteq^{\mathbb{W}}, \sqcap^{\mathbb{W}}, \sqcup^{\mathbb{W}}, \top^{\mathbb{W}}, \bot^{\mathbb{W}}\right)$ *is called a* $\mathbb{DW}$-domain *if there exists a Galois connection* $(\alpha^{\mathbb{W}}, \gamma^{\mathbb{W}})$ *from* $\mathcal{A}_{\mathbb{W}}$ *into* $\mathcal{C}(DWVars, DVar)$.

In the following, we give two $\mathbb{DW}$-domains which formalize the abstractions from Figure 5(b) and (c). The abstract domain used in Figure 5(d) will be defined in Section 4.

$\mathbb{DW}$-domain $\mathcal{A}_{\Sigma}$: To reason about the sum of data in a word, we define a $\mathbb{DW}$-domain based on an arbitrary numerical abstract domain whose dimensions represent integer program variables or terms of the form $\texttt{hd}(w)$, $\texttt{len}(w)$, and $\texttt{sum\_tl}(w)$, with $w \in DWVars$. In our experiments, we have used such a $\mathbb{DW}$-domain based on the polyhedra domain [8], denoted $\mathcal{A}_{\Sigma}$.

$\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{M}}$: To reason about multisets of data of a word, we consider the $\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{M}}$ whose elements are conjunctions of formulas of the form $T = T'$ with T and T' terms of the form $v_1 \cup v_2 \cup \cdots \cup v_p$, where $p \geq 2$. Here, $v_1, v_2, \ldots, v_p$ are variables interpreted as multisets or terms $\texttt{ms\_hd}(w)$, $\texttt{ms\_tl}(w)$, and $\texttt{ms}(d)$ with $w \in DWVars$ and $d \in DVar$. The term $\texttt{ms}(d)$ represents the singleton containing the value of the program variable d. We suppose that $v_1, v_2, \ldots, v_p$ are distinct.

Let $\top^{\mathbb{M}}$ be the formula $t = t$ and $\bot^{\mathbb{M}}$ the formula $t \cup t = t$. To define lattice operators we start by defining a procedure $\texttt{Saturate}(\mu)$, where μ is a value in $\mathcal{A}_{\mathbb{M}}$, which applies the commutativity of $=$ and $\cup$, the associativity of $\cup$, and substitutions in order to obtain new atomic formulas that are implied by the existing ones. The substitutions are applied as follows:

- if $T_1 = T_2$ is an atomic formula then we add more atomic formulas by substituting:
  - in every union term of the form $T_3 \cup T_1$, T_1 by T_2;
  - in every union term of the form $T_3 \cup T_2$, T_2 by T_1;

Let μ_1 and μ_2 be two elements in $L^{\mathbb{M}}$. We define $\mu_1 \sqsubseteq^{\mathbb{M}} \mu_2$ if for every atomic formula $T_1 = T_2$ in μ_2 there exists:

- union terms $T_1^1, T_1^2, \ldots, T_1^p$ such that $T_1 = T_1^1 \cup T_1^2 \cdots \cup T_1^p$,
- union terms $T_2^1, T_2^2, \ldots, T_2^p$ such that $T_2 = T_2^1 \cup T_2^2 \cdots \cup T_2^p$, and
- atomic formulas $T_1^i = T_2^i$, for any $1 \leq i \leq p$, in $\texttt{Saturate}(\mu_1)$.

We define $\mu_1 \sqcup^{\mathbb{M}} \mu_2$ to be the conjunction of atomic formulas that appear in both $\texttt{Saturate}(\mu_1)$ and $\texttt{Saturate}(\mu_2)$. The meet operator $\mu_1 \sqcap^{\mathbb{M}} \mu_2$ is defined to be the conjunction of atomic formulas that appear in $\texttt{Saturate}(\mu_1)$ or $\texttt{Saturate}(\mu_2)$. Since our abstract domain contains a finite number of elements, we may consider $\nabla^{\mathbb{M}} = \sqcup^{\mathbb{M}}$.

3.2 Abstract heap graphs

Abstractions of k-heap graphs as in Figure 5 are called *k-abstract heap graphs*. In the following definition, we assume that for each node of a heap graph there exists a data word variable with the same name.

Definition 3. *A* k-abstract heap over *PVar*, *DVar*, and a $\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{W}}$ *is a tuple* $\widetilde{H} = \left(N, S, V, \widetilde{W}\right)$ *where* N, S, V *are as in the definition of* k*-heap graphs, and* $\widetilde{W}$ *is an abstract value in* $\mathcal{A}_{\mathbb{W}}$ *over the data word variables* $N - \{\sharp\}$ *and the data variables DVar.*

Next, we define the domain of k-abstract heap graphs parametrized by $\mathcal{A}_{\mathbb{W}}$, denoted by $\mathcal{A}_{\mathbb{H}}(k, \mathcal{A}_{\mathbb{W}})$. Two abstract heap graphs are *isomorphic* when their underlying graphs are isomorphic. Formally, $\widetilde{H} = \left(N, S, V, \widetilde{W}\right)$ and $\widetilde{H'} = \left(N', S', V', \widetilde{W}'\right)$ are isomorphic, denoted $\widetilde{H} \sim \widetilde{H'}$, if there exists an isomorphism $h : N \to N'$ between the labeled graphs (N, S, V) and (N', S', V'). To emphasize the graph isomorphism, we may write $\widetilde{H} \sim_h \widetilde{H'}$.

The lattice operators of this domain are obtained by applying the corresponding operators from $\mathcal{A}_{\mathbb{W}}$ between abstract values which belong to isomorphic abstract heap graphs. Thus, for any $\widetilde{H} = \left(N, S, V, \widetilde{W}\right)$ and $\widetilde{H'} = \left(N', S', V', \widetilde{W}'\right)$ in $\mathcal{A}_{\mathbb{H}}(k, \mathcal{A}_{\mathbb{W}})$ such that $\widetilde{H} \sim_h \widetilde{H'}$ we have that (1) $\widetilde{H} \sqsubseteq^{\mathbb{H}} \widetilde{H'}$ if $\widetilde{W} \sqsubseteq^{\mathbb{W}} \widetilde{W}'[h(n) \mapsto n | n \in N]$ (we have unified the abstract values such that they use the same variable for isomorphic nodes) and (2) for any $\ddagger \in \{\sqcup, \sqcap, \nabla\}$, $\widetilde{H} \ddagger^{\mathbb{H}} \widetilde{H'}$ is the abstract heap $(N, S, V, \widetilde{W}'')$, where $\widetilde{W}''$ is the abstract value $\widetilde{W} \ddagger^{\mathbb{W}} \widetilde{W}'[h(n) \mapsto n | n \in N]$. The join and the widening (meet, resp.) of two non-isomorphic abstract heap graphs is $\top^{\mathbb{W}}$ ($\bot^{\mathbb{W}}$, resp.).

Proposition 1. *The entailment relation* $\sqsubseteq^{\mathbb{H}}$ *is* sound, *that is,* $\widetilde{H} \sqsubseteq^{\mathbb{H}} \widetilde{H'}$ *implies* $\gamma^{\mathbb{H}}(\widetilde{H}) \subseteq \gamma^{\mathbb{H}}(\widetilde{H'})$. *Also,* $\sqcup^{\mathbb{H}}$ *and* $\sqcap^{\mathbb{H}}$ *are the least upper bound and greatest lower bound, respectively, and* $\nabla^{\mathbb{H}}$ *is a widening operator.*

Proof: The properties of $\sqsubseteq^{\mathbb{H}}$, $\sqcup^{\mathbb{H}}$, and $\sqcap^{\mathbb{H}}$, resp. follow directly from similar properties of $\sqsubseteq^{\mathbb{W}}$, $\sqcup^{\mathbb{W}}$, and $\sqcap^{\mathbb{W}}$, resp. The proof for the widening operator relies on the fact that the heaps generated by the class of programs considered in Section 2 contain a bounded number of cut points [17]. □

Notice that $\sqsubseteq^{\mathbb{H}}$ is not *complete*. That is, we can find two abstract heaps $\widetilde{H}$ and $\widetilde{H'}$ such that $\gamma^{\mathbb{H}}(\widetilde{H}) \subseteq \gamma^{\mathbb{H}}(\widetilde{H'})$ does not imply $\widetilde{H} \sqsubseteq^{\mathbb{H}} \widetilde{H'}$.

Based on the Galois connection $(\alpha^{\mathbb{W}}, \gamma^{\mathbb{W}})$ we define the Galois connection $(\alpha^{\mathbb{H}}, \gamma^{\mathbb{H}})$ between the lattice of sets of k-heap graphs and $\mathcal{A}_{\mathbb{H}}(k, \mathcal{A}_{\mathbb{W}})$. The value of $\alpha^{\mathbb{H}}$ for a set of isomorphic k-heap graphs $\mathcal{S}$ is the underlying graph of the input heap graphs together with a value in $\mathcal{A}_{\mathbb{W}}$ obtained by applying $\alpha^{\mathbb{W}}$ to

$$\{(L, D) \mid (N, S, V, L, D) \in \mathcal{S}\}.$$

The value of $\alpha^{\mathbb{H}}$ for a set of k-heap graphs containing at least two non-isomorphic heap graphs is $\top^{\mathbb{H}}$. The concretization function $\gamma^{\mathbb{H}}$ is defined using $\gamma^{\mathbb{W}}$ in a similar manner.

3.3 Abstract heap sets

We define $\mathcal{A}_{\mathbb{HS}}(k,\mathcal{A}_{\mathbb{W}}) = \left(L^{\mathbb{HS}}(k,\mathcal{A}_{\mathbb{W}}), \sqsubseteq^{\mathbb{HS}}, \sqcap^{\mathbb{HS}}, \sqcup^{\mathbb{HS}}, \top^{\mathbb{HS}}, \bot^{\mathbb{HS}}\right)$ as a finite powerset domain corresponding to $\mathcal{A}_{\mathbb{H}}(k,\mathcal{A}_{\mathbb{W}})$. Its elements are called *k-abstract heap sets*.

Definition 4. *A k-abstract heap set over PVar, DVar, and a $\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{W}}$ is a finite set of non-isomorphic k-abstract heap graphs over PVar, DVar, and $\mathcal{A}_{\mathbb{W}}$.*

The operators associated to $\mathcal{A}_{\mathbb{HS}}(k,\mathcal{A}_{\mathbb{W}})$ and its widening operator are obtained from those of $\mathcal{A}_{\mathbb{H}}(k,\mathcal{A}_{\mathbb{W}})$ as usual [7]. The entailment relation $\sqsubseteq^{\mathbb{HS}}$ is the usual Hoare power-domain partial order [1], that is, for any A_H and A'_H in $\mathcal{A}_{\mathbb{HS}}(k,\mathcal{A}_{\mathbb{W}})$, $A_H \sqsubseteq^{\mathbb{HS}} A'_H$ iff for any $\widetilde{H} \in A_H$ there exists $\widetilde{H'} \in A'_H$ such that $\widetilde{H} \sqsubseteq^{\mathbb{H}} \widetilde{H'}$.

Let A_H and $A'_H \in L^{\mathbb{HS}}$. To make uniform the definitions of all operators of the abstract domain, we suppose that for any abstract heap $\left(N,S,V,\widetilde{W}\right) \in A_H$ for which we can not find an isomorphic abstract heap in A'_H we add to A'_H the abstract heap $(N,S,V,\bot^{\mathbb{W}})$ and vice-versa. Then, for any $\ddagger \in \{\sqcup,\sqcap,\nabla\}$ we define $A_H \ddagger^{\mathbb{HS}} A'_H$ to be the abstract heap set that contains for any two abstract heaps $\widetilde{H} = \left(N,S,V,\widetilde{W}\right) \in A_H$ and $\widetilde{H'} = \left(N',S',V',\widetilde{W'}\right) \in A'_H$ with $\widetilde{H} \sim_h \widetilde{H'}$, the abstract heap $(N,S,V,\widetilde{W''})$, where $\widetilde{W''}$ is the abstract value $\widetilde{W} \ddagger^{\mathbb{W}} \widetilde{W'}[h(n) \mapsto n | n \in N]$.

Proposition 2. *The entailment relation $\sqsubseteq^{\mathbb{HS}}$ is* sound*, that is, $A_H \sqsubseteq^{\mathbb{HS}} A'_H$ implies $\gamma^{\mathbb{HS}}(A_H) \subseteq \gamma^{\mathbb{HS}}(A'_H)$. Also, $\sqcup^{\mathbb{HS}}$ and $\sqcap^{\mathbb{HS}}$ are the least upper bound and greatest lower bound, respectively, and $\nabla^{\mathbb{HS}}$ is a widening operator.*

Proof: Directly from the properties of $\sqsubseteq^{\mathbb{H}}$, $\sqcup^{\mathbb{H}}$, $\sqcap^{\mathbb{H}}$, and $\nabla^{\mathbb{H}}$. □

As in the previous case, notice that $\sqsubseteq^{\mathbb{HS}}$ is not *complete*.

The Galois connection $(\alpha^{\mathbb{HS}}, \gamma^{\mathbb{HS}})$ between the lattice of sets of k-heap graphs and $\mathcal{A}_{\mathbb{HS}}(k,\mathcal{A}_{\mathbb{W}})$ is defined as follows. The abstraction function $\alpha^{\mathbb{HS}}$ applied to some set of k-heap graphs $\mathcal{S}$ returns a finite set of k-abstract heap graphs, one for each equivalence class of $\mathcal{S}$ with respect to the isomorphism relation. This k-abstract heap graph is defined by applying $\alpha^{\mathbb{H}}$ to the set of heap graphs in the equivalence class. The concretization function $\gamma^{\mathbb{HS}}$ is a point-wise extension of $\gamma^{\mathbb{H}}$.

3.4 Abstract postcondition operator

The abstract postcondition operator on abstract heap sets corresponding to $\mathtt{post}_k$, denoted $\mathtt{post}^{\#}_k$, is obtained by replacing every concrete transformer $F \in \{\mathtt{sglt}, \mathtt{isSglt}, \mathtt{split}, \mathtt{proj}, \mathtt{updateFirst}, \mathtt{concat}\}$ with its abstraction $F^{\#}$. For example, the abstract version of the procedure `Uncover`, denoted $\mathtt{Uncover}^{\#}$, used in $\mathtt{post}^{\#}_k(\mathtt{free}(p), A_H)$ is given in Figure 6. For any abstract heap graph $\widetilde{H} = \left(N,S,V,\widetilde{W}\right)$ in the abstract heap set A_H, $\mathtt{Uncover}^{\#}(\mathtt{A_H}, \mathtt{p})$ does one of the following:

– if $\widetilde{W}$ implies that the length of the data word associated to the node n pointed by p is 1 (i.e., $\texttt{isSglt}^{\#}(\widetilde{W},n)=1$), then the abstract heap graph is added to the output abstract heap set;
– if $\widetilde{W}$ implies that the length of the data word associated to n is strictly greater than 1 (i.e., $\texttt{isSglt}^{\#}(\widetilde{W},n)=0$), then we add to the output abstract heap set an abstract heap graph obtained (1) by adding the node m which represents the immediate successor of n and (2) by updating the abstract value in $\mathcal{A}_{\mathbb{W}}$ to $\texttt{split}^{\#}(\widetilde{W},n,m)$;
– otherwise, (i.e. $\texttt{isSglt}^{\#}(\widetilde{W},n)=-1$), we add to the output abstract heap set two abstract heap graphs: (1) an abstract heap graph obtained from $\widetilde{H}$ by adding the constraint that the length of the data word associated to n is 1 (this is done by calling the procedure $\texttt{makeSglt}^{\#}(\widetilde{W},n)$), and (2) an abstract heap graph build from $\widetilde{H}$ by adding the constraint that the length of the data word associated to n is strictly greater than 1 (this is done by calling the procedure $\texttt{makeNonSglt}^{\#}(\widetilde{W},n)$) and by applying the same transformations as in the case "$\texttt{isSglt}^{\#}(\widetilde{W},n)=0$".

```
procedure Uncover#(A_H, p) {
  A'_H = ⊥^HS;
  for each H̃ = (N, S, V, W̃) ∈ A_H do {
      n = V(p);
      if (isSglt#(W̃, n) = 1)
          A'_H = A'_H ⊔^HS {H̃};
      else if (isSglt#(W̃, n) = 0){
          N' = N ∪ {m}; // m ∉ N
          S' = S[n ↦ m, m ↦ S(n)];
          W̃' = split#(W̃, n, m);
          A'_H = A'_H ⊔^HS {(N', S', V, W̃')}; }
      else {
          A'_H = A'_H ⊔^HS {(N, S, V, makeSglt#(W̃, n))};
          N' = N ∪ {m}; // m ∉ N
          S' = S[n ↦ m, m ↦ S(n)];
          W̃' = makeNonSglt#(W̃, n);
          W̃' = split#(W̃', n, m);
          A'_H = A'_H ⊔^HS {(N', S', V, W̃')}; }
  return A'_H; }
```

Fig. 6. The abstract transformer $\texttt{Uncover}^{\#}$.

Remark 1. For any call to $\texttt{concat}^{\#}(\widetilde{W},V_1,\ldots,V_t)$ made by $\texttt{post}^{\#}_k$, the sum $|V_1|+\ldots+|V_t|$ is bounded by some constant which depends on k and on the number of program variables. This follows from the fact that the heaps generated by the class of programs considered in Section 2 contain a bounded number of cut points [17]. Consequently, we can define $\mathbb{DW}$-domains $\mathcal{A}_{\mathbb{W}}$ parametrized by k such that $\texttt{concat}^{\#}(\widetilde{W},V_1,\ldots,V_t)$ with $\widetilde{W}\in\mathcal{A}_{\mathbb{W}}$ is undefined if $|V_1|+\ldots+|V_t|$ is greater than or equal than this constant.

For any abstract transformer, we have to prove soundness and precision properties [6]. Let (α,γ) be a Galois connection associated to some abstract domain. For any con-

crete transformer F, its abstraction $F^\#$ is sound if $F(\gamma(\widetilde{W})) \subseteq \gamma(F^\#(\widetilde{W}))$, for any abstract value W. $F^\#$ is a *best abstraction* if $\alpha(F(\gamma(W))) = F^\#(W)$ and it is an *exact abstraction* if $F(\gamma(W)) = \gamma(F^\#(W))$, for any value W. The following result holds.

Theorem 1. *For any k-abstract heap set A_H in $\mathcal{A}_{\mathbb{HS}}(k, \mathcal{A}_{\mathbb{W}})$, the following hold:*
– (soundness) $\mathtt{post}_k(St, \gamma^{\mathbb{HS}}(A_H)) \sqsubseteq^{\mathbb{HS}} \gamma^{\mathbb{HS}}(\mathtt{post}^\#_k(St, A_H))$*;*
– (precision) if all the abstract transfer functions in the domain $\mathcal{A}_{\mathbb{W}}$ are best (exact, resp.) abstractions then $\mathtt{post}^\#_k$ *is also a best (exact, resp.) abstraction.*

In the following, we give sound abstract transformers for $\mathcal{A}_{\mathbb{M}}$ and $\mathcal{A}_\Sigma$.

$\mathbb{DW}$-domain $\mathcal{A}_\Sigma$: Suppose that μ is an abstract value in $\mathcal{A}_\Sigma$. Then,

- $\mathtt{sglt}^\#(\mu, x)$ adds to μ three dimensions $\mathtt{hd}(x)$, $\mathtt{len}(x)$, and $\mathtt{sum\_tl}(x)$, and outputs $\mathtt{update}^\#(\mu, \mathtt{len}(x) = 1)$, where $\mathtt{update}^\#$ is the abstract transformer in the polyhedra domain corresponding to assignments;
- $\mathtt{makeSglt}^\#(\mu, x)$ returns $\mu \sqcap^{\mathbb{Z}} \mathtt{len}(x) = 1$;
- $\mathtt{makeNonSglt}^\#(\mu, x)$ returns $\mu \sqcap^{\mathbb{Z}} \mathtt{len}(x) > 1$;
- $\mathtt{isSglt}^\#(\mu, x)$ returns returns 1 if $\mu \sqsubseteq^\Sigma \mathtt{len}(x) = 1$, it returns 0 if $\mu \sqsubseteq^\Sigma \mathtt{len}(x) > 1$, and -1, otherwise;
- $\mathtt{split}^\#(\mu, x, z)$ substitutes in μ the variable $\mathtt{sum\_tl}(x)$ by $\mathtt{sum\_tl}(x) + z[0] + \mathtt{sum\_tl}(z)$;
- $\mathtt{updateFirst}^\#(\mu, x, dt, \beta)$ applies

$$\mathtt{update}^\#(\mu, \mathtt{hd}(x) = dt\beta), \text{ if } x \in \mathit{DWVars}, \text{ and}$$
$$\mathtt{update}^\#(\mu, x = dt\beta), \text{ if } x \in \mathit{DVar},$$

  where $dt\beta$ is the expression dt in which $p \rightarrow data$ is replaced by $\mathtt{hd}(\beta(p))$;
- $\mathtt{concat}^\#(\mu, V_1, \ldots, V_p)$ replaces, for any $V_i = (s_i^1, s_i^2, \ldots, s_i^{k_i})$, $1 \leq i \leq p$, the term

$$\mathtt{sum\_tl}(s_i^1) + \mathtt{hd}(s_i^2) + \mathtt{sum\_tl}(s_i^2) + \ldots + \mathtt{hd}(s_i^{k_i}) + \mathtt{sum\_tl}(s_i^{k_i})$$

  by $\mathtt{sum\_tl}(s_i^1)$, and then it projects out from the current abstract value the variables $\mathtt{hd}(w)$, $\mathtt{len}(w)$, and $\mathtt{sum\_tl}(w)$, with $w \in (V_1 \setminus \{s_1^1\}) \cup \cdots \cup (V_p \setminus \{s_p^1\})$.

Example 1. Consider the procedure `Dispatch3` in Figure 1(b). In the following, we describe some interesting steps in the analysis of this program using the abstract domain of 1-abstract heap sets (we allow abstract heap with at most one simple node) parametrized by the data words abstract domain $\mathcal{A}_\Sigma$ presented above. Suppose that in the initial state, `head` points to a non-empty list such that the sum of its elements equals the value of the variable *s*. This is described by the abstract heap in Figure 7(a). The shape of the heap is given by the graph: a node *n* labeled by `head` connected to the distinguished node # representing *null* which is pointed by `tmp`, `grtC`, and `lessC`. The constraints regarding the length of the list and the data in the list are given by the abstract value written below the graph.

Two of the abstract heaps obtained after the first iteration of the loop are given in Figure 7(b) and (c). Notice that, the statement `tmp=x->next` produces two abstract

heaps depending on the length of the list pointed by head. If this length is 1 then we obtain an abstract heap with 2 nodes similar to the one in Figure 7(b), and, otherwise, we obtain an abstract heap with three nodes, one for the successor of n, denoted $n1$. The latter involves a call to the abstract transformer $\texttt{split}^{\#}$ which modifies the abstract value in Figure 7(a) to $\texttt{len}(n) = 1 \wedge \texttt{hd}(n) + \texttt{sum\_tl}(n) + \texttt{hd}(n1) + \texttt{sum\_tl}(n1) = s$. The abstract heaps mentioned above are produced when the test of the if statement is true, and consequently, they contain the constraint $\texttt{hd}(n) \geq C$. Analogously, when the test of the if statement is false, we obtain two abstract heaps: they contain the constraint $\texttt{hd}(n) < C$ and the labels lessC and grtC are interchanged.

After another two iterations, one of the obtained abstract heaps is the one in Figure 7(e). It is obtained by applying the abstract transformer $\texttt{Normalize}_k{}^{\#}$ on the abstract heap in Figure 7(d) which induces a call to $\texttt{concat}^{\#}$ on the abstract value from L^{Σ} contained in this abstract heap. The latter consists concatenationg the words associated to $n2$, $n1$ and n into one word which will be associated to $n2$. Thus, the length of the word associated to $n2$ in the output abstract value is updated to the sum of the lengths of the words associated to $n2$, $n1$, and n in the input abstract value. Also, we substitute the term $\texttt{hd}(n1) + \texttt{sum\_tl}(n1) + \texttt{hd}(n) + \texttt{sum\_tl}(n)$ by $\texttt{sum\_tl}(n2)$.

Notice that the abstract heaps in Figure 7(c) and (e) are similar and their join is an abstract heap defined by the same graph as Figure 7(c) and by the abstract value

$$\texttt{len}(n) \in [1,3] \wedge \texttt{hd}(n) \geq C \wedge \texttt{hd}(n) + \texttt{sum\_tl}(n) + \texttt{hd}(n1) + \texttt{sum\_tl}(n1) = s. \quad \text{(F)}$$

After some iterations, the current abstract heap set will contain abstract heaps to cover all possible cases: when the lists pointed by grtC and lessC are non-empty or when one of these lists is empty. If we apply the widening operator $\nabla^{\mathbb{HS}}$ starting with the fourth iteration, which induces a call to ∇^{Σ}, constraints like $\texttt{len}(n) \in [1,3]$ in (F) will disappear and the analysis will terminate.

Similarly, using the abstract domain for multisets $\mathcal{A}_{\mathbb{M}}$ we can prove that the union of the multisets containing the elements of the list pointed by grtC and lessC, respectively, equals the multiset containing the elements of the starting list.

$\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{M}}$: For any abstract value μ in $\mathcal{A}_{\mathbb{M}}$,

- $\texttt{sglt}^{\#}(\mu, x)$ adds to μ two dimensions for $\texttt{ms\_hd}(x)$ and $\texttt{ms\_tl}(x)$,
- $\texttt{split}^{\#}(\mu, x, z)$ replaces in μ the multiset variable $\texttt{ms\_tl}(x)$ by the union term $\texttt{ms\_tl}(x) \cup \texttt{ms\_hd}(z) \cup \texttt{ms\_tl}(z)$;
- $\texttt{isSglt}^{\#}(\mu, x)$ returns always -1;
- $\texttt{makeSglt}^{\#}(\mu, x)$ and $\texttt{makeNonSglt}^{\#}(\mu, x)$ return μ;
- $\texttt{updateFirst}^{\#}(\mu, x, dt, \beta)$ considers the following cases:
  - if $x \in DVar$ and $dt = d \in DVar$ then it adds to μ the atomic formula $\texttt{ms}(x) = \texttt{ms}(d)$,
  - if $x \in DWVars$ and $dt = p \rightarrow data$ then it adds to μ the atomic formula $\texttt{ms\_hd}(x) = \texttt{ms\_hd}(\beta(p))$,
  - otherwise, it returns $\top^{\mathbb{M}}$.
- $\texttt{concat}(\mu, V_1, \ldots, V_p)$ applies $\texttt{Saturate}(\mu)$ and then, for any $V_i = (s_i^1, s_i^2, \ldots, s_i^{k_i})$, $1 \leq i \leq p$, it replaces the union term:

$$\texttt{ms\_tl}(s_i^1) \cup \texttt{ms\_hd}(s_i^2) \cup \texttt{ms\_tl}(s_i^2) \cup \cdots \cup \texttt{ms\_hd}(s_i^{k_i}) \cup \texttt{ms\_tl}(s_i^{k_i})$$

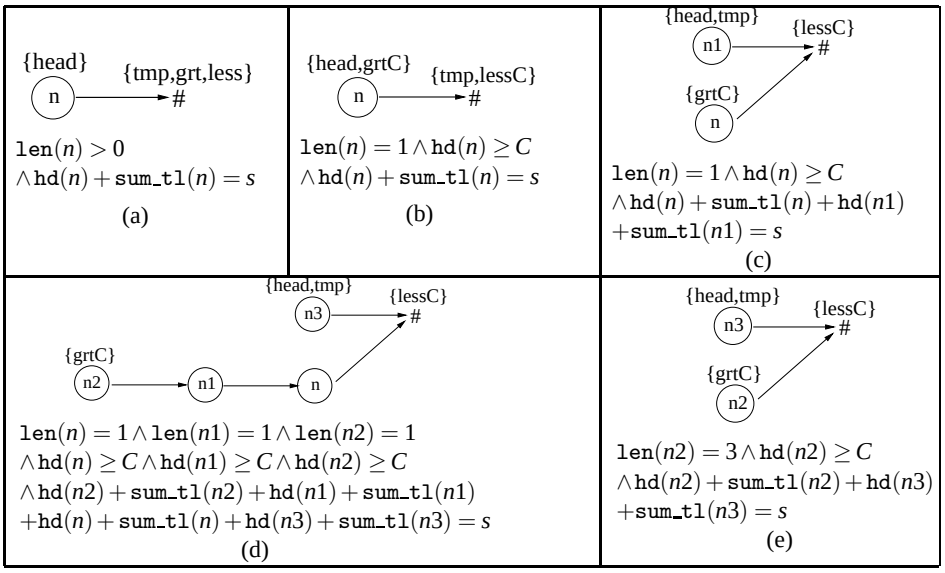


Fig. 7. Abstract heaps for the procedure `Dispatch3`

by $\texttt{ms\_tl}(s_i^1)$. Afterwards, the atomic formulas that still contain data word variables in $(V_1 \setminus \{s_1^1\}) \cup \cdots \cup (V_p \setminus \{s_p^1\})$ are deleted.

4 A $\mathbb{DW}$-domain over universally-quantified formulas

We define the $\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{U}} = \left(L^{\mathbb{U}}, \sqsubseteq^{\mathbb{U}}, \sqcap^{\mathbb{U}}, \sqcup^{\mathbb{U}}, \top^{\mathbb{U}}, \bot^{\mathbb{U}}\right)$ whose elements are universally quantified first-order formulas.

4.1 Syntax of formulas

The elements of $\mathcal{A}_{\mathbb{U}}$ are formulas of the form $E(\mathcal{V}) \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y}.\, P(\mathbf{y}, \mathcal{V}) \Rightarrow U_P(\mathbf{y}, \mathcal{V})$, where $\mathcal{V}$ is a set of data word variables. The sub-formula E is quantifier-free and it characterizes the lengths and the first symbols of the words. It is an arithmetical formula over terms $\texttt{hd}(w)$ and $\texttt{len}(w)$ with $w \in \mathcal{V}$. The variables $\mathbf{y}$ are interpreted as positions from the tail of the words represented by the variables in $\mathcal{V}$. P is a formula, called *pattern*, which constrains the positions denoted by $\mathbf{y}$. It belongs to the set $\mathcal{P}(\mathcal{V})$ of formulas obtained from a finite set $\mathcal{P}$ by substituting, in any possible way, the data words variables with the ones in $\mathcal{V}$. The set $\mathcal{P}$ is supposed to be given by the user and it is a parameter of the domain $\mathcal{A}_{\mathbb{U}}$. The formulas U_P, for any $P \in \mathcal{P}(\mathcal{V})$, are arithmetical formulas over the terms in E plus $w[y]$ and y, for any $w \in DWVars$ and $y \in \mathbf{y}$. Together with E they represent abstract values in some numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ which is also a parameter of $\mathcal{A}_{\mathbb{U}}$. For example, the following formula specifies that the word denoted by w_1 is a copy of the word denoted by w_2:

$$\texttt{len}(w_1) = \texttt{len}(w_2) \wedge \forall y_1, y_2.\ (y_1 \in \texttt{tl}(w_1) \wedge y_2 \in \texttt{tl}(w_2) \wedge y_1 = y_2) \Rightarrow w_1[y_1] = w_2[y_2],$$

and the following formula expresses the specification of `Fibonacci` from Figure 1(a):

$$\texttt{hd}(w) = 1 \wedge \forall y_1, y_2, y_3.\ ((y_1, y_2, y_3) \in \texttt{tl}(w) \wedge y_1 <_1 y_2 <_1 y_3) \Rightarrow w[y_3] = w[y_1] + w[y_2].$$

Above, the arithmetical formulas represent values from the Polyhedra domain [8]. Based on Remark 1, $\mathcal{A}_{\mathbb{U}}$ is also parametrized by an integer k.

Syntax of patterns The patterns describe a set of positions belonging to the tails of different words. They specify the word to which the positions belong, they fix an order between the positions belonging to the same word, and they put arithmetical constraints on some of these positions (which are the first on each word). For example, the formulas above contain the patterns $y_1 \in \texttt{tl}(w_1) \wedge y_2 \in \texttt{tl}(w_2) \wedge y_1 = y_2$ and $(y_1, y_2, y_3) \in \texttt{tl}(w) \wedge y_1 <_1 y_2 <_1 y_3$.

Let $(w_1, \ldots, w_q)$ and $\mathbf{w}$ be two vectors of data words variables. Also, let $\mathbf{y_i} = (y_i^1, \ldots, y_i^{p_i})$, for any $1 \leq i \leq q$, be vectors of *position variables* interpreted as integers s.t. $\mathbf{y_i} \cap \mathbf{y_j} = \emptyset$, for any $i \neq j$. A pattern $P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w})$ is a formula of the form

$$\bigwedge_{1 \leq i \leq q} P_R^i(\mathbf{y}_i, w_i) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w}), \text{ where} \tag{G}$$

– $P_R^i(\mathbf{y}_i, w_i)$ is of form $\mathbf{y_i} \in \texttt{tl}(w_i) \wedge y_i^1 \sim_1 y_i^2 \sim_2 \ldots \sim_{p_i-1} y_i^{p_i}$, where $\sim_1, \ldots, \sim_{p_i-1}$ is either the strict increasing order $<$, either the predicate "greater than or equal" $\leq$, or the "immediate successor" predicate $<_1$ (i.e., $y <_1 y'$ iff $y' = y + 1$). The formula $\mathbf{y_i} \in \texttt{tl}(w_i)$ states that the positions denoted by the variables in $\mathbf{y_i}$ belong to w_i and that $\texttt{len}(w_i) \geq |\mathbf{y_i}| + 1$ (that is, the tail of the word denoted by w_i contains at least $|\mathbf{y_i}|$ positions).
– P_L is a boolean combination of linear constraints over variables $y_1^1, \ldots, y_q^1$ and $\texttt{len}(w)$ with $w \in \mathbf{w}$. We assume that P_L does not constrain the lengths of the words in $\mathbf{w}$, that is, $\texttt{len}(w) > 0$ implies $\exists y_1^1, \ldots, y_q^1.\ P_L$, for any $w \in \mathbf{w}$.

4.2 Semantics of formulas

A model for a formula in $L^{\mathbb{U}}$ is a pair of valuations for the free variables, $L : DWVars \to \mathbb{Z}^+$ and $D : DVar \to \mathbb{Z}$. In the following we give the semantics of these formulas. The fact that (L, D) is a model of $\widetilde{W} \in L^{\mathbb{U}}$ is denoted by $(L, D) \models \widetilde{W}$.

Let $\widetilde{W}$ be the formula

$$E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1}, \ldots, \mathbf{y_q}.\ P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P.$$

We have that $(L, D) \models \phi_1 \wedge \phi_2$ iff $(L, D) \models \phi_1$ and $(L, D) \models \phi_2$. Also, $(L, D) \models E$ iff the function $f : \{l[w], w[0] \mid w \in DWVars\} \to \mathbb{Z}$, defined by $f(l[w]) = |L(w)|$ and $f(w[0]) = L(w)[0]$ belongs to $\gamma^{\mathbb{Z}}(E)$ where $\gamma^{\mathbb{Z}}$ is the concretization function associated to $\mathcal{A}_{\mathbb{Z}}$.

Let $\mathbf{y_i} = y_i^1 \ldots y_i^{p_i}$, for any $1 \leq i \leq q$, be a set of vectors of position variables. Then,

$$(L,D) \models \forall \mathbf{y_1} \ldots \forall \mathbf{y_q}. \left(\left(\bigwedge_{1 \leq i \leq q} \mathbf{y_i} \in \mathtt{tl}(w_i) \wedge y_i^1 \sim_1 y_i^2 \sim_2 \ldots \sim_{p_i-1} y_i^{p_i} \right) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \right) \Rightarrow U_P$$

iff for every $\pi : \mathbf{y_1} \cup \ldots \mathbf{y_q} \rightarrow \mathbb{N}^+$ such that

- $\mathbf{y_i} \leq |L(w_i)| + 1$, for any $1 \leq i \leq q$,
- $\pi(y_i^r) \in [1, |L(w_i)|]$, for any $1 \leq i \leq q$ and $1 \leq r \leq p_i$,
- $\pi(y_i^1) \sim_1 \pi(y_i^2) \sim_2 \ldots \sim_{p_i-1} \pi(y_i^{p_i})$, for any $1 \leq i \leq q$,
- $g : \{l[w] \mid w \in \mathbf{w}\} \cup \{y_1^1, \ldots, y_q^1\} \rightarrow \mathbb{N}^+$ defined by $g(l[w]) = |L(w)|$ and $g(y_r^1) = \pi(y_r^1)$, for any $1 \leq r \leq q$, satisfies the formula $P_L(y_1^1, \ldots, y_q^1, \mathbf{w})$,

the function

$$h : \{l[w], w[0] \mid w \in DWVars\} \cup \mathbf{y_1} \cup \ldots \cup \mathbf{y_q} \cup \{w_i[y] \mid 1 \leq i \leq q \text{ and } y \in \mathbf{y_i}\} \rightarrow \mathbb{Z}$$

defined by $h(l[w]) = |L(w)|$, $h(w[0]) = L(w)[0]$, $h(y) = \pi(y)$, for any $y \in \mathbf{y_1} \cup \ldots \cup \mathbf{y_q}$, and $h(w_i[y]) = L(w_i)[\pi(y)]$, for any $1 \leq i \leq q$ and $y \in \mathbf{y_i}$ belongs to the concretization of U_P, $\gamma^{\mathbb{Z}}(U_P)$.

4.3 Lattice operators

In this subsection, we define the entailment relation between elements of $\mathcal{A}_{\mathbb{U}}$, the join, the meet, and the widening operators. Thus, let $\widetilde{W}$ be the abstract value

$$E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1}, \ldots, \mathbf{y_q}. P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P$$

and $\widetilde{W}'$ be the value

$$E' \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1}, \ldots, \mathbf{y_q}. P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U'_P.$$

Then, $\widetilde{W} \sqsubseteq^{\mathbb{U}} \widetilde{W}'$ iff (1) $E \sqsubseteq^{\mathbb{Z}} E'$, and (2) for every pattern $P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w})$, if $E \sqsubseteq^{\mathbb{Z}} \mathtt{len}(w_i) < |\mathbf{y_i}| + 1$ does not hold, for every $1 \leq i \leq q$, then $E \sqcap^{\mathbb{Z}} U_P \sqsubseteq^{\mathbb{Z}} U'_P$.

Also, for any $\ddagger \in \{\sqcup, \sqcap, \nabla\}$, $\widetilde{W} \ddagger^{\mathbb{U}} \widetilde{W}'$ is an abstract value of the form

$$E'' \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1}, \ldots, \mathbf{y_q}. P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U''_P,$$

where

- $E \ddagger^{\mathbb{Z}} E'$,
- for any $P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \in \mathcal{P}(\mathcal{V})$, we have that:
  - if $E \sqsubseteq^{\mathbb{Z}} \mathtt{len}(w_i) < |\mathbf{y_i}| + 1$, for some $1 \leq i \leq q$, then we define $U''_P = U'_P$,

- if $E' \sqsubseteq^{\mathbb{Z}} \texttt{len}(w_i) < |\mathbf{y_i}| + 1$, for some $1 \leq i \leq q$, then we define $U''_P = U_P$,
- otherwise, $U''_P = U_P \ddagger^{\mathbb{Z}} U'_P$.

Proposition 3. *The entailment relation $\sqsubseteq^{\mathbb{U}}$ is* sound, *that is, $\widetilde{W} \sqsubseteq^{\mathbb{U}} \widetilde{W}'$ implies $\gamma^{\mathbb{U}}(\widetilde{W}) \subseteq \gamma^{\mathbb{U}}(\widetilde{W}')$. Also, $\sqcup^{\mathbb{U}}$ and $\sqcap^{\mathbb{U}}$ are the least upper bound and greatest lower bound, respectively, and $\nabla^{\mathbb{U}}$ is a widening operator.*

Proof: The properties of $\sqsubseteq^{\mathbb{U}}$, $\sqcup^{\mathbb{U}}$, $\sqcap^{\mathbb{U}}$, resp. follow directly from similar properties of $\sqsubseteq^{\mathbb{Z}}$, $\sqcup^{\mathbb{Z}}$, and $\sqcap^{\mathbb{Z}}$, resp. The proof for the widening operator relies on the widening operator $\nabla^{\mathbb{Z}}$ and on the fact that the set of patterns $\mathcal{P}$ is fixed. □

The Galois connection between the lattice $\mathcal{C}(DWVars, DVar)$ and $\mathcal{A}_{\mathbb{U}}$ is defined according to the semantics given in Section 4.2.

4.4 Closed sets of patterns

In the abstract transformer $\texttt{concat}^{\#}$ we have to transfer properties from a set of words to their concatenation. This implies that we must be able to describe data from positions which belong to intervals that overlap two or more words. To describe these sets of positions we compute for each pattern a set of patterns denoted $\texttt{Closure}(P,k)$. This computation can be done automatically and it is not trivial in the presence of arithmetical constraints.Then, $\texttt{Closure}(P,k)$ is extended as usual to sets of patterns. A set of patterns is *closed* if it equals $\texttt{Closure}(\mathcal{P},k)$, for some set of patterns $\mathcal{P}$. The precision result for $\texttt{concat}^{\#}$ given in the next sub-section is obtained only for closed sets of patterns. We begin by an example and afterwards, we give the formal definition of $\texttt{Closure}(P,k)$.

Example 2. Let P be $(y_1,y_2,y_3) \in \texttt{tl}(w) \wedge y_1 <_1 y_2 <_1 y_3$. Suppose that w is interpreted as the concatenation of two words denoted by n_1 and n_2. If we want to deduce a property of w of the form $\forall y_1,y_2,y_3.\ P \Rightarrow U$ from properties of n_1 and n_2, we have to use universally quantified formulas having as left part $P[w \leftarrow n_1]$, $P[w \leftarrow n_2]$, and one of the following patterns:

$$P_1(n_1) := (y_1) \in \texttt{tl}(n_1) \wedge y_1 = \texttt{len}(n_1) - 1,\ P_2(n_2) := (y_3) \in \texttt{tl}(n_2) \wedge y_3 = 1$$
$$P_3(n_1) := (y_1,y_2) \in \texttt{tl}(n_1) \wedge y_1 <_1 y_2 \wedge y_1 = \texttt{len}(n_1) - 2, \text{ and}$$
$$P_4(n_2) := (y_2,y_3) \in \texttt{tl}(n_2) \wedge y_2 <_1 y_3 \wedge y_2 = 1.$$

These patterns characterize any three consecutive positions in the word denoted by w. Using P_1 and P_2, we capture the case when y_1 is the last position of n_1 and y_2, y_3 are the first two positions of n_2. Because patterns characterize positions in tails of words, y_2 does not appear explicitly. Its data can be characterized using the quantifier-free part. The case when y_1 and y_2 are the last positions of n_1 and y_3 is the first position of n_2 is considered using P_2. The pattern P_3 describes the case when y_1, y_2, and y_3 are the first three positions of n_2. Finally, $P[w \leftarrow n_1]$ and $P[w \leftarrow n_2]$ consider the situations when all the positions belong to the same word.

Formal definition of $\texttt{Closure}(P,k)$**:** Let $P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w})$ be a pattern

$$\left(\bigwedge_{1 \leq i \leq q} \left(\mathbf{y_i} \in \texttt{tl}(w_i) \wedge y_i^1 \sim_1 y_i^2 \sim_2 \ldots \sim_{p_i-1} y_i^{p_i}\right) \right) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w})$$

and let $\mathbf{u}_1,\ldots,\mathbf{u_q}$ be vectors of data word variables. Suppose that w_i represents the concatenation of the words denoted by $\mathbf{u_i}$, $1 \leq i \leq q$. The procedure $\texttt{Closure}(P, \mathbf{u}_1, \ldots, \mathbf{u}_q)$ contains two steps:

1. we define all the patterns which constrain the positions $\mathbf{y_i}$, for any $1 \leq i \leq q$, such that they belong to one of the words in $\mathbf{u_i}$ and they satisfy the ordering constraints in P_R^i and the arithmetical constraints in P_L;
2. for any pattern P' obtained in the first step and for any subset $\mathbf{u} \subseteq \mathbf{u}_1 \cup \ldots \cup \mathbf{u}_q$ we put in $\texttt{Closure}(P, \mathbf{u}_1, \ldots, \mathbf{u}_q)$ a pattern containing all the atomic formulas over position variables which are constrained, by P', to belong to a word denoted by one of the variables in $\mathbf{u}$.

Following the steps described above, $\texttt{Closure}(P, \mathbf{u}_1, \ldots, \mathbf{u}_q)$ is defined as follows:

Compute $\texttt{Tuples}(P_R^i, v_1, \ldots, v_m)$**:** Let $v_1,\ldots, v_m$ be a sequence of data words variables and let $P_R^i(\mathbf{y}_i, w_i)$ be the formula:

$$\mathbf{y}_i \in \texttt{tl}(w_i) \wedge y_i^1 \sim_1 y_i^2 \sim_2 \ldots \sim_{j-1} y_i^j,$$

where $\sim_1, \ldots, \sim_{j-1} \in \{<, \leq, <_1\}$. Intuitively, $\texttt{Tuples}(P_R^i, v_1, \ldots, v_m)$ shows all the possible ways of choosing j positions satisfying the order constraint in P_R^i on the word representing the concatenation of the words $v_1,\ldots,v_m$. Roughly, it will consider all the possible ways of choosing an arbitrary number of positions on the word v_1, an arbitrary number of positions on the word v_2, etc.
Clearly, if $m = 1$ then $\texttt{Tuples}(P_R^i, v_1, \ldots, v_m) = P_R^i[w_i \leftarrow v_1]$.
Otherwise, if $m \geq 2$, we begin by considering the case when we choose positions that are not first symbols of the words $v_1,\ldots,v_m$. Thus, we consider m' formulas, $\phi_1, \ldots, \phi_{m'}$, which characterize j positions on the the concatenation of the words $v_1,\ldots,v_m$ that are not first symbols of these words:

$$\begin{array}{l}
\phi_1 = (y_i^1, \ldots, y_i^{q_1}) \in \texttt{tl}(v_{s_1}) \wedge y_i^1 \sim_1 \ldots \sim_{q_1-1} y_i^{q_1} \\
\phi_2 = (y_i^{q_1+1}, \ldots, y_i^{q_2}) \in \texttt{tl}(v_{s_2}) \wedge y_i^{q_1+1} \sim_{q_1+1} \ldots \sim_{q_2-1} y_i^{q_2} \\
\ldots \\
\phi_{m'} = (y_i^{q_{m'-1}+1}, \ldots, y_i^j) \in \texttt{tl}(v_{s_{m'}}) \wedge y_i^{q_{m'-1}+1} \sim_{q_{m'-1}+1} \ldots \sim_{j-1} y_i^j,
\end{array}$$

where the following hold:
- $1 \leq m' \leq m$ and $1 \leq s_1 < s_2 < \ldots < s_{m'} \leq m$,
- $1 \leq q_1 < q_2 < \ldots < q_{m'-1} \leq j$,
- if $m' > 1$ then $\sim_{q_1} \neq <_1, \ldots, \sim_{q_{m'-1}} \neq <_1$.

Above, ϕ_1 characterizes q_1 positions belonging to the word v_{s_1}, for some $1 \leq s_1 \leq m$, which obey the same restrictions as the first q_1 positions in P_R^i. Then, ϕ_2 characterizes q_2 positions belonging to the word v_{s_2}, for some $1 \leq s_2 \leq m$ greater than

s_1, which obey the same restrictions as the next $q_2 - q_1$ positions in P_R^i. Obviously, the positions $y_i^{q_1}$ and $y_i^{q_1+1}$ can not be immediate successors because we have considered the case when $y_i^{q_1+1}$ can not be the first symbol of v_{s_2}. Thus, $\pi_{q_1} \neq 1$. The same happens with all the remaining formulas ϕ_3,...,$\phi_{m'}$.
Then $\phi_1 \wedge \ldots \wedge \phi_{m'} \in \texttt{Tuples}(P_R^i, v_1, \ldots, v_m)$, for any s_1,...,$s_{m'}$ and q_1,...,$q_{m'-1}$ as above.
Now, for the general case, we consider m' formulas ψ_1, ..., $\psi_{m'}$ as follows:

$$\psi_1 = (y_i^{t_0}, \ldots, y_i^{q_1}) \in \texttt{tl}(v_{s_1}) \wedge y_i^{t_0} \sim_{t_0} \ldots \sim_{q_1-1} y_i^{q_1}$$
$$\psi_2 = (y_i^{q_1+t_1}, \ldots, y_i^{q_2}) \in \texttt{tl}(v_{s_2}) \wedge y_i^{q_1+t_1} \sim_{q_1+t_1} \ldots \sim_{q_2-1} y_i^{q_2}$$
$$\ldots$$
$$\psi_{m'} = (y_i^{q_{m'-1}+t_{m'-1}}, \ldots, y_i^{j}) \in \texttt{tl}(v_{s_{m'}}) \wedge y_i^{q_{m'-1}+t_{m'-1}} \sim_{q_{m'-1}+t_{m'-1}} \ldots \sim_{j-1} y_i^{j},$$

where the following hold:
- $1 \leq m' \leq m$ and $1 \leq s_1 < s_2 < \ldots < s_{m'} \leq m$,
- $1 \leq t_0 < q_1 < q_2 < \ldots < q_{m'-1} \leq j$,
- $t_0 < s_1 + 1$ and $t_r \leq s_{r+1} - s_r + 1$, for any $1 \leq r < m'$,
- $q_r + t_r < q_{r+1}$, for any $1 \leq r < m' - 1$, and $q_{m'-1} + t_{m'-1} < j$,
- if $m' > 1$ and $t_{r+1} = 1$ then $\sim_{q_r} \neq <_1$, for any $1 \leq r < m' - 1$.

Above, ψ_1 describes $q_1 - t_0$ positions belonging to the word v_{s_1}, for some $1 \leq s_1 \leq m$, which obey the same restrictions as the positions $y_i^{t_0}$,...,$y_i^{q_1}$ in P_R^i. It is supposed that the first $t_0 - 1$ positions are mapped to first symbols of some of the words v_2,...,v_{s_1}. For this, we must have $t_0 < s_1 + 1$. Then, ψ_2 describes $q_2 - (q_1 + t_1)$ positions belonging to the word v_{s_2}, for some $1 \leq s_2 \leq m$, which obey the same restrictions as the positions $y_i^{q_1+t_1}$,...,$y_i^{q_2}$ in P_R^i. It is supposed that the positions $y_i^{q_1+1}$,...,$y_i^{q_1+t_1-1}$ are mapped to first symbols of some of the words v_{s_1+1},...,v_{s_2}. Consequently, $t_1 \leq s_2 - s_1 + 1$ must hold. If $t_1 = 1$, then between $y_i^{q_1}$ and $y_i^{q_1+1}$ there is at least one symbol (the first symbol of v_{s_2}). Consequently, π_{q_1} must be different from 1. The same happens for the other formulas.
Then $\psi_1 \wedge \ldots \wedge \psi_{m'} \in \texttt{Tuples}(P_R^i, v_1, \ldots, v_m)$, for any s_1,...,$s_{m'}$, t_0, q_1,...,$q_{m'-1}$, and t_1,...,$t_{m'-1}$ as above.

The output in the first step of $\texttt{Closure}(P, \mathbf{u}_1, \ldots, \mathbf{u}_q)$**:** We compute in this step the set of all patterns of the form:

$$\bigwedge_{1 \leq i \leq q} Q_R^i(\mathbf{y}_i, \mathbf{u_i}) \wedge P_L'(y_1^1, \ldots, y_q^1, \mathbf{w}) \wedge P_L'', \text{ where}$$

- $Q_R^i(\mathbf{y}_i, \mathbf{u_i}) \in \texttt{Tuples}(P_R^i, \mathbf{u_i})$,
- $P_L'(y_1^1, \ldots, y_q^1, \mathbf{w})$ is obtained from $P_L(y_1^1, \ldots, y_q^1, \mathbf{w})$ by:
  - for any $1 \leq i \leq q$, we substitute $l[w_i]$ with $l[u_i^1] + l[u_i^2] + \cdots + l[u_i^{j_i}]$, where $\mathbf{u_i} = (u_i^1, u_i^2, \ldots, u_i^{j_i})$,
  - for any $1 \leq i \leq q$, if y_i^1 is a position of the word u_i^{o+1}, for some $1 < o < j_i$, then we substitute y_i^1 with $y_i^1 + l[u_i^1] + \ldots + l[u_i^o]$.
- P_L'' is a conjunction of linear constraints build as follows:

- for any $1 \leq i \leq q$, for any sub-formula of $Q_R^i(\mathbf{y}_i, \mathbf{u_i})$ of the form

$$(y_r, \ldots, y_{r'}) \in \mathtt{tl}(v) \wedge y_r <_{\pi_r} \ldots <_{\pi_{r'-1}} y_{r'},$$

with $r < r' < |\mathbf{y_i}|$, if $\pi_r = \ldots = \pi_{r'-1} = \pi_r = 1$ (π_r is known from P_R^i) then we add to P_L'' the constraint $y_1 = \mathtt{len}(v) - r$.

The output in the second step of $\mathtt{Closure}(P, \mathbf{u}_1, \ldots, \mathbf{u}_q)$**:** Let P' be a pattern computed in the first step of the form:

$$\bigwedge_{1 \leq i \leq q} Q_R^i(\mathbf{y}_i, \mathbf{u_i}) \wedge P_L'(y_1^1, \ldots, y_q^1, \mathbf{w}) \wedge P_L''.$$

Also, let $\mathbf{u} \subseteq \mathbf{u}_1 \cup \ldots \cup \mathbf{u}_q$. We denote by $\mathbf{y_i}' \subseteq \mathbf{y_i}$ the set of variables y for which there exist some variable $u \in \mathbf{u}$ and some vector $\mathbf{y} \subseteq \mathbf{y_i}$ with $y \in \mathbf{y}$ such that $\mathbf{y} \in \mathtt{tl}(u)$ is an atomic formula in $Q_R^i(\mathbf{y}_i, \mathbf{u_i})$. Let $Q_R^i(\mathbf{y}_i, \mathbf{u_i} \cap \mathbf{u})$ denote the conjunction of atomic formulas in $Q_R^i(\mathbf{y}_i, \mathbf{u_i})$ over position variables in $\mathbf{y_i}'$. Then, $\mathtt{Closure}(P, \mathbf{u}_1, \ldots, \mathbf{u}_q)$ contains the pattern

$$\left(\bigwedge_{1 \leq i \leq q} Q_R^i(\mathbf{y}_i, \mathbf{u_i} \cap \mathbf{u}) \right) \wedge P_L^{\mathbf{u}},$$

where $P_L^{\mathbf{u}}$ is the quantifier-free formula equivalent to $\exists \mathbf{Y}.\ \left(P_L'(y_1^1, \ldots, y_q^1, \mathbf{w}) \wedge P_L''\right)$. Here, $\mathbf{Y}$ contains all the position variables in $P_L'(y_1^1, \ldots, y_q^1, \mathbf{w}) \wedge P_L''$ which are not present in $\bigwedge_{1 \leq i \leq q} Q_R^i(\mathbf{y}_i, \mathbf{u_i} \cap \mathbf{u})$.

Notice that if the pattern P contains no arithmetical constraints then $\mathtt{Closure}(P, k) = \mathtt{Closure}(P, k+1)$, for any k greater than or equal to the number of position variables in P.

By taking all the sets $\mathtt{Closure}(P, \mathbf{u}_1, \ldots, \mathbf{u}_q)$ which correspond to concatenations of at most C words, where C is the constant mentioned in Remark 1, we obtain $\mathtt{Closure}(P, k)$. Then, $\mathtt{Closure}(P, k)$ is extended as usual to sets of patterns. A set of patterns is *closed* if it equals $\mathtt{Closure}(\mathcal{P}, k)$, for some $\mathcal{P}$.

4.5 Abstract transformers

In this section, we define the abstract transformers associated to the $\mathbb{DW}$ domain $\mathcal{A}_{\mathbb{U}}$. We prove that these abstract transformers are sound with respect to the concrete operations and we identify conditions under which they are precise. First, we will present a particular class of abstract values for which we can prove the precision of the abstract transformers. In the following, the projection of some element $\widetilde{X}$ from some abstract domain $\mathcal{A}$ on some set of variables V is denoted $\widetilde{X}\downarrow V$. The projection of an abstract element $\widetilde{X}$ on all the variables except some set V is denoted $\widetilde{X}\uparrow V$.

Closed abstract values In general, there may be more than one abstract value in $L^{\mathbb{U}}$ having the same concretization. For example,

$$\begin{gathered}\mathtt{hd}(w) = 0 \wedge \mathtt{len}(w) = 6 \\ \wedge \forall y_1.\ (y_1) \in \mathtt{tl}(w) \Rightarrow (1 \leq w[y_1] \leq 5) \wedge \forall y_1, y_2.\ ((y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 < y_2) \Rightarrow w[y_1] < w[y_2].\end{gathered}$$

and

$$\begin{gathered}\mathtt{hd}(w) = 0 \wedge \mathtt{len}(w) = 6 \\ \wedge \forall y_1.\ (y_1) \in \mathtt{tl}(w) \Rightarrow (w[y_1] = y_1) \wedge \forall y_1, y_2.\ ((y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 < y_2) \Rightarrow \top^{\mathbb{Z}}.\end{gathered}$$

represent the same data word $w = 012345$. In some of the results that follow we need a canonical representation for the abstract values in $L^{\mathbb{U}}$. Such representations are called *closed abstract values*.

Definition 5. *An abstract value $W \in L^{\mathbb{U}}$ is* closed *if $\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) = W$.*

The intuition behind the notion of closure is given by the following lemma. Roughly, an abstract value W is closed iff it is the smallest (w.r.t. $\sqsubseteq^{\mathbb{U}}$) between all values having the same concretization as W.

Lemma 1. *An abstract value $W \in L^{\mathbb{U}}$ is closed iff for any $W' \in L^{\mathbb{U}}$ with $\gamma^{\mathbb{U}}(W') = \gamma^{\mathbb{U}}(W)$, we have that $W \sqsubseteq^{\mathbb{U}} W'$.*

Proof: " $\Rightarrow$ " Let $W' \in L^{\mathbb{U}}$ such that $\gamma^{\mathbb{U}}(W) = \gamma^{\mathbb{U}}(W')$. By the fact that W is closed, we have that $\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W')) = \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) = W$. By the fact that $(\alpha^{\mathbb{U}}, \gamma^{\mathbb{U}})$ is a Galois connection, from $\gamma^{\mathbb{U}}(W') \subseteq \gamma^{\mathbb{U}}(W')$ we obtain that $\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W')) \sqsubseteq^{\mathbb{U}} W'$. The latter, implies $W \sqsubseteq^{\mathbb{U}} W'$.
" $\Leftarrow$ " from the definition of the Galois connection we have that $\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) \sqsubseteq^{\mathbb{U}} W$. Let $W' = \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))$. If we can prove that $\gamma^{\mathbb{U}}(W) = \gamma^{\mathbb{U}}(W')$ then, by the hypothesis, we obtain that $W \sqsubseteq^{\mathbb{U}} W' = \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))$.

Since $\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) \sqsubseteq^{\mathbb{U}} \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))$ and (α, γ) is a Galois connection, we obtain that $\gamma^{\mathbb{U}}(W) \subseteq \gamma^{\mathbb{U}}(W')$. Then, since $\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) \sqsubseteq^{\mathbb{U}} W$ and γ is monotonic, we obtain that $\gamma^{\mathbb{U}}(W') \subseteq \gamma^{\mathbb{U}}(W)$. □

For any $W \in L^{\mathbb{U}}$, we define its *closure* as the abstract element $\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))$. The following result holds.

Lemma 2. *For any $W \in L^{\mathbb{U}}$, its closure is a closed abstract value*

Proof: Let $V = \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))$. We have to prove that $V = \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(V))$, that is,

$$\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) = \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)))).$$

To this we try to prove

$$\gamma^{\mathbb{U}}(W) = \gamma^{\mathbb{U}}(\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))).$$

To prove the implication from left to right, we start from

$$\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) \sqsubseteq^{\mathbb{U}} \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)),$$

which by the property of the Galois connection, implies,

$$\gamma^{\mathbb{U}}(W) \subseteq \gamma^{\mathbb{U}}(\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))).$$

To prove the implication from right to left, we start from

$$\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W)) \sqsubseteq^{\mathbb{U}} W,$$

which by the monotonicity of the concretization function $\gamma^{\mathbb{U}}$, implies

$$\gamma^{\mathbb{U}}(\alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))) \subseteq \gamma^{\mathbb{U}}(W).$$

□

The definition and the lemma above hold also for other abstract domains, instead of $\mathcal{A}_{\mathbb{U}}$. In the following, we will give a characterization for some class of closed abstract values which are defined on some specific class of patterns called *simple patterns*.

Definition 6. *A pattern $P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q)$ with $\mathbf{y_i} = y_i^1 \ldots y_i^{p_i}$, for any $1 \leq i \leq q$, of the form:*

$$\bigwedge_{1 \leq i \leq q} \mathbf{y_i} \in \mathtt{tl}(w_i) \wedge y_i^1 \leq y_i^2 \leq \ldots \leq y_i^{p_i}$$

is called a simple *pattern. An abstract value belonging to the abstract domain $\mathcal{A}_{\mathbb{U}}$ parametrized by a set of simple patterns is called a* simple abstract value.

For any simple abstract value $W \in \mathcal{A}_{\mathbb{U}}$, we define $\gamma^{\mathbb{U}}(W) \mid_{fst \cup len}$ as the set which contains, for any $(L,D) \in \gamma^{\mathbb{U}}(W)$, the function

$$f_{(L,D)} : \{\mathtt{hd}(w), \mathtt{len}(w), d \mid w \in \mathcal{V}, d \in \mathit{DVar}\} \to \mathbb{Z},$$

defined by $f_{(L,D)}(\mathtt{hd}(w)) = L(w)[0]$ and $f_{(L,D)}(\mathtt{len}(w)) = |L(w)|$, for any $w \in \mathcal{V} \subseteq \mathit{DWVars}$, and $f_{(L,D)}(d) = D(d)$, for any $d \in \mathit{DVar}$. Also, for any $P \in \mathcal{P}(\mathcal{V})$ as in Definition 6, we define $\gamma^{\mathbb{U}}(W) \mid_P$ as the set which contains, for any $(L,D) \in \gamma^{\mathbb{U}}(W)$, all the functions

$$g_{(L,D)} : \Big(\mathbf{y} \cup \{w_i[y_i^j] \mid 1 \leq i \leq q, 1 \leq j \leq p_i\} \cup \{\mathtt{hd}(w), \mathtt{len}(w), d \mid w \in \mathcal{V}, d \in \mathit{DVar}\}\Big) \to \mathbb{Z},$$

where $\mathbf{y} = \mathbf{y_1} \cup \ldots \cup \mathbf{y_q}$, such that the values assigned to $\mathbf{y}$ by $g_{(L,D)}$ satisfy P, $g_{(L,D)}(w[y]) = L(w)[g_{(L,D)}(y)]$, for any $w \in \{w_1, \ldots, w_q\}$ and $y \in \mathbf{y}$, $g_{(L,D)}(\mathtt{hd}(w)) = L(w)[0]$ and $g_{(L,D)}(\mathtt{len}(w)) = |L(w)|$, for any $w \in \mathcal{V}$, and $g_{(L,D)}(d) = D(d)$, for any $d \in \mathit{DVar}$.

Lemma 3. *If the numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ contains an exact projection operator $\uparrow$ and an exact meet operator $\sqcap^{\mathbb{Z}}$ (i.e., $\gamma^{\mathbb{Z}}(X \sqcap^{\mathbb{Z}} Y) = \gamma^{\mathbb{Z}}(X) \cap \gamma^{\mathbb{Z}}(Y)$) then, for any simple abstract value $W \in \mathcal{A}_{\mathbb{U}}$ of the form*

$$W ::= E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1}, \ldots, \mathbf{y_q}.\, P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P,$$

we have that W is closed iff

1. $\gamma^{\mathbb{U}}(W)\mid_{fst\cup len} = \gamma^{\mathbb{Z}}(E)$,
2. *for any* $P \in \mathcal{P}(DWVars)$ $\gamma^{\mathbb{U}}(W)\mid_P = \gamma^{\mathbb{Z}}(U_P)$,
3. *the abstract values* E, *and* U_P, *for any* $P \in \mathcal{P}(DWVars)$, *are closed.*

Proof: " $\Rightarrow$ " First, we want to prove that $\gamma^{\mathbb{U}}(W)\mid_{fst\cup len} = \gamma^{\mathbb{Z}}(E)$. Since W is closed, by the definition of the abstraction function $\alpha^{\mathbb{U}}$, we obtain that

$$\alpha^{\mathbb{Z}}(\gamma^{\mathbb{U}}(W)\mid_{fst\cup len}) = E. \tag{H}$$

This implies that $\alpha^{\mathbb{Z}}(\gamma^{\mathbb{U}}(W)\mid_{fst\cup len}) \sqsubseteq^{\mathbb{Z}} E$, which by the property of the Galois connection, implies

$$\gamma^{\mathbb{U}}(W)\mid_{fst\cup len} \subseteq \gamma^{\mathbb{Z}}(E).$$

Then, property (H) implies $E \sqsubseteq^{\mathbb{Z}} \alpha^{\mathbb{Z}}(\gamma^{\mathbb{U}}(W)\mid_{fst\cup len})$, which together with $\alpha^{\mathbb{Z}}(\gamma^{\mathbb{Z}}(E)) \sqsubseteq^{\mathbb{Z}} E$ implies

$$\alpha^{\mathbb{Z}}(\gamma^{\mathbb{Z}}(E)) \sqsubseteq^{\mathbb{Z}} \alpha^{\mathbb{Z}}(\gamma^{\mathbb{U}}(W)\mid_{fst\cup len}).$$

The latter, by the property of the Galois connection, implies

$$\gamma^{\mathbb{Z}}(E) \subseteq \gamma^{\mathbb{Z}}(\alpha^{\mathbb{Z}}(\gamma^{\mathbb{U}}(W)\mid_{fst\cup len}))$$

Remember that the formula U_P is defined over the variables in E, therefore from the definition of $\gamma^{\mathbb{U}}$ and by the fact that the projection operator in $\mathcal{A}_{\mathbb{Z}}$ is exact, we have that $\gamma^{\mathbb{U}}(W)\mid_{fst\cup len}$ equals

$$\gamma^{\mathbb{Z}}(E) \cap \bigcap_{P\in\mathcal{P}(\mathcal{V})} \gamma^{\mathbb{Z}}(U_P{\uparrow}\mathbf{y}_P),$$

where $\mathbf{y}_P$ includes the position variables in P and the terms of the form $w[y]$ with y a position variable in P. Thus,

$$\gamma^{\mathbb{Z}}(E) \subseteq \gamma^{\mathbb{Z}}(\alpha^{\mathbb{Z}}(\gamma^{\mathbb{Z}}(E) \cap \bigcap_{P\in\mathcal{P}(\mathcal{V})} \gamma^{\mathbb{Z}}(U_P{\uparrow}\mathbf{y}_P))),$$

which by the fact that $\sqcap^{\mathbb{Z}}$ is exact implies

$$\gamma^{\mathbb{Z}}(E) \subseteq \gamma^{\mathbb{Z}}(\alpha^{\mathbb{Z}}(\gamma^{\mathbb{Z}}(E \sqcap^{\mathbb{Z}} \sqcap^{\mathbb{Z}}_{P\in\mathcal{P}(\mathcal{V})} U_P{\uparrow}\mathbf{y}_P))).$$

Since $\gamma^{\mathbb{Z}} \circ \alpha^{\mathbb{Z}} \circ \gamma^{\mathbb{Z}} = \gamma^{\mathbb{Z}}$, we obtain that

$$\gamma^{\mathbb{Z}}(E) \subseteq \gamma^{\mathbb{Z}}\left(E \sqcap^{\mathbb{Z}} \sqcap^{\mathbb{Z}}_{P\in\mathcal{P}(\mathcal{V})} U_P{\uparrow}\mathbf{y}_P\right).$$

Using again that $\sqcap^{\mathbb{Z}}$ is exact we conclude that

$$\gamma^{\mathbb{Z}}(E) \subseteq \left(\gamma^{\mathbb{Z}}(E) \cap \bigcap_{P\in\mathcal{P}(\mathcal{V})} \gamma^{\mathbb{Z}}(U_P{\uparrow}\mathbf{y}_P)\right) = \gamma^{\mathbb{U}}(W)\mid_{fst\cup len}.$$

The prove of the second fact follows the same ideas.

" $\Leftarrow$ " Let $W' = \alpha^{\mathbb{U}}(\gamma^{\mathbb{U}}(W))$. From the definiton of $\alpha^{\mathbb{U}}, \gamma^{\mathbb{U}}$ follows that

$$W' = \alpha^{\mathbb{Z}}(\gamma^{\mathbb{U}}(W) \mid_{fst \cup len}) \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \alpha^{\mathbb{Z}}(\gamma^{\mathbb{U}}(W) \mid_P).$$

Then, using hypothesises 1 and 2 we obtain that

$$W' = \alpha^{\mathbb{Z}}(\gamma^{\mathbb{Z}}(E)) \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \alpha^{\mathbb{Z}}(\gamma^{\mathbb{Z}}(U_P)).$$

Moreover, using the third hypothesis we conclude that $W' = E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} U_P$, therefore $W' = W$.

□

Next, we describe a procedure which, for some class of abstract values $W \in L^{\mathbb{U}}$ returns closed abstract values having the same concretization as W. This procedure works only for abstract values defined over *simple patterns*. It relies on the existence of a similar procedure for abstract values in $\mathcal{A}_{\mathbb{Z}}$.

The closure procedure Let W be a simple abstract element in $L^{\mathbb{U}}$, with

$$W ::= E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1}, \ldots, \mathbf{y_q}.\, P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P.$$

This procedure builds a closed abstract value, denoted $\mathtt{Cl}(W)$, such that $\gamma^{\mathbb{U}}(\mathtt{Cl(W)}) = \gamma^{\mathbb{U}}(W)$.

If E is unsatisfiable (that is, $\gamma^{\mathbb{Z}}(E)$ is empty) then $\mathtt{Cl}(W)$ is the bottom element $\bot^{\mathbb{U}}$. Otherwise, we define $\mathtt{CL}(W)$ to be the abstract value

$$E^S \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1}, \ldots, \mathbf{y_q}.\, P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P^S,$$

where the numeric abstract values are obtained in the following steps:

Step 1: We set $E^S = E$ and we enforce the quantifier-free part of $\mathtt{Cl}(W)$ from the universal formulas of W. Thus, for any $P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \in \mathcal{P}(\mathcal{V})$, such that $E \sqsubseteq^{\mathbb{Z}} \bigwedge_{i=1}^{q} \mathtt{len}(w_i) > 1$ we apply,

$$E^S = E^S \sqcap^{\mathbb{Z}} (U_P{\downarrow}\, \{\mathtt{len}(w), \mathtt{hd}(w) \mid w \in \mathcal{V}\})$$

Step 2: We set $U_P^S = U_P \sqcap^{\mathbb{Z}} E^S$, for any $P \in \mathcal{P}(\mathcal{V})$. Then, we enforce universal formulas based on other universal formulas. To see the intuition behind this consider the following example. Let $\mathcal{V} = \{w\}$, $\mathcal{P}(\mathcal{V}) = \{(y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 \leq y_2, y \in \mathtt{tl}(w)\}$, and W as follows:

$$\forall y_1, y_2.\ ((y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 \leq y_2) \Rightarrow U(y_1, y_2) \wedge \forall y.\ (y \in \mathtt{tl}(w)) \Rightarrow \top^{\mathbb{Z}},$$

where $U(y_1, y_2) = w[y_1] < w[y_2] \wedge w[y_1] \geq 2 \wedge w[y_2] \leq 5$.

Notice that the sub-formula with two universal variables implies that every data symbol in the word is between 2 and 5. This property can be expressed by a formula with only one universal variable but it is not implied by the second conjunct of W. The procedure `Cl` enforces this sub-formula according to this remark. Thus, it replaces the second conjunct of W by $\forall y.\ ((y) \in \mathtt{tl}(w)) \Rightarrow \left(\top^{\mathbb{Z}} \sqcap^{\mathbb{Z}} U(y)\right)$, where $U(y)$ is a formula deduced from the first conjunct of W. We start from the fact that y_1 and y_2 can be interpreted to all symbols of w starting from the second one till the last one. Consequently, to obtain a property of all symbols starting with the second one we define

$$U(y) ::= \left(((U(y_1,y_2)\uparrow y_2)[y_1 \leftarrow y]) \sqcap^{\mathbb{Z}} ((U(y_1,y_2)\uparrow y_1)[y_2 \leftarrow y])\right).$$

The same approach can be extended to more general patterns. Thus, let $P \in \mathcal{P}(\mathcal{V})$ be a pattern of the form

$$P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q) ::= \bigwedge_{1 \le i \le q} \mathbf{y_i} \in \mathtt{tl}(w_i) \wedge y_i^1 \le y_i^2 \le \ldots \le y_i^{p_i}.$$

In the following, we show how we can enforce U_P^S using the information stored in the other universal formulas. Thus, let $P' \in \mathcal{P}(\mathcal{V})$ be another pattern of the form

$$P'(\mathbf{y'_1}, w'_1 \ldots, \mathbf{y'_{q'}}, w'_{q'}) ::= \bigwedge_{1 \le i \le q'} \mathbf{y'_i} \in \mathtt{tl}(w'_i) \wedge y_i^1 \le y_i^2 \le \ldots \le y_i^{p'_i},$$

such that there exists $1 \le p \le min(q,q')$ with $w_i = w'_i$, for any $1 \le i \le p$ and the length of the words described by the pattern P' is at least 2, i.e. $E \sqsubseteq^{\mathbb{Z}} \bigwedge_{i=1}^{q} \mathtt{len}(w_i) > 1$.
Let Σ be the set of tuples $\sigma = (\sigma_1, \ldots, \sigma_p)$, where $\sigma_i : \mathbf{y'_i} \to \mathbf{y_i}$ maps elements of $\mathbf{y'_i}$ to elements of $\mathbf{y_i}$. We have two cases:

- if $p_i > p'_i$ then σ_i is any injective total mapping,
- if $p_i \le p'_i$ then σ_i maps p_i elements of $\mathbf{y'_i}$ into distinct elements of $\mathbf{y_i}$.

To enforce U_P^S using the information stored in $U_{P'}^S$, we do the following: for any $\sigma = (\sigma_1, \ldots, \sigma_p) \in \Sigma$, if $Y_\sigma \subseteq \mathbf{y'_1} \cup \ldots \cup \mathbf{y'_p}$ is the set of variables not appearing in the domain of some σ_i then we apply

$$U_P^S = U_P^S \sqcap^{\mathbb{Z}} \left(\left(U_{P'}^S \uparrow Y_\sigma\right)[y \leftarrow \sigma(y)]\right).$$

Step 3: For any universal sub-formula corresponding to some pattern P of the form

$$\bigwedge_{1 \le i \le q} \mathbf{y_i} \in \mathtt{tl}(w_i) \wedge y_i^1 \le y_i^2 \le \ldots \le y_i^{p_i},$$

for which $E^S \sqsubseteq^{\mathbb{Z}} \mathtt{len}(w_i) <= 1$, for some $1 \le i \le q$, we define $U_P^S = \bot^{\mathbb{Z}}$. This is possible because the right part can be any element of $\mathcal{A}_{\mathbb{Z}}$, even $\bot^{\mathbb{Z}}$, since the guard has only the empty model.

Step 4: For any universal sub-formula not considered in Step 3, corresponding to some pattern P of the form

$$\bigwedge_{1 \le i \le q} \mathbf{y_i} \in \mathtt{tl}(w_i) \wedge y_i^1 \le y_i^2 \le \ldots \le y_i^{p_i},$$

we apply

$$U_P^S = U_P^S \sqcap^{\mathbb{Z}} \left(\bigwedge_{\substack{1 \le i \le q \\ 1 \le j \le p_i}} 1 \le y_i^j \le \mathtt{len}(w_i) \right).$$

Step 5: For any universal sub-formula corresponding to some pattern P, we apply on U_P^S the canonization procedure from the numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$.

The following theorem states the correctness of the procedure above.

Theorem 2. *Let $W \in L^{\mathbb{U}}$ be a simple abstract value. Then, $\mathtt{Cl}(W)$ is a closed abstract value with $\gamma^{\mathbb{U}}(\mathtt{Cl}(W)) = \gamma^{\mathbb{U}}(W)$.*

Remark 2. The procedure above can not be extended to more general patterns. Thus, if we consider patterns of the form

$$\bigwedge_{1 \le i \le q} \mathbf{y_i} \in \mathtt{tl}(w_i) \wedge y_i^1 < y_i^2 < \ldots < y_i^{p_i}$$

then some of the universal properties depend on the length of the words. For example, let $\mathcal{V} = \{w\}$, $\mathcal{P}(\mathcal{V}) = \{(y_1, y_2) \in w \wedge y_1 < y_2, y_1 \in w\}$, and W be the abstract element

$$\begin{gathered} \mathtt{hd}(w) = 0 \wedge \mathtt{len}(w) = 6 \\ \wedge \forall y_1.\ (y_1) \in \mathtt{tl}(w) \Rightarrow (1 \le w[y_1] \le 5) \\ \wedge \forall y_1, y_2.\ ((y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 < y_2) \Rightarrow w[y_1] < w[y_2]. \end{gathered}$$

Notice that the two universal formulas in W and the fact that the length of w is 6 induce the following property

$$\forall y_1.\ (y_1) \in \mathtt{tl}(w) \Rightarrow (w[y_1] = y_1) \tag{I}$$

and consequently, a smaller abstract value than W (w.r.t. $\sqsubseteq^{\mathbb{U}}$) is

$$\begin{gathered} \mathtt{hd}(w) = 0 \wedge \mathtt{len}(w) = 6 \\ \wedge \forall y_1.\ (y_1) \in \mathtt{tl}(w) \Rightarrow (w[y_1] = y_1 \wedge 1 \le w[y_1] \le 5) \\ \wedge \forall y_1, y_2.\ ((y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 < y_2) \Rightarrow w[y_1] < w[y_2]. \end{gathered}$$

The procedure that we have presented uses the projection from $\mathcal{A}^{\mathbb{Z}}$, which in this case cannot induce a relation between positions and data.

If we consider the patterns describing consecutive positions, i.e. of the form

$$\bigwedge_{1 \le i \le q} \mathbf{y_i} \in \mathtt{tl}(w_i) \wedge y_i^1 <_1 y_i^2 <_1 \ldots <_1 y_i^{p_i}$$

we encounter a similar difficulty. For example, let $\mathcal{V} = \{w\}$, $\mathcal{P}(\mathcal{V}) = \{(y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 <_1 y_2, y_1 \in w\}$, and W be the abstract element

$$\begin{gathered} \mathtt{hd}(w) = 0 \wedge \mathtt{len}(w) = 6 \\ \wedge \forall y_1.\ (y_1) \in \mathtt{tl}(w) \Rightarrow (1 \le w[y_1] \le 5) \wedge \forall y_1, y_2.\ ((y_1, y_2) \in \mathtt{tl}(w) \wedge y_1 <_1 y_2) \Rightarrow w[y_2] = w[y_1] + 1. \end{gathered}$$

As in the previous case, the universal formulas and the fact that the length of w is 6 induce the following property

$$\forall y_1.\ (y_1) \in \mathtt{tl}(w) \Rightarrow (w[y_1] = y_1)$$

which is not obtaind from projections since it does not take into consideration the length of w and the succesor relation between y_1 and y_2.

In the following we formally define the abstract transformers. To obtain the precision results (best abstract transformers) we suppose that last step of their definition consists in applying the procedure of $\mathcal{A}^{\mathbb{Z}}$ that computes closed abstract values, for all abstract elements in $\mathcal{A}^{\mathbb{Z}}$ that define the current abstract element of $\mathcal{A}^{\mathbb{U}}$.

The projection operator First, we give the definition of the projection operator and then we present correctness and precision results.

Let $\widetilde{W}$ be an element in $\mathcal{A}_{\mathbb{U}}$ of the following form:

$$E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\ P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P.$$

For any data word variable w, we want to define an abstract element $\mathtt{proj}^{\#}(\widetilde{W}, w)$ which contains no reference of w such that its concretization is an over-approximation of the concretization of $\widetilde{W}$ when considering only words not denoted by w. We start by projecting out from E the variables $\mathtt{len}(w)$ and $\mathtt{hd}(w)$ corresponding to the length and the first symbol of w. Next, we consider only the universally quantified conjuncts over data words variables whose lenght is strictly greather then one:

$$\forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\ P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P \text{ such that } E \sqsubseteq^{\mathbb{Z}} \bigwedge_{i=1}^{q} \mathtt{len}(w_i) > 1. \qquad \text{(J)}$$

Then with respect to the these universal formulas, we do the following:

- for any universal sub-formula as in (J) containing the data word variable w, we project out from the right part of the implication, U_P, the position variables $\mathbf{y}$ and the terms build over the variable w, $\mathtt{len}(w)$ and $\mathtt{hd}(w)$, and then we apply the meet operator in $\mathcal{A}_{\mathbb{Z}}$ between the obtained abstract value and the quantifier-free part of $\widetilde{W}$. Formally, we apply

$$E = E \sqcap^{\mathbb{Z}} (U_P{\uparrow}(\mathbf{y} \cup \{w[y] \mid y \in \mathbf{y}\} \cup \{\mathtt{len}(w), \mathtt{hd}(w)\}))\,.$$

- for any sub-formula of $\widetilde{W}$ as in (J) of the form:

$$\forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\ \left(\bigwedge_{1 \leq i \leq q} P_R^i(\mathbf{y}_i, w_i) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \right) \Rightarrow U_P, \qquad \text{(K)}$$

such that $w \neq w_i$, for all $1 \leq i \leq q$, and $w \in \mathbf{w}$, let P' be the pattern

$$\bigwedge_{1 \leq i \leq q} P_R^i(\mathbf{y}_i, w_i) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \uparrow \mathtt{len}(w),$$

where $P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \uparrow \mathtt{len}(w)$ is the quantifier-free Presburger formula corresponding to $\exists \mathtt{len}(w).\ P_L(y_1^1, \ldots, y_q^1, \mathbf{w})$. If there exists $P'' \in \mathcal{P}(\mathcal{V} \setminus \{w\})$ with the same number of data words variables as P' and the same number of position variables on each data word such that $P'' \Rightarrow P'$ ($\Rightarrow$ is the usual implication between quantifier-free Presburger formulas) then we modify the universal formula corresponding to P'' by

$$U_{P''} = U_{P''} \sqcap^{\mathbb{Z}} (U_P \uparrow \{\mathtt{len}(w), \mathtt{hd}(w)\}).$$

– for any sub-formula of $\widetilde{W}$ of the form:

$$\forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\ \left(\bigwedge_{1 \leq i \leq q} P_R^i(\mathbf{y}_i, w_i) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \right) \Rightarrow U_P, \tag{L}$$

such that $w = w_j$, for some $1 \leq j \leq q$, let P' be the pattern

$$\bigwedge_{1 \leq i \leq q, i \neq j} P_R^i(\mathbf{y}_i, w_i) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \uparrow \{\mathtt{len}(w), y_j^1\}.$$

If there exists $P'' \in \mathcal{P}(DWVars \setminus \{w\})$ with the same number of data words variables as P' and the same number of position variables on each data word such that $P'' \Rightarrow P'$ then we modify the universal formula corresponding to P'' by

$$U_{P''} = U_{P''} \sqcap^{\mathbb{Z}} (U_P \uparrow (\mathbf{y}_j \cup \{w[y] \mid y \in \mathbf{y}_j\} \cup \{\mathtt{len}(w), \mathtt{hd}(w)\})).$$

The correctness and the precision of the projection operator are proved in the following results.

Theorem 3. *Let $\mathcal{A}_{\mathbb{U}}$ be as above such that the numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ contains a sound projection operator and an exact meet operator. For any $\widetilde{W}$ in $\mathcal{A}_{\mathbb{U}}$, the following holds*

$$\alpha^{\mathbb{U}}\left(\{(\mathtt{proj}(L,w), D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) \sqsubseteq^{\mathbb{U}} \mathtt{proj}^{\#}(\widetilde{W}, w).$$

Theorem 4. *Let $\mathcal{A}_{\mathbb{U}}$ be as above such that the numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ contains a sound projection operator and an exact meet operator. If the projection operator in $\mathcal{A}_{\mathbb{Z}}$ is exact then*

$$\alpha^{\mathbb{U}}\left(\{(\mathtt{proj}(L,w), D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) = \mathtt{proj}^{\#}(\widetilde{W}, w),$$

for any closed abstract value $\widetilde{W}$.

The abstract transformer $\texttt{sglt}^{\#}$ Let $\widetilde{W}$ be an element in $L^{\mathbb{U}}$ of the form

$$E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\, P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P.$$

The output of $\texttt{sglt}^{\#}(\widetilde{W}, x)$ is

$$E' \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V} \cup x)} \forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\, P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P,$$

where E' is obtained from E by adding two dimensions for $\texttt{len}(x)$ and $\texttt{hd}(x)$ and by applying $E' = E \sqcap^{\mathbb{Z}} \texttt{len}(x) = 1$, and for every $P \in \mathcal{P}(\mathcal{V} \cup x) \setminus \mathcal{P}(\mathcal{V})$, $U_P = \bot^{\mathbb{U}}$.

Theorem 5. *Let $\mathcal{A}_{\mathbb{U}}$ be as above such that the numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ contains an exact meet operator. For any $\widetilde{W}$ in $\mathcal{A}_{\mathbb{U}}$, the following holds*

$$\alpha^{\mathbb{U}}\left(\{(\texttt{sglt}\,(L,x)\,,D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) \sqsubseteq^{\mathbb{U}} \texttt{sglt}^{\#}(\widetilde{W}, x).$$

Theorem 6. *Let $\mathcal{A}_{\mathbb{U}}$ be as above such that the numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ contains an exact meet operator. If the abstract transformer in $\mathcal{A}_{\mathbb{Z}}$ corresponding to assignments $x = 1$, where x is an integer variable, is exact then,*

$$\alpha^{\mathbb{U}}\left(\{(\texttt{sglt}\,(L,x)\,,D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) = \texttt{sglt}^{\#}(\widetilde{W}, x),$$

for any closed abstract value $\widetilde{W}$.

The abstract transformer $\texttt{isSglt}^{\#}$ Let $\widetilde{W}$ be an element in $\mathcal{A}_{\mathbb{U}}$. Then, $\texttt{isSglt}^{\#}(\widetilde{W}, x)$ returns 1 if

$$E \sqsubseteq^{\mathbb{Z}} l[x] = 1,$$

it returns 0 if $E \sqsubseteq^{\mathbb{Z}} l[x] > 1$, and -1, otherwise.

The abstract transformer $\texttt{updateFirst}^{\#}$ Let $\widetilde{W}$ be an element in $\mathcal{A}_{\mathbb{U}}$ of the following form

$$E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\, P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P,$$

$x \in \mathcal{V} \cup DVar$, dt a data expression, and $\beta : PVar \rightarrow DWVars$.

Let $dt\beta$ be the expression dt in which $p \rightarrow data$ is replaced by $\texttt{hd}(\beta(p))$. The abstract value $\texttt{updateFirst}^{\#}(\widetilde{W}, x, dt, \beta)$ is obtained from $\widetilde{W}$ by replacing the quantifier-free part with

$$\texttt{update}^{\#}(E, \texttt{hd}(x) = dt\beta), \text{ if } x \in \mathcal{V}, \text{ or}$$
$$\texttt{update}^{\#}(E, x = dt\beta), \text{ if } x \in DVar$$

and for each pattern $p \in \mathcal{P}(\mathcal{V})$, the corresponding abstract element from $\mathcal{A}_{\mathbb{Z}}$ is

$$\texttt{update}^{\#}(U_P, \texttt{hd}(x) = dt\beta) \text{if } x \in \mathcal{V}, \text{ or}$$
$$\texttt{update}^{\#}(U_P, x = dt\beta), \text{ if } x \in DVar.$$

where $\texttt{update}^{\#}$ is the abstract transformer in $\mathcal{A}_{\mathbb{Z}}$ corresponding to assignments.

The next results prove the correctness of the abstract transformer $\texttt{updateFirst}^{\#}$ and identify conditions under which it is a best abstract transformer.

Theorem 7. *Let $\widetilde{W}$ be an element in $\mathcal{A}_{\mathbb{U}}$ such that the numerical abstract domain $\mathcal{A}^{\mathbb{Z}}$ contains a sound assignment operator, then the following holds*

$$\alpha^{\mathbb{U}}\left(\{(\texttt{updateFirst}(L,x,dt,\beta),D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) \sqsubseteq^{\mathbb{U}} \texttt{updateFirst}^{\#}(\widetilde{W},x,dt,\beta).$$

Theorem 8. *Let $\widetilde{W}$ be an element in $\mathcal{A}_{\mathbb{U}}$ such that the numerical abstract domain $\mathcal{A}^{\mathbb{Z}}$ contains a sound assignment operator. If the assignment abstract transformer in $\mathcal{A}_{\mathbb{Z}}$ is exact for data expressions of the form dt then*

$$\alpha^{\mathbb{U}}\left(\{(\texttt{updateFirst}(L,x,dt,\beta),D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) =^{\mathbb{U}} \texttt{updateFirst}^{\#}(\widetilde{W},x,dt,\beta),$$

for any closed abstract value $\widetilde{W}$.

The abstract transformer $\texttt{split}^{\#}$ The procedure $\texttt{split}^{\#}(\widetilde{W},x,z)$ with $\widetilde{W} \in \mathcal{A}_{\mathbb{U}}$ splits the data word represented by x into a word representing its first symbol and a word representing its tail. In the output, the first word is represented by x and the second one by z (z is not a variable in $\widetilde{W}$). First, the abstract values from $\mathcal{A}_{\mathbb{Z}}$ are updated such that the length of x becomes 1 and the length of z is $\texttt{len}(x)-1$. Then, the universally quantified formulas in $\widetilde{W}$ that characterize the tail of x should be removed. Before doing this, we use them to generate (1) relations between the first symbol of z and the first symbol of other words which are used to straighten the quantifier-free part, and (2) universal formulas that characterize the tail of z. We begin by an example and then we give the formal definition.

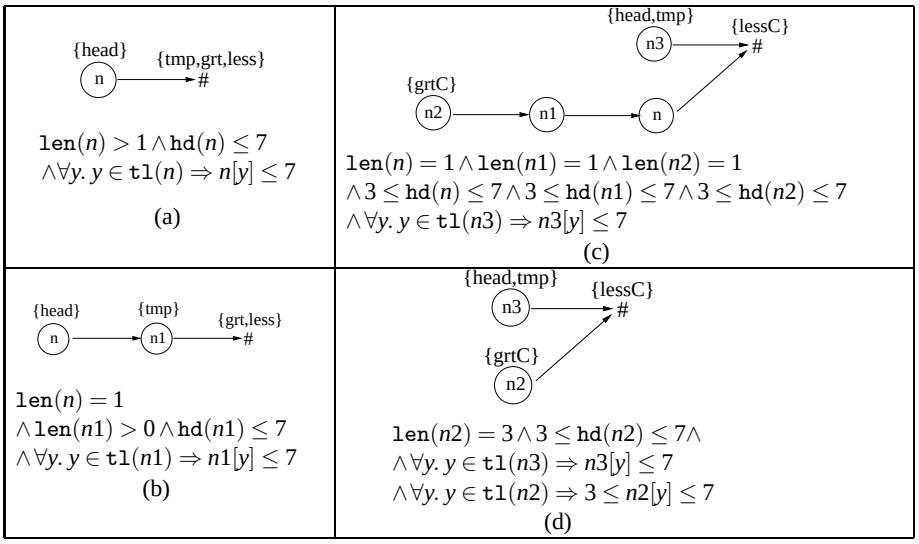


Fig. 8. Abstract heaps for the procedure `Dispatch3`

Example 3. Suppose that we analyze the procedure `Dispatch3` from Section 1 using the abstract domain of 1-abstract heap sets (we allow abstract heaps with at most one simple node) parametrized by the $\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{U}}$ over the set of patterns $\mathcal{P} = \{y \in w\}$, the Polyhedra domain, and $k = 1$. Also, suppose that the initial state is described by the k-abstract heap in Figure 8(a). Then, during the first iteration of the loop, the output of $\texttt{tmp} = \texttt{head} \rightarrow \texttt{next}$ is pictured in Figure 8(b). The abstract value from $\mathcal{A}_{\mathbb{U}}$ is obtained by applying $\texttt{split}^{\#}(\widetilde{W}, n, n1)$, where $\widetilde{W}$ is the formula in Figure 8(a). The constraint $\texttt{hd}(n1) \leq 7$ and the universal formula from $\texttt{split}^{\#}(\widetilde{W}, n, n1)$ are implied by the universal formula in $\widetilde{W}$.

Let $\widetilde{W}$ be an element in $\mathcal{A}_{\mathbb{U}}$ of the following form

$$E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\, P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U_P.$$

Updating the quantifier-free part: For every $P \in \mathcal{P}(\mathcal{V})$ of the form

$$P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w}) ::= \bigwedge_{1 \leq i \leq q} P_R^i(\mathbf{y}_i, w_i) \wedge P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \qquad \text{(M)}$$

such that $w_j = x$, for some $1 \leq j \leq q$, if

$$P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \wedge E \wedge y_j^1 = 1$$

is a satisfiable Presburger formula, we define E'_{U_P}:

$$E'_{U_P} = \left(U_P \uparrow \left(\mathbf{y_1} \cup \ldots \cup \left(\mathbf{y_j} \setminus \{y_j^1\}\right) \cup \ldots \cup \mathbf{y_q}\right)\right)\left[w_j[y_j^1] \leftarrow \texttt{hd}(z)\right].$$

Let E'' be the greatest lower bound in $\mathcal{A}_{\mathbb{Z}}$ between the value E in $\widetilde{W}$ to which we have added a new dimension for $\texttt{hd}(z)$ and all the values E'_{U_P} associated to patterns $P \in \mathcal{P}(DWVars)$ speaking about the data word x.

The quantifier-free part of the abstract value outputted by $\texttt{split}^{\#}(\widetilde{W}, x, z)$ is obtained by adding to E'' one more dimension for $\texttt{len}(z)$ and by applying the following:

$$E'' = E''[\texttt{len}(x) \leftarrow \texttt{len}(z) + 1]$$
$$E'' = \texttt{update}^{\#}(E'', \texttt{len}(x) := 1),$$

where $\texttt{update}^{\#}$ is the abstract transformer in $\mathcal{A}_{\mathbb{Z}}$ corresponding to assignments.

Updating universal formulas: With respect to universal formulas, we preserve the ones that do not characterize the tail of the data word denoted by x or that do not use the term $\texttt{len}(x)$ for the length of x. An universal formula characterizing the data of the words $w_1,\ldots,w_q$ with $w_j = x$, for some $1 \leq j \leq q$, may imply an universal formula characterizing the data of $w_1,\ldots,w_{j-1},w_{j+1},\ldots,w_q$ or a formula characterizing the data of $w_1,\ldots,w_{j-1},z,w_{j+1},\ldots,w_q$.

By this transformer, we add the data word variable z and consequently, the outputted abstract value will contain new universal formulas corresponding to patterns $P(\mathbf{y_1}, w_1 \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \in DWVars$ with $z = w_j$, for some $1 \leq j \leq q$, or $z \in \mathbf{w}$. This set of

patterns is denoted $\mathcal{P}'$. These universal formulas are deduced from universal formulas speaking about the data word x as follows. To start, we add to $\widetilde{W}$ universal formulas

$$\forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\ P(\mathbf{y_1}, w_1, \ldots, \mathbf{y_q}, w_q, \mathbf{w}) \Rightarrow U'_P,$$

with $U'_P = \top^{\mathbb{Z}}$, for any $P \in \mathcal{P}'$, and $U_{P'} = U_P$, otherwise. Then, we consider all patterns speaking about x and eventually, we modify all the universal formulas defined above.

Therefore, for every $P \in \mathcal{P}(\mathcal{V})$ like in (M) such that $w_j = x$, for some $1 \leq j \leq q$, we do the following:

- if $P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \wedge E \wedge y_j^1 = 1$ is a satisfiable Presburger formula then, using the universal formula corresponding to P, we may find relations between the first symbol of the word z, $\mathtt{hd}(z)$ and the tail of other words (including the word denoted by z). Let P' be the pattern

$$P' ::= \bigwedge_{1 \leq i \leq q, i \neq j} P_R^i(\mathbf{y}_i, w_i) \wedge P_R^j(\mathbf{y}_j \setminus \{y_j^1\}, z) \wedge \left(\left(P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \uparrow y_j^1 \right) [\mathtt{len}(x) \leftarrow \mathtt{len}(z) + 1] \right),$$

  where $P_R^j(\mathbf{y}_j \setminus \{y_j^1\}, z)$ is obtained from $P_R^j(\mathbf{y}_j, z)$ by forgetting the position variable y_j^1 (if $|\mathbf{y_j}| = 1$ then we delete P_R^j). If there exists $P'' \in \mathcal{P}(\mathcal{V} \cup \{z\})$ with the same number of data words variables as P' and the same number of position variables on each data word such that $P'' \Rightarrow P'$ ($\Rightarrow$ is the usual implication between quantifier-free Presburger formulas) then we modify the abstract value $U'_{P''}$ corresponding to P'' by

$$U'_{P''} = U'_{P''} \sqcap^{\mathbb{Z}} \left(U_P \begin{bmatrix} x[y_j^1] \leftarrow \mathtt{hd}(z), \\ y_j^k \leftarrow y_j^k + 1 \\ x[y_j^k] \leftarrow z[y_j^k] \\ y_j^1 \leftarrow 1 \end{bmatrix} \right).$$

- if $P_L(y_1^1, \ldots, y_q^1, \mathbf{w}) \wedge E \wedge y_j^1 > 1$ is a satisfiable Presburger formula then, using the universal formula corresponding to P, we may find relations between the tail of the word z and the tail of other words. Thus, let P' be the pattern

$$P' ::= \bigwedge_{1 \leq i \leq q, i \neq j} P_R(\mathbf{y}_i, w_i) \wedge P_R(\mathbf{y}_j, z) \wedge \left(P_L\left(y_1^1, \ldots, y_q^1, \mathbf{w}\right) [\mathtt{len}(x) \leftarrow \mathtt{len}(z) + 1] \right).$$

  If there exists $P'' \in \mathcal{P}(\mathcal{V} \cup \{z\})$ with the same number of data words variables as P' and the same number of position variables on each data word such that $P'' \Rightarrow P'$ then we modify the output universal formula corresponding to P'' by

$$U'_{P''} = U'_{P''} \sqcap^{\mathbb{Z}} \left(U_P \left[y_j^k \leftarrow y_j^k + 1, x[y_j^k] \leftarrow z[y_j^k] \right] \right)$$

Finally, if the term $\mathtt{len}(x)$ appears in universal formula then this term should be substituted with $\mathtt{len}(z) + 1$.

The next results prove the correctness of the abstract transformer $\mathtt{split}^{\#}$ and identify conditions under which it is a best abstract transformer.

Theorem 9. *Let $\mathcal{A}_{\mathbb{U}}$ be an abstract domain as above parametrized by a set of patterns $\mathcal{P}$ and by a numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ which contains a sound projection operator and an exact meet operator. For any $\widetilde{W}$ in $\mathcal{A}_{\mathbb{U}}$, the following holds*

$$\alpha^{\mathbb{U}}\left(\{(\texttt{split}\,(L,x,z),D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) \sqsubseteq^{\mathbb{U}} \texttt{split}^{\#}(\widetilde{W},x,z).$$

Theorem 10. *Let $\mathcal{A}_{\mathbb{U}}$ be an abstract domain as above parametrized by a set of patterns $\mathcal{P}$ and by a numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ which contains a sound projection operator and an exact meet operator. If (1) $\mathcal{P}$ contains only simple patterns (2) the projection operator, $\sqsubseteq^{\mathbb{Z}}$, and the abstract transformer in $\mathcal{A}_{\mathbb{Z}}$ corresponding to assignments $x = z-1$, where x,z are integer variables, are exact then,*

$$\alpha^{\mathbb{U}}\left(\{(\texttt{split}\,(L,x,z),D) \mid (L,D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) = \texttt{split}^{\#}(\widetilde{W},x,z),$$

for any $\widetilde{W}$ a closed abstract value.

The abstract transformer $\texttt{concat}^{\#}$**:** Let $V_1,\ldots,V_t$ be vectors of data word variables. Then, $\texttt{concat}^{\#}(\widetilde{W},V_1,\ldots,V_t)$ transforms $\widetilde{W}$ such that the first variable in V_i will represent a word which is the concatenation of the words denoted by the variables in V_i, for every $1 \leq i \leq t$. There are three main steps in the definition of this operation: (1) using the constraints from the quantifier-free part, we identify the maximal sub-vectors of V_i containing only variables which represent singletons (words of length 1), (2) we replace the constraints from the quantifier-free part characterizing these singletons with universally quantified formulas that describe abstractions of the concatenations of these sequences, and (3) we apply transformations on the current formula which correspond to concatenations of words of length strictly greater than 1. In the third step, we replace constraints in the form of universally quantified formulas with new universally quantified formulas that describe abstractions of concatenations. In order to be precise during the third step, we have to consider closed sets of patterns. We start be a couple of examples and then, we give the formal definition of this abstract transformer.

Example 4 (Concatenating sequences of singletons). Suppose that we continue the analysis from Example 3. After several iterations of the loop, we obtain the abstract heap in Figure 8(d) which is obtained by applying $\texttt{Normalize}_k^{\#}$ on the abstract heap in Figure 8(c). The formula in Figure 8(d) is obtained by applying $\texttt{concat}^{\#}(\widetilde{W}',(n2,n1,n))$, where $\widetilde{W}'$ is the formula in Figure 8(c). Since the length of the words denoted by $n2$, $n1$, and n is 1, we have to apply the second step in $\texttt{concat}^{\#}$. We search for a formula $\forall y.\ y \in \texttt{tl}(n2) \Rightarrow U$ which holds when $n2$ is interpreted to the concatenation of the singletons represented by $n2$, $n1$, and n. Since the length of the concatenation is 3, there are only two values for y such that $y \in \texttt{tl}(n2)$, $y = 1$ and $y = 2$. To these values we associate two abstract values U_1 and U_2 obtained from the quantifier-free part of $\widetilde{W}'$ (1) by substituting $\texttt{hd}(n1)$ and $\texttt{hd}(n2)$, respectively, by $n2[y]$, (2) by updating $\texttt{len}(n2)$ to 3, and (3) by projecting out terms containing data word variables in $\{n1,n2\}$ (using a projection operator defined in $\mathcal{A}_{\mathbb{Z}}$. In this case, we obtain two identical formulas U_{γ_1} and U_{γ_2} of the form $3 \leq n2[y] \leq 7 \wedge 3 \leq n2[0] \leq 7 \wedge l[n2] = 3$, and we define $U = U_{\gamma_1} \sqcup^{\mathbb{Z}} U_{\gamma_2}$.

Example 5 (Concatenating words of length greater than 1). Suppose that we analyze the procedure `Fibonacci` from Section 1 using the abstract domain of 3-abstract heap sets parametrized by $\mathcal{A}_{\mathbb{U}}$ over $\mathcal{P} = \texttt{Closure}(P,3)$, where $P := (y_1,y_2,y_3) \in \texttt{tl}(n) \wedge y_1 <_1 y_2 <_1 y_3$, the Polyhedra numerical domain, and $k = 3$. The analysis starts from an initial state in which `head` points to a non-empty list. After executing some iterations of the loop, we obtain an abstract heap having 7 nodes in a row, n_i, $1 \leq i \leq 6$, and # such that n_1 and n_6 are pointed by the program variables `head` and `x`, resp. We apply $\texttt{Normalize}_k^{\#}$ which calls $\texttt{concat}^{\#}(\widetilde{W},(n_1,n_2,n_3,n_4,n_5))$, where $\widetilde{W}$ is the formula in $L^{\mathbb{U}}$ associated to this abstract heap. The formula $\widetilde{W}$ is a conjunction between the quantifier-free part

$$E := \texttt{len}(n_1) = 5 \wedge \texttt{hd}(n_1) = 1 \wedge \texttt{hd}(n_2) = 8 \wedge \texttt{hd}(n_3) = 13 \wedge \texttt{hd}(n_4) = 21 \\ \wedge \texttt{hd}(n_5) = 34 \wedge m1 = 13 \wedge m2 = 21$$

and some universally-quantified formulas, including

$$\forall y_1,y_2,y_3.\ ((y_1,y_2,y_3) \in \texttt{tl}(n_1) \wedge y_1 <_1 y_2 <_1 y_3) \Rightarrow (n_1[y_3] = n_1[y_1] + n_1[y_2]).$$

We identify the sub-vector of variables (n_2,n_3,n_4,n_5) representing singletons and we apply the second step in $\texttt{concat}^{\#}$. Consequently, the data words variables n_3, n_4, and n_5 are removed and new universally quantified formulas are added corresponding to the patterns in $\mathcal{P}$ and the data word variable n_2. Now, the word represented by n_1 satisfies the same constraints as in $\widetilde{W}$ and the word represented by n_2 is an abstraction of the concatenation of the singletons denoted by n_2, n_3, n_4, and n_5. One of formulas generated during the final step of $\texttt{concat}^{\#}$ has the form $\forall y_1,y_2,y_3.\ P \Rightarrow U$. The value U is the join of several numerical abstract values representing properties of three consecutive positions on the concatenation of the words denoted by n_1 and n_2. These abstract values are obtained using the quantifier-free part and the abstract values from the formulas associated to the patterns in $\texttt{Closure}(P,n_1n_2)$ from Example 2. For example, the abstract value representing the property of the last symbol in n_1 and the first two symbols in n_2 is the meet between the abstract value associated to $P_1(n_1)$, the quantifier-free part in which $\texttt{hd}(n_1)$ is substituted by $n[y_2]$, and the right part associated to $P_2(n_2)$.

Let $\widetilde{W}$ be an abstract element of the form

$$E \wedge \bigwedge_{P \in \mathcal{P}(\mathcal{V})} \forall \mathbf{y_1} \ldots \forall \mathbf{y_q}.\ P(\mathbf{y_1},\ldots,\mathbf{y_q}) \Rightarrow U_P.$$

The three steps of the procedure $\texttt{concat}^{\#}(\widetilde{W},V_1,\ldots,V_t)$ are defined as follows:

Step 1: we begin by identifying maximal sub-vectors $\mathbf{v} = v_1 \ldots v_r$, $r > 1$, of variables from V_i, for any $1 \leq i \leq t$, such that each component of these sub-vectors represents a data word of length one, i.e. $E \sqsubseteq^{\mathbb{Z}} l[v_j] = 1$, for all $1 \leq j \leq r$. We denote by *SgVars* the subset of *DWVars* that contains, for every such sub-vector, all the elements without the first one.

Step 2: For each sub-vector $\mathbf{v} = v_1 \ldots v_r$ identified in the previous step, we apply to E and all U_P, $P \in \mathcal{P}(\mathcal{V})$, the abstract transformer corresponding to the assignment $l[w] = r$ (since each word represented by some v_i is of length 1, then the word representing the concatenation has the length r).

We continue by searching for universally quantified properties which hold over these sub-vectors. We begin by considering universal properties that describe only these sub-vectors and then, we will consider properties that relate these sub-vectors to other words in $\widetilde{W}$.

For every q-tuple $(\mathbf{v}_1,\ldots,\mathbf{v}_q)$ of sub-vectors as above, where $\mathbf{v}_i = (v_i^0,\ldots,v_i^{t_i})$, for any $1 \leq i \leq q$, and for every pattern $P(\mathbf{y}_1, v_1^0, \ldots, \mathbf{y}_q, v_q^0, \mathbf{w}) \in \mathcal{P}$ with $|\mathbf{y}_i| \leq |\mathbf{v}_i| - 1$, for any $1 \leq i \leq q$, we want to discover a universal property of the form

$$\forall \mathbf{y}_1 \ldots \forall \mathbf{y}_q.\, P(\mathbf{y}_1, v_1^0, \ldots, \mathbf{y}_q, v_q^0, \mathbf{w}) \Rightarrow U_P,$$

which is true when for any $1 \leq i \leq q$, v_i^0 will be interpreted to the word $\mathtt{hd}(v_i^0)\ldots\mathtt{hd}(v_i^{t_i})$. Each v_i^0 will represent the concatenation of the words in $\mathbf{v}_i$.

To this, let Π_P be the set of all possible mappings $\pi : \mathbf{y_1} \cup \ldots \cup \mathbf{y_q} \to \mathbb{N}^+$ between position variables in P and positions in the tail of the words represented by v_1^0,...,v_q^0 defined above. Thus, the variables in $\mathbf{y_i}$ are mapped by any $\pi \in \Pi_P$ to values from the interval $[1, t_i]$, for all $1 \leq i \leq q$, such that E implies

$$P(\mathbf{y}_1, v_1^0, \ldots, \mathbf{y}_q, v_q^0, \mathbf{w})\,[y \leftarrow \pi(y) \mid y \in \mathbf{y_1} \cup \ldots \cup \mathbf{y_q}].$$

For each $\pi \in \Pi_P$ we denote by E_π the abstract element obtained from E in two steps:

- we introduce terms denoting symbols in the new words represented by v_1^0,...,v_q^0 at positions represented by the position variables in P according to the mapping π. Thus, if $\pi(y_i^j) = s$, for some $1 \leq i \leq q$ and $1 \leq j \leq |\mathbf{y}_i|$, then we substitute the term denoting the only symbol in v_i^s, $\mathtt{hd}(v_i^s)$, by the term $v_i^0[y_i^j]$;
- we project out all terms containing variables in *SgVars*.

Formally, E_π is

$$\left(E\left[\mathtt{hd}(v_i^{\pi(y_i^j)}) \leftarrow w_i[y_i^j] \mid 1 \leq i \leq q, 1 \leq j \leq |\mathbf{y_i}|\right]\right) \upharpoonright \{\mathtt{len}(v), \mathtt{hd}(v) \mid v \in \mathit{SgVars}\}.$$

Then, we define the abstract element $U^P \in \mathcal{A}_\mathbb{Z}$ mentioned above by

$$U^P = \bigsqcup_{\pi \in \Pi_P}^{\mathbb{Z}} E_\pi.$$

Now, we continue by searching for universal properties that relate words obtained by concatenation as above to other words described by $\widetilde{W}$. W.l.o.g. we suppose that $P \in \mathcal{P}(\mathcal{V})$ is a pattern that speaks about a set of words w_1,...,w_q not in the sub-vectors above and a set of words v_1^0,...,v_r^0 obtained by concatenation, of the following form:

$$\bigwedge_{1 \leq i \leq q} P_R^i(\mathbf{y}_i, w_i) \wedge \bigwedge_{1 \leq i \leq r} Q_R^i(\mathbf{x}_i, v_i^0) \wedge P_L(y_1^1, \ldots, y_q^1, x_1^1, \ldots, x_r^1, \mathbf{w}),$$

where $\mathbf{w} \subseteq \mathit{DWVars} \setminus \mathit{SgVars}$.

If P_L is a conjunction between a formula $P_L^1(y_1^1,\ldots,y_q^1,\mathbf{w})$ and a formula $P_L^2(x_1^1,\ldots,x_r^1,\mathbf{w})$ and if $\mathcal{P}(\mathcal{V})$ contains a pattern

$$P'(\mathbf{y}_1,w_1,\ldots,\mathbf{y}_q,w_q,\mathbf{w}) ::= \bigwedge_{1\le i\le q} P_R^i(\mathbf{y}_i,w_i)\wedge P_L^1(y_1^1,\ldots,y_q^1,\mathbf{w}),$$

then let $\Pi_{P''}$ be the set of mappings $\pi:\mathbf{x}_1\cup\ldots\cup\mathbf{x}_r\to\mathbb{N}^+$ defined as above for the pattern

$$P''(\mathbf{x}_1,v_1^0,\ldots,\mathbf{x}_r,v_r^0,\mathbf{w}) ::= \bigwedge_{1\le i\le r} Q_R^i(\mathbf{x}_i,v_i^0)\wedge P_L''(x_1^1,\ldots,x_r^1,\mathbf{w}).$$

We define the abstract element U_P corresponding to P by:

$$U^P = \bigsqcup_{\pi\in\Pi_{P''}}^{\mathbb{Z}} EU_\pi,$$

where EU_π is similar to E_π except the fact that all the substitutions and projections are applied to $E\sqcap^{\mathbb{Z}}U_{P'}$ instead of E.
After deducing all the new universally quantified properties we modify E by projecting out all variables in *SgVars*.

Step 3: Suppose that, for any $1\le i\le t$, V_i' is the vector of data word variables obtained from V_i by replacing each sub-vector $\mathbf{v}=v_1\ldots v_r$ considered in the first step with the data word variable v_1.
For any $1\le i\le t$, we apply to E and U_P, $P\in\mathcal{P}(\mathcal{V})$, the abstract transformer corresponding to the assignment $l[z_i]=\sum_{w\in V_i'}\mathtt{len}(w)$ (the length of the word representing the concatenation of the words represented by variables in V_i' is the sum of the lengths of all these words).
W.l.o.g. suppose that P is a pattern in $\mathcal{P}(DWVars)$ that speaks about a set of words w_1,...,w_q in $DWVars\setminus\{V_1^0,\ldots,V_t^0\}$ and a set of words V_1^0,...,V_r^0 representing the concatenations of V_1',...,V_r', of the following form:

$$\bigwedge_{1\le i\le q} P_R^i(\mathbf{y}_i,w_i)\wedge\bigwedge_{1\le i\le r} Q_R^i(\mathbf{x}_i,V_i^0)\wedge P_L(y_1^1,\ldots,y_q^1,x_1^1,\ldots,x_r^1,\mathbf{w}),$$

where $\mathbf{w}\subseteq(DWVars\setminus(V_1\cup\ldots\cup V_t))\cup\{V_1^0,\ldots,V_t^0\}$.
We search for an abstract element U_P such that the universal property

$$\forall\mathbf{y}_1\ldots\forall\mathbf{y}_q,\forall\mathbf{x}_1\ldots\forall\mathbf{x}_r.\,P(\mathbf{y}_1,w_1,\ldots,\mathbf{y}_q,w_q,\mathbf{x}_1,V_1^0,\ldots,\mathbf{x}_q,V_q^0,\mathbf{w})\Rightarrow U_P,$$

is true when for any $1\le i\le r$, V_i^0 is interpreted to the word representing the concatenation of V_i'.
To this, we use the patterns from $\mathtt{Closure}(P,w_1,\ldots,w_q,V_1',\ldots,V_r')$. Thus, let P' be a pattern in $\mathtt{Closure}(P,w_1,\ldots,w_q,V_1',\ldots,V_r')$ of the form:

$$\bigwedge_{1\le i\le q} P_R^i(\mathbf{y}_i,w_i)\wedge\bigwedge_{1\le i\le r} T_R^i(\mathbf{x}_i,V_i^0)\wedge P_L'(y_1^1,\ldots,y_q^1,x_1^1,\ldots,x_r^1,\mathbf{w}),$$

where $T_R^i \in \texttt{Tuples}(Q_R^i, V_i')$. As we have seen in the definition of $\texttt{Tuples}(Q_R^i, V_i')$, some variables in $\mathbf{x}_i$ may be omitted from T_R^i. It is supposed that they will be mapped to first symbols of words in V_i'. Thus, we identify a set of partial mappings $\Gamma_{P'}$ between terms of the form $\texttt{hd}(v)$ with $v \in V_1' \cup \ldots \cup V_r'$ and variables in $\mathbf{x}_1 \cup \ldots \cup \mathbf{x}_r$ depending on each T_R^i as follows.
Suppose that $V_i' = v_1 \ldots v_m$ and $\mathbf{x}_i = x_i^1 \ldots x_i^j$. By definition, $T_R^i = \psi_1 \wedge \ldots \wedge \psi_{m'}$, where

$$\psi_1 = (x_i^{t_0}, \ldots, x_i^{q_1}) \in \texttt{tl}(v_{s_1}) \wedge x_i^{t_0} \sim_{t_0} \ldots \sim_{q_1-1} x_i^{q_1}$$
$$\psi_2 = (x_i^{q_1+t_1}, \ldots, x_i^{q_2}) \in \texttt{tl}(v_{s_2}) \wedge x_i^{q_1+t_1} \sim_{q_1+t_1} \ldots \sim_{q_2-1} x_i^{q_2}$$
$$\ldots$$
$$\psi_{m'} = (x_i^{q_{m'-1}+t_{m'-1}}, \ldots, x_i^{j}) \in \texttt{tl}(v_{s_{m'}}) \wedge x_i^{q_{m'-1}+t_{m'-1}} \sim_{q_{m'-1}+t_{m'-1}} \ldots \sim_{j-1} x_i^{j},$$

such that:
- $1 \leq m' \leq m$ and $1 \leq s_1 < s_2 < \ldots < s_{m'} \leq m$,
- $1 \leq t_0 < q_1 < q_2 < \ldots < q_{m'-1} \leq j$,
- $t_0 < s_1 + 1$ and $t_r \leq s_{r+1} - s_r + 1$, for any $1 \leq r < m'$,
- $q_r + t_r < q_{r+1}$, for any $1 \leq r < m' - 1$, and $q_{m'-1} + t_{m'-1} < j$,
- if $m' > 1$ and $t_{r+1} = 1$ then $\sim_{q_r} \neq <_1$, for any $1 \leq r < m' - 1$.

For each $1 \leq r < m' - 1$, we add to $\Gamma_{P'}$ all the possible mappings between the terms $\texttt{hd}(v_{s_r+1})$,...,$\texttt{hd}(v_{s_{r+1}})$ and the terms $V_i^0[x_i^{q_r+1}]$,...,$V_i^0[x_i^{q_r+t_r-1}]$.
Then, we define the abstract element U_P by

$$U_P = \bigsqcup_{\substack{P' \in \texttt{Closure}(P, w_1, \ldots, w_q, V_1', \ldots, V_r') \\ \gamma \in \Gamma_{P'}}} {}^{\mathbb{Z}} \left(\left(U_{P'} \sqcap^{\mathbb{Z}} E\right)\gamma \right) \uparrow \left(\left(V_1' \setminus \{V_1^0\}\right) \cup \ldots \cup \left(V_t' \setminus \{V_t^0\}\right) \right),$$

where $\left(U_{P'} \sqcap^{\mathbb{Z}} E\right)\gamma$ is obtained from the abstract element $U_{P'} \sqcap^{\mathbb{Z}} E$ by applying the substitution γ.
After deducing all the new universally quantified properties we modify E by projecting out all variables in $\left(\left(V_1' \setminus \{V_1^0\}\right) \cup \ldots \cup \left(V_t' \setminus \{V_t^0\}\right)\right)$.

Concerning this abstract transformer we can prove the following results.

Theorem 11. *Let $\mathcal{A}_{\mathbb{U}}$ be an abstract domain as above parametrized by a set of patterns $\mathcal{P}$ and by a numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ which contains a sound projection operator and an exact meet operator. For any abstract value $\widetilde{W}$ in $\mathcal{A}_{\mathbb{U}}$, we have that*

$$\alpha^{\mathbb{U}}\left(\{(\texttt{concat}(L, V_1, \ldots, V_t), D) \mid (L, D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) \sqsubseteq^{\mathbb{U}} \texttt{concat}^{\#}(\widetilde{W}, V_1, \ldots, V_t).$$

Theorem 12. *Let $\mathcal{A}_{\mathbb{U}}$ be an abstract domain as above parametrized by a set of patterns $\mathcal{P}$ and by a numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$ which contains a sound projection operator and an exact meet operator. If (1) $\mathcal{P}$ is closed and it contains only simple patterns (2) the projection operator, $\sqsubseteq^{\mathbb{Z}}$, and the abstract transformer in $\mathcal{A}_{\mathbb{Z}}$ corresponding to assignments $x = z_1 + \ldots + z_t$, where $x, z_1, \ldots, z_t$ are integer variables, are exact then,*

$$\alpha^{\mathbb{U}}\left(\{\texttt{concat}((L, D), V_1, \ldots, V_t) \mid (L, D) \in \gamma^{\mathbb{U}}(\widetilde{W})\}\right) = \texttt{concat}^{\#}(\widetilde{W}, V_1, \ldots, V_t),$$

for any $\widetilde{W}$ a closed abstract value

The precision results hold only if they are applied on closed abstract values. Fortunately, when considering simple patterns, all the abstract transformers preserve the closure property, that is, they output closed values when applied on closed values.

5 Experimental results

We have implemented the general method presented in this paper, i.e., the abstract reachability analysis using the $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{W}})$ abstract domain. Our implementation is generic in three dimensions. First, the $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{W}})$ abstract domain is interfaced with the APRON platform [15], so we are able to use the fix-point computation engines provided by this platform; currently, we are using INTERPROC. Second, the implementations of the $\mathbb{DW}$-domains can be plugged in the $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{W}})$ domain. We have implemented the $\mathbb{DW}$-domain $\mathcal{A}_{\Sigma}$ as well as the $\mathcal{A}_{\mathbb{U}}$ domain for a significant class of patterns allowing to handle a large class of programs. Currently, we are working on the implementation of the $\mathcal{A}_{\mathbb{M}}$ domain. Third, the implemented $\mathbb{DW}$-domains are generic on the numerical domain $\mathcal{A}_{\mathbb{Z}}$ used to represent data and length constraints. For this, we use again the APRON interface to access domains like octagons or polyhedra.

We have carried out experiments on a wide spectrum of programs including programs performing list traversal to search or to update data, programs with destructive updates and changes in the shape (e.g., list dispatch or reversal, sorting algorithms such as insertion sort), and programs computing complex arithmetical relations. We present hereafter some[3] of the specifications that can be synthesized using our approach.

Ordering and data preservation constraints: For sorting algorithms or the algorithms testing data ordering, our tool was able to synthesize constraints with respect to order preservation: $\forall y_1, y_2.\, \texttt{head} \xrightarrow{+} y_1 \xrightarrow{+} y_2 \Rightarrow \texttt{data}(y_1) \leq \texttt{data}(y_2)$ and $\forall y.\, \texttt{head} \xrightarrow{+} y \Rightarrow \texttt{data}(\texttt{head}) \leq \texttt{data}(y)$. Although, the multiset $\mathbb{DW}$-domain $\mathcal{A}_{\mathbb{M}}$ is not yet implemented we were able to manually check the preservation of the data in the list. On the other hand, the tool has synthesized using $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\Sigma})$ a weaker property namely the preservation of the sum of the elements of the list and the preservation of the length of the list. Another example for which the analysis synthesizes the constraint of sum and length preservation is the program doing a copy with reversal of a list `head` into a list `rev`: $\sum_{\texttt{head} \xrightarrow{*} y} \texttt{data}(y) = \sum_{\texttt{rev} \xrightarrow{*} y} \texttt{data}(y) \wedge \texttt{len}(\texttt{head} \xrightarrow{+} \texttt{null}) = \texttt{len}(\texttt{rev} \xrightarrow{+} \texttt{null})$.

Relating data and lengths of lists: Consider the program `Dispatch3` in Figure 1(b). Using the domain $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\Sigma})$, respectively $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{U}})$, the tool synthesized the post-condition (C), respectively (A), given in the introduction. Moreover, we are able to obtain constraints relating data and lengths inside the universal constraints, e.g., in the program initializing the data in a list `head` with the first even numbers. When analyzing this program with $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{U}})$, the generated post-condition contains the constraint $\texttt{data}(\texttt{head}) = 0 \wedge \forall y.\, \texttt{head} \xrightarrow{+} y \Rightarrow \texttt{data}(y) = 2 \times \texttt{len}(\texttt{head} \xrightarrow{+} y)$ meaning that the data stored in the ith cell of the list is equal to $2i$.

Relations over different lists: Our tool is able to generate constraints relating data in different lists. Consider the program which copies in the list `new` each da-

[3] A detailed presentation is available at `http://www.liafa.jusieu.fr/cinv/`.

tum of the list head incremented by 2 (the two lists have equal length). Using the domain $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{U}})$, the tool generates the post-condition $\mathtt{data}(\mathtt{head}) + 2 = \mathtt{data}(\mathtt{new}) \wedge \mathtt{len}(\mathtt{head} \xrightarrow{+} \mathtt{null}) = \mathtt{len}(\mathtt{new} \xrightarrow{+} \mathtt{null}) \wedge \forall y_1, y_2.\ (\mathtt{head} \xrightarrow{+} y_1 \wedge \mathtt{new} \xrightarrow{+} y_2 \wedge \mathtt{len}(\mathtt{head} \xrightarrow{+} y_1) = \mathtt{len}(\mathtt{new} \xrightarrow{+} y_2)) \Rightarrow \mathtt{data}(y_1) + 2 = \mathtt{data}(y_2)$. Using more complex patterns, not yet implemented in our domain, we synthesize manually the post-condition of the program copying in sequence two list A and B into a third list C, i.e., $\forall y_1, y_2.\ (\mathtt{B} \xrightarrow{*} y_1 \wedge \mathtt{C} \xrightarrow{*} y_2 \wedge y_2 = \mathtt{len}(\mathtt{A} \xrightarrow{*} \mathtt{null}) + y_1) \Rightarrow \mathtt{data}(y_1) = \mathtt{data}(y_2)$.

The analysis with $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\Sigma})$ of the program that creates a copy of a list, generates the post-condition $2 \times \mathtt{len}(\mathtt{head} \xrightarrow{*} \mathtt{null}) + \sum_{\mathtt{head} \xrightarrow{*} y} \mathtt{data}(y) = \sum_{\mathtt{new} \xrightarrow{*} y} \mathtt{data}(y)$.

Complex arithmetical relations: We have applied our tool on the Fibonacci example using $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{U}})$ over different sets of patterns. The constraint (D) given in the introduction is generated using the pattern with three universals successively ordered (and the patterns in its closure). Using a pattern with two universals, we obtain that the list head is sorted, and using a pattern with one universal, we obtain that $\forall y.\ \mathtt{head} \xrightarrow{+} y \Rightarrow (\mathtt{data}(y) + 1 \geq \mathtt{len}(\mathtt{head} \xrightarrow{+} y))$. Furthermore, the constraint (E) (in the introduction) is generated using $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\Sigma})$.

Performances: Each of the examples has been carried out in less than 1 second using between 4KB to 63MB. The most expensive example is the insertion sort (with destructive updates) which takes 0.99s and 62.2MB. Traversal algorithms such as search and local update algorithms, require only few hundredths of a second, e.g., 0.02s for the maximum calculation. Properties of programs such as Fibonacci are generated in few tenths of seconds, e.g., 0.42s for (E).

6 Conclusion and related work

We have defined powerful invariant synthesis techniques for a significant class of programs manipulating dynamic lists with unbounded data. Future work includes (1) extending the framework to handle a wider class of data structures, e.g. doubly-linked lists, composed data structures, (2) developing heuristic techniques for automatic synthesis of the patterns used in $\mathcal{A}_{\mathbb{U}}$, and (3) defining other abstract domains for data sequences, in particular, domains based on different classes of universally quantified formulas.

Related Work: Invariant synthesis for programs with dynamic data structures has been addressed using different approaches including constraint solving [2, 13], abstract interpretation [14, 9–12, 18, 19], Craig interpolants [16], and automata-theoretic techniques [3, 4]. The contributions of our paper are (1) a generic framework for combining an abstraction for the heap with various abstraction for data sequences, (2) new abstract domains on data sequences to reason about aspects beyond the reach of the existing methods such as the sum or the multiset of all elements in a sequence, as well as a new domain for generating an expressive class of first order universal formulas, and (3) precision results of the abstract transformers for a significant class of programs. Several works [14, 9, 18] consider invariant synthesis for programs with uni-dimensional arrays of integers. These programs can be straightforwardly encoded in our framework. In [12], a synthesis technique for universally quantified formulas is presented. Our technique differs from this one by the type of user guiding information. Indeed, the quantified formulas considered in [12] are of the form $\forall \mathbf{y}.\ F_1 \Rightarrow F_2$, where F_2 must

be given by the user. In contrast, our approach fixes the formulas in left hand side of the implication and synthesizes the right hand side. Therefore, the two approaches are in principle incomparable. The techniques in [14, 9] are applicable to programs with arrays. The class of invariants they can generate is included in the one handled by our approach using $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{U}})$. These techniques are based on an automatically generated finite partitioning of the array indices. We consider a larger class of programs for which these techniques can not be applied. The analysis introduced in [18] for programs with arrays can synthesize invariants on multisets of the elements in array fragments. This technique differs from ours based on the domain $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{U}})$ by the fact that it can not be applied directly to programs with dynamic lists. Finally, the analysis in [11] combines a numerical abstract domain with a shape analysis. It is not restricted by the class of data structures but it considers only properties related to the shape and to the size of the memory, assuming that data have been abstracted away. Our approach is less general concerning shape properties but it is more expressive concerning properties on data.

References

1. S. Abramsky and A. Jung. Domain theory. volume 3 of *Handbook of Logic in Computer Science*, pages 1–168. Clarendon Press, 2007.
2. D. Beyer, T.A. Henzinger, R. Majumdar, and A. Rybalchenko. Invariant synthesis for combined theories. In *Proc. of VMCAI*, volume 4349 of *LNCS*, pages 378–394. Springer, 2007.
3. A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In *CAV*, volume 4144 of *LNCS*, pages 517–531. Springer, 2006.
4. M. Bozga, P. Habermehl, R. Iosif, F. Konecný, and T. Vojnar. Automatic verification of integer array programs. In *Proc. of CAV*, volume 5643 of *LNCS*, pages 157–172, 2009.
5. R. Clarisó and J. Cortadella. The octahedron abstract domain. In *Proc. of SAS*, volume 3148 of *LNCS*, pages 312–327. Springer, 2004.
6. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
7. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of POPL*, pages 269–282, 1979.
8. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. of POPL*, pages 84–96, 1978.
9. D. Gopan, T.W. Reps, and S. Sagiv. A framework for numeric analysis of array operations. In *Proc. of POPL*, pages 338–350, 2005.
10. A. Gotsman, J. Berdine, and B. Cook. Interprocedural shape analysis with separated heap abstractions. In *Proc. of SAS*, volume 4134 of *LNCS*, pages 240–260. Springer, 2006.
11. S. Gulwani, T. Lev-Ami, and M. Sagiv. A combination framework for tracking partition sizes. In *Proc. of POPL*, pages 239–251, 2009.
12. S. Gulwani, B. McCloskey, and A. Tiwari. Lifting abstract interpreters to quantified logical domains. In *Proc. of POPL*, pages 235–246, 2008.
13. A. Gupta, R. Majumdar, and A. Rybalchenko. From tests to proofs. In *Proc. of TACAS*, volume 5505 of *LNCS*, pages 262–276. Springer, 2009.
14. N. Halbwachs and M. Péron. Discovering properties about arrays in simple programs. In *Proc. of PLDI*, pages 339–348, 2008.
15. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *Proc. of CAV*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
16. R. Jhala and K.L. McMillan. Array abstractions from proofs. In W. Damm and H. Hermanns, editors, *Proc. of CAV*, volume 4590 of *LNCS*, pages 193–206. Springer, 2007.

17. R. Manevich, E. Yahav, G. Ramalingam, and S. Sagiv. Predicate abstraction and canonical abstraction for singly-linked lists. In *VMCAI*, volume 3385 of *LNCS*, pages 181–198, 2005.
18. V. Perrelle and N. Halbwachs. An analysis of permutations in arrays. In *Proc. of VMCAI*, volume 5944 of *LNCS*, pages 279–294. Springer, 2010.
19. S. Sagiv, T.W. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, 2002.