Shape Analysis in a Functional Language by Using Regular Languages (Extended Version)

Technical Report TR-8-13 ⋆

Manuel Montenegro
montenegro@fdi.ucm.es

Ricardo Peña
ricardo@sip.ucm.es

Clara Segura
csegura@sip.ucm.es

Universidad Complutense de Madrid, Spain

Abstract. Shape analysis is concerned with the compile-time determination of the 'shape' the heap may take at runtime, meaning by this the pointer chains that may happen within, and between, the data structures built by the program. This includes detecting alias and sharing between the program variables.
Functional languages facilitates somehow this task due to the absence of variable updating. Even though, sharing and aliasing are still possible. We present an abstract interpretation-based analysis computing precise information about these relations. In fact, the analyis gives an information more precise than just the existence of sharing. It informs about the path through which this sharing takes place. This information is critical in order to get a modular analysis and not to lose precision when calling an already analysed function.
The main innovation with respect to the literature is the use of regular languages to specify the possible pointer paths from a variable to its descendants. This additional information makes the analysis much more precise while still being affordable in terms of efficiency. We have implemented it and give convincing examples of its precision.
Keywords: functional languages, abstract interpretation, shape analysis, points-to analysis, regular languages.

1 Motivation

Shape analysis is concerned with statically determining the connections between program variables through pointers in the heap that may occur at runtime. As particular cases it includes sharing and alias between variables. To know the shape of the heap for every possible program execution is undecidable in general, but the analysis computes an over-approximation of this shape. This means that it may include relations that will never happen at runtime.

Much work has been done in imperative languages (see Sec. 7), specially for C. There, the sharing detection is aggravated by the fact that variables are mutable, and they may point to different places at different times. We have addressed the problem for a first order functional language. This simplifies some of the difficulties since variables do not mutate. A consequence is that the inferred relations are immutable considering different parts of the program text. Another consequence is that the heap is never updated. It can only be increased with new data structures, or decreased by the garbage collector. But the latter cannot produce effects in its live part.

Our analysis puts the emphasis on three properties: (1) modularity; (2) precision; and (3) efficiency. For the sake of scalability, it is important for the analysis to be modular. The results obtained for a function should summarize the shape information so that the user functions should be able to compute all the sharing produced when calling it. Looked at from outside, and given that the language is functional, a function may only create sharing between its result and its arguments, or between the results themselves, but it can never create new sharing between the arguments. The internal variables become dead after the call, so the result of analysing a function only contains its input-output sharing behaviour. Differently from previous works,

⋆ Work partially funded by the projects TIN2008-06622-C03-01/TIN (STAMP), and S2009/TIC-1465 (PROMETIDOS).

```
unshuffle []     = ([],[])
unshuffle (x:xs) = (x:ys2, ys1)
                   where (ys1,ys2) = unshuffle xs
merge []     ys  = ys
merge (x:xs) []  = x:xs
merge (x:xs) (y:ys) | x <= y    = x : merge xs     (y:ys)
                    | otherwise = y : merge (x:xs) ys
msort []  = []
msort [x] = [x]
msort xs  = merge (msort xs1) (msort xs2)
            where (xs1, xs2) = unshuffle xs
```

Fig. 1: *mergesort* algorithm in *Full-Safe*

we compute the *paths* through which this sharing may occur in a precise way. This information is used to propagate to the caller the sharing created by a call. In this way, large programs can be analysed with a cost linear in the number of functions.

The motivation for our analysis is a type system we have developed for a functional language with explicit memory disposal [9]. This feature may create dangling pointers at runtime. The language also provides automatically allocated and deallocated heap regions, instead of having a runtime garbage collector. This feature can never create dangling pointers, so it plays no role in the current work and we will not mention it anymore. We have proved that passing successfully the type inference phase gives total guarantee that there will not be such dangling pointers. For typechecking a function, it is critical to know at compile time which variables may point to the disposed data structures, and for this a precise sharing analysis was needed. Nevertheless, we believe that the sharing analysis presented here could be equally useful for other purposes, since it provides precise information about the heap shape. Note that some shapes, such as cyclic or doubly chained lists, cannot be created by a functional language, so they are out of the scope of our analysis. But, in some cases, the analysis is capable of asserting that a given structure is a tree, i.e. it does not have internal sharing.

Our prior prototype shape analysis done in [10] was correct but imprecise, specially in function applications and **case** expressions. The reason for this is that it does not suffice knowing that two variables share a common descendant. We should more precisely know through which paths this sharing occurs.

The main contribution of this paper with respect to [10] is the incorporation of regular languages to our abstract domain. Each word of the language defines a pointer path within a data structure. Having regular languages introduces additional problems such as how to combine them during the analysis, how to compare them, and specially how to guarantee that a fixpoint will be reached after a finite number of iterations. We show that we have increased the precision of our prior analysis, and that the new problems can be tackled with a reasonable efficiency.

The plan of the paper is as follows: Sec. 2 provides a mild introduction to the analysis via a small example. Then, Sections 3, 4 and 5 contain all the technical material about the abstract domain, abstract interpretation rules, correctness, widening, decidability, and cost of the operations done on regular expressions. Sec. 6 presents our implementation and gives more examples. Finally, Sec. 7 concludes and discuss some related work.

2 Shape Analysis by Example

Our reference language *Safe* is a first-order eager language with a syntax similar to Haskell's. Fig. 1 shows a mergesort algorithm written in *Full-Safe*. The compiler's front-end processes *Full-Safe* and produces a bare-bones functional language called *Core-Safe*. This transformation desugars pattern matching into **case** expressions, transforms **where** clauses into **let** expressions, collapses several function-defining equations into a single one, and ensures unique names for the variables. In Fig. 2 we show a simplified *Core-Safe*'s syntax. A program *prog* is a sequence of possibly recursive polymorphic data and function definitions followed by a main expression e whose value is the program result. The abbreviation $\overline{x_i}$ stands for $x_1 \cdots x_n$, for some n. In Fig. 3 we show the translation to *Core-Safe* of the *msort* function of Fig. 1.

$$
\begin{array}{lll}
\mathit{prog} \rightarrow \overline{\mathit{data}_i}; \overline{\mathit{dec}_j}; e & & \{\textit{Core-Safe}\text{ program}\} \\
\mathit{dec} \;\rightarrow\; f\ \overline{x_i}\ = e & & \{\text{recursive, polymorphic function definition}\} \\
e \quad\rightarrow\; c & & \{\text{literal of a basic type}\} \\
\qquad\;\; \mid x & & \{\text{variable}\} \\
\qquad\;\; \mid f\ \overline{a_i} & & \{\text{function application}\} \\
\qquad\;\; \mid C\ \overline{a_i} & & \{\text{constructor application}\} \\
\qquad\;\; \mid \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2 & & \{\text{non-recursive, monomorphic } \mathbf{let}\} \\
\qquad\;\; \mid \mathbf{case}\ x\ \mathbf{of}\ \overline{\mathit{alt}_i} & & \{\text{case expression}\} \\
\mathit{alt} \;\; \rightarrow C\ \overline{x_i} \rightarrow e & & \{\text{case alternative}\}
\end{array}
$$

Fig. 2: Simplified *Core-Safe* syntax

```
msort xs = case xs of
             []   -> []
             x:xx -> case xx of []  -> x:[]
                                _:_ -> let p  = unshuffle xs             in
                                       let y1 = case p of (s1,s2) -> s1 in
                                       let y2 = case p of (w1,w2) -> w2 in
                                       let z1 = msort y1                 in
                                       let z2 = msort y2                 in
                                       merge z1 z2
```

Fig. 3: Function *msort* in *Core-Safe*

Our shape analysis infers the following sharing information for the functions `unshuffle` and `merge`:

$$
\begin{array}{ll}
\Sigma(\mathit{unshuffle}) & = \{\mathit{res} \xrightarrow{12^*1+22^*1} \bullet \xleftarrow{2^*1} \mathit{xs}\} \\
\Sigma(\mathit{merge}) & = \{\mathit{res} \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} \mathit{xs},\ \mathit{res} \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} \mathit{ys}, \\
& \quad\ \mathit{res} \xrightarrow{2^*} \bullet \xleftarrow{2^*} \mathit{xs},\ \mathit{res} \xrightarrow{2^*} \bullet \xleftarrow{2^*} \mathit{ys}\}
\end{array}
$$

The meaning for `unshuffle` is the following: the resulting tuple *res* of calling the function with an input list *xs*, may share the elements of this list. Moreover, the path reaching a common descendant, in the case of *res*, begins either with a 1 or a 2 (this should be understood as descending to the left or to the right element of the tuple), and then follows by the path 2^*1, by this meaning that we should take the tail of the (left or right) list a number of times, and then take the head. From *xs*'s point of view of, the common descendant can be reached by a similar path 2^*1.

The meaning for `merge` is quite precise: the resulting list *res* may share its elements with any of the input lists *xs* and *ys*, and additionally one or more tails of *res* may be shared with one or more tails of both *xs* and *ys*. This is what the path 2^* means.

When analysing `msort`'s code of Fig. 3, we have the information about `unshuffle` and `merge` available. By substituting the actual arguments for the formal ones, we get the following relations:

$$
\begin{array}{ll}
R_1 = \{p \xrightarrow{12^*1+22^*1} \bullet \xleftarrow{2^*1} \mathit{xs}\} \\
R_2 = \{\mathit{res} \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} z_1,\ \mathit{res} \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} z_2, \\
\quad\ \ \mathit{res} \xrightarrow{2^*} \bullet \xleftarrow{2^*} z_1,\ \mathit{res} \xrightarrow{2^*} \bullet \xleftarrow{2^*} z_2\}
\end{array}
$$

The **case** and **let** expressions in `msort` introduce more relations:

$$
R_3 = \{x \xrightarrow{\epsilon} \bullet \xleftarrow{1} \mathit{xs},\ y_1 \xrightarrow{\epsilon} \bullet \xleftarrow{1} p,\ y_2 \xrightarrow{\epsilon} \bullet \xleftarrow{2} p\}
$$

From these relations, we can derive other by reflexivity, symmetry and transitivity, such as:

$$
R_4 = \{y_1 \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} \mathit{xs},\ y_2 \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} \mathit{xs},\ y_1 \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} y_2\}
$$

In the first iteration of `msort`'s analysis, the only relation that can be inferred between its result and its argument is $xs \xrightarrow{1} \bullet \xleftarrow{1} res$. This is due to the third line. The rest of the code gives us sharing information between the internal variables, but this cannot be propagated to the arguments, because in the internal recursive calls to `msort` we have nothing to start with. But, by interpreting the internal calls with the sharing information $\Sigma_1(msort) = \{xs \xrightarrow{1} \bullet \xleftarrow{1} res\}$, we get the following bigger result: $\Sigma_2(msort) = \{xs \xrightarrow{2^*1+1} \bullet \xleftarrow{2^*1+1} res\}$. If we interpret the code a third time by using this information when interpreting the internal calls, we get again the same result. So, a fixed point has been reached, and we consider this result as a correct approximation of the sharing created by `msort`.

It is worthwhile to remark that our prior analysis [10] of the same program gave us the additional spurious sharing information $\{z_2 \longrightarrow \bullet \longleftarrow z_1\}$, meaning that a descendant of z_2 is shared by z_1 (the regular languages were absent in that analysis). Having spurious relations is not incorrect, but just imprecise. Since we used this analysis to type a destructive version of `msort`, using in turn a destructive version of `merge`, our type system rejected the function because of this additional sharing. The cause of this imprecision was a worse analysis of **case** expressions and function applications due to the absence of the paths represented by the regular languages.

3 The Analysis

We formally define here the analysis approximating the runtime sharing relations between the program variables. At this point, types have already been inferred, so the analysis can ask for type-related issues, such as the positions of constructor descendants, their types, and the like.

3.1 Sharing relation

In order to capture sharing, we define a binary relation between variables:

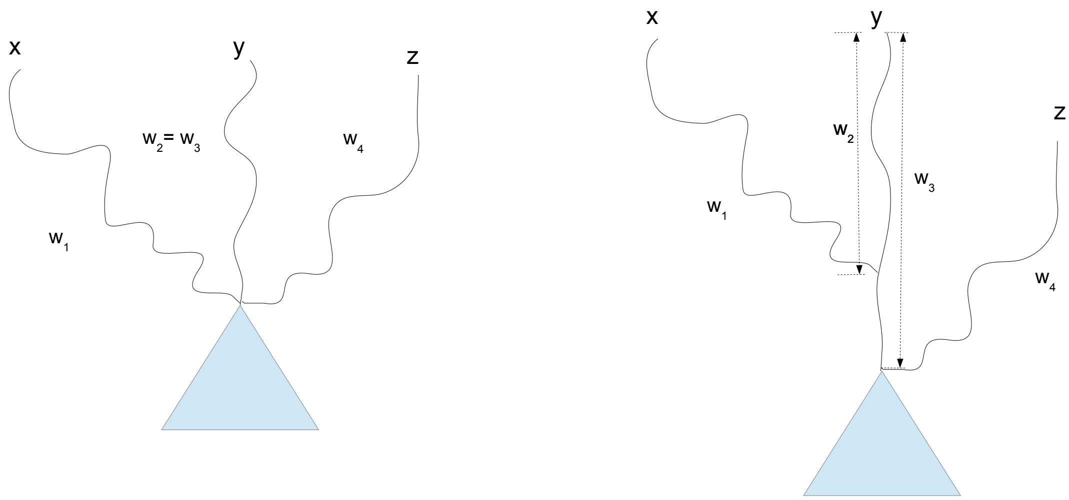
Definition 1. *Given two variables x and y, in scope in an expression, a sharing relation is a set of two pairs $\{(x, p_1), (y, p_2)\}$ specifying that x and y share a common descendant. Moreover, the regular languages denoted by p_1 and p_2 respectively define the possible pointer chains through which x and y reach their common descendant. We shall denote this sharing relation either by $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ or $y \xrightarrow{p_2} \bullet \xleftarrow{p_1} x$.*

For the sake of readability, we shall assume in the following p_1 and p_2 to be regular expressions that denote regular languages, but the actual implementation does not use them, though. Notice that, if $p_1 = \epsilon$, then x is a descendant of y, and symmetrically for p_2.

The regular languages have pairs i_C as alphabet symbols, where i is a natural number starting at 1, and C is a data constructor. The symbol i_C denotes a singleton pointer path in the heap passing through the i-th argument of constructor C. For instance, $x \xrightarrow{2^*_:} \bullet \xleftarrow{1_{(,)}} y$ indicates that a tail of the list x is pointed-to by the first element of the tuple y. In the examples, we shall usually omit the constructor.

The relation $\xrightarrow{p_1} \bullet \xleftarrow{p_2}$ is symmetric by definition and reflexive by writing $p_1 = p_2 = \epsilon$. But the transitivity does not hold, i.e. $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ and $y \xrightarrow{p_3} \bullet \xleftarrow{p_4} z$, with $p_2 \neq \epsilon$, does not necessarily imply $x \xrightarrow{p_1} \bullet \xleftarrow{p_4} z$. However, the transitivity holds in some cases, for example when y reaches its common descendant with x through *the same path* as it reaches its common descendant with z, as shown in Figure 4a.

More generally, we can investigate the languages denoted by p_2 and p_3, and decide whether a path in p_2 coincides with, or is a prefix of, a path in p_3 (as shown in Figure 4b), or the other way around. In these cases, there may exist a sharing path through y between x and z. Notice that both p_2 and p_3 are upper approximations to the actual runtime paths, so the risk of imprecision is still there, but if there are no such paths we are certain that there will not be paths at runtime either, and we can safely omit a tuple relating x and z from the sharing relation. The rules computing the sharing derived by transitivity are explained in detail in Section 3.4.



(a) (b)

Fig. 4: Particular cases of transitivity

3.2 The abstract interpretation

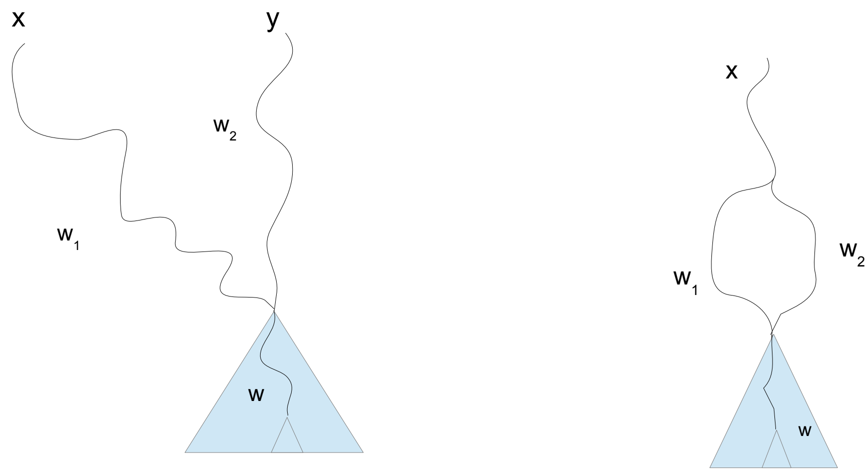
Based on the above considerations, we define an abstract interpretation S (meaning *sharing*) which, given an expression e and a set R containing an upper approximation to the sharing relations between the variables in scope in e, delivers another set R_{res} (*res* stands for result) containing (an upper approximation to) all the relations between the result of evaluating e, named *res*, and its variables in scope. To be precise, R and R_{res} must record at least the *minimum* information needed in order to compute all possible sharing, i.e. if we have $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ in R or R_{res}, and p_3 denotes all possible paths inside the data structure pointed-to by x and y, then we understand that $x \xrightarrow{p_1 \cdot p_3} \bullet \xleftarrow{p_2 \cdot p_3} y$ is implicitly included in the relation.

Notice that this means that:

- If two variables x and y share a substructure in the heap as in Figure 5a, there must exist a sharing relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ containing at least the paths w_1 and w_2, leading to the first point of confluence. Their extensions with a common path w need not.
- In case a variable x has internal sharing, as shown in Figure 5b, there must exist a sharing relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} x$ containing at least the paths w_1 and w_2 leading to the first point of confluence.

In order to achieve a modular analysis, it is very important to reflect the result of the analysis of a function f in a *function signature environment*, so that when the analysis finds calls to f in the body of another function g, it uses this knowledge to compute the sharing relations for g. We keep function signatures in a global environment Σ, so that $\Sigma(f)$ is a set R_{res} containing the sharing relations between the result of calling f and its arguments. The interpretation $S[\![e]\!]\ R\ \Sigma$ gives us the relations between (the normal form of) e and its variables in scope, provided Σ gives us correct approximations to the sharing relations of the functions called from e.

The rules for expressions are explained in detail in Section 3.3. The interpretation S_d of a function definition $f\ x_1 \ldots x_n = e_f$ begins with the interpretation of its body. It is straightforward to extract the signature of the function, which just describes the relations between the result of e_f and its formal arguments $\overline{x_j}^n$, which are the only variables in scope. In case f is recursive, the interpretation is run several times, by starting with an empty signature for f and then computing the least fixpoint. Each iteration updates f's signature in the signature environment:



(a) (b)

Fig. 5: At least paths w_1 and w_2 must be recorded in a sharing relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ (a) or $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} x$ (b)

$$
\begin{array}{ll}
S\,[\![c]\!]\ R\ \Sigma & = R \\
S\,[\![x]\!]\ R\ \Sigma & = R\ \uplus^*_{res}\ \{res \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x\} \\
S\,[\![C\ \overline{a_i}^m]\!]\ R\ \Sigma & = R\ \uplus^*_{res}\ \{res \xrightarrow{j_C} \bullet \xleftarrow{\epsilon} a_j \mid\ j \in \{1..m\},\ var(a_j)\} \\
S\,[\![g\ \overline{a_i}^m]\!]\ R\ \Sigma & = R\ \uplus^*_{res}\ \Sigma(g)[\overline{a_j/x_j}^m] \\
S\,[\![\mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2]\!]\ R\ \Sigma & = (S\,[\![e_2]\!]\ R_1\ \Sigma)\backslash\{x_1\} \\
\quad \mathbf{where}\ R_1 = (S[\![e_1]\!]\ R\ \Sigma)[x_1/res] & \\
S\,[\![\mathbf{case}\ x\ \mathbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}]\!]\ R\ \Sigma & = \bigcup_i (S\,[\![e_i]\!]\ R_i\ \Sigma)\backslash\{\overline{x_{ij}}^{n_i}\} \\
\quad \mathbf{where}\ R_i = R\ \uplus^*_{\overline{x_{ij}}}\ \{x \xrightarrow{j_{C_i}} \bullet \xleftarrow{\epsilon} x_{ij} \mid j \in \{1..n_i\}\} &
\end{array}
$$

Fig. 6: Definition of the abstract interpretation S

$$
\begin{array}{l}
S_d[\![f\ x_1 \ldots x_n = e_f]\!]\ \Sigma = \mathit{fix}\ (\lambda\Sigma\,.\,\Sigma[f \to S[\![e_f]\!]\ R_0\ \Sigma])\ \Sigma_0 \\
\quad \mathbf{where}\ \ \Sigma_0 = \Sigma[f \to \emptyset] \\
\qquad\qquad\ \ R_0 = \{x_i \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x_i \mid i \in \{1..n\}\}
\end{array}
$$

where $\Sigma[f \to R]$ either adds signature R for f or replaces it in case there was already one for it. Notice that the right hand side of the function definition is analysed starting with a neutral initial relation R_0 in which each argument is only related to itself. This means that the signatures are computed assuming that all the parameters are disjoint and they do not present internal sharing in addition to the trivial sharing relation given by R_0. When they are not, the function caller knows the additional sharing of the actual arguments and the rule for application merges both information, as we will see in Section 3.3.

As function S is monotonic over a lattice, the least fixpoint exists and could be computed using Kleene's ascending chain if the chain were finite. We come back to this issue in Section 5.

3.3 Interpretation of expressions

The interpretation defined in Figure 6 does a top-down traversal of a function definition, accumulating these relations as soon as bound variables become free variables.

The notation $R[y/x]$ means the substitution of the variable y for the variable x in the relation R. In order to avoid name capture, y must be fresh in R. The operator $R\backslash\{x\}$ removes from R any tuple containing the variable x. The union operator $\cup$ is the usual set union. The closure operation $R_1 \uplus_x^* R_2$ takes a relation R_1 and completes it by adding R_2 and the tuples involving x that can be derived by transitivity. This operation also generates the reflexive relation $x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x$. We explain this operator in detail in Section 3.4.

An important invariant of the rules presented in Figure 6 is that, in each occurrence of $S\ [\![e]\!]\ R\ \Sigma$, the set R contains an upper approximation of *all* the sharing relations that at runtime may happen between the variables in scope in e. Also, the set R_{res} returned by $S\ [\![e]\!]\ R\ \Sigma$ enjoys the same property. It is easy to check that if the property holds for the original call $S\ [\![e_f]\!]\ R_0\ \Sigma$, then the rules preserve it.

The rule for a constant c introduces no new sharing. The rule of a variable x specifies that the result is an alias of x, and $\uplus_{res}^*$ propagates to the result the variables to which x is related.

When a constructor application $C\ \overline{a_i}^m$ is returned as a result, parent-child sharing relations are created with the constructor's children. These are added to the current set R, and then the closure computes all the derived sharing.

When a function application $g\ \overline{a_i}^m$ is returned as a result, first we get from g's signature the sharing relations between g's result and its formal arguments. These are copied by replacing the formal arguments by the actual ones, and then added to the current set. As before, the closure computation does the rest.

The **let** rule is almost self-explanatory: first e_1 is analysed and the sharing computed for e_1's result is assigned to the new variable in scope x_1. Using this enriched set R_1 as assumption, the main expression e_2 is analysed, and its result is the result of the whole **let** expression.

Finally, a **case** expression introduces the pattern variables $\overline{x_{ij}}^{n_i}$ in the scope of a branch e_i. Their sharing relations are derived from the parent x's ones by first adding the child-parent relation between each x_{ij} and x, and then computing the closure. After analysing the branches, the least upper bound of all the analyses must be computed, expressing the fact that at compile time it is not known which branch will be taken at runtime.

It is important to see whether the relations inferred by the analysis are well-typed. For instance, we could have a relation $x \xrightarrow{p_x} \bullet \xleftarrow{p_y} y$ in which the descendant reached from x and p_1 had a type t, while the descendant reached from y and p_y had a different type t'. This would obviously be a spurious relation since in well-typed programs, an ill-typed sharing may not occur at runtime.

The expression $type(t, p)$ returns the type computed starting at the type t, and then descending through the constructors of the words in p according to its type and to the child chosen at each step. In our language this type can be statically computed. Let t_x be the type computed by the compiler for the variable x.

Definition 2. *We say that the relation* $x \xrightarrow{p_x} \bullet \xleftarrow{p_y} y$ *is well-typed if* $type(t_x, p_x) = type(t_y, p_y)$.

Lemma 1. *If the relations in* R *and* Σ *are well-typed, then for every expression* e*, the relations in* $S[\![e]\!]\ R\ \Sigma$ *are well-typed.*

Proof. By induction on the rules used to compute $R' = S[\![e]\!]\ R\ \Sigma$, we will prove a stronger property, namely that if $x \xrightarrow{p_x} \bullet \xleftarrow{p_y} y \in R'$, then for all $w_x \in L(p_x), w_y \in L(p_y)$ the relation $x \xrightarrow{w_x} \bullet \xleftarrow{w_y} y$ is well-typed, and $type(t_x, w_x) = type(t_x, p_x)$, and $type(t_y, w_y) = type(t_y, p_y)$.

By inspection of the rules of Fig. 6, it is easy to check that every relation explicitly introduced there is well-typed, and that the path expressions are just basic ones such as j_C or ϵ. So, they satisfy the desired property. Also, substituting a variable for another one with the same type, preserves the property. We concentrate then on the closure operator of Fig. 7, and more specifically on the rule TR.

Let us assume that for every $w_1 \in L(p_1), w_2 \in L(p_2), w_3 \in L(p_3), w_4 \in L(p_4)$, the relations $x \xrightarrow{w_1} \bullet \xleftarrow{w_2} y$ and $y \xrightarrow{w_3} \bullet \xleftarrow{w_4} z$ are well-typed. Let $w_5 \in L(p_3|_{p_2})$. We know this language not to be empty because of the condition $p_2 \ll p_3$. Let $t_1 = type(t_x, p_1) = type(t_y, p_2)$ and $t_2 = type(t_y, p_3) = type(t_z, p_4)$.

Given w_5, there exist words $w_2 \in L(p_2), w_3 \in L(p_3)$ such that $w_3 = w_2 \cdot w_5$. Let $w_1 \in L(p_1)$. We have then the following equalities:

$$type(t_x, p_1 \cdot p_3|_{p_2}) = type(t_x, w_1 \cdot w_5) = type(t_1, w_5) =$$
$$type(type(t_y, w_2), w_5) = type(t_y, w_3) = type(t_y, p_3) = type(t_z, p_4)$$

$$
\begin{aligned}
R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \overset{\text{def}}{=}_{y \neq x} \; & R \cup \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \cup \{x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x\} \\
& \cup \{x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_4} z \mid y \xrightarrow{p_3} \bullet \xleftarrow{p_4} z \in R\} \\
& \cup \{x \xrightarrow{p_1} \bullet \xleftarrow{p_4 \cdot p_2|_{p_3}} z \mid y \xrightarrow{p_3} \bullet \xleftarrow{p_4} z \in R\} \\
& \cup \{x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_1 \cdot p_4|_{p_2}} x \mid y \xrightarrow{p_3} \bullet \xleftarrow{p_4} y \in R\}
\end{aligned}
$$

$$
\begin{aligned}
R \uplus^*_x R' \quad \overset{\text{def}}{=} \quad & \overline{R \uplus_x \{\{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \mid x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R', y \neq x\}} \\
& \cup \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} x \mid x \xrightarrow{p_1} \bullet \xleftarrow{p_2} x \in R'\}
\end{aligned}
$$

Fig. 7: Definition of the closure operation

Then, for all $w_x \in L(p_1 \cdot p_3|_{p_2}), w_y \in L(p_4)$, the relation $x \xrightarrow{w_x} \bullet \xleftarrow{w_y} z$ is well-typed.

3.4 The closure of a relation

The closure operation $\uplus^*_x$ is defined in terms of the simpler one $\uplus_x$, which completes a relation set R with a new relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$, where $y \neq x$, by adding the relations that bind x to the variables contained in R, and are derived by transitivity. Both operators are defined in Figure 7.

The inclusion of R and the relations $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$, $x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x$ is self-explanatory. We shall concentrate on the remaining relations shown in the last lines of the definition.

The second line corresponds to the case illustrated in Figure 4b, while the third one corresponds to the symmetric case. These relations involve the derivative operator $\_|\_$ whose meaning is:

$$p_1|_{p_2} = \{w_3 \mid \exists w_2 \in L(p_2).w_2 w_3 \in L(p_1)\}.$$

If p_1 and p_2 denote regular languages so do $p_1|_{p_2}$, and in Section 5 we explain how to compute it[1]. In the second line of the definition of $\uplus_x$ the language describing $p_1 \cdot p_3|_{p_2}$ might be empty. In this case we can discard the corresponding sharing relation from the result of the closure operation. If it is not empty then there exists a word $w_2 \in L(p_2)$ such that it its a prefix of another word $w_3 \in L(p_3)$, so we can start from x, follow a path $w_1 \in L(p_1)$, and then follow the path w_2 without the prefix w_3 (which results in a path of $L(p_3|_{p_2})$) in order to reach the common descendant of x and z. The third line of $\uplus_x$ is applicable when a path of p_3 is a prefix of a path of p_2, and works similarly.

The fourth line deals with the case in which variable x gets internal sharing through variable y, shown in Figure 8. This happens when the path w_2 through which y reaches its common descendant with x is a prefix of both paths w_3 and w_4 representing the internal sharing of y. Then $p_3|_{p_2}$ and $p_4|_{p_2}$ are not empty, and contain respectively the paths w_3 and w_4 without the prefix w_2, which prepended with $w_1 \in L(p_1)$ represent two paths of internal sharing from variable x.

In spite of the restrictions of the $\uplus$ operator, we could replace the $\uplus^*$ operator by a sequence of $\uplus$ operations in all the rules but in function application, because only in the application non-trivial reflexive relations may be added.

In fact, operation $R \uplus^*_x R'$ is used to define the confluence of information happening in a function call. R represents the context of the call, while R' represents the sharing generated by the function between the result and the arguments.

Its definition is divided into two parts:

1. First, we take each relation in R' of the form $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ such that $y \neq x$ and apply the previous transitivity operator incrementally. This is well defined because operator $\uplus_x$ is in a sense *commutative*,

[1] When $p_2 = \{a\}$, the language $p_1|_a$ is sometimes called the *derivative* of $L(p_1)$ with respect to a, and it is denoted $a \backslash L$, being $L = L(p_1)$.

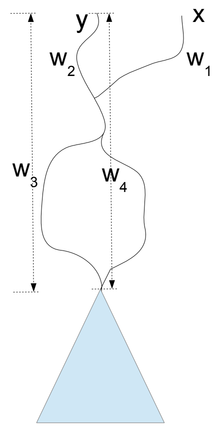


Fig. 8: Transitivity with internal sharing.

as we will prove in Section 4. So the order in which we add the relations of R' is not relevant: the final result may be different but *equivalent*, in the sense that it records the same information.

This part reflects the interaction of the context with the function definition.

2. Second, we just add those reflexive relations $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} x \in R'$. In the abstract interpretation, this kind of relations only appear in the application of a function: it may happen that the result of a function f has internal sharing, so a relation $res \xrightarrow{p_1} \bullet \xleftarrow{p_2} res \in \Sigma(f)$. It is not necessary to apply transitivity here because the internal sharing of *res* either comes from the function itself (i.e. is reflected in R') or through a real argument which already has internal sharing (i.e. is reflected in R). Transitivity, as we will prove in Section 4, would only add redundant information.

4 Correctness

In this section we provide the main results needed to prove the analysis is well-defined and correct.

4.1 Heap properties

Definition 3. *Let h be a heap, $p, q \in \operatorname{dom} h$, and $v \in \mathcal{V}$. We say that q is an immediate v-successor of p (written $p \overset{v}{\dashrightarrow}_h q$) iff $h(p) = (j, C\ \overline{v_i}^n)$ for some j, C and $\overline{v_i}^n$, and there exists some $i \in \{1..n\}$ such that $q = v_i$ and $v = i_C$. Analogously, assume a word w in $\mathcal{V}^*$. A pointer $q \in \operatorname{dom} h$ is a w-successor of p (written $p \overset{w}{\Longrightarrow}_h q$) if there exists a sequence of pointers $p_0, \ldots, p_n$ $(n \geq 0)$ and a sequence of positions $v_1, \ldots, v_n \in \mathcal{V}$ such that $w = v_1 \cdots v_n$ and:*

$$p = p_0 \overset{v_1}{\dashrightarrow}_h p_1 \overset{v_2}{\dashrightarrow}_h \ldots \overset{v_n}{\dashrightarrow} p_n = q$$

We are mostly interested in the fact that two given variables are pointing to a common pointer p, rather than the p itself. That is why we shall use the notation

$$p_1 \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} p_2 \text{ (in } h)$$

to denote the existence of a pointer p such that $p_1 \overset{w_1}{\Longrightarrow}_h p$ and $p_2 \overset{w_2}{\Longrightarrow}_h p$. Moreover, the following diagram:

$$\begin{tikzcd} & p_3 \arrow[d, Rightarrow, "w_3"'] & \\ p_1 \arrow[r, Rightarrow, "w_1"] & \bullet & p_2 \arrow[l, Rightarrow, "w_2"'] \end{tikzcd} \qquad (\text{in } h)$$

denotes the existence of a p such that $p_1 \overset{w_1}{\Longrightarrow}_h p$, $p_2 \overset{w_2}{\Longrightarrow}_h p$ and $p_3 \overset{w_3}{\Longrightarrow}_h p$.

Lemma 2 (Closure preservation). *Let us assume an execution $E \vdash h, e \Downarrow h', v$. For every pointer $p \in \operatorname{dom} h'$, variable $x \in \operatorname{dom} E$, and $w \in \mathcal{V}^*$:*

$$E(x) \overset{w}{\Longrightarrow}_h p \textit{ if and only if } E(x) \overset{w}{\Longrightarrow}_{h'} p$$

As a consequence of this, $E(x) \overset{w}{\Longrightarrow}_{h'} p$ implies $p \in \operatorname{dom} h$.

Lemma 3 (Path extension). *For any heap h, pointers $p_1, p_2 \in \operatorname{dom} h$, and words $w_1, w_2, w_3 \in \mathcal{V}^*$, if $p_1 \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} p_2\,(\text{in } h)$ and there exists some p' such that $p_1 \overset{w_1 w_3}{\Longrightarrow}_h p'$, then it holds that $p_1 \overset{w_1 w_3}{\Longrightarrow} \bullet \overset{w_2 w_3}{\Longleftarrow} p_2\,(\text{in } h)$.*

Lemma 4 (Path splitting). *Assume a heap h, two pointers $p, q \in \operatorname{dom} h$ and a word $w \in \mathcal{V}^*$ such that $p \overset{w}{\Longrightarrow}_h q$. For every $w_1, w_2 \in \mathcal{V}^*$ such that $w = w_1 w_2$ and every $p' \in \operatorname{dom} h$, the fact $p \overset{w_1}{\Longrightarrow}_h p'$ implies $p' \overset{w_2}{\Longrightarrow}_h q$.*

Lemma 5 (Sharing lemma). *Assume an execution $E \vdash h, e \Downarrow h', v$ and a pointer $p \in \operatorname{dom} h$ and two paths $w_p, w_v \in \mathcal{V}^*$. If*

$$p \overset{w_p}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} v\ (\text{in } h')$$

there exists a variable y occurring free in e and a path $w_y \in \mathcal{V}^*$ such that

$$\begin{tikzcd} & E(y) \arrow[d, Rightarrow, "w_y"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad (\text{in } h')$$

Proof. By induction on the size of the $\Downarrow$-derivation. We distinguish cases on the expression e being executed.

- **Case** $e \equiv c$
  In this case v is not a pointer, so $p \overset{w_p}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} v\ (\text{in } h')$ never holds and the lemma holds vacuously.
- **Case** $e \equiv x$
  We know that $v \overset{\epsilon}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} v\ (\text{in } h')$ and, by Lemma 3, $v \overset{w_v}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} v\ (\text{in } h')$. With the hypothesis, we get:

  $$\begin{tikzcd} & v \arrow[d, Rightarrow, "w_v"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad (\text{in } h')$$

  or, equivalently,

  $$\begin{tikzcd} & E(x) \arrow[d, Rightarrow, "w_v"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad (\text{in } h')$$

  Hence the lemma holds for $y = x$ and $w_y = w_v$.

- **Case** $e \equiv C\ \overline{a_i}^n$
  Assume $p \overset{w_p}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} v$ (in h'). The word w_v must be different from ϵ, since otherwise we would have $p \overset{w_p}{\Longrightarrow}_{h'} v$ which implies, by Lemma 2, $p \overset{w_p}{\Longrightarrow}_h v$, leading to a contradiction since v is a fresh pointer not appearing in h.
  Therefore, we assume that $w_v \neg \epsilon$. By construction of v, there must be an integer $i \in \{1..n\}$ and a word w'_v such that $w_v = i_C w'_v$, so it holds that $v \overset{i_C}{\dashrightarrow}_{h'} E(a_i)$. Hence we can apply the path splitting result (Lemma 4) to $v \overset{w_v}{\Longrightarrow} \bullet \overset{w_p}{\Longleftarrow} p$ (in h') in order to get $E(a_i) \overset{w'_v}{\Longrightarrow} \bullet \overset{w_p}{\Longleftarrow} p$ (in h'). On the other hand, the relation $E(a_i) \overset{\epsilon}{\Longrightarrow} \bullet \overset{i_C}{\Longleftarrow} v$ (in h') can be extended by Lemma 3 so as to get $E(a_i) \overset{w'_v}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} v$ (in h'). We obtain the following situation:

$$\begin{tikzcd} & E(a_i) \arrow[d, Rightarrow, "w'_v"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad \text{(in } h'\text{)}$$

  Hence the lemma holds for $y = a_i$ and $w_y = w'_v$.
- **Case** $e \equiv f\ \overline{a_i}^n$
  Assume that $f\ \overline{x_i} = e_f$ is the definition of the function being called, and let us define $E_g = [\overline{y_i \mapsto E(a_i)}^n]$. The following judgement is a direct descendant of the main judgement $E \vdash h, e \Downarrow h', v$:

$$E_g \vdash h, e_g \Downarrow h', v$$

  By induction hypothesis, there exists a variable y and a word w_y such that:

$$\begin{tikzcd} & E_g(y) \arrow[d, Rightarrow, "w_y"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad \text{(in } h'\text{)}$$

  and the variable y occurs free in e_f. This implies that $y = y_i$ for some $i \in \{1..n\}$. We get $E_g(y) = E_g(y_i) = E(a_i)$, so the diagram above can be rewritten as follows:

$$\begin{tikzcd} & E(a_i) \arrow[d, Rightarrow, "w_y"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad \text{(in } h'\text{)}$$

  Therefore the lemma holds for $y = a_i$.
- **Case** $e \equiv$ **let** $x_1 = e_1$ **in** e_2
  The following $\Downarrow$-judgements are direct descendants of $E \vdash h, e \Downarrow h', v$:

$$\begin{array}{l} E \vdash h, e_1 \Downarrow h_1, v_1 \\ \underbrace{E \uplus [x_1 \mapsto v_1]}_{E_1} \vdash h_1, e_2 \Downarrow h', v \end{array}$$

By induction hypothesis on the $\Downarrow$-judgement of e_2 we get:

$$\begin{tikzcd} & E_1(y_2) \arrow[d, Rightarrow, "w_2"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad \text{(in } h'\text{)} \tag{1}$$

If $y_2 \neq x_1$ the Lemma holds by taking $y = y_2$ and $w_y = w_2$, since $E_1(y_2) = E(y_2)$. If $y_2 = x_1$ the relation $p \overset{w_p}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} v_1$ (in h') follows from the diagram above, since $E_1(y_2) = E_1(x_1) = v_1$. Moreover, both p and v_1 belong to the domain h_1, so we can apply Lemma 2 in order to get $p \overset{w_p}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} v_1$ (in h_1). From this fact we can apply induction hypothesis on the $\Downarrow$-judgement of e_1 so as to get:

$$\begin{tikzcd} & E(y_1) \arrow[d, Rightarrow, "w_1"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v_1 \arrow[l, Rightarrow, "w_2"'] \end{tikzcd} \qquad \text{(in } h_1\text{)}$$

By applying closure preservation (Lemma 2) this relation also holds in h'.

$$\begin{tikzcd} & E(y_1) \arrow[d, Rightarrow, "w_1"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v_1 \arrow[l, Rightarrow, "w_2"'] \end{tikzcd} \qquad \text{(in } h'\text{)} \tag{2}$$

for some y_1 occurring free in e_1 and some word w_1. From the situation depicted in (1) we get $v_1 \overset{w_2}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} v$ (in h'), hence we obtain from (2) the following situation:

$$\begin{tikzcd} & E(y_1) \arrow[d, Rightarrow, "w_1"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad \text{(in } h'\text{)}$$

Hence the lemma holds for $y = y_1$ and $w_y = w_1$.

- **Case** $e \equiv$ **case** $x\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^{n}$
  There exists some $p_x \in \text{dom}\ \ h$, $r \in \{1..n\}$ such that $E(x) = p_x$ and $h(p_x) = C_r\ \overline{v_j}^{n_r}$ for some $\overline{v_j}^{n_r}$, so a direct descendant of $E \vdash h, e \Downarrow h', v$ is the following $\Downarrow$-judgement:

  $$\underbrace{E \uplus [\overline{x_{rj} \mapsto v_j}^{n_r}]}_{E_r} \vdash h, e_r \Downarrow h', v$$

  Assume that $p \overset{w_p}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} v$ (in h') for some $p \in \text{dom}\, h$. By induction hypothesis on this judgement we get:

  $$\begin{tikzcd} & E_r(y_r) \arrow[d, Rightarrow, "w_r"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad \text{(in } h'\text{)}$$

for some y_r occurring free in e_r. If $y_r \notin \{\overline{x_{rj}}^{n_r}\}$ the Lemma holds by choosing $y = y_r$ and $w_y = w_r$, since $E_r(y_r) = E(y_r)$. Now we assume that $y_r = x_{rj}$ for some $j \in \{1..n_r\}$. By the definition of E_r, it holds that $E_r(x) \overset{j_{C_r}}{\dashrightarrow} E_r(y_r)$, so we can transform the diagram above into the following one:

$$\begin{tikzcd} & E_r(x) \arrow[d, Rightarrow, "j_{C_r} w_r"'] & \\ p \arrow[r, Rightarrow, "w_p"] & \bullet & v \arrow[l, Rightarrow, "w_v"'] \end{tikzcd} \qquad \text{(in } h')$$

And the lemma holds by taking $y = x$ and $w_y = j_{C_r} w_r$, since $E_r(x) = E(x)$.

$\square$

4.2 Properties of the abstract interpretation

Lemma 6 (Conservative abstract interpretation). *Let e be an expression, Σ a signature environment and R a set of sharing relations. Then $R \subseteq S\ [\![e]\!]\ R\ \Sigma$.*

Proof. By structural induction on e. All cases are straightforward.

Lemma 7. *Let p_1, p_2 and p_3 be path expressions. Then:*

1. $L((p_1 \cdot p_2)|_{p_3}) = L(p_1|_{p_3} \cdot p_2) \cup L(p_2|_{(p_3|_{p_1})})$.
2. $L(p_1|_{p_2 \cdot p_3}) = L((p_1|_{p_2})|_{p_3})$.

Proof. Let us start with the $\subseteq$ inclusion of (1). Assume a word $w \in L((p_1 \cdot p_2)|_{p_3})$. Then, there exist $w_3 \in L(p_3)$ and $w_{12} \in L(p_1 \cdot p_2)$ such that $w_{12} = w_3 w$. Moreover, there exists a pair of words $w_1 \in L(p_1)$, $w_2 \in L(p_2)$ such that $w_{12} = w_1 w_2$, and therefore $w_1 w_2 = w_3 w$. It follows that either w_1 is a prefix of w_3 or viceversa. In the first case we can assume the existence of a word w' such that $w_3 = w_1 w'$, from which it follows that $w' \in L(p_3|_{p_1})$. We can rewrite the equality $w_1 w_2 = w_3 w$ into $w_1 w_2 = w_1 w' w$ or, equivalently, $w_2 = w' w$. Therefore, $w \in L(p_2|_{(p_3|_{p_1})})$. On the other hand, if w_3 is a prefix of w_1 then there exists a word w'' such that $w_1 = w_3 w''$. This implies $w'' \in L(p_1|_{p_3})$. By rewriting the above mentioned equality $w_1 w_2 = w_3 w$ into $w_3 w'' w_2 = w_3 w$ we obtain $w = w'' w_2$, or equivalently, $w \in L(p_1|_{p_3} \cdot p_2)$.

Regarding the $\supseteq$ inclusion of (1), assume $w \in L(p_1|_{p_3} \cdot p_2) \cup L(p_2|_{(p_3|_{p_1})})$. If $w \in L(p_1|_{p_3} \cdot p_2)$ there exist $w_1 \in L(p_1)$, $w_2 \in L(p_2)$, $w_3 \in L(p_3)$ and $w_{13} \in L(p_1|_{p_3})$ such that $w = w_{13} w_2$ and $w_1 = w_3 w_{13}$. By appending w_2 to both sides of the last equality we obtain $w_1 w_2 = w_3 w_{13} w_2 = w_3 w$, so $w \in L((p_1 \cdot p_2)|_{p_3})$. If $w \in L(p_2|_{(p_3|_{p_1})})$ then there exist $w_1 \in L(p_1)$, $w_2 \in L(p_2)$, $w_3 \in L(p_3)$ and $w_{31} \in L(p_3|_{p_2})$ such that $w_2 = w_{31} w$ and $w_3 = w_1 w_{31}$. By appending w to the latter equation it holds that $w_3 w = w_1 w_{31} w = w_1 w_2$, so $w \in L((p_1 \cdot p_2)|_{p_3})$.

Now we prove (2), starting with the $\subseteq$ inclusion. Assume $w \in L(p_1|_{p_2 \cdot p_3})$, so there exist $w_1 \in L(p_2)$, $w_2 \in L(p_2)$, and $w_3 \in L(p_3)$ satisfying $w_1 = w_2 w_3 w$. This implies $w_3 w \in L(p_1|_{p_2})$, and consequently, $w \in L((p_1|_{p_2})|_{p_3})$. The $\supseteq$ inclusion is proved similarly: assume $w \in L((p_1|_{p_2})|_{p_3})$. There exist $w_1 \in L(p_1)$, $w_2 \in L(p_2)$, $w_3 \in L(p_3)$ and $w_{12} \in L(p_1|_{p_2})$ such that $w_{12} = w_3 w$ and $w_1 = w_2 w_{12}$. By substituting the first equation into the second one we obtain $w_1 = w_2 w_3 w$ and hence $w \in L(p_1|_{p_2 \cdot p_3})$. $\square$

Lemma 8 (Monotonicity of the closure operation). *Let R and R' be two sets of relations, and $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ a sharing relation such that $y \neq x$. If $R \subseteq R'$ then:*

$$R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \subseteq R' \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$$

An immediate consequence is that $R \uplus_x^* R'' \subseteq R' \uplus_x^* R''$.

Proof. It follows trivially from the definition of $\uplus_x$.

Definition 4. *A set of sharing relations R is included in R' (written $R \sqsubseteq R'$) if for every sharing relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R$ and every pair of words $w_1 \in L(p_1)$, $w_2 \in L(p_2)$ there exists a sharing relation $x \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y \in R'$ such that $w_1 \in L(p'_1)$ and $w_2 \in L(p'_2)$. Two sets of relations R and R' are said to be* equivalent *(written $R \equiv R'$) if both $R \sqsubseteq R'$ and $R' \sqsubseteq R$ hold.*

Lemma 9 (Commutativity of the closure operation). *Let R be a set of sharing relations and $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$, $x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'$ a pair of sharing relations such that $y \neq x$ and $y' \neq x'$. Let us define $R_{x,x'}$ and $R_{x',x}$ as follows:*

$$R_{x,x'} \stackrel{def}{=} (R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}) \uplus_{x'} \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\}$$
$$R_{x',x} \stackrel{def}{=} (R \uplus_{x'} \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\}) \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$$

If $y' \neq x$ then $R_{x,x'} \sqsubseteq R_{x',x}$.

Proof. Let us define R_x and $R_{x'}$ as follows:

$$R_x \stackrel{\text{def}}{=} R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$$
$$R_{x'} \stackrel{\text{def}}{=} R \uplus_{x'} \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\}$$

From the definition of the closure operator it is easy to see that $R \subseteq R_x \subseteq R_{x,x'}$ and $R \subseteq R_{x'} \subseteq R_{x',x}$.

Assume $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x,x'}$ and two words $w_1 \in L(q_1)$, $w_2 \in L(q_2)$. If $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_x$, this means that $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$ but, since $R \subseteq R_{x'}$, we can apply Lemma 8 so as to obtain

$$z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x'} \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$$

or, equivalently, $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x',x}$. Therefore, the lemma holds in this case. As a consequence, we can assume in the following that $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \notin R_x$, but belongs to $R_{x,x'}$, so there are five possibilities left:

1. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x'\}$
2. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\}$
3. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1 \cdot p_3|_{p'_2}} \bullet \xleftarrow{p_4} z_3 \mid y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R_x\}$.
4. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p_4 \cdot p'_2|_{p_3}} z_3 \mid y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R_x\}$.
5. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1 \cdot p_3|_{p'_2}} \bullet \xleftarrow{p'_1 \cdot p_4|_{p'_2}} x' \mid y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} y' \in R_x\}$.

The lemma holds trivially under the first two assumptions, because both $x' \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x'$ and $x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'$ are contained within $R_{x'}$, and hence belong to $R_{x',x}$. We shall concentrate on the remaining three cases.

Let us start with the third case, i.e. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 = x' \xrightarrow{p'_1 \cdot p_3|_{p'_2}} \bullet \xleftarrow{p_4} z_3$ for some $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R_x$. Without loss of generality we assume that $z_1 = x'$, $z_2 = z_3$, $q_1 = p'_1 \cdot p_3|_{p'_2}$ and $q_2 = p_4$. We distinguish six cases according to the definition of R_x.

- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R$.
  In this case $x' \xrightarrow{p'_1 \cdot p_3|_{p'_2}} \bullet \xleftarrow{p_4} z_3 \in R \uplus_{x'} \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\} = R_{x'}$ which, in turn, is a subset of $R_{x',x}$. Therefore, $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x',x}$.
- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in \{x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x\}$.
  This implies $p_3 = \epsilon$, $p_4 = \epsilon$, $z_3 = x$, and $y' = x$, the latter of which leading to a contradiction, since we are assuming that $y' \neq x$. This case cannot hold.

- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$.
  Since $y' \neq x$, the only possibility is that $y' = y$, $z_3 = x$, $p_4 = p_1$, and $p_3 = p_2$. It is obvious that $y' \xrightarrow{p_2'} \bullet \xleftarrow{p_1'} x' \in R_{x'}$ or, equivalently, $y \xrightarrow{p_2'} \bullet \xleftarrow{p_1'} x' \in R_{x'}$. This implies the following fact,

$$x' \xrightarrow{p_1' \cdot p_2|_{p_2'}} \bullet \xleftarrow{p_1} x \in R_{x'} \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} = R_{x',x}$$

  which, since $z_1 = x'$, $q_1 = p_1' \cdot p_3|_{p_2'} = p_1' \cdot p_2|_{p_2'}$, $q_2 = p_4 = p_1$, and $z_2 = z_3 = x$ leads to

$$z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x'} \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} = R_{x',x}$$

  and the lemma holds in this case.
- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 = x \xrightarrow{p_1 \cdot p_5|_{p_2}} \bullet \xleftarrow{p_6} z_4$ for some $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_4 \in R$.
  Again, since $y' \neq x$, it follows that $y' = z_4$, $x = z_3$, $p_3 = p_6$, and $p_4 = p_1 \cdot p_5|_{p_2}$. As a consequence:

$$q_1 = p_1' \cdot p_3|_{p_2'} = p_1' \cdot p_6|_{p_2'} \qquad q_2 = p_4 = p_1 \cdot p_5|_{p_2} \tag{3}$$

  In this way, the sharing relation $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_4 \in R$ can be rewritten as $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R$. Therefore:

$$\begin{aligned}
& y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R \\
\Rightarrow\ & x' \xrightarrow{p_1' \cdot p_6|_{p_2'}} \bullet \xleftarrow{p_5} y \in R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\} \\
\Rightarrow\ & x' \xrightarrow{p_1' \cdot p_6|_{p_2'}} \bullet \xleftarrow{p_1 \cdot p_5|_{p_2}} x \in (R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\}) \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \\
\Leftrightarrow\ & x' \xrightarrow{p_1' \cdot p_6|_{p_2'}} \bullet \xleftarrow{p_1 \cdot p_5|_{p_2}} x \in R_{x',x}
\end{aligned}$$

  But, since $x' = z_1$ and $x = z_3 = z_2$, and by applying the facts shown in (3), we obtain $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x',x}$, which proves the lemma.
- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 = x \xrightarrow{p_1} \bullet \xleftarrow{p_6 \cdot p_2|_{p_5}} z_4$ for some $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_4 \in R$.
  This implies $y' = z_4$, $z_3 = x$, $p_3 = p_6 \cdot p_2|_{p_5}$, $p_4 = p_1$, and hence

$$q_1 = p_1' \cdot p_3|_{p_2'} = p_1' \cdot (p_6 \cdot p_2|_{p_5})|_{p_2'} \qquad q_2 = p_4 = p_1$$

  The language of q_1 can be decomposed as follows by using Lemma 7:

$$\begin{aligned}
L(q_1) &= L(p_1' \cdot p_6|_{p_2'} \cdot p_2|_{p_5}) \cup L(p_1' \cdot (p_2|_{p_5})|_{p_2'|_{p_6}}) \\
&= L(p_1' \cdot p_6|_{p_2'} \cdot p_2|_{p_5}) \cup L(p_1' \cdot p_2|_{p_5 \cdot p_2'|_{p_6}})
\end{aligned}$$

  Since $w_1 \in L(q_1)$, we distinguish cases:
  1. $w_1 \in L(p_1' \cdot p_6|_{p_2'} \cdot p_2|_{p_5})$
     Since $y' = z_4$, we get $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R$ and therefore:

$$\begin{aligned}
& y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R \\
\Rightarrow\ & x' \xrightarrow{p_1' \cdot p_6|_{p_2'}} \bullet \xleftarrow{p_5} y \in R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\} \\
\Rightarrow\ & x' \xrightarrow{p_1' \cdot p_6|_{p_2'} \cdot p_2|_{p_5}} \bullet \xleftarrow{p_1} x \in (R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\}) \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \\
\Leftrightarrow\ & x' \xrightarrow{p_1' \cdot p_6|_{p_2'} \cdot p_2|_{p_5}} \bullet \xleftarrow{p_1} x \in R_{x',x}
\end{aligned}$$

     The latter sharing relation can be rewritten as $z_1 \xrightarrow{p_1' \cdot p_6|_{p_2'} \cdot p_2|_{p_5}} \bullet \xleftarrow{q_2} z_2$, which proves the lemma.

2. $w_1 \in L(p'_1 \cdot p_2|_{p_5 \cdot p'_2|_{p_6}})$
   As in the previous case, we start from $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R$:

$$\begin{aligned}
& y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R \\
\Rightarrow\ & x' \xrightarrow{p'_1} \bullet \xleftarrow{p_5 \cdot p'_2|_{p_6}} y \in R \uplus_{x'} \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\} \\
\Rightarrow\ & x' \xrightarrow{p'_1 \cdot p_2|_{p_5 \cdot p'_2|_{p_6}}} \bullet \xleftarrow{p_1} x \in (R \uplus_{x'} \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\}) \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \\
\Leftrightarrow\ & x' \xrightarrow{p'_1 \cdot p_2|_{p_5 \cdot p'_2|_{p_6}}} \bullet \xleftarrow{p_1} x \in R_{x',x}
\end{aligned}$$

   Therefore, $z_1 \xrightarrow{p'_1 \cdot p_2|_{p_5 \cdot p'_2|_{p_6}}} \bullet \xleftarrow{q_2} z_2 \in R_{x',x}$ and the lemma holds.
3. **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 = x \xrightarrow{p_1 \cdot p_5|_{p_2}} \bullet \xleftarrow{p_1 \cdot p_6|_{p_2}} x$ for some $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y \in R$. This case is not possible as $y' \neq x$.

After exploring these six possibilities, we go back to the original case distinction,

1. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x'\}$
2. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\}$
3. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1 \cdot p_3|_{p'_2}} \bullet \xleftarrow{p_4} z_3 \mid y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R_x\}$.
4. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p_4 \cdot p'_2|_{p_3}} z_3 \mid y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R_x\}$.
5. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p'_1 \cdot p_3|_{p'_2}} \bullet \xleftarrow{p'_1 \cdot p_4|_{p'_2}} x' \mid y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} y' \in R_x\}$.

and concentrate on the fourth case, i.e. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 = x' \xrightarrow{p'_1} \bullet \xleftarrow{p_4 \cdot p'_2|_{p_3}} z_3$ for some $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R_x$. Again, we assume without loss of generality that $z_1 = x'$, $z_2 = z_3$, $q_1 = p'_1$, and $q_2 = p_4 \cdot p'_2|_{p_3}$, and we distinguish cases according to the definition of R_x.

- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in R$.
  We obtain $x' \xrightarrow{p'_1} \bullet \xleftarrow{p_4 \cdot p'_2|_{p_3}} z_3 \in R \uplus_{x'} \{x' \xrightarrow{p'_1} \bullet \xleftarrow{p'_2} y'\} = R_{x'}$, so it follows that $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x'} \subseteq R_{x',x}$.
- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in \{x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x\}$.
  This case cannot hold, since $y' \neq x$.
- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$.
  Since $y' \neq x$, the only possibility is that $y' = y$, $z_3 = x$, $p_4 = p_1$, and $p_3 = p_2$. It is obvious that $y' \xrightarrow{p'_2} \bullet \xleftarrow{p'_1} x' \in R_{x'}$ or, equivalently, $y \xrightarrow{p'_2} \bullet \xleftarrow{p'_1} x' \in R_{x'}$. This implies the following fact,

$$x' \xrightarrow{p'_1} \bullet \xleftarrow{p_1 \cdot p'_2|_{p_2}} x \in R_{x'} \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} = R_{x',x}$$

  which, since $z_1 = x'$, $q_1 = p'_1$, $q_2 = p_4 \cdot p'_2|_{p_3} = p_1 \cdot p'_2|_{p_2}$, and $z_2 = z_3 = x$ leads to

$$z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_{x'} \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} = R_{x',x}$$

  which proves the lemma.
- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 = x \xrightarrow{p_1 \cdot p_5|_{p_2}} \bullet \xleftarrow{p_6} z_4$ for some $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_4 \in R$.
  Again, since $y' \neq x$, it follows that $y' = z_4$, $x = z_3$. $p_3 = p_6$, and $p_4 = p_1 \cdot p_5|_{p_2}$. As a consequence:

$$q_1 = p'_1 \qquad q_2 = p_4 \cdot p'_2|_{p_3} = p_1 \cdot p_5|_{p_2} \cdot p'_2|_{p_6}$$

  By using Lemma 7 we can show that

$$L(q_2) = L(p_1 \cdot p_5|_{p_2} \cdot p'_2|_{p_6}) \subseteq L(p_1 \cdot (p_5 \cdot p'_2|_{p_6})|_{p_2}) \tag{4}$$

The sharing relation $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_4$ can be rewritten as $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y'$ and hence:

$$
\begin{aligned}
& y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R \\
\Rightarrow\ & x' \xrightarrow{p_1'} \bullet \xleftarrow{p_5 \cdot p_2'|_{p_6}} y \in R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\} \\
\Rightarrow\ & x' \xrightarrow{p_1'} \bullet \xleftarrow{p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2}} x \in (R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\}) \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \\
\Leftrightarrow\ & x' \xrightarrow{p_1'} \bullet \xleftarrow{p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2}} x \in R_{x',x}
\end{aligned}
$$

But, since $x' = z_1$ and $x = z_3 = z_2$, we obtain $z_1 \xrightarrow{q_1} \bullet \xleftarrow{p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2}} z_2 \in R_{x',x}$, which proves the lemma, since it follows from (4) that $w_2 \in L(p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2})$.

- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 = x \xrightarrow{p_1} \bullet \xleftarrow{p_6 \cdot p_2|_{p_5}} z_4$ for some $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_4 \in R$.
This implies $y' = z_4$, $z_3 = x$, $p_3 = p_6 \cdot p_2|_{p_5}$, and $p_4 = p_1$, and hence

$$q_1 = p_1' \qquad q_2 = p_4 \cdot p_2'|_{p_3} = p_1 \cdot p_2'|_{p_6 \cdot p_2|_{p_5}}$$

As in the previous case, we recall Lemma 7 in order to establish:

$$L(q_2) = L(p_1 \cdot p_2'|_{p_6 \cdot p_2|_{p_5}}) = L(p_1 \cdot (p_2'|_{p_6})|_{p_2|_{p_5}}) \subseteq L(p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2}) \tag{5}$$

Since $y' = z_4$, we get $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R$ and therefore:

$$
\begin{aligned}
& y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y' \in R \\
\Rightarrow\ & x' \xrightarrow{p_1'} \bullet \xleftarrow{p_5 \cdot p_2'|_{p_6}} y \in R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\} \\
\Rightarrow\ & x' \xrightarrow{p_1'} \bullet \xleftarrow{p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2}} x \in (R \uplus_{x'} \{x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'\}) \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \\
\Leftrightarrow\ & x' \xrightarrow{p_1'} \bullet \xleftarrow{p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2}} x \in R_{x',x}
\end{aligned}
$$

The latter sharing relation can be rewritten as $z_1 \xrightarrow{q_1} \bullet \xleftarrow{p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2}} z_2$, which proves the lemma, since $w_2 \in L(p_1 \cdot (p_5 \cdot p_2'|_{p_6})|_{p_2})$ by (5).

- **Case** $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z_3 = x \xrightarrow{p_1 \cdot p_5|_{p_2}} \bullet \xleftarrow{p_1 \cdot p_6|_{p_2}} x$ for some $y \xrightarrow{p_5} \bullet \xleftarrow{p_6} y \in R$. This case is not possible as $y' \neq x$.

Finally we consider the fifth case, i.e. $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in \{x' \xrightarrow{p_1' \cdot p_3|_{p_2'}} \bullet \xleftarrow{p_1' \cdot p_4|_{p_2'}} x' \mid y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} y' \in R_x\}$. As $y' \neq x$, the only possibility is that $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} y' \in R$ and consequently $z_1 \xrightarrow{q_1} \bullet \xleftarrow{q_2} z_2 \in R_x' \subseteq R_{x',x}$ and we are done.

□

Corollary 1. *Under the same conditions as Lemma 9, if $y' \neq x$ and $x' \neq y$ then $R_{x,x'} \equiv R_{x',x}$.*

Proof. Since $y' \neq x$ we apply Lemma 9 so as to obtain $R_{x,x'} \sqsubseteq R_{x',x}$. By swapping the roles of the relations $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ and $x' \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y'$ in the definitions of $R_{x,x'}$ and $R_{x',x}$ we can apply Lemma 9 again so as to get $R_{x',x} \sqsubseteq R_{x,x'}$. Therefore, $R_{x,x'} \equiv R_{x',x}$. □

Definition 5. *A* generalised substitution *θ is a set of pairs of variables, where the pair (x, y) specifies that x is going to be replaced by y. The* domain *of θ (denoted* dom θ*) is the set of variables x such that $(x, z) \in \theta$ for some z. The notation $[z/x] \in \theta$ is defined as follows:*

$$[z/x] \in \theta \Leftrightarrow_{def} (x, z) \in \theta \vee (x \notin \text{dom } \theta \wedge x = z)$$

If R is a set of sharing relations and θ is a generalised substitution, the set $R\theta$ is defined as follows:

$$R\theta = \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \mid x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in R, [x/x'] \in \theta, [y/y'] \in \theta\} \tag{6}$$

A generalised substitution θ is said to be injective *whenever $[x/z_1], [x/z_2] \in \theta$ implies $z_1 = z_2$. The inverse of a generalised substitution θ^{-1} is defined by $\theta^{-1} = \{(x, y) \mid (y, x) \in \theta\}$.*

It can be easily shown that $[x/y] \in \theta$ iff $[y/x] \in \theta^{-1}$. By abuse of notation we denote by $[y_1/x_1, \ldots, y_n/x_n]$ the substitution $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. It is easy to see that the notation $R[\overline{a_i/x_i}^n]$ used above is a particular case of (6).

As an example, assume $\theta = [a_1/x, a_2/x, a_2/y, a_3/y]$ and R defined as follows:

$$R = \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y, y \xrightarrow{p_3} \bullet \xleftarrow{p_4} z\}$$

Hence we get:

$$\begin{aligned} R\theta = \{ \; & a_1 \xrightarrow{p_1} \bullet \xleftarrow{p_2} a_3, a_2 \xrightarrow{p_1} \bullet \xleftarrow{p_2} a_3, a_1 \xrightarrow{p_1} \bullet \xleftarrow{p_2} a_4, a_2 \xrightarrow{p_1} \bullet \xleftarrow{p_2} a_4, \\ & a_3 \xrightarrow{p_3} \bullet \xleftarrow{p_4} z, a_4 \xrightarrow{p_3} \bullet \xleftarrow{p_4} z\} \end{aligned}$$

Lemma 10 (Properties of substitution). *Let R, R_1, R_2 be sets of relations and θ a generalised substitution. Then:*

1. $(R_1 \cup R_2)\theta = R_1\theta \cup R_2\theta$.
2. *If $x \notin \mathrm{dom}(\theta) \cup ran(\theta)$, then $(R \uplus_x^* R')\theta = R\theta \uplus_x^* R'\theta$.*
3. $R\theta\theta^{-1} \supseteq R$.
4. *If θ is injective, then $R\theta\theta^{-1} = R$.*

Proof. We prove (1) as follows:

$$\begin{aligned} & x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in (R_1 \cup R_2)\theta \\ \Leftrightarrow\ & \exists x', y'.x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in (R_1 \cup R_2)\theta \wedge [x/x'], [y/y'] \in \theta \\ \Leftrightarrow\ & \exists x', y'.(x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in R_1 \vee x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in R_2) \wedge [x/x'], [y/y'] \in \theta \\ \Leftrightarrow\ & (\exists x', y'.x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in R_1 \wedge [x/x'], [y/y'] \in \theta) \;\; \vee \\ & (\exists x', y'.x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in R_2 \wedge [x/x'], [y/y'] \in \theta) \\ \Leftrightarrow\ & x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R_1\theta \vee x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R_2\theta \\ \Leftrightarrow\ & x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R_1\theta \cup R_2\theta \end{aligned}$$

In order to prove (2) it is enough to prove that given a variable $y \neq x$:

$$(R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\})\theta = R\theta \uplus_x^* \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$$

From the fact that $x \notin \mathrm{dom}(\theta) \cup ran(\theta)$ and (1) we have property (2) by definition of $\uplus_x^*$.

Notice that the right hand side is well defined because by hypothesis $x \notin \mathrm{dom}(\theta) \cup \mathrm{ran}(\theta)$, so each z such that$[z/y] \in \theta$ holds that $z \neq x$. First we prove $\subseteq$.

$$\begin{aligned} & u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in (R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\})\theta \\ \Leftrightarrow\ & \exists u', v'.u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' \in R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\} \wedge [u/u'], [v/v'] \in \theta \end{aligned}$$

Now we can distinguish the following cases:

$u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' \in R$ Then by definition of substitution $u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in R\theta \subseteq R\theta \uplus_x^* \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$.

$u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' = x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ Similarly, $u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta \subseteq R\theta \uplus_x^* \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$.

$u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' = x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x$ As $x \notin \mathrm{dom}(\theta)$, $u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v = x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x \in R\theta \uplus_x^* \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$, by definition of closure.

$u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' = x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_4} z$ such that $y \xrightarrow{p_3} \bullet \xleftarrow{p_4} z \in R$ and $p_3|_{p_2} \neq \emptyset$. Then for each z', y' such that $[z'/z], [y'/y] \in \theta$ we know that $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z' \in R\theta$ and $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$, and consequently $x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_4} z' \in R\theta \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y'\} \subseteq R\theta \uplus_x^* \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$. In particular it holds for $z' = v$ as $v' = z$ and $[v/v'] \in \theta$, and so we are done.

$u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' = x \xrightarrow{p_1} \bullet \xleftarrow{p_4 \cdot p_2|_{p_3}} z$ such that $y \xrightarrow{p_3} \bullet \xleftarrow{p_4} z \in R$ and $p_2|_{p_3} \neq \emptyset$. Similar to the previous case.

$u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' = x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_1 \cdot p_4|_{p_2}} x$ such that $y \xrightarrow{p_3} \bullet \xleftarrow{p_4} y \in R$ and $p_3|_{p_2} \neq \emptyset$, $p_4|_{p_2} \neq \emptyset$. Similar to the previous cases.

Now we prove $\supseteq$. Let $u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in R\theta \uplus^*_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$. As $y \neq x$ and $x \notin \mathrm{dom}(\theta) \cup \mathrm{ran}(\theta)$ we only have to distinguish the following cases:

$u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in R\theta$ Then there exist u' and v' such that $u' \xrightarrow{q_1} \bullet \xleftarrow{q_2} v' \in R$ and $[u/u'], [v/v'] \in \theta$. As $R \subseteq R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$ then trivially $u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in (R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\})\theta$.

$u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$ This means that $u = x$ and $[v/y] \in \theta$. As $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$, trivially $u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v \in (R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\})\theta$.

$u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v = x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x$ Relation $x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x \in R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$ and consequently also in $(R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\})\theta$ because $x \notin \mathrm{dom}(\theta)$.

$u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v = x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_4} z'$ such that there exist $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$ and $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z' \in R\theta$ such that $[y'/y] \in \theta$, $p_3|_{p_2} \neq \emptyset$. So there exists z such that $y \xrightarrow{p_3} \bullet \xleftarrow{p_4} z \in R$ and $[z'/z] \in \theta$. Consequently, by definition of $\uplus_x$, $x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_4} z \in R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$ and finally $x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_4} z' \in (R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\})\theta$.

$u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v = x \xrightarrow{p_1} \bullet \xleftarrow{p_4 \cdot p_2|_{p_3}} z'$ such that there exist $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$ and $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} z' \in R\theta$ such that $[y'/y] \in \theta$, $p_2|_{p_3} \neq \emptyset$. This case is similar to the previous one.

$u \xrightarrow{q_1} \bullet \xleftarrow{q_2} v = x \xrightarrow{p_1 \cdot p_3|_{p_2}} \bullet \xleftarrow{p_1 \cdot p_4|_{p_2}} x$ such that there exist $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}\theta$ and $y' \xrightarrow{p_3} \bullet \xleftarrow{p_4} y' \in R\theta$ such that $[y'/y] \in \theta$, $p_3|_{p_2} \neq \emptyset$, $p_4|_{p_2} \neq \emptyset$. This case is similar to the previous ones.

In order to prove (3), let us assume $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R$. There must exist some variables x', y' such that $[x'/x], [y'/y] \in \theta$. Therefore, $x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in R\theta$. Moreover, $[x/x'], [y/y'] \in \theta^{-1}$, so we get $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R\theta\theta^{-1}$.

We have only to prove the $\subseteq$ inclusion of (4), since the $\supseteq$ follows from (3). Let us assume $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R\theta\theta^{-1}$. By the definition of substitution there must be some x', y' such that $x' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y' \in R\theta$ and $[x/x'], [y/y'] \in \theta^{-1}$, the latter of which implies that $[x'/x], [y'/y] \in \theta$. Moreover, by applying the definition of substitution again, there exist some x'', y'' such that $x'' \xrightarrow{p_1} \bullet \xleftarrow{p_2} y'' \in R$ and $[x'/x''], [y'/y''] \in \theta$. Since θ is injective it holds that $x = x''$ and $y = y''$, from which we get $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R$. ◻

4.3 Notion of correct approximation

Now we define when a set of relations correctly approximates the real sharing in a heap and the notion of correct signature. The first definition reflects the fact that at least the minimum sharing must be recorded in the relations, i.e. the paths leading to the first point of confluence must be recorded, while their extensions with a common path need not. Notice that this means that in case of internal sharing, each point of internal confluence must also be recorded.

A correct function signature must record enough sharing information to be able to approximate each possible call to that function, i.e. each possible execution of the body. The operational semantics of Core-Safe can be found at [9]. It is a standard big-step operational eager semantics: judgment $E \vdash h, e \Downarrow h', v$ means that expression e in a variable environment E and initial heap h evaluates to value v and the heap changes to h'. If in a heap h there exists an actual sharing between two variables x and y through respective pointer paths w_1 and w_2, we say that there exists a *sharing condition* in h and denote it by $E(x) \xRightarrow{w_1} \bullet \xLeftarrow{w_2} E(y)(\text{in } h)$.

Definition 6. *A sharing relation* $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ *is said to* approximate *a sharing condition* $E(x) \xRightarrow{w_1} \bullet \xLeftarrow{w_2} E(y)$ (in h) *iff there exists a word* w *such that* $w_1 \in L(p_1 w)$ *and* $w_2 \in L(p_2 w)$.

Definition 7. *Let R be a set of sharing relations, E a runtime environment, and h a heap. We say that R is a correct approximation of E and h, denoted $R \succeq (E, h)$, iff for every pair of variables $x, y \in \operatorname{dom} E$, and pair of words $w_1, w_2 \in \mathcal{V}^*$ if the condition $E(x) \stackrel{w_1}{\Longrightarrow} \bullet \stackrel{w_2}{\Longleftarrow} E(y)$ (in h) holds, it is approximated by a sharing relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ in R.*

Lemma 11 (Properties of correct approximations). *For any R, R', E, h, x, z, v of their respective types:*

1. *If $R \succeq (E, h)$ and $R \equiv R'$, then $R' \succeq (E, h)$.*
2. *If $R \succeq (E, h)$ and $R \sqsubseteq R'$ then $R' \succeq (E, h)$.*
3. *If $R \succeq (E \uplus [\overline{x_i \mapsto v}^n], h)$ then $R[\overline{z_i / x_i}^n] \succeq (E \uplus [\overline{z_i \mapsto v}^n], h)$.*
4. *If $R \succeq (E \uplus [x \mapsto v], h)$ then $R \backslash \{x\} \succeq (E, h)$.*

Proof. The three properties follow trivially from the definition of $\succeq$.

So far we have used the notation $R[\overline{z_i/x_i}^n]$ to denote the substitution of the x_i variables appearing in each side of R by their corresponding z_i. In this context, the $[\overline{z_i/x_i}^n]$ can be interpreted as a function θ that maps each x_i to z_i. When proving the correctness analysis below, we shall use a slight generalization of substitutions in which each θ is a *relation*, rather than a function.

Definition 8 (Correct signature). *A set R of relations is a correct signature for a function definition $f\ \overline{x_i}^n = e_f$ iff for each execution $E_f \vdash h, e_f \Downarrow h', v$ of the body of the function and every set of relations R' such that $R' \succeq (E_f, h)$ it holds that $R' \uplus^*_{res} R \succeq (E_f \uplus [res \mapsto v], h')$. A signature environment Σ is said to be correct iff every signature it contains is correct.*

4.4 Correctness

Correctness of the analysis is divided into two steps. First, we prove that given correct signatures of the functions which are called from an expression, the interpretation of the expression is correct. Then, we prove that the interpretation of a function generates a correct signature. For both theorems we need to prove that the transitive closure operator is correct, which we show in the following two lemmas. The second lemma concerns the case in which a variable gets internal sharing through another variable having internal sharing, i.e. the fourth line of operator $\uplus_x$ definition.

Lemma 12 (Transitive closure lemma). *Let us assume a runtime environment E, a heap h, a set of sharing relations R, some variables x, y, z, (with $y \neq z$) words w_x, w_y, w_z, and paths p_{xy}, p_{yx}, p_{yz}, p_{zy} such that the following holds:*

$$E(x) \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E(y)\ (\text{in } h),\ \textit{approximated by}\ x \xrightarrow{p_{xy}} \bullet \xleftarrow{p_{yx}} y \in R$$
$$E(z) \stackrel{w_z}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E(y)\ (\text{in } h),\ \textit{approximated by}\ z \xrightarrow{p_{zy}} \bullet \xleftarrow{p_{yz}} y$$

Then there exists a sharing relation $x \xrightarrow{p_{xz}} \bullet \xleftarrow{p_{zx}} z \in R \uplus_z \{y \xrightarrow{p_{yz}} \bullet \xleftarrow{p_{zy}} z\}$ which approximates $E(x) \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E(z)$ (in h).

Proof. From the assumptions of the lemma we know that there exist two words w and w' such that the following hold:

$$\begin{array}{l}
\text{(A) } w_x \in L(p_{xy} w) \\
\text{(B) } w_y \in L(p_{yx} w) \\
\text{(C) } w_y \in L(p_{yz} w') \\
\text{(D) } w_z \in L(p_{zy} w'),
\end{array}$$

and we have to prove the existence of a word w'' such that:

$$\begin{array}{l}
\text{(E) } w_x \in L(p_{xz} w'') \\
\text{(F) } w_z \in L(p_{zx} w'')
\end{array}$$

By (B) and (C) there exist two words $w_{yx} \in L(p_{yx})$ and $w_{yz} \in L(p_{yz})$ such that $w_y = w_{yx}w = w_{yz}w'$, so either w_{yx} is a prefix of w_{yz}, or w_{yz} is a prefix of w_{yx}. We distinguish cases:

- **Case 1:** w_{yx} is a prefix of w_{yz}
  This implies the existence of a word w_0 such that $w_{yz} = w_{yx}w_0$. Hence $w_y = w_{yz}w' = w_{yx}w_0w'$ and $w_y = w_{yx}w$, so $w = w_0w'$. Moreover, since $w_{yx}w_0 \in L(p_{yz})$ and $w_{yx} \in L(p_{yx})$ it holds that $w_0 \in L(p_{yz}|_{p_{yx}})$.
  The lemma holds if we take $p_{xz} = p_{xy} \cdot p_{yz}|_{p_{yx}}$, $p_{zx} = p_{zy}$, and $w'' = w'$, as by definition of $\uplus_z$ we obtain:

  $$z \xrightarrow{p_{zy}} \bullet \xleftarrow{p_{xy} \cdot p_{yz}|_{p_{yx}}} \in R \uplus_z \{z \xrightarrow{p_{zy}} \bullet \xleftarrow{p_{yz}} y\}$$

  or, equivalently, $x \xrightarrow{p_{xz}} \bullet \xleftarrow{p_{zx}} z \in R \uplus_z \{z \xrightarrow{p_{zy}} \bullet \xleftarrow{p_{yz}} y\}$.
  Moreover, (A) implies the existence of a $w_{xy} \in L(p_{xy})$ such that $w_x = w_{xy}w$. Therefore (E) holds since:

  $$w_x = w_{xy}w = w_{xy}w_0w' \in L(p_{xy} \cdot p_{yz}|_{p_{yx}} \cdot w') = L(p_{xz}w'')$$

  Finally, (D) implies the existence of a $w_{zy} \in L(p_{zy})$ such that $w_z = w_{zy}w'$. Therefore (F) holds since:

  $$w_z = w_{zy}w' \in L(p_{zy}w') = L(p_{zx}w'')$$

- **Case 2:** w_{yz} is a prefix of w_{yx}. It is dual to the previous case. There exists an w_0 such that $w_{yx} = w_{yz}w_0$, which implies that $w_0 \in L(p_{yx}|_{p_{yz}})$. On the other hand, by (B) and (C) we get $w' = w_0w$, and the Lemma holds if we take $p_{zx} = p_{zy} \cdot p_{yx}|_{p_{yz}}$, $p_{xz} = p_{xy}$, and $w'' = w$. By definition of $\uplus_z$ operator we get, indeed:

  $$z \xrightarrow{p_{zy} \cdot p_{yx}|_{p_{yz}}} \bullet \xleftarrow{p_{xy}} x \in R \uplus_z \{z \xrightarrow{p_{zy}} \bullet \xleftarrow{p_{yz}} y\}$$

  In addition, from (A) and (D) there are two words $w_{xy} \in L(p_{xy})$ and $w_{zy} \in L(p_{zy})$ such that $w_x = w_{xy}w$ and $w_z = w_{zy}w'$. This allows us to prove (E) and (F):

  $$w_x = w_{xy}w \in L(p_{xy}w) = L(p_{xz}w'')$$

  $$w_z = w_{zy}w' = w_{zy}w_0w \in L(p_{zy} \cdot p_{yx}|_{p_{yz}}w) = L(p_{zx}w'')$$

$\square$

Lemma 13 (Transitive self-closure lemma). *Let us assume a runtime environment E, a heap h, a set of sharing relations R, some variables x, y (with $x \neq y$), words w_x, w_y, w_1, w_2 and paths p_{x1}, p_{x2}, p_{xy}, p_{yx} such that the following holds:*

$$E(x) \overset{w_xw_1}{\Longrightarrow} \bullet \overset{w_xw_2}{\Longleftarrow} E(x) \text{ (in } h\text{), being approximated by } x \xrightarrow{p_{x1}} \bullet \xleftarrow{p_{x2}} x \in R$$
$$E(x) \overset{w_x}{\Longrightarrow} \bullet \overset{w_y}{\Longleftarrow} E(y) \text{ (in } h\text{), being approximated by } x \xrightarrow{p_{xy}} \bullet \xleftarrow{p_{yx}} y$$

Then there exists a sharing relation $y \xrightarrow{p_{y1}} \bullet \xleftarrow{p_{y2}} y \in R \uplus_y \{x \xrightarrow{p_{xy}} \bullet \xleftarrow{p_{yx}} y\}$ which approximates $E(y) \overset{w_yw_1}{\Longrightarrow} \bullet \overset{w_yw_2}{\Longleftarrow} E(y)$ (in h).

Proof. If $w_1 = w_2$ the lemma trivially holds because $E(y) \overset{w_yw_1}{\Longrightarrow} \bullet \overset{w_yw_2}{\Longleftarrow} E(y)$ (in h) is approximated by $y \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} y$. So, let assume that $w_1 \neq w_2$.

From the assumptions of the lemma we know that there exist two words w and w' such that:

(A) $w_xw_1 \in L(p_{x1}w)$
(B) $w_xw_2 \in L(p_{x2}w)$
(C) $w_x \in L(p_{xy}w')$
(D) $w_y \in L(p_{yx}w')$,

and we have to prove the existence of a word w'' such that:

$$\text{(E) } w_y w_1 \in L(p_{y1} w'')$$
$$\text{(F) } w_y w_2 \in L(p_{y2} w'')$$

By (D) there exists a word $w_{yx} \in L(p_{yx})$ such that $w_y = w_{yx} w'$.

By (A) and (B) there exist two words $w_{x1} \in L(p_{x1}$ and $w_{x2} \in L(p_{x2}$ such that $w_x w_1 = w_{x1} w$ and $w_x w_2 = w_{x2} w$. As $w_1 \neq w_2$ trivially $w_{x1} \neq w_{x2}$. By (C) there exists a word w_{xy} such that $w_x = w_{xy} w'$.

So $w_{xy} w' w_1 = w_{x1} w$ and $w_{xy} w' w_2 = w_{x2} w$.

We distinguish the following cases:

- **Case 1:** w_{xy} is a prefix of both w_{x1} and w_{x2}.
  This implies the existence of w_{xy1} and w_{xy2} such that $w_{x1} = w_{xy} w_{xy1}$ and $w_{x2} = w_{xy} w_{xy2}$. Consequently $w_{xy1} \in L(p_{x1}|_{p_{xy}})$ and $w_{xy2} \in L(p_{x2}|_{p_{xy}})$. Also, $w_{xy} w' w_1 = w_{x1} w = w_{xy} w_{xy1} w$ and $w_{xy} w' w_2 = w_{x2} w = w_{xy} w_{xy2} w$, which implies that $w' w_1 = w_{xy1} w$ and $w' w_2 = w_{xy2} w$ (*).
  If we take $p_{y1} = p_{yx} \cdot p_{x1}|_{p_{xy}}$ and $p_{y2} = p_{yx} \cdot p_{x2}|_{p_{xy}}$, we know by definition that $y \xrightarrow{p_{y1}} \bullet \xleftarrow{p_{y2}} y \in R \uplus_y \{x \xrightarrow{p_{xy}} \bullet \xleftarrow{p_{yx}} y\}$. We can take $w'' = w$ because by (*) $w_y w_1 = w_{yx} w' w_1 = w_{yx} w_{xy1} w \in L(p_{y1} w)$ and $w_y w_2 = w_{yx} w' w_2 = w_{yx} w_{xy2} w \in L(p_{y2} w)$.
- **Case 2:** w_{x1} and w_{x2} are prefix of w_{xy}. This case is not possible because we would have that $E(x) \overset{w_{x1}}{\Longrightarrow} \bullet \overset{w_{x2}}{\Longleftarrow} E(x)$ (in h), with $w_{x1} \neq w_{x2}$ both prefixes of a unique path w_x.
- **Case 3:** w_{x1} is prefix of w_{xy} and w_{xy} is prefix of w_{x2}. Again, this case is not possible because we would have that w_{x1} is prefix of w_{x2}, and as $w_{x1} \neq w_{x2}$ then there exists $w''' \neq \epsilon$ such that $E(x) \overset{w_{x1}}{\Longrightarrow} \bullet \overset{w_{x1} w'''}{\Longleftarrow} E(x)$ (in h), which is not possible in our heaps.
- **Case 4:** w_{x2} is prefix of w_{xy} and w_{xy} is prefix of w_{x1} (symmetric to the previous one)

The following theorem establishes the correctness of the abstract interpretation modulo the correctness of function signatures.

Theorem 1. *Assume an expression e, a set of sharing relations R and a correct signature environment Σ. If S $[\![e]\!]$ R $\Sigma = R'$, then for every execution $E \vdash h, e \Downarrow h', v$ in which $R \succeq (E, h)$, it holds that $R' \succeq (E \uplus [res \mapsto v], h')$.*

Proof. Let us denote the environment $E \uplus [res \mapsto v]$ by E'. We have to prove that for every $x, y \in \operatorname{dom} E'$ the runtime sharing condition $E'(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(y)$ (in h') implies the existence of a sharing relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R'$ and a word w such that $w_1 \in L(p_1 w)$ and $w_2 \in L(p_2 w)$. If x and y are distinct from *res*, then by Lemma 2 we know that $E(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E(y)$ (in h), and since $R \succeq (E, h)$, there exists a relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R$ and a word w satisfying the same conditions, but since R is a subset of R' (by Lemma 6), the Theorem holds when x and y are distinct from *res*, so henceforth we shall assume that at least one x and y is the *res* variable. We shall assume without loss of generality that $y = res$. We proceed by induction on the size of the $\Downarrow$-derivation. We distinguish cases on the structure of e.

- **Case** $e \equiv c$
  Since $E'(res)$ is not a pointer, the condition $E'(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(res)$ (in h') cannot hold for any x, so the Lemma holds vacuously in the case $y = res$.
- **Case** $e \equiv z$
  Assume $E'(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(res)$ (in h'), since $E'(res) = v = E(z)$ we obtain $E'(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E(z)$ (in h').
  $x \neq res$ We can substitute $E(x)$ for $E'(x)$ and by closure preservation (Lemma 2) this relation also holds in h, so we can assume the existence of a sharing relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} z \in R$ approximating this sharing (both in h and h'). As $E'(z) \overset{\epsilon}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E'(res)$ (in h'), by path extension lemma (Lemma 3) also $E'(z) \overset{w_2}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(res)$ (in h'). Relation $res \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} z$ approximates this sharing. Then, by transitive closure lemma there exists a relation $x \xrightarrow{p_3} \bullet \xleftarrow{p_4} z$ in $R \uplus_{res} \{res \longrightarrow^{\epsilon} \bullet \xleftarrow{\epsilon} z\} \subseteq R'$, approximating $E'(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(res)$ (in h').

$x = res$ We have $E(z) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E(z)$ (in h'), and so in h by closure preservation. This implies the existence of a sharing relation $z \overset{p_1}{\longrightarrow} \bullet \overset{p_2}{\longleftarrow} z \in R$ approximating this sharing. Relation $res \overset{\epsilon}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} z$ approximates $E'(z) \overset{\epsilon}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E'(res)$ (in h').

Then, by transitive self-closure lemma (Lemma 13), there exists a relation $res \overset{p_3}{\longrightarrow} \bullet \overset{p_4}{\longleftarrow} res \in R \uplus_{res} \{res \overset{\epsilon}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} z\} \subseteq R'$ approximating $E'(res) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(res)$ (in h').

- **Case** $e \equiv C\ \overline{a_i}^n$

Let us assume first that $E'(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(res)$ (in h') with $x \neq res$. That is, $E'(x) = E(x)$. The word w_2 must be distinct from ϵ, since otherwise we would have $E(x) \overset{w_1}{\Longrightarrow}_{h'} E'(res)$ and by Lemma 2 it would follow that $E(x) \overset{w_1}{\Longrightarrow}_h E'(res)$, leading to a contradiction, as $E'(res)$ does not appear in h. Therefore, let us assume that $w_2 = j_C w_2'$ for some $j \in \{1..n\}$. This implies that $E'(res) \overset{j_C}{\Longrightarrow}_{h'} E(a_j)$ and, by path splitting (Lemma 4) we get $E(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E(a_j)$ (in h'). This implies, by closure preservation, $E(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E(a_j)$ (in h) and, since R correctly approximates (E, h), there is a relation $x \overset{p_1}{\longrightarrow} \bullet \overset{p_2}{\longleftarrow} a_j \in R$ approximating this sharing (both in h and h').

As $E'(res) \overset{j_C}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E(a_j)$ (in h'), by path extension lemma, also $E'(res) \overset{j_C w_2'}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E(a_j)$ (in h'), which is approximated by $res \overset{j_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_j$.

By transitive closure lemma there exists a relation $res \overset{p_3}{\longrightarrow} \bullet \overset{p_4}{\longleftarrow} x \in R \uplus_{res} \{res \overset{j_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_j\} \subseteq R'$ approximating $E'(x) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E'(res)$ (in h'), which proves the result when $x \neq res$.

If $x = res$ we have to distinguish cases:

1. $w_1 = w_2$. The runtime relation $E'(res) \overset{w_1}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} E'(res)$ (in h') is correctly approximated by the relation $res \overset{\epsilon}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} res \in R'$.
2. $w_1 = \epsilon, w_2 \neq \epsilon$. This case cannot hold, as w_2 must be of the form $j_C w_2'$ for some w_2 and some $j \in \{1..n\}$. This would imply, by path splitting (Lemma 4), $E(a_j) \overset{w_2'}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E'(res)$ (in h') or, in other words, $E(a_j) \overset{w_2'}{\Longrightarrow}_{h'} E'(res)$. By closure preservation (Lemma 2) we could replace h' by h in this relation leading to a contradiction, as $E'(res)$ does not belong to the domain of h.
3. $w_1 \neq \epsilon, w_2 = \epsilon$. Similarly as above, this case cannot hold.
4. $w_1 \neq w_2, w_1 \neq \epsilon, w_2 \neq \epsilon$. In this case w_1 must be of the form $i_C w_1'$ and w_2 must be of the form $j_C w_2'$ for some words w_1', w_2' and some constructor positions $i, j \in \{1..n\}$. By path splitting we obtain the following relations in h':

$$E'(res) \overset{i_C}{\Longrightarrow} E(a_i) \overset{w_1'}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E(a_j) \overset{j_C}{\Longleftarrow} E'(res)$$

We have to distinguish two cases $i_C \neq j_C$ and $i_C = j_C$.

$i_C \neq j_C$ In this case the relation $E(a_i) \overset{w_1'}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E(a_j)$ also holds in h by closure preservation. Hence there exists a sharing relation $a_i \overset{p_1}{\longrightarrow} \bullet \overset{p_2}{\longleftarrow} a_j \in R$ (1) approximating this sharing.

From the facts $E'(res) \overset{i_C}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E'(a_i)$ (in h') and $E'(res) \overset{j_C}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E'(a_j)$ (in h') by path extension lemma, also $E'(res) \overset{i_C w_1'}{\Longrightarrow} \bullet \overset{w_1'}{\Longleftarrow} E'(a_i)$ (in h') and $E'(res) \overset{j_C w_2'}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E'(a_j)$ (in h'), which are respectively approximated by $res \overset{i_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_i$ (2) and $res \overset{j_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_j$ (3).

By transitive closure lemma applied to (1) and (2) there exists a relation $res \overset{p_3}{\longrightarrow} \bullet \overset{p_4}{\longleftarrow} a_j \in R \uplus_{res} \{res \overset{i_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_i\}$ (4) approximating $E'(res) \overset{i_C w_1'}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E'(a_j)$ (in h'). Again by transitive closure lemma applied to (4) and (3) there exists a relation $res \overset{p_5}{\longrightarrow} \bullet \overset{p_6}{\longleftarrow} res \in R \uplus_{res} \{res \overset{i_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_i\} \uplus_{res} \{res \overset{i_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_j\} \subseteq R'$ approximating $E'(res) \overset{i_C w_1'}{\Longrightarrow} \bullet \overset{j_C w_2'}{\Longleftarrow} E'(res)$ (in h').

$i_C = j_C$ In this case the relation $E(a_i) \overset{w_1'}{\Longrightarrow} \bullet \overset{w_2'}{\Longleftarrow} E(a_i)$ holds in h by closure preservation. Hence there exists a sharing relation $a_i \overset{p_1}{\longrightarrow} \bullet \overset{p_2}{\longleftarrow} a_i \in R$ approximating this sharing. Relation $E'(res) \overset{i_C}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E'(a_i)$ (in h') is approximated by $res \overset{i_C}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} a_i$. Then by transitive self-closure lemma

(Lemma 13), there exists a relation $res \xrightarrow{p_3} \bullet \xleftarrow{p_4} res \in R \uplus_{res} \{res \xrightarrow{i_C} \bullet \xleftarrow{\epsilon} a_i\} \subseteq R'$ approximating $E'(res) \overset{i_C w'_1}{\Longrightarrow} \bullet \overset{i_C w'_2}{\Longleftarrow} E'(res)$ (in h') and we are done.

- **Case** $e \equiv g\ \overline{a_i}^n$
Let us assume that $g\ \overline{y_i}^n = e_g$ is the function definition of g, and assume the following execution:

$$\frac{E_g \vdash h, e_g \Downarrow h', v}{E \vdash h, g\ \overline{a_i}^n \Downarrow h', v} \qquad \text{where } E_g = [\overline{y_i \mapsto E(a_i)}^n]$$

Let E_0 be a runtime environment such that $E = E_0 \uplus [\overline{a_i \mapsto E(a_i)}^n]$. By assumption the following relation holds:

$$R \succeq (E_0 \uplus [\overline{a_i \mapsto E(a_i)}^n], h)$$

By Lemma 11 we can replace each a_i by its y_i so as to get:

$$R[\overline{y_i/a_i}^n] \succeq (E_0 \uplus [\overline{y_i \mapsto E(a_i)}^n], h)$$

Notice that $[\overline{y_i/a_i}^n]$ is not a standard substitution, but a generalised one. Moreover, since all the y_i are distinct, this substitution is injective. We can leave out the E_0 from this approximation relation so as to get:

$$R[\overline{y_i/a_i}^n] \succeq ([\overline{y_i \mapsto E(a_i)}^n], h)$$

which follows trivially from the previous one. From the definition of correct signature, we obtain:

$$R[\overline{y_i/a_i}^n] \uplus^*_{res} \Sigma(g) \succeq ([\overline{y_i \mapsto E(a_i)}^n] \uplus [res \mapsto v], h')$$

Now we substitute the a_i for the y_i in the environment of the right-hand side by using Lemma 11:

$$(R[\overline{y_i/a_i}^n] \uplus^*_{res} \Sigma(g))[\overline{a_i/y_i}^n] \succeq ([\overline{a_i \mapsto E(a_i)}^n] \uplus [res \mapsto v], h') \tag{7}$$

and we use the properties of Lemma 10 in order to transform the left-hand side:

$$\begin{array}{ll} & (R[\overline{y_i/a_i}^n] \uplus^*_{res} \Sigma(g))[\overline{a_i/y_i}^n] \\ = & \qquad \{\text{by Lemma 10 (2) as } res \neq \overline{y_i}, \overline{a_i}\} \\ & R[\overline{y_i/a_i}^n][\overline{a_i/y_i}^n] \uplus^*_{res} \Sigma(g)[\overline{a_i/y_i}^n] \\ = & \qquad \{\text{by Lemma 10 (4), since } [\overline{y_i/a_i}^n] \text{ is injective }\} \\ & R \uplus^*_{res} \Sigma(g)[\overline{a_i/y_i}^n] \end{array}$$

Therefore we can rewrite (7) so as to get:

$$R \uplus^*_{res} \Sigma(g)[\overline{a_i/y_i}^n] \succeq ([\overline{a_i \mapsto E(a_i)}^n] \uplus [res \mapsto v], h')$$

Notice that R is a subset of the left-hand side, and R correctly approximates E_0, so we can add E_0 to the right-hand side:

$$R \uplus^*_{res} \Sigma(g)[\overline{a_i/y_i}^n] \succeq (E_0 \uplus [\overline{a_i \mapsto E(a_i)}^n] \uplus [res \mapsto v], h')$$

which is equivalent to $R' \succeq (E', h')$.

- **Case** $e \equiv \textbf{let } x_1 = e_1 \textbf{ in } e_2$
We get the following execution:

$$\frac{E \vdash h, e_1 \Downarrow h_1, v_1 \quad E \uplus [x_1 \mapsto v_1] \vdash h_1, e_2 \Downarrow h', v}{E \vdash h, \textbf{let } x_1 = e_1 \textbf{ in } e_2 \Downarrow h', v}$$

Since $R \succeq (E, h)$ we can apply the induction hypothesis on the $\Downarrow$-derivation of e_1 and obtain:

$$S\ [\![e_1]\!]\ R\ \Sigma \succeq (E \uplus [res \mapsto v], h_1)$$

By Lemma 11 we can substitute x_1 for *res* in order to get:

$$(S\ [\![e_1]\!]\ R\ \Sigma)[x_1/res] \succeq (E \uplus [x_1 \mapsto v_1], h_1)$$

Let us denote the left-hand side by R_1. Now we can apply the induction hypothesis on the derivation of e_2,

$$S\ [\![e_2]\!]\ R_1\ \Sigma \succeq (E \uplus [x_1 \mapsto v_1] \uplus [res \mapsto v], h')$$

and apply Lemma 11 again,

$$(S\ [\![e_2]\!]\ R_1\ \Sigma)\backslash\{x_1\} \succeq (E \uplus [res \mapsto v], h')$$

which is what we wanted to prove.

- **Case** $e \equiv$ **case** x **of** $\overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^{n}$ Assume $h(E(x)) = C_r\ v_1 \cdots v_{n_r}$. In this case we get the following execution:

$$\frac{\overbrace{E \uplus [\overline{x_{rj} \mapsto v_j}^{n_r}]}^{E_r} \vdash h, e_r \Downarrow h', v}{E \vdash h, \textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^{n} \Downarrow h', v}$$

Now we prove that $R_r = R \uplus^*_{\overline{x_{rj}}} \{x \xrightarrow{j_{C_r}} \bullet \xleftarrow{\epsilon} x_{rj} \mid j \in \{1..n_r\}\} \succeq (E_r = E \uplus [\overline{x_{rj} \mapsto v_j}^{n_r}], h)$. Assume two variables $u, v \in \mathrm{dom}(E_r)$ such that $E_r(u) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E_r(v)$ (in h). We distinguish the following cases:

$u, v \in \mathrm{dom}(E)$. Then as $R \succeq (E, h)$, there exists $u \xrightarrow{p_1} \bullet \xleftarrow{p_2} v \in R$ approximating this sharing. As $R \subseteq R_r$, the property holds.

$u = y \in \mathrm{dom}(E)$**,** $v = x_{rj}$**,** $j \in \{1..n_r\}$. By hypothesis $h(E(x)) = C_r\ v_1 \cdots v_{n_r}$, so $E(y) \overset{w_1}{\Longrightarrow} \bullet \overset{j_{C_r} w_2}{\Longleftarrow} E(x)$(in h). As $R \succeq (E, h)$ there exists $y \xrightarrow{p_1} \bullet \xleftarrow{p_2} x \in R$ approximating this sharing.
From $E(x) \overset{j_{C_r}}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E_r(x_{rj})$(in h), by path extension (Lemma 3) we obtain $E(x) \overset{j_{C_r} w_2}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E_r(x_{rj})$(in h), which is approximated by $x \xrightarrow{j_{C_r}} \bullet \xleftarrow{\epsilon} x_{rj}$.
By transitive closure (Lemma 12) there exists a sharing relation $y \xrightarrow{p_3} \bullet \xleftarrow{p_4} x_{rj} \in R_r$ approximating $E_r(y) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E_r(x_{rj})$ (in h).

$u = x_{ri}$**,** $v = x_{rj}$**,** $i, j \in \{1..n_r\}$**,** $i \neq j$. In this case, $E(x) \overset{j_{C_r} w_2}{\Longrightarrow} \bullet \overset{i_{C_r} w_1}{\Longleftarrow} E(x)$ (in h). As $R \succeq (E, h)$ there exists $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} x \in R$ approximating this sharing.
From $E(x) \overset{j_{C_r}}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E_r(x_{rj})$(in h), by path extension (Lemma 3) we obtain $E(x) \overset{j_{C_r} w_2}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E_r(x_{rj})$(in h), which is approximated by $x \xrightarrow{j_{C_r}} \bullet \xleftarrow{\epsilon} x_{rj}$. By transitive closure (Lemma 12) there exists a sharing relation $x \xrightarrow{p_3} \bullet \xleftarrow{p_4} x_{rj} \in R_r$ approximating (1) $E(x) \overset{i_{C_r} w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E_r(x_{rj})$ (in h).
From $E(x) \overset{i_{C_r}}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E_r(x_{ri})$ (in h), by path extension (Lemma 3) we obtain (2) $E(x) \overset{i_{C_r} w_1}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} E_r(x_{ri})$ (in h), which is approximated by $x \xrightarrow{i_{C_r}} \bullet \xleftarrow{\epsilon} x_{ri}$. By transitive closure (Lemma 12) applied to (1) and (2) there exists a sharing relation $x_{ri} \xrightarrow{p_5} \bullet \xleftarrow{p_6} x_{rj} \in R_r$ approximating $E_r(x_{ri}) \overset{w_1}{\Longrightarrow} \bullet \overset{w_2}{\Longleftarrow} E_r(x_{rj})$ (in h).

$u = x_{ri} = v$**,** $i \in \{1..n_r\}$ In this case, $E(x) \overset{i_{C_r} w_2}{\Longrightarrow} \bullet \overset{i_{C_r} w_1}{\Longleftarrow} E(x)$ (in h). As $R \succeq (E, h)$ there exists $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} x \in R$ approximating this sharing. Relation $E(x) \overset{i_{C_r}}{\Longrightarrow} \bullet \overset{\epsilon}{\Longleftarrow} E(x_{ri})$ (in h) is approximated by $x \xrightarrow{i_C} \bullet \xleftarrow{\epsilon} x_{ri}$. Then, by transitive self-closure lemma (Lemma 13), there exists a relation $x_{ri} \xrightarrow{p_3} \bullet \xleftarrow{p_4} x_{ri}$ approximating $E(x_{ri}) \overset{w_2}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} E(x_{ri})$ (in h).

Then, by induction hypothesis,

$$S\ [\![e_r]\!]\ R_r\ \Sigma \succeq (E_r \uplus [res \mapsto v], h')$$

and by Lemma 11

$$S\ [\![e_r]\!]\ R_r\ \Sigma\backslash\{\overline{x_{rj}}^{n_r}\} \succeq (E \uplus [res \mapsto v], h')$$

Consequently,

$$\bigcup_i (S\ [\![e_i]\!]\ R_i\ \Sigma\backslash\{\overline{x_{ij}}^{n_i}\}) \succeq (E \uplus [res \mapsto v], h')$$

□

Now we prove that the interpretation of a function returns a correct signature. A signature records the sharing between the result and the arguments of the function assuming these are disjoint and without internal sharing. However, a real call to the function may not satisfy such assumtption. Given the real configuration (E,h), we define an hypothetical execution where both the environment $\widehat{E}$ and the heap $\widehat{h}$ contain the same information as (E,h) but meeting the separation property. The signature of the function captures the sharing information corresponding to this hypothetical execution.

Definition 9. *Let (E,h) and $(\widehat{E},\widehat{h})$ be two configurations such that $dom\ E = dom\ \widehat{E}$. A mapping $\gamma : dom\ \widehat{h} \to dom\ h$ is said to be an entanglement from $(\widehat{E},\widehat{h})$ to (E,h), iff:*

1. *For every pointer $\widehat{p} \in dom\ \widehat{h}$, if $\widehat{h}(\widehat{p}) = C\ \widehat{v}_1 \cdots \widehat{v}_n$, then $h(\gamma(\widehat{p})) = C\ \gamma(\widehat{v}_1)\ \cdots \gamma(\widehat{v}_n)$.*
2. *For every variable $x \in dom\ \widehat{E}$, $\gamma(\widehat{E}(x)) = E(x)$.*

As an example, assume a function definition $f\ x\ y = C\ y$. Its signature consists of the following relations: $\{res \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} res, res \xrightarrow{1_C} \bullet \xleftarrow{\epsilon} y\}$. Assume we execute a call $f\ z\ z$ where $E(z) = p$, $h(p) = C'\ p'\ p'$, $h(p') = C''\ 3$, i.e. $E_f = [x \mapsto p, y \mapsto p]$. In this case x and y are not disjoint and also contain internal sharing. We can define $(\widehat{E}_f,\widehat{h})$ such that $\widehat{E}_f(x) = p_1$, $\widehat{E}_f(y) = p_2$, $\widehat{h}(p_1) = C'\ p_1'\ p_1''$, $\widehat{h}(p_2) = C'\ p_2'\ p_2''$ and $\widehat{h}(p_1') = \widehat{h}(p_1'') = \widehat{h}(p_2') = \widehat{h}(p_2'') = C''\ 3$. Then $\gamma(p_1) = \gamma_(p_2) = p$, $\gamma(p_1') = \gamma(p_1'') = \gamma(p_2') = \gamma(p_2'') = p'$ is an entanglement from $(\widehat{E}_f,\widehat{h})$ to (E_f,h).

Lemma 14. *Let γ be an entanglement from $(\widehat{E},\widehat{h})$ to (E,h). For every $p \in dom\ h$ and $\widehat{p} \in dom\ \widehat{h}$ such that $\gamma(\widehat{p}) = p$.*

1. *If $\widehat{p} \overset{w}{\Longrightarrow}_{\widehat{h}} \widehat{q}$ for some $\widehat{q}$ and w, then $p \overset{w}{\Longrightarrow}_h \gamma(\widehat{q})$.*
2. *If $p \overset{w}{\Longrightarrow}_h q$ for some q, then there exists another $\widehat{q} \in dom\ \widehat{h}$ such that $\widehat{p} \overset{w}{\Longrightarrow}_h \widehat{q}$ and $\gamma(\widehat{q}) = q$.*

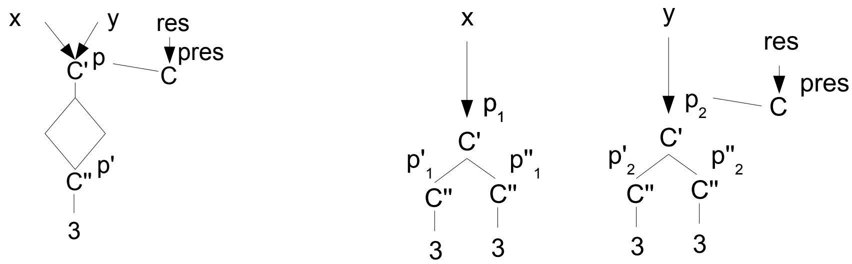
The following lemma proves that both the hypothetical and the real execution proceed in parallel and that the information inside the heap is the same although with a different shape. In Figure 9 we show the final heaps of the executions corresponding to the previous example.

Lemma 15. *Assume an execution $E \vdash h, e \Downarrow h', v$ and a configuration $(\widehat{E},\widehat{h})$. For every entanglement γ from $(\widehat{E},\widehat{h})$ to (E,h) there exist some $\widehat{h}'$, $\widehat{v}'$ and γ' such that:*

1. *$\widehat{E} \vdash \widehat{h}, e \Downarrow \widehat{h}', \widehat{v}$.*
2. *γ' is a conservative extension of γ. That is, $\gamma \subseteq \gamma'$.*
3. *γ' is an entanglement from $(\widehat{E},\widehat{h}')$ to (E,h').*
4. *$\gamma'(\widehat{v}) = v$.*

Proof. By induction on the size of the $\Downarrow$-derivation. We distinguish cases w.r.t. the structure of e.

- **Case** $e \equiv c$
  If we define $\widehat{v} = v = c$, $\widehat{h}' = \widehat{h}$, and $\gamma' = \gamma$, so γ' is a conservative extension of γ. We also get $\widehat{E} \vdash \widehat{h}, c \Downarrow \widehat{h}, c$, and γ' entangles $(\widehat{E},\widehat{h}')$ into (E,h') since $h = h'$ and γ entangles $(\widehat{E},\widehat{h})$ into (E,h). Finally, we get $\gamma'(\widehat{v}) = \gamma(c) = c = v$.



(a) (b)

Fig. 9: Final heaps in the real execution (a), and the untangled one (b).

- **Case** $e \equiv x$
  In this case we get $v = E(x)$ and $h' = h$. Let us define $\widehat{v} \stackrel{\text{def}}{=} \widehat{E}(x)$, $\gamma' \stackrel{\text{def}}{=} \gamma$, and $\widehat{h}' \stackrel{\text{def}}{=} \widehat{h}$, so we can obtain the derivation $\widehat{E} \vdash \widehat{h}, e \Downarrow \widehat{h}', \widehat{v}$. Obviously, γ' is a conservative extension of γ and an entanglement from $(\widehat{E}, \widehat{h}')$ to (E, h'), as $h = h'$ and γ is an entanglement from $(\widehat{E}, \widehat{h})$ to (E, h) by assumption. Moreover, $\gamma'(\widehat{v}) = \gamma(\widehat{E}(x)) = E(x) = v$, so the lemma holds in this case.
- **Case** $e \equiv C\, a_1 \cdots a_n$
  Assume a mapping γ that entangles $(\widehat{E}, \widehat{h})$ into (E, h). Under this case, we get $h' = h \uplus [p \mapsto C\ E(a_1) \cdots E(a_n)]$, where p is a fresh variable. The lemma holds with the following choices of $\widehat{h}'$, $\widehat{v}'$ and γ',
  $$\widehat{h}' \stackrel{\text{def}}{=} \widehat{h} \uplus [\widehat{p} \mapsto C\ \widehat{E}(a_1) \cdots \widehat{E}(a_n)] \qquad \widehat{v}' \stackrel{\text{def}}{=} \widehat{p} \qquad \gamma' \stackrel{\text{def}}{=} \gamma \uplus [\widehat{p} \mapsto p]$$
  being $\widehat{p}$ a fresh pointer not appearing in dom $\widehat{h}$. Indeed, it holds that $\widehat{E} \vdash \widehat{h}, e \Downarrow \widehat{h}', \widehat{p}$, and that γ' is a conservative extension of γ. In order to prove that γ' is an entanglement from $(\widehat{E}, \widehat{h}')$ to (E, h'), we show that
  $$h(\gamma'(\widehat{p})) = h(p) = C\ E(a_1) \cdots E(a_n) = C\ \gamma(\widehat{E}(a_1)) \cdots \gamma(\widehat{E}(a_n))$$
  whereas the entanglement condition also holds for the remaining pointers in dom $\widehat{h}$ by assumption. Finally, it holds that $\gamma(\widehat{v}) = \gamma(\widehat{p}) = p = v$, which proves the lemma.
- **Case** $e \equiv f\ a_1 \cdots a_n$
  In this case we get:
  $$\frac{f\ \overline{y_i} = e_f \qquad \overbrace{[\overline{y_i \mapsto E(a_i)}^{n}]}^{E_f} \vdash h, e_f \Downarrow h', v}{E \vdash h, f\ a_1 \cdots a_n \Downarrow h', v}$$
  In order to apply the induction hypothesis, let us define $\widehat{E}_f \stackrel{\text{def}}{=} [\overline{y_i \mapsto \widehat{E}(a_i)}^{n}]$. If γ entangles $(\widehat{E}, \widehat{h})$ into (E, h), it is easy to see that $\gamma(\widehat{E}_f(y_i)) = \gamma(\widehat{E}(a_i)) = E(a_i) = E_f(y_i)$, so γ also entangles $(\widehat{E}_f, \widehat{h})$ into (E_f, h). Therefore, by induction hypothesis there exists a heap $\widehat{h}'$, a result $\widehat{v}$, and an entanglement γ' from $(\widehat{E}_f, \widehat{h}')$ to (E_f, h') such that:
  $$\widehat{E}_f \vdash \widehat{h}, e_f \Downarrow \widehat{h}', \widehat{v} \qquad \gamma \subseteq \gamma' \qquad \gamma'(\widehat{v}) = v$$
  From the first fact we obtain the following derivation:
  $$\frac{f\ \overline{y_i} = e_f \qquad [\overline{y_i \mapsto \widehat{E}(a_i)}^{n}] \vdash \widehat{h}, e_f \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, f\ a_1 \cdots a_n \Downarrow \widehat{h}', \widehat{v}}$$

Now let us prove that γ' is an entanglement from $(\widehat{E}, \widehat{h}')$ to (E, h'). We only have to prove that $\gamma'(\widehat{E}(x)) = E(x)$ for every $x \in \text{dom } \widehat{E}$. Therefore, assume $x \in \text{dom } \widehat{E} = \text{dom } E$. We know that γ entangles $(\widehat{E}, \widehat{h})$ into (E, h), and that $E(x) \in \text{dom } h$, so $\gamma(\widehat{E}(x)) = E(x)$. But this implies that $\gamma'(\widehat{E}(x)) = E(x)$, for γ' being a conservative extension of γ. Therefore, the lemma holds.

- **Case** $e \equiv \textbf{let } x_1 = e_1 \textbf{ in } e_2$

  Assume the following execution of e:

$$\frac{E \vdash h, e_1 \Downarrow h_1, v_1 \qquad E \uplus [x_1 \mapsto v_1] \vdash h_1, e_2 \Downarrow h', v}{E \vdash h, \textbf{let } x_1 = e_1 \textbf{ in } e_2 \Downarrow h', v}$$

We can apply the induction hypothesis on the $\Downarrow$-derivation of e_1 and assume the existence of a heap $\widehat{h}_1$, a value $\widehat{v}_1$, and an entanglement γ_1 from $(\widehat{E}, \widehat{h}_1)$ to (E, h_1) such that:

$$\widehat{E} \vdash \widehat{h}, e_1 \Downarrow \widehat{h}_1, \widehat{v}_1 \qquad \gamma \subseteq \gamma_1 \qquad \gamma_1(\widehat{v}_1) = v_1$$

In order to apply the induction hypothesis on the $\Downarrow$-derivation of e_2. If we define $\widehat{E}_1 \stackrel{\text{def}}{=} \widehat{E} \uplus [x_1 \mapsto \widehat{v}_1]$ and $E_1 = E \uplus [x_1 \mapsto v_1]$, we have to prove that γ_1 entangles $(\widehat{E}_1, \widehat{h}_1)$ into (E_1, h_1), but this follows from the fact that γ_1 entangles $(\widehat{E}, \widehat{h}_1)$ into (E, h_1) and that $\gamma(\widehat{E}_1(x_1)) = \gamma(\widehat{v}_1) = v_1 = E(x_1)$. Therefore, we apply the induction hypothesis so as to get a heap $\widehat{h}'$, a value $\widehat{v}$ and an entanglement γ' from $(\widehat{E}_1, \widehat{h}')$ to (E_1, h') such that:

$$\widehat{E}_1 \vdash \widehat{h}_1, e_2 \Downarrow \widehat{h}', \widehat{v} \qquad \gamma \subseteq \gamma_1 \subseteq \gamma' \qquad \gamma'(\widehat{v}') = v$$

As a consequence, we get the following derivation:

$$\frac{\widehat{E} \vdash \widehat{h}, e_1 \Downarrow \widehat{h}_1, \widehat{v}_1 \qquad \widehat{E} \uplus [x_1 \mapsto \widehat{v}_1] \vdash \widehat{h}_1, e_2 \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, \textbf{let } x_1 = e_1 \textbf{ in } e_2 \Downarrow \widehat{h}', \widehat{v}}$$

Moreover, it easy to show that if γ' entangles $(\widehat{E} \uplus [x_1 \mapsto v_1], \widehat{h}')$ into $(E \uplus [x_1 \mapsto v_1], h')$, then it also entangles $(\widehat{E}, \widehat{h}')$ into (E, h').

- **Case** $e \equiv \textbf{case } x \textbf{ of } \overline{C_i \ \overline{x_{ij}}^{n_i} \to e_i}^{n}$

  Assume $h(E(x)) = C_r \ v_1 \cdots v_{n_r}$. In this case we get the following execution:

$$\frac{\overbrace{E \uplus [\overline{x_{rj} \mapsto v_j}^{n_r}]}^{E_r} \vdash h, e_r \Downarrow h', v}{E \vdash h, \textbf{case } x \textbf{ of } \overline{C_i \ \overline{x_{ij}}^{n_i} \to e_i}^{n} \Downarrow h', v}$$

Let us define $\widehat{E}_r = \widehat{E} \uplus [\overline{x_{rj} \mapsto \widehat{v}_j}^{n_r}]$, where the $\widehat{v}_j$ are given by $\widehat{h}(\widehat{E}(x)) = C_r \ \widehat{v}_1 \cdots \widehat{v}_n$. Since γ entangles $(\widehat{E}, \widehat{h})$ into (E, h), we get $\gamma(\widehat{E}_r(x_{rj})) = \gamma(\widehat{v}_j) = v_j = E_r(x_{rj})$ for every $j \in \{1..n_r\}$. Therefore, γ is also an entanglement from $(\widehat{E}_r, \widehat{h})$ to (E_r, h), so we apply the induction hypothesis so as to get a heap $\widehat{h}'$, a value $\widehat{v}$ and an entanglement γ' from $(\widehat{E}_r, \widehat{h}')$ to (E_r, h') such that:

$$\widehat{E}_r \vdash \widehat{h}, e_r \Downarrow \widehat{h}', \widehat{v} \qquad \gamma \subseteq \gamma' \qquad \gamma'(\widehat{v}) = v$$

As in the previous case (**let** expressions), we can show that γ entangles $(\widehat{E}, \widehat{h}')$ into (E, h') by just removing the $\overline{x_{rj}}^{n_r}$ bindings from $\widehat{E}_r$ and E_r. Moreover, from the first fact we obtain the following derivation:

$$\frac{\widehat{E} \uplus [\overline{x_{rj} \mapsto \widehat{v}_j}^{n_r}] \vdash \widehat{h}, e_r \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, \textbf{case } x \textbf{ of } \overline{C_i \ \overline{x_{ij}}^{n_i} \to e_i}^{n} \Downarrow \widehat{h}', \widehat{v}}$$

which proves the lemma.

Lemma 16. *Let $E \vdash h, e \Downarrow h', v$ be an execution. Given a configuration $(\widehat{E}, \widehat{h})$ and an entanglement γ from $(\widehat{E}, \widehat{h})$ to (E, h), assume we execute e under $(\widehat{E}, \widehat{h})$ so as to get $\widehat{E} \vdash \widehat{h}, e \Downarrow \widehat{h}', \widehat{v}$ (see Lemma 15). Then, for every variable $z \in dom\ E$ such that $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v} v$ (in h') there exists a variable $y \in dom\ E$ and a word w_y such that $\widehat{E}(y) \xRightarrow{w_y} \bullet \xLeftarrow{w_v} \widehat{v}$ (in $\widehat{h}'$) and $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_y} E(y)$ (in h).*

Proof. By induction on the $\Downarrow$-derivation of e.

- **Case** $e \equiv c$
  The result is a literal, so $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v} v$ (in h') does not hold for any x. The lemma holds vacuously.
- **Case** $e \equiv x$
  In this case it holds that $h' = h$ and $\widehat{h}' = \widehat{h}$, so $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v} v$ (in h') is equivalent to $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v} E(x)$ (in h). This implies the existence of a pointer q such that $E(x) \xRightarrow{w_v}_h q$, which, in turn, leads to $\widehat{E}(x) \xRightarrow{w_v}_{\widehat{h}} \widehat{q}$ for some pointer $\widehat{q}$ such that $\gamma(\widehat{q}) = q$. Therefore it holds that $\widehat{E}(x) \xRightarrow{w_v} \bullet \xLeftarrow{w_v} \widehat{E}(x)$ (in $\widehat{h} = \widehat{h}'$), and the lemma follows by taking $y = x$ and $w_y = w_v$.
- **Case** $e \equiv C\ a_1 \cdots a_n$
  We get $h' = h \uplus [p \mapsto C\ E(a_1) \cdots E(a_n)]$, where p is a fresh pointer not appearing in h. Analogously, we get $\widehat{h}' = \widehat{h} \uplus [\widehat{p} \mapsto C\ \widehat{E}(a_1) \cdots \widehat{E}(a_n)]$, being $\widehat{p}$ another fresh pointer. Assume there exists a variable z such that $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v} p$ (in h'). The word w_v must be nonempty, since otherwise we would get $E(z) \xRightarrow{w_z}_{h'} p$ and, by closure preservation, $E(z) \xRightarrow{w_z}_h p$, contradicting the fact that $p \notin \text{dom}\ h$. Therefore, we assume that $w_v = j_C w'_v$ for some $j \in \{1..n\}$. The Lemma holds by taking $y = a_j$ and $w_y = w'_v$. Indeed, we get $p \xRightarrow{j_C}_{h'} E(a_j) \xRightarrow{w'_v} \bullet \xLeftarrow{w_z} E(z)$ (in h'), which leads to $E(a_j) \xRightarrow{w'_v} \bullet \xLeftarrow{w_z} E(z)$ (in h) by closure preservation. Moreover, since $\widehat{E}(a_j) \xRightarrow{\epsilon} \bullet \xLeftarrow{j_C} \widehat{p}$ (in $\widehat{h}'$) holds, we can extend both paths with w'_v so as to get $\widehat{E}(a_j) \xRightarrow{w'_v} \bullet \xLeftarrow{w_v} \widehat{p}$ (in $\widehat{h}'$).
- **Case** $e \equiv f\ a_1 \cdots a_n$, where $f\ y_1 \cdots y_n = e_f \in \Sigma$
  Assume a variable z such that $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v} v$ (in h'). The set of free variables in e is $\{a_1, \ldots, a_n\}$. By sharing lemma, one of these must be involved in this sharing relation, so there exist an $i \in \{1..n\}$ and a word w_a such that $E(a_i) \xRightarrow{w_a} \bullet \xLeftarrow{w_z} E(z)$ (in h') and $E(a_i) \xRightarrow{w_a} \bullet \xLeftarrow{w_v} v$ (in h'). By closure preservation, we can transform the former into $E(a_i) \xRightarrow{w_a} \bullet \xLeftarrow{w_z} E(z)$ (in h), whereas the latter can be rewritten as $E_f(y_i) \xRightarrow{w_a} \bullet \xLeftarrow{w_v} v$ (in h'), where $E_f \stackrel{\text{def}}{=} [\overline{y_i \mapsto E(a_i)}^n]$. If we define $\widehat{E}_f = [\overline{y_i \mapsto \widehat{E}(a_i)}^n]$, it can be shown that γ is an entanglement from $(\widehat{E}_f, \widehat{h})$ to (E_f, h), so we can apply the induction hypothesis on the judgement $E_f \vdash h, e_f \Downarrow h', v$. Therefore, there exists a $j \in \{1..n\}$ and a word w_y such that $E_f(y_j) \xRightarrow{w_y} \bullet \xLeftarrow{w_a} E_f(y_i)$ (in h) and $\widehat{E}_f(y_j) \xRightarrow{w_y} \bullet \xLeftarrow{w_v} \widehat{v}$ (in $\widehat{h}'$), where $\widehat{E}_f \vdash \widehat{h}, e_f \Downarrow \widehat{h}', \widehat{v}$ is the untangled execution. We can unfold the definition of E_f in these relations so as to get $E(a_j) \xRightarrow{w_y} \bullet \xLeftarrow{w_a} E(a_i)$ (in h) and $\widehat{E}(a_j) \xRightarrow{w_y} \bullet \xLeftarrow{w_a} \widehat{v}$ (in $\widehat{h}'$). Finally, from $E(a_j) \xRightarrow{w_y} \bullet \xLeftarrow{w_a} E(a_i)$ (in h) and $E(a_i) \xRightarrow{w_a} \bullet \xLeftarrow{w_z} E(z)$ (in h) it follows that $E(a_j) \xRightarrow{w_y} \bullet \xLeftarrow{w_z} E(z)$ (in h), which proves the Lemma by taking $y = a_j$.
- **Case** $e \equiv$ **let** $x_1 = e_1$ **in** e_2
  Assume $z \in \text{dom}\ E$ such that $E(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v} v$ (in h'), and the following derivation:

$$\frac{E \vdash h, e_1 \Downarrow h_1, v_1 \qquad \overbrace{E \uplus [x_1 \mapsto v_1]}^{E_1} \vdash h_1, e_2 \Downarrow h', v}{E \vdash h, \textbf{let } x_1 = e_1 \textbf{ in } e_2 \Downarrow h', v}$$

By Lemma 15 there we get an analogous $\Downarrow$-derivation under the configuration $(\widehat{E}, \widehat{h})$,

$$\frac{\widehat{E} \vdash \widehat{h}, e_1 \Downarrow \widehat{h}_1, \widehat{v}_1 \qquad \overbrace{\widehat{E} \uplus [x_1 \mapsto \widehat{v}_1]}^{\widehat{E}_1} \vdash \widehat{h}_1, e_2 \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, \textbf{let } x_1 = e_1 \textbf{ in } e_2 \Downarrow \widehat{h}', \widehat{v}}$$

and an entanglement γ_1 from $(\widehat{E} \uplus [x_1 \mapsto \widehat{v}_1], \widehat{h}_1)$ to $(E \uplus [x_1 \mapsto v_1], h_1)$. Therefore, by induction hypothesis on the $\Downarrow$-derivation of e_2, there exist a variable y_1 and a word w_{y_1} such that $E_1(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E_1(z)$ (in h_1) and $\widehat{E}_1(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$). We distinguish cases:

$y_1 \neq x_1$

In this case $\widehat{E}_1(y_1) = \widehat{E}(y_1)$, and $E_1(y_1) = E(y_1)$. Hence, by closure preservation lemma, the condition $E_1(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E_1(z)$ (in h_1) can be rewritten as $E(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E(z)$ (in h), and $\widehat{E}_1(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$) as $\widehat{E}(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$). Therefore, the lemma holds by taking $y_1 = y$ and $w_y = w_{y_1}$.

$y_1 = x_1$

In this case we rewrite $E_1(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E_1(z)$ (in h_1) as $v_1 \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E(z)$ (in h_1), and $\widehat{E}_1(y_1) \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$) as $\widehat{v}_1 \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), so we can apply the induction hypothesis on the $\Downarrow$-derivation of e_1 in order to obtain a variable $y \in \mathrm{dom}\ E$ and a word w_y such that $E(y) \stackrel{w_y}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E(z)$ (in h) and $\widehat{E}(y) \stackrel{w_y}{\Longrightarrow} \bullet \stackrel{w_{y_1}}{\Longleftarrow} \widehat{v}_1$ (in $\widehat{h}_1$). Since $\widehat{h}_1 \subseteq \widehat{h}'$, we can substitute $\widehat{h}'$ for $\widehat{h}_1$ in the latter condition. As a result, and together with $\widehat{v}_1 \stackrel{w_{y_1}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), we get $\widehat{E}(y) \stackrel{w_y}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), which proves the lemma.

- **Case** $e \equiv$ **case** x **of** $\overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n$

  Assume that $h(E(x)) = C_r\ \overline{v_j}^{n_r}$, so $\widehat{h}(\widehat{E}(x)) = C_r\ \overline{\widehat{v}_j}^{n_r}$, where $\gamma(\widehat{v}_j) = v_j$ for every $j \in \{1..n_r\}$. Again, let us compare the executions of e under (E, h) and $(\widehat{E}, \widehat{h})$:

$$\frac{\overbrace{E \uplus [\overline{x_{rj} \mapsto v_j}^{n_r}]}^{E_r} \vdash h, e_r \Downarrow h', v}{E \vdash h, \textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n \Downarrow h', v}$$

$$\frac{\overbrace{\widehat{E} \uplus [\overline{x_{rj} \mapsto \widehat{v}_j}^{n_r}]}^{\widehat{E}_r} \vdash \widehat{h}, e_r \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, \textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n \Downarrow \widehat{h}', \widehat{v}}$$

  Assume a z such that $E(z) \stackrel{w_z}{\Longrightarrow} \bullet \stackrel{v}{\Longleftarrow} v$ (in h'), so $E_r(z) \stackrel{w_z}{\Longrightarrow} \bullet \stackrel{v}{\Longleftarrow} v$ (in h'). In the proof of Lemma 15 it was shown that γ is an entanglement from $(\widehat{E}_r, \widehat{h})$ to (E_r, h), so we can apply the induction hypothesis and assume the existence of a variable $y_r \in \mathrm{dom}\ E_r$ and a word w_{y_r} such that $E_r(y_r) \stackrel{w_{y_r}}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E_r(z)$ (in h) and $\widehat{E}_r(y_r) \stackrel{w_{y_r}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$). If y_r is none of the $\overline{x_{ij}}^{n_i}$ we can replace the E_r by E in these sharing conditions (since $z \in \mathrm{dom}\ E$), and obtain the desired result by taking $y = y_r$ and $w_y = w_r$. On the other hand, if $y = x_{ij}$ for some $j \in \{1..n_r\}$ the lemma holds by taking $y = x$ and $w_y = j_{C_r} w_r$. Indeed, it holds that $E(x) \stackrel{j_{C_r}}{\Longrightarrow}_h E_r(x_{ij}) \stackrel{w_{y_r}}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E_r(z)$ (in h), and $\widehat{E}(x) \stackrel{j_{C_r}}{\Longrightarrow}_{\widehat{h}} \widehat{E}_r(x_{ij}) \stackrel{w_{y_r}}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), where the $\stackrel{j_{C_r}}{\Longrightarrow}_{\widehat{h}}$ can be replaced by a $\stackrel{j_{C_r}}{\Longrightarrow}_{\widehat{h}'}$, since $\widehat{h} \subseteq \widehat{h}'$. This proves the lemma.

Lemma 17. *Let $E \vdash h, e \Downarrow h', v$ be an execution. Given a configuration $(\widehat{E}, \widehat{h})$ and an entanglement γ from $(\widehat{E}, \widehat{h})$ to (E, h), assume we execute e under $(\widehat{E}, \widehat{h})$ so as to get $\widehat{E} \vdash \widehat{h}, e \Downarrow \widehat{h}', \widehat{v}$ (see Lemma 15). Then, for every w_1, w_2 such that $v \stackrel{w_2}{\Longrightarrow} \bullet \stackrel{w_1}{\Longleftarrow} v$ (in h') holds, but $\widehat{v} \stackrel{w_2}{\Longrightarrow} \bullet \stackrel{w_1}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$) does not, either*

- *there exist two variables $y \neq z \in$ dom E and two words w_y, w_z such that:*
  1. *$E(y) \stackrel{w_y}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E(z)$ (in h).*
  2. *$\widehat{E}(y) \stackrel{w_y}{\Longrightarrow} \bullet \stackrel{w_1}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$).*
  3. *$\widehat{v} \stackrel{w_2}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} \widehat{E}(z)$ (in $\widehat{h}'$).*
- *or, there exists a variable $z \in$ dom E and words w_v, w_z, w'_1, w'_2 such that*
  1. *$E(z) \stackrel{w_z w'_1}{\Longrightarrow} \bullet \stackrel{w_z w'_2}{\Longleftarrow} E(z)$ (in h).*
  2. *$\widehat{v} \stackrel{w_v}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} \widehat{E}(z)$ (in $\widehat{h}'$).*

3. $w_1 = w_v w_1'$ *and* $w_2 = w_v w_2'$

Proof. By induction on the size of the $\Downarrow$-derivation corresponding to the execution of e. We distinguish cases:

- **Case** $e \equiv c$
  In this case v is a literal, so the relation $v \xRightarrow{w_2} \bullet \xLeftarrow{w_1} v$ (in h') does not hold, and the lemma holds vacuously.
- **Case** $e \equiv x$
  In this case we get $h' = h$, $\widehat{h}' = \widehat{h}$, $v = E(x)$, and $\widehat{v} = \widehat{E}(x)$. The second option holds with $z = x$, $w_z = \epsilon = w_v$, $w_1' = w_1$ and $w_2' = w_2$. $E(x) \xRightarrow{w_1} \bullet \xLeftarrow{w_2} E(x)$ (in h) holds by assumption and the rest is trivial.
- **Case** $e \equiv C\ \overline{a_i}^n$
  We obtain $h' = h \uplus [p \mapsto C\ \overline{E(a_i)}^n]$ and $\widehat{h}' = \widehat{h} \uplus [\widehat{p} \mapsto C\ \overline{\widehat{E}(a_i)}^n]$, where p and $\widehat{p}$ are fresh pointers in their respective heaps. Assume $p \xRightarrow{w_2} \bullet \xLeftarrow{w_1} p$ (in h'). Let us distinguish cases:
  - $w_1 = \epsilon$, $w_2 = \epsilon$
    In this case we would get $\widehat{p} \xRightarrow{\epsilon} \bullet \xLeftarrow{\epsilon} \widehat{p}$ (in $\widehat{h}'$), which contradicts the assumption of the lemma.
  - $w_1 = \epsilon$, $w_2 = j_C w_2'$ for some $j \in \{1..n\}$ and w_2'
    This case cannot hold, since that would imply $E(a_j) \xRightarrow{w_2'}_{h'} p$ and, by closure preservation, $E(a_j) \xRightarrow{w_2'}_{h} p$, which contradicts the fact that p does not occur in h.
  - $w_2 = \epsilon$, $w_1 = i_C w_1'$ for some $i \in \{1..n\}$ and w_1'
    As in the previous case, this would imply $E(a_i) \xRightarrow{w_1'}_{h} p$, in contradiction with p being a fresh pointer.
  - $w_1 = i_C w_1'$, $w_2 = j_C w_2'$ for some $i, j \in \{1..n\}$, w_1', and w_2'
    This situation is depicted as follows:

$$p \xRightarrow{i_C} E(a_i) \xRightarrow{w_1'} \bullet \xLeftarrow{w_2'} E(a_j) \xLeftarrow{j_C} p \text{ (in } h') \tag{8}$$

    If $i_C \neq j_C$, the first option of the lemma holds with the following choice of y, z, w_y, and w_z:

$$y \stackrel{\text{def}}{=} a_i \qquad z \stackrel{\text{def}}{=} a_j \qquad w_y \stackrel{\text{def}}{=} w_1' \qquad w_z \stackrel{\text{def}}{=} w_2'$$

    The first conclusion is rewritten as $E(a_i) \xRightarrow{w_1'} \bullet \xLeftarrow{w_2'} E(a_j)$ (in h), which follows from (49) and the closure preservation property. The second one is rewritten as $\widehat{E}(a_i) \xRightarrow{w_1'} \bullet \xLeftarrow{w_1} \widehat{v}$ (in $\widehat{h}'$), which follows (by appending w_1' to both sides) from the relation $\widehat{E}(a_i) \xRightarrow{\epsilon} \bullet \xLeftarrow{i_C} \widehat{v}$ (in $\widehat{h}'$). Similarly, the third conclusion can be obtained from $\widehat{v} \xRightarrow{j_C} \bullet \xLeftarrow{\epsilon} \widehat{E}(a_j)$ (in $\widehat{h}'$), by appending w_2' to both sides.
    If $i_C = j_C$, then:

$$p \xRightarrow{i_C} E(a_i) \xRightarrow{w_1'} \bullet \xLeftarrow{w_2'} E(a_i) \xLeftarrow{i_C} p \text{ (in } h') \tag{9}$$

    and the second part of the lemma holds trivially with $z = a_i$, $w_z = \epsilon$ and $w_v = i_C$.
- **Case** $e \equiv f\ \overline{a_i}^n$
  Assume the following derivations:

$$\frac{f\ \overline{y_i} = e_f \qquad \overbrace{[\overline{y_i \mapsto E(a_i)}^n]}^{E_f} \vdash h, e_f \Downarrow h', v}{E \vdash h, f\ \overline{a_i}^n \Downarrow h', v}$$

$$\frac{f\ \overline{y_i} = e_f \qquad \overbrace{[\overline{y_i \mapsto \widehat{E}(a_i)}^n]}^{\widehat{E}_f} \vdash \widehat{h}, e_f \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, f\ \overline{a_i}^n \Downarrow \widehat{h}', \widehat{v}}$$

  From the assumptions it can be easily shown that γ entangles $(\widehat{E}_f, \widehat{h})$ into (E_f, h). If $v \xRightarrow{w_2} \bullet \xLeftarrow{w_1} v$ (in h') we can apply the induction hypothesis.

If we assume the existence of some variables $y_i \neq y_j$, and words w_{y_i}, w_{y_j} such that $E_f(y_i) \overset{w_{y_i}}{\Longrightarrow} \bullet \overset{w_{y_j}}{\Longleftarrow} E_f(y_j)$ (in h), $\widehat{E}_f(y_i) \overset{w_{y_i}}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), and $\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{w_{y_j}}{\Longleftarrow} \widehat{E}_f(y_j)$ (in $\widehat{h}'$). The conclusions of the first option of the lemma follow trivially from these facts by taking $y \overset{\text{def}}{=} a_i$, $z \overset{\text{def}}{=} a_j$, $w_y = w_{y_i}$, and $w_z = w_{y_j}$. On the other hand, if there exists y_i and words w_{y_i}, w_1', w_2', w_v such that $E_f(y_i) \overset{w_{y_i} w_1'}{\Longrightarrow} \bullet \overset{w_{y_i} w_2'}{\Longleftarrow} E_f(y_i)$ (in h), $\widehat{v} \overset{w_v}{\Longrightarrow} \bullet \overset{w_{y_i}}{\Longleftarrow} \widehat{E}_f(y_i)$ (in $\widehat{h}'$) and $w_1 = w_v w_1'$, $w_2 = w_v w_2'$; then the second part of the lemma holds by taking $z \overset{\text{def}}{=} a_i$ and $w_z = w_{y_i}$.

- **Case** $e \equiv \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2$
  If we execute e under (E, h) we get the following $\Downarrow$-derivation,

$$\frac{E \vdash h, e_1 \Downarrow h_1, v_1 \qquad \overbrace{E \uplus [x_1 \mapsto v_1]}^{E_1} \vdash h_1, e_2 \Downarrow h', v}{E \vdash h, \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2 \Downarrow h', v}$$

besides the following one, obtained when executing e under $(\widehat{E}, \widehat{h})$:

$$\frac{\widehat{E} \vdash \widehat{h}, e_1 \Downarrow \widehat{h}_1, \widehat{v}_1 \qquad \overbrace{\widehat{E} \uplus [x_1 \mapsto \widehat{v}_1]}^{\widehat{E}_1} \vdash \widehat{h}_1, e_2 \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2 \Downarrow \widehat{h}', \widehat{v}}$$

Assume that $v \overset{w_2}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} v$ (in h'), but $\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$) does not hold. From the proof of Lemma 15 we know that γ entangles $(\widehat{E}_1, \widehat{h}_1)$ into (E_1, h_1), so we can apply the induction hypothesis on the $\Downarrow$-derivation of e_2.
If there are two variables $y_1 \neq z_1 \in \text{dom}\ E_1$ and a pair of words w_{y_1}, w_{z_1} such that:

$$E_1(y_1) \overset{w_{y_1}}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} E_1(z_1) \text{ (in } h_1) \tag{10}$$

$$\widehat{E}_1(y_1) \overset{w_{y_1}}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}') \tag{11}$$

$$\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} \widehat{E}_1(z_1) \text{ (in } \widehat{h}') \tag{12}$$

Let us distinguish cases:

- $y_1 \neq x_1$, $z_1 \neq x_1$
  In this case $E_1(y_1) = E(y_1)$, and $E_1(z_1) = E(z_1)$, so (10) is rewritten as $E(y_1) \overset{w_{y_1}}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} E(z_1)$ (in h_1) and, by closure preservation, $E(y_1) \overset{w_{y_1}}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} E(z_1)$ (in h). On the other hand, the sharing conditions (11) and (12) are rewritten as $\widehat{E}(y_1) \overset{w_{y_1}}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$) and $\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} \widehat{E}(z_1)$ (in $\widehat{h}'$) respectively. The lemma follows by taking $y \overset{\text{def}}{=} y_1$, $z \overset{\text{def}}{=} z_1$, $w_y \overset{\text{def}}{=} w_{y_1}$, and $w_z \overset{\text{def}}{=} w_{z_1}$.
- $y_1 = x_1$, $z_1 \neq x_1$
  The sharing conditions (10–12) are rewritten as follows:

$$v_1 \overset{w_{y_1}}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} E(z_1) \text{ (in } h_1) \tag{13}$$

$$\widehat{v}_1 \overset{w_{y_1}}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}') \tag{14}$$

$$\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} \widehat{E}(z_1) \text{ (in } \widehat{h}') \tag{15}$$

  We apply Lemma 16 to (13) in order to assume the existence of a variable y and a word w_y such that:

$$E(y) \overset{w_y}{\Longrightarrow} \bullet \overset{w_{z_1}}{\Longleftarrow} E(z_1) \text{ (in } h) \tag{16}$$

$$\widehat{E}(y) \overset{w_y}{\Longrightarrow} \bullet \overset{w_{y_1}}{\Longleftarrow} \widehat{v}_1 \text{ (in } \widehat{h}_1 \subseteq \widehat{h}') \tag{17}$$

  The lemma follows with the choices of $z \overset{\text{def}}{=} z_1$ and $w_z \overset{\text{def}}{=} w_{z_1}$. Indeed, the first conclusion follows from (16). The second conclusion follows from (14) and (17), whereas the third one follows from (15).

- $y_1 \neq x_1$, $z_1 = x_1$
  This case is similar to the previous one.
- $y_1 = x_1$, $z_1 = x_1$. This case cannot happen if $y_1 \neq z_1$

On the other hand, if there is a variable z_1 and words w_v, w_{z_1}, w'_1, w'_2 such that

$$E_1(z_1) \overset{w_{z_1}w'_1}{\Longrightarrow} \bullet \overset{w_{z_1}w'_2}{\Longleftarrow} E_1(z_1) \text{ (in } h_1) \tag{18}$$

$$\widehat{E}_1(z_1) \overset{w_{z_1}}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}') \tag{19}$$

and $w_1 = w_v w'_1$, $w_2 = w_v w'_2$; then we have two cases:

- $z_1 \neq x_1$. In this case we can rewrite (18) and (19) as

$$E(z_1) \overset{w_{z_1}w'_1}{\Longrightarrow} \bullet \overset{w_{z_1}w'_2}{\Longleftarrow} E(z_1) \text{ (in } h) \tag{20}$$

$$\widehat{E}(z_1) \overset{w_{z_1}}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}') \tag{21}$$

  and the lemma holds by taking $z = z_1$ and $w_z = w_{z_1}$.
- $z_1 = x_1$ By rewritting (18) and (19) we get

$$v_1 \overset{w_{z_1}w'_1}{\Longrightarrow} \bullet \overset{w_{z_1}w'_2}{\Longleftarrow} v_1 \text{ (in } h_1) \tag{22}$$

$$\widehat{v}_1 \overset{w_{z_1}}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}') \tag{23}$$

  We can assume that $\widehat{v}_1 \overset{w_{z_1}w'_1}{\Longrightarrow} \bullet \overset{w_{z_1}w'_2}{\Longleftarrow} \widehat{v}_1$ (in $\widehat{h}_1$) does not hold, since otherwise we would get $\widehat{v}_1 \overset{w_{z_1}w'_1}{\Longrightarrow} \bullet \overset{w_{z_1}w'_2}{\Longleftarrow} \widehat{v}_1$ (in $\widehat{h}'$) by the fact that $\widehat{h}_1 \subseteq \widehat{h}'$, and obtain from (23) that $\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), which would contradict the assumption of the theorem. Therefore, we can apply the induction hypothesis on the $\Downarrow$-derivation of e_1.
  Then either there are two variables $z \neq y \in \operatorname{dom} E$ and two words w_z, w_y such that:

$$E(z) \overset{w_z}{\Longrightarrow} \bullet \overset{w_y}{\Longleftarrow} E(y) \text{ (in } h) \tag{24}$$

$$\widehat{E}(z) \overset{w_z}{\Longrightarrow} \bullet \overset{w_{z_1}w'_1}{\Longleftarrow} \widehat{v}_1 \text{ (in } \widehat{h}_1 \subseteq \widehat{h}') \tag{25}$$

$$\widehat{v}_1 \overset{w_{z_1}w'_2}{\Longrightarrow} \bullet \overset{w_y}{\Longleftarrow} \widehat{E}(y) \text{ (in } \widehat{h}_1 \subseteq \widehat{h}') \tag{26}$$

  In this case first conclusion of the lemma is (24); the second one follows from (25) and (23), whereas the third one follows from (26) and (23).
  Or there is a variable $z \in \operatorname{dom} E$ and words w_z, w''_1, w''_2, w''_v such that

$$E(z) \overset{w_z w''_1}{\Longrightarrow} \bullet \overset{w_z w''_2}{\Longleftarrow} E(z) \text{ (in } h) \tag{27}$$

$$\widehat{E}(z) \overset{w_z}{\Longrightarrow} \bullet \overset{w''_v}{\Longleftarrow} \widehat{v}_1 \text{ (in } \widehat{h}_1 \subseteq \widehat{h}') \tag{28}$$

  and $w_{z_1}w'_1 = w''_v w''_1$, $w_{z_1}w'_2 = w''_v w''_2$. Here, we have to distinguish again two cases:
  1. If w_{z_1} is a prefix of w''_v, i.e. $w''_v = w_{z_1}w'$. Then, $w''_1 = w'w'_1$ and $w''_2 = w'w'_2$, and the lemma holds for z: from (27) we have that

$$E(z) \overset{(w_z w')w'_1}{\Longrightarrow} \bullet \overset{(w_z w')w'_2}{\Longleftarrow} E(z) \text{ (in } h) \tag{29}$$

     and from (28) and (23) we have that

$$\widehat{E}(z) \overset{w_z w'}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}_1 \subseteq \widehat{h}') \tag{30}$$

     and $w_1 = w_v w'_1$ and $w_2 = w_v w'_2$.

2. If w''_v is a prefix of w_{z_1}, i.e. $w_{z_1} = w''_v w'$. Then, $w'_1 = w' w''_1$ and $w'_2 = w' w''_2$ and the lemma holds for z: from (27) we have that

$$E(z) \xRightarrow{w_z w''_1} \bullet \xLeftarrow{w_z w''_2} E(z) \text{ (in } h) \tag{31}$$

and from (28) and (23) we have that

$$\widehat{E}(z) \xRightarrow{w_z} \bullet \xLeftarrow{w_v w'} \widehat{v} \text{ (in } \widehat{h}_1 \subseteq \widehat{h}') \tag{32}$$

and $w_1 = (w_v w')w''_1$ and $w_2 = (w_v w')w''_2$.

- **Case** $e \equiv \textbf{case } x \textbf{ of } \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n$
We get the following derivations:

$$\frac{\overbrace{E \uplus [\overline{x_{rj} \mapsto v_j}^{n_r}]}^{E_r} \vdash h, e_r \Downarrow h', v}{E \vdash h, \textbf{case } x \textbf{ of } \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n \Downarrow h', v}$$

$$\frac{\overbrace{\widehat{E} \uplus [\overline{x_{rj} \mapsto \widehat{v}_j}^{n_r}]}^{\widehat{E}_r} \vdash \widehat{h}, e_r \Downarrow \widehat{h}', \widehat{v}}{\widehat{E} \vdash \widehat{h}, \textbf{case } x \textbf{ of } \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n \Downarrow \widehat{h}', \widehat{v}}$$

Assume we have $v \xRightarrow{w_2} \bullet \xLeftarrow{w_1} v$ (in h'), but not $\widehat{v} \xRightarrow{w_2} \bullet \xLeftarrow{w_1} \widehat{v}$ (in $\widehat{h}'$), and that the r-th branch is executed, that is, $h(E(x)) = C_r\ \overline{v_j}^{n_r}$ for some $\overline{v_j}^{n_r}$. We apply the induction hypothesis to the $\Downarrow$-derivation of e_r. If there are two variables $y_r \neq z_r \in \text{dom } E_r$ and two words w_{y_r}, w_{z_r} such that:

$$E_r(y_r) \xRightarrow{w_{y_r}} \bullet \xLeftarrow{w_{z_r}} E_r(z_r) \text{ (in } h) \tag{33}$$

$$\widehat{E}_r(y_r) \xRightarrow{w_{y_r}} \bullet \xLeftarrow{w_1} \widehat{v} \text{ (in } \widehat{h}') \tag{34}$$

$$\widehat{v} \xRightarrow{w_2} \bullet \xLeftarrow{w_{z_r}} \widehat{E}_r(z_r) \text{ (in } \widehat{h}') \tag{35}$$

As in the case of **let** expressions, we further distinguish cases:

- $y_r \in \text{dom } E$, $z_r \in \text{dom } E$
We can substitute E for E_r in the relations (33–35), which leads to the three conclusions of the lemma with $y \stackrel{\text{def}}{=} y_r$, $z \stackrel{\text{def}}{=} z_r$, $w_y \stackrel{\text{def}}{=} w_{y_r}$, and $w_z \stackrel{\text{def}}{=} w_{z_r}$.
- $y_r \notin \text{dom } E$, $z_r \in \text{dom } E$
Then $y_r = x_{rj}$ for some $j \in \{1..n_r\}$. We rewrite (33–35) as follows:

$$v_j \xRightarrow{w_{y_r}} \bullet \xLeftarrow{w_{z_r}} E(z_r) \text{ (in } h) \tag{36}$$

$$\widehat{v}_j \xRightarrow{w_{y_r}} \bullet \xLeftarrow{w_1} \widehat{v} \text{ (in } \widehat{h}') \tag{37}$$

$$\widehat{v} \xRightarrow{w_2} \bullet \xLeftarrow{w_{z_r}} \widehat{E}(z_r) \text{ (in } \widehat{h}') \tag{38}$$

But notice that $E(x) \xRightarrow{j_{C_r}}_h v_j$, and $\widehat{E}(x) \xRightarrow{j_{C_r}}_{\widehat{h}} \widehat{v}_j$, where the latter also holds in $\widehat{h}'$. Hence we get, by (36) and (37):

$$E(x) \xRightarrow{j_{C_r} w_{y_r}} \bullet \xLeftarrow{w_{z_r}} E_r(z_r) \text{ (in } h) \tag{39}$$

$$\widehat{E}(x) \xRightarrow{j_{C_r} w_{y_r}} \bullet \xLeftarrow{w_1} \widehat{v} \text{ (in } \widehat{h}') \tag{40}$$

Therefore, if we define $y \stackrel{\text{def}}{=} x$, $z \stackrel{\text{def}}{=} z_r$, $w_y \stackrel{\text{def}}{=} j_{C_r} w_{y_r}$, and $w_z \stackrel{\text{def}}{=} w_{z_r}$, the required result follows from (38), (39), and (40).

- $y_r \in \text{dom } E$, $z_r \notin \text{dom } E$
  This case is symmetrical to the previous one.
- $y_r \notin \text{dom } E$, $z_r \notin \text{dom } E$
  Then, $y_r = x_{rj}$ and $z_r = x_{ri}$ for some $j \neq i \in \{1..n_r\}$. Relations (33–35) are rewritten as follows:

$$v_j \overset{w_{y_r}}{\Longrightarrow} \bullet \overset{w_{z_r}}{\Longleftarrow} v_i \text{ (in } h\text{)} \tag{41}$$

$$\widehat{v}_j \overset{w_{y_r}}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}'\text{)} \tag{42}$$

$$\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{w_{z_r}}{\Longleftarrow} \widehat{v}_i \text{ (in } \widehat{h}'\text{)} \tag{43}$$

  Now we get $E(x) \overset{j_{C_r}}{\Longrightarrow}_h v_j$ and $E(x) \overset{i_{C_r}}{\Longrightarrow}_h v_i$ both of which hold in h' as well. Analogously, $\widehat{E}(x) \overset{j_{C_r}}{\Longrightarrow}_{\widehat{h}} \widehat{v}_j$ and $\widehat{E}(x) \overset{i_{C_r}}{\Longrightarrow}_{\widehat{h}} \widehat{v}_i$, and so in $\widehat{h}'$. Hence we get from (41–43):

$$E(x) \overset{j_{C_r} w_{y_r}}{\Longrightarrow} \bullet \overset{i_{C_r} w_{z_r}}{\Longleftarrow} E(x) \text{ (in } h\text{)} \tag{44}$$

$$\widehat{E}(x) \overset{j_{C_r} w_{y_r}}{\Longrightarrow} \bullet \overset{w_1}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}'\text{)} \tag{45}$$

$$\widehat{v} \overset{w_2}{\Longrightarrow} \bullet \overset{i_{C_r} w_{z_r}}{\Longleftarrow} \widehat{E}(x) \text{ (in } \widehat{h}'\text{)} \tag{46}$$

  The lemma follows from these three assumptions with $y \overset{\text{def}}{=} x$, $z \overset{\text{def}}{=} x$, $w_y \overset{\text{def}}{=} j_{C_r} w_{y_r}$, and $w_z \overset{\text{def}}{=} i_{C_r} w_{z_r}$.

If there is a variable $y_r \in \text{dom } E_r$ and words w_{y_r}, w_v, w_1', w_2' such that:

$$E_r(y_r) \overset{w_{y_r} w_1'}{\Longrightarrow} \bullet \overset{w_{y_r} w_2'}{\Longleftarrow} E_r(y_r) \text{ (in } h\text{)} \tag{47}$$

$$\widehat{E}_r(y_r) \overset{w_{y_r}}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}'\text{)} \tag{48}$$

and $w_1 = w_v w_1'$ and $w_2 = w_v w_2'$.
We distinguish two cases:

- $y_r \in \text{dom } E$, then the lemma holds with $z = y_r$.
- $y_r \notin \text{dom } E$. Then $y_r = x_{r_j}$ for some $j \in \{1..n_r\}$ and from (47) we get:

$$E(x) \overset{j_{C_r}}{\Longrightarrow} E_r(x_{rj}) \overset{w_{y_r} w_1'}{\Longrightarrow} \bullet \overset{w_{y_r} w_2'}{\Longleftarrow} E_r(x_{rj}) \overset{j_{C_r}}{\Longleftarrow} p \text{ (in } h'\text{)} \tag{49}$$

  and from (50) we get:

$$\widehat{E}_r(x) \overset{j_{C_r} w_{y_r}}{\Longrightarrow} \bullet \overset{w_v}{\Longleftarrow} \widehat{v} \text{ (in } \widehat{h}'\text{)} \tag{50}$$

  so the lemma holds for $z = x$.

For the same heap several entanglements may be defined, but we are interested in a configuration $(\widehat{E}, \widehat{h})$, where everything is untangled, as shown in the previous example. This is because, then $R_0 = \{x_i \overset{\epsilon}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} x_i \mid i = 1..n\}$ correctly approximates its sharing.

Lemma 18. *For any configuration (E, h) there exists another configuration $(\widehat{E}, \widehat{h})$ and an entanglement γ from $(\widehat{E}, \widehat{h})$ to (E, h) such that the set $\{x \overset{\epsilon}{\longrightarrow} \bullet \overset{\epsilon}{\longleftarrow} x \mid x \in \textit{dom } E\}$ is a correct approximation of $(\widehat{E}, \widehat{h})$.*

Proof. Let us define the following function ψ which untangles the closure of a pointer p in a heap h and yields the corresponding entanglement γ:

$$\begin{aligned}
\psi(c, h) &= (c, \emptyset, \emptyset) \\
\psi(p, h) &= (\widehat{p}, [\widehat{p} \mapsto C\ \widehat{v}_1 \ldots \widehat{v}_i] \uplus \widehat{h}_1 \uplus \cdots \uplus \widehat{h}_n, [\widehat{p} \mapsto p] \uplus \gamma_1 \uplus \cdots \uplus \gamma_n) \\
&\quad \textbf{where}\ \ C\ v_1 \ldots v_n = h(p) \\
&\quad\phantom{\textbf{where}}\ \ (\widehat{v}_i, \widehat{h}_i, \gamma_i) = \psi(v_i, h) \text{ for all } i \in \{1..n\} \\
&\quad\phantom{\textbf{where}}\ \ \widehat{p} \text{ is a fresh pointer}
\end{aligned}$$

Without loss of generality we can assume that h is finite, closed, and acyclic, so we can prove the following property by induction on the length of the longest pointer chain that can be followed[2] starting from v: If $(\widehat{v}, \widehat{h}, \gamma) = \psi(v, h)$ then:

1. γ is an homomorphism from $\widehat{h}$ to h. That is, for every $\widehat{p} \in \text{dom } \widehat{h}$, if $\widehat{h}(\widehat{p}) = C\ \widehat{v}_1 \cdots \widehat{v}_n$, then $h(\gamma(\widehat{p})) = C\ \gamma(\widehat{v}_1) \cdots \gamma(\widehat{v}_n)$.
2. $\gamma(\widehat{v}) = v$.
3. If $\widehat{v} \xRightarrow{w_1} \bullet \xLeftarrow{w_2} \widehat{v}$ (in $\widehat{h}$) then $w_1 = w_2$.

- **Base case:** If v is a literal the second property follows by the convention of $\gamma(c) = c$, whereas the first and the third hold vacuously. If v is a pointer without children, assume a constructor C such that $h(v) = C$. Then we get $\widehat{h} = [\widehat{v} \mapsto C]$, and $\gamma = [\widehat{v} \mapsto v]$, from which the first two properties follow trivially. Finally, if $\widehat{v} \xRightarrow{w_1} \bullet \xLeftarrow{w_2} \widehat{v}$ (in $\widehat{h}$) then $w_1 = w_2 = \epsilon$.
- **Inductive step:** Assume v is a pointer for which $h(v) = C\ \overline{v_i}^n$. If we define, for each $i \in \{1..n\}$ $(\widehat{v}_i, \widehat{h}_i, \gamma_i) \stackrel{\text{def}}{=} \psi(v_i, h)$, we get $\widehat{h} = [\widehat{v} \mapsto C\ \overline{\widehat{v}_i}^n] \uplus \widehat{h}_1 \uplus \cdots \uplus \widehat{h}_n$ and $\gamma = [\widehat{v} \mapsto v] \uplus \gamma_1 \uplus \cdots \uplus \gamma_n$. Moreover, the three properties above follow for each of the $(\widehat{v}_i, \widehat{h}_i, \gamma_i)$ by induction hypothesis, so let us prove them for $(\widehat{v}, \widehat{h}, \gamma)$:
    1. Assume $\widehat{p} \in \text{dom } \widehat{h}$. If $\widehat{p} = \widehat{v}$, then we get $h(\gamma(\widehat{v})) = h(v) = C\ \overline{v_i}^n = C\ \overline{\gamma(\widehat{v}_i)}^n$, with the last step justified by the property $\gamma(\widehat{v}_i) = v_i$ given by the induction hypothesis. If $\widehat{p} \in \text{dom } \widehat{h}_i$, the property follows from γ_i being an homomorphism from $\widehat{h}_i$ to h_i.
    2. Trivial, by the definition of γ.
    3. Assume $\widehat{v} \xRightarrow{w_1} \bullet \xLeftarrow{w_2} \widehat{v}$ (in $\widehat{h}$). If $w_1 = w_2 = \epsilon$ we are done. If $w_1 = j_C w_1'$ for some $j \in \{1..n\}$ and $w_2 = \epsilon$, then we would get $\widehat{v} \xRightarrow{w_1}_{\widehat{h}} \widehat{q}$ for some pointer $\widehat{q} \in \text{dom } \widehat{h}_j$, but also $\widehat{v} \xRightarrow{w_1}_{\widehat{h}} \widehat{v}$, which implies $\widehat{v} = \widehat{q}$, and contradicts the fact that $\widehat{v}$ is a fresh variable. A similar contradiction would we obtained if we assume an empty w_1 and a nonempty w_2. The only remaining case is $w_1 = j_C w_1'$ and $w_2 = i_C w_2'$ for some $i, j \in \{1..n\}$. If $i \neq j$ we would get $\widehat{v}_j \xRightarrow{w_1'} \bullet \xLeftarrow{w_2'} \widehat{v}_i$ (in $\widehat{h}'$), contradicting the fact that h_j and h_i have disjoint domains (as demanded by $\uplus$). If $i = j$ we would get $\widehat{v}_i \xRightarrow{w_1'} \bullet \xLeftarrow{w_2'} \widehat{v}_i$ (in $\widehat{h}$), which implies $w_1' = w_2'$ by induction hypothesis. Therefore $w_1 = w_2$.

Now we have proved these three properties, let us define for every variable $x \in \text{dom } E$ the triple $(\widehat{v}_x, \widehat{h}_x, \gamma_x) \stackrel{\text{def}}{=} \psi(E(x), h)$, and $\widehat{h}$, $\widehat{E}$, γ as follows:

$$\widehat{h} = \biguplus_{x \in \text{dom } E} \widehat{h}_i \qquad \widehat{E} = [x \mapsto \widehat{v}_x \mid x \in \text{dom } E] \qquad \gamma = \biguplus_{x \in \text{dom } E} \gamma_i$$

Since all the pointers generated by the ψ function are fresh, we can ensure that $\widehat{h}$ and γ are well-defined. The property of γ being an homomorphism from $\widehat{h}$ to h follows from the fact of each γ_i being an homomorphism from $\widehat{h}_i$ to h. Moreover, E and $\widehat{E}$ share the same definition domain, and for every $x \in \text{dom } E$ we get $\gamma(\widehat{E}(x)) = \gamma(\widehat{v}_x) = E(x)$. As a consequence, we have proved that γ is an entanglement from $(\widehat{E}, \widehat{h})$ to (E, h). The only thing left to prove is that $(\widehat{E}, \widehat{h})$ is correctly approximated by $\{x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x \mid x \in \text{dom } E\}$. Assume $x, y \in \text{dom } \widehat{E}$ and two words w_x,w_y such that $\widehat{E}(x) \xRightarrow{w_x} \bullet \xLeftarrow{w_y} \widehat{E}(y)$ (in $\widehat{h}$). If $x \neq y$ we would get that $\widehat{h}_x$ and $\widehat{h}_y$ would have a pointer in common, contradicting the disjointness of $\widehat{h}_x$ and $\widehat{h}_y$. If $x = y$, we get $\widehat{E}(x) \xRightarrow{w_x} \bullet \xLeftarrow{w_y} \widehat{E}(x)$ (in $\widehat{h}_x$) which implies that $w_1 = w_2$. Therefore, the relation $\widehat{E}(x) \xRightarrow{w_x} \bullet \xLeftarrow{w_y} \widehat{E}(y)$ (in $\widehat{h}$) is rewritten as $\widehat{E}(x) \xRightarrow{w_x} \bullet \xLeftarrow{w_x} \widehat{E}(x)$ (in $\widehat{h}$), which is correctly approximated by $x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x$.

In the example above, signature is $R' = S\ [\![e_f]\!]\ R_0\ \Sigma = \{res \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} res, res \xrightarrow{1_C} \bullet \xleftarrow{\epsilon} y\}$. The environment of the call is approximated by $R = \{x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} y, x \xrightarrow{1_{C'}} \bullet \xleftarrow{2_{C'}} x, y \xrightarrow{1_{C'}} \bullet \xleftarrow{2_{C'}} y\}$. So the final

[2] If v is a literal we assume this length, by convention, to be zero.

sharing is approximated by $R \uplus^*_{res} R'$ which merges the context of the call with the signature, and contains $\{res \xrightarrow{1_C} \bullet \xleftarrow{\epsilon} y, res \xrightarrow{1_C} \bullet \xleftarrow{\epsilon} x, res \xrightarrow{1_C \cdot 1_{C'}} \bullet \xleftarrow{1_C \cdot 2_{C'}} res, res \xrightarrow{1_C \cdot 1'_C} \bullet \xleftarrow{2_{C'}} x, res \xrightarrow{1_C \cdot 1'_C} \bullet \xleftarrow{2_{C'}} y, res \xrightarrow{1_C \cdot 2'_C} \bullet \xleftarrow{1_{C'}} x, res \xrightarrow{1_C \cdot 2'_C} \bullet \xleftarrow{1_{C'}} y\}$. This happens for each R approximating a context call, so R' is a correct signature for f. We prove this in the following theorem.

Theorem 2. *Assume a function definition $f\ \overline{x_i}^n = e_f$, a set of relations $R_0 = \{x_i \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} x_i \mid i = 1..n\}$, and an environment Σ with correct signatures. If $R' = S\ [\![e_f]\!]\ R_0\ \Sigma$, then R' is a correct signature for f.*

Proof. Assume a configuration (E, h) with $\mathrm{dom}\ E = \{\overline{x_i}^n\}$ and a set of relations R such that $R \succeq (E, h)$. If we execute e_f under the configuration (E, h) we get $E \vdash h, e_f \Downarrow h', v$ for some h', v. We have to show that $R \uplus^*_{res} R' \succeq (E', h')$, where $E' \stackrel{\text{def}}{=} E \uplus [res \mapsto v]$. By Lemma 18 there exists a mapping γ which entangles a configuration $(\widehat{E}, \widehat{h})$ into (E, h), where $(\widehat{E}, \widehat{h})$ is correctly approximated by R_0. Assume we execute e_f under the untangled configuration so as to get $\widehat{E} \vdash \widehat{h}, e_f \Downarrow \widehat{h}', \widehat{v}$ for some $\widehat{h}'$ and $\widehat{v}$. By correctness theorem (Theorem 1) we know that $R' \succeq (\widehat{E}', \widehat{h}')$, where $\widehat{E}' \stackrel{\text{def}}{=} \widehat{E} \uplus [res \mapsto \widehat{v}]$.

In order to prove that $R \uplus^*_{res} R' \succeq (E', h')$ let us assume that $E'(x) \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E'(y)$ (in h') for some w_x and w_y and distinguish cases:

- $x \neq res$, $y \neq res$
  Then we get $E(x) \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E(y)$ (in h') and, by closure preservation, $E(x) \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E(y)$ (in h), which is correctly approximated by a relation $x \xrightarrow{p_x} \bullet \xleftarrow{p_y} y \in R \subseteq R \uplus^*_{res} R'$.
- $x = res$, $y \neq res$
  Then we get $v \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E(y)$ (in h') which implies, by Lemma 16, the existence of a variable $z \in \mathrm{dom}\ E$ and a word w_z such that $E(z) \stackrel{w_z}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E(y)$ (in h) and $\widehat{v} \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} \widehat{E}(z)$ (in $\widehat{h}'$). By Lemma 14, also $v \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_z}{\Longleftarrow} E(z)$ (in h'). Since $R \succeq (E, h)$, the first relation is approximated by a sharing relation $z \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R$, whereas the second one is approximated by a sharing relation $res \xrightarrow{p_3} \bullet \xleftarrow{p_4} z \in R'$. Therefore, by Lemma 12, the relation $v \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E(y)$ (in h') is approximated by a sharing relation in $R \uplus^*_{res} R'$.
- $x \neq res$, $y = res$
  This case is analogous to the previous one.
- $x = res$, $y = res$
  Then $E'(x) \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} E'(y)$ (in h') is translated into $v \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} v$ (in h'). If $\widehat{v} \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$) then there is some relation $res \xrightarrow{p_x} \bullet \xleftarrow{p_y} res \in R'$ approximating it, since R' correctly approximates $(\widehat{E}', \widehat{h}')$. In this case the theorem holds, as R' is contained within $R \uplus^*_{res} R'$. If $\widehat{v} \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$) does not hold, we can apply Lemma 17.
  In case the first part of the lemma holds, there exist two variables $z_1 \neq z_2$ and two words w_1, w_2 such that:
  1. $E(z_1) \stackrel{w_1}{\Longrightarrow} \bullet \stackrel{w_2}{\Longleftarrow} E(z_2)$ (in h), which is approximated by a relation $z_1 \xrightarrow{p_1} \bullet \xleftarrow{p_2} z_2 \in R$.
  2. $\widehat{E}(z_1) \stackrel{w_1}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), which is approximated by a relation $z_1 \xrightarrow{p_3} \bullet \xleftarrow{p_4} res \in R'$.
  3. $\widehat{v} \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_2}{\Longleftarrow} \widehat{E}(z_2)$ (in $\widehat{h}'$), which is approximated by a relation $res \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_2 \in R'$.

  The second relation can be rewritten, by Lemma 14, as $E(z_1) \stackrel{w_1}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} v$ (in h'), whereas we can replace the h in the first relation by h', for the latter being a superset of h. Therefore it holds that $v \stackrel{w_y}{\Longrightarrow} \bullet \stackrel{w_2}{\Longleftarrow} E(z_2)$ (in h'), which, by Lemma 12, is approximated by a relation $res \xrightarrow{p_7} \bullet \xleftarrow{p_8} z_2 \in R \uplus_{res} \{res \xrightarrow{p_4} \bullet \xleftarrow{p_3} z_1\}$. Now we translate the third relation above into $v \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_2}{\Longleftarrow} E(z_2)$ (in h'), and combine it with $v \stackrel{w_y}{\Longrightarrow} \bullet \stackrel{w_2}{\Longleftarrow} E(z_2)$ (in h') so as to get $v \stackrel{w_x}{\Longrightarrow} \bullet \stackrel{w_y}{\Longleftarrow} v$ (in h'), which by Lemma 12 is approximated by a sharing relation $res \xrightarrow{p_9} \bullet \xleftarrow{p_{10}} res \in (R \uplus_{res} \{res \xrightarrow{p_4} \bullet \xleftarrow{p_3} z_1\}) \uplus_{res} \{res \xrightarrow{p_5} \bullet \xleftarrow{p_6} z_2\} \subseteq R \uplus^*_{res} R'$.
  Otherwise, the second part of the lemma holds, i.e. there exists a variable $z \in \mathrm{dom}\ E$ such that
  1. $E(z) \stackrel{w_z w'_1}{\Longrightarrow} \bullet \stackrel{w_z w'_2}{\Longleftarrow} E(z)$ (in h), which is approximated by a relation $z \xrightarrow{p_1} \bullet \xleftarrow{p_2} z \in R$.
  2. $\widehat{E}(z) \stackrel{w_z}{\Longrightarrow} \bullet \stackrel{w_v}{\Longleftarrow} \widehat{v}$ (in $\widehat{h}'$), which is approximated by a relation $z \xrightarrow{p_3} \bullet \xleftarrow{p_4} res \in R'$.

and $w_x = w_v w_1'$ and $w_y = w_v w_2'$. By Lemma 13, there exists a sharing relation $res \xrightarrow{p_5} \bullet \xleftarrow{p_6} res \in R \uplus_{res} \{z \xrightarrow{p_3} \bullet \xleftarrow{p_4} res\} \subseteq R \uplus^*_{res} R'$ approximating $v \xRightarrow{w_x} \bullet \xLeftarrow{w_y} v$ (in h').

5 Implementation Issues and Cost

The analysis presented in the previous section contains some tests and operations that deserve a detailed comment in order to see whether all of them are decidable, and what their costs are.

Since the number of bound variables in a function definition is finite, so is the number of tuples in R. A relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ may occur multiple times in R with different p_1 and p_2 but, as we will see, always with different types. Then, all the set union operations are decidable.

In the implementation, we represent regular languages by non-deterministic finite automata (NFA). We will denote them by $A = (\Sigma, Q, i, F, \delta)$. This facilitates some of the operations needed on regular languages. These are the following:

1. To test whether a regular language L is empty, i.e. $L = \{\}$.
2. Given regular languages L_1 and L_2, to compute its concatenation $L_1.L_2$.
3. Given regular languages L_1 and L_2, to compute $L_1|_{L_2}$.
4. Given regular languages L_1 and L_2, to test whether $L_1 \subseteq L_2$.

The emptiness test can be achieved [5] by looking for a final state that is reachable from the initial one. If $n = |Q|$ is the number of states of A, the algorithm costs $O(n^2)$.

Given NFA automata A_1 and A_2, the automaton recognizing $L(A_1).L(A_2)$ can be constructed with a cost $O(n)$, just by connecting with ϵ-transitions the final states of A_1 to the initial one of A_2.

Given NFA automata $A_1 = (\Sigma_1, Q_1, i_1, F_1, \delta_1)$ and $A_2 = (\Sigma_2, Q_2, i_2, F_2, \delta_2)$, the automaton recognizing $L(A_1)|_{L(A_2)}$ is more involved. In fact, we have not found in the literature an algorithm to compute it, and have invented our own:

1. Compute the automaton A_2' by adding to A_2 transitions with every symbol in $\Sigma = \Sigma_1 \cup \Sigma_2$, from every final state of A_2 to itself. It is clear that A_2' recognizes $L(A_2).\Sigma^*$.
2. Compute $A_3 = A_1 \cap A_2'$. It recognizes the words of $L(A_1)$ beginning with a word of $L(A_2)$. The construction implies that the states of A_3 are all the pairs of $Q_1 \times Q_2$.
3. Build an automaton A_4 with a fresh state q_0 as the initial one. If $L(A_1) \cap L(A_2)$ is empty, then add ϵ-transitions from q_0 to every state $(q_i, p_j) \in A_3$ such that q_i is a non-final state of A_1 and p_j is a final state of A_2'. If $L(A_1) \cap L(A_2)$ is not empty, then q_i in $(q_i, p_j) \in A_3$ can be any state of A_1.
4. Remove from A_4 all the states non-reachable from q_0. The resulting automaton exactly recognizes $L(A_1)|_{L(A_2)}$.

The dominant costs of the algorithm are the cartesian product and the state reachability computation, both in $O(n^2)$.

Given NFA automata A_1 and A_2, $L(A_1) \subseteq L(A_2)$ if and only if $L(A_1) \cap L(A_2) = L(A_1)$, so inclusion is a particular case of equality. Unfortunately, equality cannot be directly computed on NFA's. They must be converted to deterministic finite automata (DFA), and then their equality tested with the well-known table-filling algorithm [5], which has a cost $O(n^2)$. But the conversion from NFA to DFA has a worst-case cost in $O(n^3 2^n)$. This is because the number of states of the DFA are subsets of the NFA set of states, and can in theory be up to 2^n. As we will see, the equality of languages must be tested once every fixpoint iteration.

When interpreting the body of a recursive function f, we start by setting an empty signature for f, i.e. $\Sigma(f) = \emptyset$. It is easy to show that the interpretation is monotonic in the lattice:

$$\langle \mathcal{M}(Var_f \times \mathcal{P}(\Sigma^*) \times \mathcal{P}(\Sigma^*) \times Var_f), \emptyset, \top, \sqsubseteq, \cup, \cap \rangle$$

where $\mathcal{M}$ stands for 'multiset of', Var_f are the bound variables of f, Σ^* is the top regular language, and $\top$ is the maximum relation. We need to ensure that no two tuples with the same type exist relating the same variables. So, at the end of each iteration, the following collapsing rule is used:

$$\frac{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \in R \qquad x \xrightarrow{p_3} \bullet \xleftarrow{p_4} y \in R \qquad type(x, p_1) = type(x, p_3)}{\text{replace in } R \text{ the two tuples by } x \xrightarrow{p_1+p_3} \bullet \xleftarrow{p_2+p_4} y}\ OR$$

Should not we use this rule, the abstract domain, regarding only the relations between program variables, would be infinite. The order relation between two tuples relating the same pair of variables, and having the same type, is as follows:

$$x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y \ \sqsubseteq \ x \xrightarrow{p_1'} \bullet \xleftarrow{p_2'} y$$

if $L(p_1) \subseteq L(p_1')$ and $L(p_2) \subseteq L(p_2')$. Let us call $\mathcal{I}_f\ \Sigma$ to the interpretation of e_f with current signature environment Σ, returning Σ with f's signature updated. By monotonicity, we have:

$$\emptyset \sqsubseteq \mathcal{I}_f\ \emptyset \sqsubseteq \mathcal{I}_f\ (\mathcal{I}_f\ \emptyset) \sqsubseteq \ldots \sqsubseteq (\mathcal{I}_f)^i\ \emptyset \sqsubseteq \ldots$$

Disregarding the regular languages, this chain is finite because so is Var_f, and the number of different types of the program. Then, the least fixpoint can be reached after a finite number of iterations. If n is the number of f's formal arguments, then at most n iterations are needed. This is because functional languages have no variable updates, and then there never may arise sharing relations between the formal arguments as a consequence of the function body actions. The only possible relations will be between the function's result and its arguments.

Considering now the regular languages, infinite ascending chains are possible, i.e. one can obtain infinite chains $L_1 \subseteq L_2 \subseteq L_3 \subseteq \ldots$.

The least upper bound of such a sequence of regular languages needs not to be a regular one. But, at least, there always exists the regular language Σ^* greater than any other one. In order to ensure termination of the fixpoint computation, we use the following *widening* technique [2]:

1. Based on the form of the automata denoting the increasing language sequence, and by using some heuristics, we guess an automaton A such that $\bigcup_i L_i \subseteq L(A)$. Then, we iterate the interpretation by using this automaton as an assumption in f's signature.
2. If A is a fixpoint or a post-fixpoint, then we are done. Otherwise, we use Σ^* as the upper bound of the sequence. In terms of precision, $x \xrightarrow{\Sigma^*} \bullet \xleftarrow{\Sigma^*} y$ is completely uninformative about the paths through which x and y share their common descendant.

The heuristic consists of comparing the automata sequence obtained for a given relation $x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y$ in the successive iterations, and discovering growing sequences reaching three or more states related by the same alphabet symbol. For example q_1, q_2, q_3, with $(q_1, a, q_2), (q_2, a, q_3) \in \delta$. These sequences are collapsed into a single state class q, with a single iterative transition $(q, a, q) \in \delta$. The resulting automata is compared with the non-widened one, to ensure that they are equivalent regarding the remaining transitions. In all the examples we have tried, this heuristic appears to be enough to reach a fixed point.

We pay now attention to the asymptotic cost of the whole interpretation. We choose the size n of a function to be its number of bound variables. This figure is linearly related to the size of its abstract syntax tree, and to the number of lines of its source code. How is n related to the size of the inferred automata in terms of their number of states? It is easy to check that every bound variable y introduces a relation $x \xrightarrow{j_C} \bullet \xleftarrow{\epsilon} y$ with a prior bound variable x. This increases by one the number of states of the y relations with respect to those of the x relations. So, the automata number of states grow from one to the abstract syntax tree height, when going from the initial expression to the deepest ones. Assuming a reasonably balanced syntax tree, we consider $\log n$ to be an accurate bound to the automata size.

```
last xs = case xs of
            x:xx -> case xx of
                       []    -> {* R1 *} x
                       y:yy -> {* R2 *} last xx
```

Fig. 10: Definition of the function `last`

If a function definition has n bound variables, and considering as a constant the number of different types, in the worst case there can be up to $O(n^2)$ tuples in the current relation R. The computation of a single closure operation $R \uplus_x \{x \xrightarrow{p_1} \bullet \xleftarrow{p_2} y\}$ (see Fig. 7) introduces as many relations $x \xrightarrow{p_x} \bullet \xleftarrow{p_z} z$ as prior relations $y \xrightarrow{p_y} \bullet \xleftarrow{p_z} z$ are there in R, i.e. $O(n)$ in the worst case. A single iteration of the abstract interpretation will compute one such closure for every bound variable, giving an upper bound of $O(n^2)$ new relations per iteration. For each one, two languages $A_1|_{A_2} . A_3$ must be computed, giving a total cost of $O(n^2 \log^2 n)$ per iteration.

It has been said that the number of iterations is at most the function's number of arguments, which is usually small. Even if it is not, in practice it suffices to perform only three iterations of the analysis before applying the widening, and then an additional iteration in order to to check that the fixpoint has been reached. This checking is the most expensive operation of the analysis. A maximum of $O(n^2)$ languages are tested for equality, giving a total theoretical cost of $O(n^2 2^{\log n} \log^3 n)$ in the worst case, i.e. $O(n^3 \log^3 n)$.

A worst-case cost of $O(n^3 \log^3 n)$ is by no means a low one, but we consider it to be rather pessimistic. We remark that we are assuming each variable to be related to each other, and all conversions from NFA to DFA to produce an exponential blow-up of states. This leads us to think that this theoretical cost is almost never reached. Also, in functional programming it is common to write small functions. So, the number n of bound variables can be expected to remain below 20 for most of the functions (the reader is invited to check this assertion for the functions presented in this paper).

In practice our analysis is affordable for medium-size functions. More importantly, it is modular, because once a function definition is analysed, all its relevant information is recorded in the signature environment. Hence, the compilation of a big program is still linear in the program size, even if analysing each individual function of size n takes a time in $O(n^3 \log^3 n)$.

We have implemented the analysis presented here, which has been integrated into our *Safe* compiler, written in Haskell. We have extended the HaLeX library [15], which manipulates regular languages, with new operations such as language intersection, derivation and equality. While the implementation of the abstract interpretation rules of Fig. 6 is straightforward, the closure operation defined in Fig. 7 is much more involved.

Even though the automata library is not particularly efficient and there is much space for optimization, our prototype implementation is able to analyse a file with 40 small functions similar to `msort`, in less than ten seconds in a standard laptop computer.

In order to illustrate the analysis, we present in Fig. 10 the code of a function `last` computing the last element of a non-empty list. By iterating once the interpretation, and in the places marked in the text, we get the following two sets:

$$
\begin{aligned}
R_1 &= \{xs \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} xs\} \uplus_x \{xs \xrightarrow{1} \bullet \xleftarrow{\epsilon} x\} \uplus_{xx} \\
&\quad\ \{xs \xrightarrow{2} \bullet \xleftarrow{\epsilon} xx\} \\
R_2 &= R_1 \uplus_y \{xx \xrightarrow{1} \bullet \xleftarrow{\epsilon} y\} \uplus_{yy} \{xx \xrightarrow{2} \bullet \xleftarrow{\epsilon} yy\}
\end{aligned}
$$

Then $\Sigma_1 = \mathcal{I}_{last}\ \{last \mapsto \emptyset\} = \{res \xrightarrow{\epsilon} \bullet \xleftarrow{1} xs\}$, where we omit the reflexive relations. By applying again the interpretation, we get:

$$
\begin{aligned}
\Sigma_2 = \mathcal{I}_{last}\ \{last \mapsto \Sigma_1\} &= \{res \xrightarrow{\epsilon} \bullet \xleftarrow{21} xs\}\ \cup \\
&\quad\ \{res \xrightarrow{\epsilon} \bullet \xleftarrow{1} xs\} \\
&= \{res \xrightarrow{\epsilon} \bullet \xleftarrow{21+1} xs\}
\end{aligned}
$$

The language 21 is obtained by the closure $\{res \xrightarrow{\epsilon} \bullet \xleftarrow{1} xx\} \uplus_{res} \{xs \xrightarrow{2} \bullet \xleftarrow{\epsilon} xx\}$. In the next round, we get $\Sigma_3 = \{res \xrightarrow{\epsilon} \bullet \xleftarrow{2(21+1)+1} xs\}$ Applying now the widening step, we get $\Sigma_3 = \{res \xrightarrow{\epsilon} \bullet \xleftarrow{2^*1+21+1} xs\}$, and by applying the interpretation once more:

$$\begin{aligned}\mathcal{I}_{last}\ \{last \mapsto \Sigma_3\} &= \{res \xrightarrow{\epsilon} \bullet \xleftarrow{2(2^*1+21+1)} xs\}\ \cup \\ &\quad\ \{res \xrightarrow{\epsilon} \bullet \xleftarrow{1} xs\} \\ &= \{res \xrightarrow{\epsilon} \bullet \xleftarrow{2(2^*1+21+1)+1} xs\}\end{aligned}$$

The final test is $2(2^*1 + 21 + 1) + 1 \subseteq 2^*1 + 21 + 1$ which returns *true* because all the words in the left language are also in the right one. Notice that the right expression could be further simplified to 2^*1. This language clearly expresses that the result of `last` is a descendant of the argument list that can be reached by taking the tail of the list a number of times and then by taking the head.

6 Case Studies

Besides the examples already shown in the paper, we have applied our analysis to some additional ones involving list and binary tree manipulations. The following functions show how our analysis can also detect internal sharing in the data structure given as a result. This is useful to know whether a given data structure is laid out in memory without overlapping.

```
buildTree x 0 = Empty
buildTree x n = Node (buildTree x (n-1)) x (buildTree x (n-1))

buildTreeSh x 0 = Empty
buildTreeSh x n = let t = buildTree x (n-1) in Node t x t
```

The shape analysis yields the results given below. We also include the inferred sharing relations of the `append`, `partition` and `qsort` functions that make up a typical *Quicksort* implementation:

$$\begin{aligned}
buildTree\ x\ n :&\ \{res \xrightarrow{(1+3)^*2} \bullet \xleftarrow{\epsilon} x \\
&\ ,\ res \xrightarrow{(1+3)^*2} \bullet \xleftarrow{(1+3)^*2} res\} \\
buildTreeSh\ x\ n :&\ \{res \xrightarrow{(1+3)^*2} \bullet \xleftarrow{\epsilon} x \\
&\ ,\ res \xrightarrow{(1+3)^*2} \bullet \xleftarrow{(1+3)^*2} res \\
&\ ,\ res \xrightarrow{(1+3)^*} \bullet \xleftarrow{(1+3)^*} res\} \\
append\ xs\ ys :&\ \{res \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} xs,\ res \xrightarrow{2^*} \bullet \xleftarrow{\epsilon} ys\} \\
partition\ p\ xs :&\ \{res \xrightarrow{12^*1+22^*1} \bullet \xleftarrow{2^*1} xs\} \\
qsort\ xs :&\ \{res \xrightarrow{2^*1} \bullet \xleftarrow{2^*1} xs\}
\end{aligned}$$

7 Related Work and Conclusions

There exist many different analyses dedicated to extracting information about the heap, mainly in imperative languages where pointers are explicitly used and may be reassigned. *Alias analysis* is one of the most studied. It tries to detect program variables that point to the same memory location. *Pointer analysis* aims at determining the storage locations a pointer can point to, so it may be also used to detect aliases in a program. These analyses are used in many different applications such as live variable analysis for register allocation and constant propagation. In [11, 4, 12] we can find surveys about pointer analysis applied to imperative languages from the 80's. *Shape analysis* [14, 8, 13] tries to approximate the 'shape' of the heap-allocated structures. That information has been used, for example, for binding time optimizations.

The level of detail of all these analyses mainly depends on the user of the analysis. Our analysis tries to capture a kind of sharing information more refined than alias and pointer analysis may provide, and in fact both are subsumed in our relations: if $x \xrightarrow{\epsilon} \bullet \xleftarrow{\epsilon} y$, then, x and y are aliases; if $x \xrightarrow{j} \bullet \xleftarrow{\epsilon} y$, then x points to y (i.e. y is the j-th child of the data structure x). Shape analysis is nearer to our needs.

Jones and Muchnick [8] associate sets of k-limited graphs to each program point in order to approximate the sharing relations between variables. The k limits the length of the paths in the graphs modeling the heap in order to make the domain finite and obtain the minimal fixpoint by iteration. The graphs obtained after the abstract execution of a program instruction must be transformed in order to maintain themselves k-limited. Our widening operator ressembles this operation. Our path relations are in general uncomparable in precision to these sets of limited graphs. First, having sets of graphs may provide more precision because our union operation loses information: adding $x \xrightarrow{p_1+p_3} \bullet \xleftarrow{p_2+p_4} y$ introduces combinations of paths $x \xrightarrow{p_1} \bullet \xleftarrow{p_4} y$ and $x \xrightarrow{p_3} \bullet \xleftarrow{p_2} y$ which did not exist previously. Second, paths longer than k may be more precise that k-limited graphs: $x \xrightarrow{2221} \bullet \xleftarrow{\epsilon} y$ indicating that y is the fifth element of the list x is more precise than saying in a 2-limited graph that y shares in an *unknown* way with x after the path 22. Additionally, the cost of having sets of graphs is doubly exponential in the number of variables.

In order to reduce the cost to polinomial, Reps [13] formulated the analysis as a graph-reachability problem over the dependence graph generated from the program. The reachability is defined in terms of those (context-free) paths one is interested in. The fixpoint calculation in this case is also finite because he just records the information about the variables, not the exact paths. We need the paths in order to make the analysis more precise as shown in the *mergesort* example, that is why we need the widening. The use of context-free paths in our framework would make undecidable most of our tests.

Other related works are those devoted to compile-time garbage collection, such as [6, 7]. The first one tries to save creating a new array when updating an array that is only referenced once. The second one provides an analysis also detecting when a cell is referenced at most once by the subsequent computation. Its aim is to destroy the cell after its last use so that it can be reused by the runtime system. Both analyses are done on a first-order eager functional language. After these ones, there have been many similar analyses, usually known as *usage analyses* (e.g. [16, 1, 17, 3]) whose aim is to detect when a cell is used at most once and then, either to recover or to avoid to update it, when the language is lazy. These analyses do not try to know *which* other data structures points to a particular cell, but rather *how many* of them do it, and in this sense they are simpler. The nearer to our problem is [7] since it pursues an aim similar to that of *Safe*: to save memory. The main difference is that, in our case, it is the programmer who decides to destroy a cell and the compiler just analyses whether doing this is safe or not. So, the programmer may have destructive and non-destructive versions of the same function and uses the first one in contexts where it is safe to do it. In [7] it is the compiler who decides to destroy the cell, when it is safe to do it in *all* the contexts in which the function is called. A single unsafe context will avoid to recover the cell in all the safe ones. Another important difference is that our analysis is modular, while theirs need to analyse the program as a whole. This makes it unpractical for big programs.

References

1. Erik Barendsen and Sjaak Smetsers. Uniqueness typing for functional languages with graph rewriting semantics. *Mathematical Structures in Computer Science*, 6(6):579–612, December 1996.
2. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *Proc. 4th ACM Symp. on Principles of Prog. Languages*, pages 238–252. ACM, 1977.
3. J. Gustavsson and J. Sveningsson. A Usage Analysis with Bounded Usage Polymorphism and Subtyping. In *Selected Papers of the 12th International Workshop on Implementation of Functional Languages, IFL'00*, volume 2011 of *LNCS*, pages 140–157. Springer-Verlag, 2001.
4. M. Hind. Pointer analysis: Haven't we solved this problem yet? In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering, PASTE'01*, pages 54–61. ACM Press, 2001.
5. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 2nd edition, 2001.

6. P. Hudak. A Semantic Model of Reference Counting and its Abstraction (Detailed Summary). In *ACM Symposium on Lisp and Functional Programming*, pages 351–363. ACM, 1986.
7. T. P. Jensen and T. AE. Mogensen. A Backwards Analysis for Compile-Time Garbage Collection. In *European Symposium on Programming*, pages 227–239. LNCS 432, Springer, 1990.
8. N. D. Jones and S. S. Muchnick. Flow analysis and optimization of lisp-like structures. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '79, pages 244–256. ACM, 1979.
9. M. Montenegro, R. Peña, and C. Segura. A Type System for Safe Memory Management and its Proof of Correctness. In *ACM Principles and Practice of Declarative Programming, PPDP'08, Valencia, Spain, July. 2008*, pages 152–162, 2008.
10. R. Peña, C. Segura, and M. Montenegro. A Sharing Analysis for SAFE. In *Selected Papers Trends in Functional Programming, TFP'06*, pages 109–128. Intellect, 2007.
11. V. Raman. Pointer analysis – a survey. CS203 UC Santa Cruz, `http://www.soe.ucsc.edu/~vishwa/publications/Pointers.pdf`, 2004.
12. D. Rayside. Points–to analysis. `http://www.cs.utexas.edu/~pingali/CS395T/2012sp/lectures/points-to.pdf`, 2005.
13. T. Reps. Shape analysis as a generalized path problem. In *Proceedings of the 1995 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, PEPM '95, pages 1–11. ACM, 1995.
14. J. C. Reynolds. Automatic computation of data set definitions. In *IFIP Congress (1)*, pages 456–461, 1968.
15. João Saraiva. HaLeX: A Haskell Library to Model, Manipulate and Animate Regular Languages. In *Proc. ACM Workshop on Functional and Declarative Programming in Education*, University of Kiel. Tech. Report 0210, pages 133–140, 2002.
16. D. N. Turner, P. L. Wadler, and C. Mossin. Once upon a type. In *7'th International Conference on Functional Programming and Computer Architecture*, pages 1–11, La Jolla, California, June 1995. ACM Press.
17. K. Wansbrough and S. L. Peyton Jones. Once upon a polymorphic type. In *The Twenty-sixth ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Antonio, Texas, January 1999.