# A Space Consumption Analysis By Abstract Interpretation[☆]

Technical Report 01/11
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Manuel Montenegro, Ricardo Peña, Clara Segura

*Departamento de Sistemas Informáticos y Computación,*
*Facultad de Informática, Universidad Complutense de Madrid,*
*c/Profesor José García Santesmases s/n, 28040 Madrid, Spain*

**Abstract**

*Safe* is a first-order functional language with an implicit region-based memory system. A static analysis for inferring regions and a type system have been presented elsewhere. In this paper we present a new analysis aimed at inferring upper bounds for heap and stack consumption. It is based on abstract interpretation, being the abstract domain the set of all $n$-ary monotonic functions from real non-negative numbers to a real non-negative result. This domain turns out to be a complete lattice under the usual $\sqsubseteq$ relation on functions. Our interpretation is monotonic in this domain and the solution we seek is the least fixpoint of the interpretation.

We first explain the abstract domain and some correctness properties of the interpretation rules with respect to the language semantics, then present the inference algorithms for recursive functions, and finally illustrate the approach with the upper bounds obtained by our implementation for some case studies.

*Keywords:* Regions, space consumption, abstract interpretation.

## 1. Introduction

The first-order functional language *Safe* has been developed in the last few years as a research platform for analysing and formally certifying two properties of programs related to memory management: absence of dangling pointers and having an upper bound to memory consumption. Two features make *Safe* different from conventional functional languages: (a) a region based memory management system which does not need a garbage collector; and (b) a programmer may ask for explicit destruction of memory cells, so that they could be reused by the program. These characteristics, together with the above certified properties, make *Safe* useful for programming small devices where memory requirements are rather strict and where garbage collectors are a burden in service availability.

The *Safe* compiler is equipped with a battery of static analyses which infer such properties [10, 12, 13]. These analyses are carried out on an intermediate language called *Core-Safe* explained below. We have developed a

[☆]Work partially funded by the projects TIN2008-06622-C03-01/TIN (STAMP), S-0505/ TIC/ 0407 (PROMESAS) and the MEC FPU grant AP2006-02154.

*Email addresses:* montenegro@fdi.ucm.es (Manuel Montenegro), ricardo@sip.ucm.es (Ricardo Peña), csegura@sip.ucm.es (Clara Segura)

resource-aware operational semantics of *Core-Safe* [11] producing not only values but also exact figures on the heap and stack consumption of a particular running. The code generation phases have been certified in a proof assistant [4, 5], so that there is a formal guarantee that the object code actually executed in the target machine (the JVM [9]) will exactly consume the figures predicted by the semantics.

Regions are dynamically allocated and deallocated. The compiler 'knows' which data lives in each region. Thanks to that, it can compute an upper bound to the space consumption of every region and so and upper bound to the total heap consumption. Adding to this a stack consumption analysis would result in having an upper bound to the total memory needs of a program.

In this work we present a static analysis aimed at inferring upper bounds for individual *Safe* functions, for expressions, and for the whole program. These have the form of $n$-ary mathematical functions relating the input argument sizes to the heap and stack consumption made by a *Safe* function, and include as particular cases multivariate polynomials of any degree. Given the complexity of the inference problem, even for a first-order language like *Safe*, we have identified three separate aspects which can be independently studied and solved: (1) Having an upper bound on the size of the call-tree deployed at runtime by each recursive *Safe* function; (2) Having upper bounds on the sizes of all the expressions of a recursive *Safe* function. These are defined as the number of cells needed by the normal form of the expression; and (3) Given the above, having an inference algorithm to get upper bounds for the stack and heap consumption of a recursive *Safe* function.

Several approaches to solve (1) and (2) have been proposed in the literature (see the Related Work section). We have obtained promising results for them by using rewriting systems termination proofs [10]. In case of success, these tools return multivariate polynomials of any degree as solutions. This work presents a possible solution to (3) by using abstract interpretation. It should be considered as a *proof-of-concept* paper: we investigate how good the upper bounds obtained by the approach are, provided we have the best possible solutions for problems (1) and (2). In the case studies presented below, we have introduced by hand the bounds to the call-tree and to the expression sizes.

The abstract domain is the set of all monotonic, non-negative, $n$-ary functions having real number arguments and real number result. This infinite domain is a complete lattice, and the interpretation is monotonic in the domain. So, fixpoints are the solutions we seek for the memory needs of a recursive *Safe* function. An interesting feature of our interpretation is that we usually start with an over-approximation of the fixpoint, but we can obtain tighter and tighter safe upper bounds just by iterating the interpretation any desired number of times.

The plan of the paper is as follows: Section 2 gives a brief description of our language; Section 3 introduces the abstract domain; Sections 4 and 5 give the abstract interpretation rules and some proof sketches about their correctness, while Section 6 is devoted to our inference algorithms for recursive functions; in Section 7 we apply them to some case studies, and finally in Section 8 we give some account on related and future work.

## 2. Safe in a Nutshell

*Safe* is polymorphic and has a syntax similar to that of (first-order) Haskell. In *Full-Safe* in which programs are written, regions are implicit. These are inferred when *Full-Safe* is desugared into *Core-Safe* [13], where regions are explicit:

$$
\begin{array}{lcl}
\mathit{prog} & \rightarrow & \overline{\mathit{data}_i}^{\,n}; \overline{\mathit{dec}_j}^{\,m}; e \\
\mathit{data} & \rightarrow & \textbf{data}\ T\ \overline{\alpha_i}^{\,n}\ @\ \overline{\rho_j}^{\,m} = \overline{C_k\ \overline{t_{ks}}^{\,n_k}\ @\ \rho_m}^{\,l} \\
\mathit{dec} & \rightarrow & f\ \overline{x_i}^{\,n}\ @\ \overline{r_j}^{\,l} = e \\
e & \rightarrow & c \mid x \mid C\ \overline{a_i}^{\,n}\ @\ r \mid f\ \overline{a_i}^{\,n}\ @\ \overline{r_j}^{\,l} \mid \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2 \mid \\
 & & \textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{\,n_i} \rightarrow e_i}^{\,n}
\end{array}
$$

The allocation and deallocation of regions is bound to function calls: a *working region* called *self* is allocated when entering the call and deallocated when exiting it. So, at any execution point only a small number of regions, kept in an invocation stack, are alive. The data structures built at *self* will die at function termination, as the following treesort algorithm shows:

```
treesort xs = inorder (mkTree xs)
```

First, the original list `xs` is used to build a search tree by applying function `mkTree` (not shown). The tree is traversed in inorder to produce the sorted list. The tree is not part of the result of the function, so it will be built in the working region and will die when the `treesort` function returns. The *Core-Safe* version of `treesort` showing the inferred type and regions is the following:

```
treesort :: [a] @ rho1 -> rho2 -> [a] @ rho2
treesort xs @ r = let t = mkTree xs @ self
                  in inorder t @ r
```

Variable `r` of type `rho2` is an additional argument in which `treesort` receives the region where the output list should be built. This is passed to the `inorder` function. However `self` is passed to `mkTree` to instruct it that the intermediate tree should be built in `treesort`'s *self* region.

In Fig. 1 we show the *Core-Safe* big-step semantic rules, where extra annotations have been added in order to obtain additional information as a side effect of evaluating an expression. We will have judgments of the form:

$$E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s)$$

meaning that expression $e$ is evaluated in an environment $E$ using the $td$ topmost positions in the stack, and in a heap $(h, k)$ with $0..k$ active regions. As a result, a heap $(h', k)$ and a value $v$ are obtained, and a resource vector $(\delta, m, s)$ is consumed, which is explained below in detail.

Notice that $k$ does not change because the number of active regions increases by one at each application and decreases by one at each function return, and all applications during $e$'s evaluation have been completed. A heap $h$ is a mapping between pointers and constructor cells $(j, C\ \overline{v_i}^n)$, where $j$ is the cell region. The first component of the resource vector is a partial function $\delta : \mathbb{N} \to \mathbb{N}$ giving for each active region $i$ the difference between the cells in the final and initial heaps. By $dom(\delta)$ we denote the subset of $\mathbb{N}$ in which $\delta$ is defined. By $|\delta|$ we mean the sum $\sum_{n \in dom(\delta)} \delta(n)$ giving the total balance of cells. The remaining components $m$ and $s$ respectively give the *minimum* number of fresh cells in the heap and of words in the stack needed to successfully evaluate $e$. When $e$ is the main expression, these figures give us the total memory needs of a particular run of the *Safe* program. It is easy to see that $m \geq |\delta|$. For a full description of the semantics and the abstract machine see [11]. There, there is an additional rule for a let expression in which the auxiliary expression is a constructor application. This is more efficient than having a general let expression. In order to simplify proofs in this paper we have preferred a single let. Everything can be easily adapted to having several optimized ones.

### *2.1. Counting the number of recursive calls*

An important precondition for the correctness of the algorithms described in the following sections is the fact that we use upper bounds of the actual number of recursive and base calls, and the maximum number of nested calls. In order to take these figures into account we add extra annotations to the big-step operational semantics of Figure 1. We will have judgments of the form:

$$E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s), (n_t, n_b, l)_f$$

$$E \vdash h, k, td, c \Downarrow h, k, c, ([\,], 0, 1)\ [Lit]$$

$$E[x \mapsto v] \vdash h, k, td, x \Downarrow h, k, v, ([\,], 0, 1)\ [Var]$$

$$\frac{j \leq k \quad fresh(p)}{E \vdash h, k, td, C\ \overline{a_i}^n\ @\ r \Downarrow h \uplus [p \mapsto (j, C\ \overline{v_i}^n)], k, p, ([j \mapsto 1], 1, 1)}\ [Cons]$$

$$\frac{(f\ \overline{x_i}^n\ @\ \overline{r_j}^l = e) \in \Sigma \quad [\overline{x_i \mapsto E(a_i)}^n, \overline{r_j \mapsto E(r'_j)}^l, self \mapsto k+1] \vdash h, k+1, n+l, e \Downarrow h', k+1, v, (\delta, m, s)}{E \vdash h, k, td, f\ \overline{a_i}^n\ @\ \overline{r'_j}^l \Downarrow h'|_k, k, v, (\delta|_k, m, \max\{n+l, s+n+l-td\})}\ [App]$$

$$\frac{E \vdash h, k, td, a_1 \Downarrow h, k, v_1, ([\,], 0, 1) \quad E \vdash h, k, td, a_2 \Downarrow h, k, v_2, ([\,], 0, 1)}{E \vdash h, k, td, a_1 \oplus a_2 \Downarrow h, k, v_1 \oplus v_2, ([\,], 0, 2)}\ [Primop]$$

$$\frac{\begin{array}{c}E \vdash h, k, 0, e_1 \Downarrow h', k, v_1, (\delta_1, m_1, s_1) \\ E \cup [x_1 \mapsto v_1] \vdash h', k, td+1, e_2 \Downarrow h'', k, v, (\delta_2, m_2, s_2)\end{array}}{E \vdash h, k, td, \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2 \Downarrow h'', k, v, (\delta_1 + \delta_2, \max\{m_1, |\delta_1| + m_2\}, \max\{2 + s_1, 1 + s_2\})}\ [Let]$$

$$\frac{C = C_r \quad E \cup [\overline{x_{r_i} \mapsto v_i}^{n_r}] \vdash h, k, td + n_r, e_r \Downarrow h', k, v, (\delta, m, s)}{E[x \mapsto p] \vdash h[p \mapsto (j, C\ \overline{v_i}^n)], k, td, \mathbf{case}\ x\ \mathbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \rightarrow e_i}^n \Downarrow h', k, v, (\delta, m, s + n_r)}\ [Case]$$

Figure 1: Resource-Aware Operational semantics of *Safe* expressions

where $n_t$ is the total number of calls to $f$ occurring in the evaluation of $e$ (including the current call, since we assume that $f$ is the context function) from which $n_b$ calls correspond to base cases. The number of recursive childs in the call tree can be obtained by substracting $n_b$ from $n_t$. The maximum number of nested calls is reflected in $l$.

The resulting rules are shown in Figure 2. The $(\delta, m, s)$ annotations are left out for simplicity. All of them require no explanation, except the one corresponding to **let** expressions. In this case we sum the number of total calls from each subexpression and subtract 1 (otherwise we would count the actual call twice). With regard to the resulting $n_b$, if both subexpressions contain recursive calls we just add the corresponding $n_b$'s, otherwise we only consider the number of base calls of the subexpression not having recursive calls. This is specified by means of the $\oplus$ operator, defined as follows:

$$x \oplus_{n_{t1}, n_{t2}} y = \begin{cases} x & \text{if } n_{t2} = 1 \\ y & \text{if } n_{t1} = 1 \\ x + y & \text{e.o.c} \end{cases}$$

By simple inspection of the rules one can prove that $n_t \geq n_b$ and hence the expression $n_{b1} \oplus_{n_{t1}, n_{t2}} n_{b2}$ in $[Let]$ is well-defined. The following Lemma shows an important property of these annotations.

**Lemma 1.** *Let $e$ be an expression such that the following judgment holds for some $E$, $h$, $k$, $td_i$, $h'$,$v$, $\delta$, $m$, $s$, $n_t$, $n_b$ and $l$:*

$$E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s), (n_t, n_b, l)_f \tag{1}$$

*Let us assume that there are $p$ direct recursive calls to $f$ in the derivation of (1). That is, for each $i \in \{1 \ldots p\}$ there exist some $E_i$, $h_i$, $h'_i$, $v_i$, $\delta_i$, $m_i$, $s_i$, $n_{t,i}$, $n_{b_i}$ and $l_i$ such that:*

$$E_i \vdash h_i, k+1, td_i, e_f \Downarrow h'_i, k+1, v_i, (\delta_i, m_i, s_i), (n_{t,i}, n_{b,i}, l_i)_f$$

*belongs to (1). Therefore it holds that:*

$$n_t = 1 + \sum_{i=1}^{p} n_{t,i} \qquad n_b = \sum_{i=1}^{p} n_{b,i} \qquad l = 1 + \max\{l_i \mid i \in \{1 \ldots p\}\}$$

$$E \vdash h, k, td, c \Downarrow h, k, c, (1,1,1)_f \ [Lit]$$

$$E[x \mapsto v] \vdash h, k, td, x \Downarrow h, k, v, (1,1,1)_f \ [Var]$$

$$\frac{j \leq k \quad fresh(p)}{E \vdash h, k, td, C\ \overline{a_i}^n \ @\ r \Downarrow h \uplus [p \mapsto (j, C\ \overline{v_i}^n)], k, p, (1,1,1)_f} \ [Cons]$$

$$\frac{E \vdash h, k, td, a_1 \Downarrow h, k, v_1, (1,1,1) \quad E \vdash h, k, td, a_2 \Downarrow h, k, v_2, (1,1,1)}{E \vdash h, k, td, a_1 \oplus a_2 \Downarrow h, k, v_1 \oplus v_2, (1,1,1)} \ [Primop]$$

$$\frac{g \neq f \quad (g\ \overline{x_i}^n \ @\ \overline{r_j}^t = e) \in \Sigma \quad [\overline{x_i \mapsto E(a_i)}^n, \overline{r_j \mapsto E(r'_j)}^t, self \mapsto k+1] \vdash h, k+1, n+l, e \Downarrow h', k+1, v, (n_t, n_b, l)_f}{E \vdash h, k, td, g\ \overline{a_i}^n \ @\ \overline{r'_j}^t \Downarrow h'|_k, k, v, (n_t, n_b, l)_f} \ [App - NonRec]$$

$$\frac{(f\ \overline{x_i}^n \ @\ \overline{r_j}^t = e) \in \Sigma \quad [\overline{x_i \mapsto E(a_i)}^n, \overline{r_j \mapsto E(r'_j)}^t, self \mapsto k+1] \vdash h, k+1, n+l, e \Downarrow h', k+1, v, (n_t, n_b, l)_f}{E \vdash h, k, td, f\ \overline{a_i}^n \ @\ \overline{r'_j}^t \Downarrow h'|_k, k, v, (n_t+1, n_b, l+1)_f} \ [App - Rec]$$

$$\frac{\begin{array}{c} E \vdash h, k, 0, e_1 \Downarrow h', k, v_1, (n_{t1}, n_{b1}, l_1)_f \\ E \cup [x_1 \mapsto v_1] \vdash h', k, td+1, e_2 \Downarrow h'', k, v, (n_{t2}, n_{b2}, l_2)_f \end{array}}{E \vdash h, k, td, \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2 \Downarrow h'', k, v, (n_{t1} + n_{t2} - 1, n_{b1} \oplus_{n_{t1}, n_{t2}} n_{b2}, \max\{l_1, l_2\})_f} \ [Let]$$

$$\frac{C = C_r \quad E \cup [\overline{x_{r_i} \mapsto v_i}^{n_r}] \vdash h, k, td + n_r, e_r \Downarrow h', k, v, (n_t, n_b, l)_f}{E[x \mapsto p] \vdash h[p \mapsto (j, C\ \overline{v_i}^n)], k, td, \mathbf{case}\ x\ \mathbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n \Downarrow h', k, v, (n_t, n_b, l)_f} \ [Case]$$

Figure 2: Big-step operational semantics enriched with number of calls

## 3. Function Signatures

A *Core-Safe* function is defined as a $n + m$ argument expression:

$$f :: t_1 \to \ldots t_n \to \rho_1 \to \ldots \rho_m \to t$$
$$f\ x_1 \cdots x_n \ @\ r_1\ \cdots r_m = e_f$$

A function may charge space costs to heap regions and to the stack. In general, these costs depend on the *sizes* of the function arguments. For example,

```
copy xs @ r = case xs of []   -> [] @ r
                         y:ys -> let zs = copy ys @ r in
                                 let rs = (y:zs) @ r in rs
```

charges as many cells to region $r$ as the input list size. We define the size of an algebraic type term to be the number of cells of its recursive spine, which means that it is always at least 1.

We define the size of a boolean value to be zero. However, for a natural or a real number we take its value because frequently space costs depend on the value of a numeric argument.

As a consequence, all the costs, sizes and needs of $f$ can be expressed as functions on $f$'s argument sizes:

$$\mathbb{F} = \{\eta : (\mathbb{R}^+ \cup \{+\infty\})^n \to \mathbb{R}^+ \cup \{+\infty, -\infty\} \ \mid \ \eta \text{ is monotonic}\}$$

Cost or size $+\infty$ will be used to represent that we are not able to infer a bound (either because it does not exist or because the analysis is not powerful enough). We use $-\infty$ to express that the cost or size is not defined. As an example, the following size function

$$\lambda xs. \begin{cases} xs - 3 & \text{if } xs \geq 4 \\ -\infty & \text{otherwise} \end{cases}$$

where $xs$ represents the size of a list, is undefined for those sizes smaller than 4, i.e. for those lists with less than 3 elements.

The ordering on set $\mathbb{R}$ augmented with $\pm\infty$ is as usual:

$$-\infty \leq 0 \wedge \forall x \in \mathbb{R}^+.x \leq +\infty$$

so

$$-\infty \sqcup x = x \quad +\infty \sqcup x = +\infty$$

Arithmetic monotone operations with $\pm\infty$ are defined as follows, where $x \in \mathbb{R}^+$ while $y \in \mathbb{R}^+ \cup \{+\infty, -\infty\}$:

$$\begin{array}{ll} -\infty + y = -\infty & +\infty + x = +\infty \\ -\infty * y = -\infty & +\infty * x = +\infty \end{array}$$

The domain of space cost functions $(\mathbb{F}, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ is a complete lattice, where $\sqsubseteq$ is the usual order between functions, and the rest of the components are standard. Notice that it is closed by the operations $\{+, \sqcup, *\}$.

Function $f$ above may charge space costs to a maximum of $m+1$ regions: it may create cells in any output region $r_1 \ldots r_m$, and additionally in its *self* region. Each region $r$ has a region type $\rho$. We denote by $R^f_{in}$ the set of input region types of $f$, by $R^f_{out}$ the set of its output region types, and by $R^f_{arg}$ the set of argument region types. For example, $R^{treesort}_{in} = \{\rho_1\}$ and $R^{treesort}_{out} = R^{treesort}_{arg} = \{\rho_2\}$. We denote $R_f = R^f_{arg}$.

Looked from outside, the charges to the *self* region are not visible, as this region disappears when the function returns. So $\mathbb{D} = \{\Delta : R_f \to \mathbb{F}\}$ is the complete lattice of functions that describe the space costs charged by $f$ to every visible region. In the following we will call abstract heaps to the functions $\Delta \in \mathbb{D}$.

**Definition 1.** *A function* signature *for $f$ is a triple $(\Delta_f, \mu_f, \sigma_f)$, where $\Delta_f$ belongs to $\mathbb{D}$, and $\mu_f, \sigma_f$ belong to $\mathbb{F}$.*

The aim is that $\Delta_f$ describes (an upper bound to) the space costs charged by $f$ to every visible region, (i.e. the increment in live memory due to a call to $f$), and $\mu_f, \sigma_f$ respectively describe (an upper bound to) the heap and stack *needs* in order to execute $f$ without running out of space (i.e. the maximal increment in live memory during $f$'s evaluation).

We abbreviate $\lambda\overline{x_i}^n.c$ by $c$, when $c \in \mathbb{R}^+$. By $[\,]_f$ we denote the constant function $\lambda\rho \in R_f.\lambda\overline{x_i}^n.0$, and by $[\rho' \to n]_f$ we denote the following function:

$$\lambda\rho \in R_f.\lambda\overline{x_i}^n.\begin{cases} 0 & \text{if } \rho \neq \rho' \\ n & \text{if } \rho = \rho' \end{cases}$$

By $|\Delta|$ we mean $\sum_{\rho \in dom(\Delta)} \Delta\ \rho$. If $\eta \in \mathbb{F}$, by *dom* $\eta$ de denote the set of values where $\eta$ does not evaluate to $-\infty$, i.e.:

$$dom\ \eta \stackrel{\text{def}}{=} \{\overline{x} \mid \eta\ \overline{x} \neq -\infty\}$$

By abuse of notation, if $\Delta \in \mathbb{D}$, *dom* $\Delta$ is the intersection of the domains of *all* functions contained in $\Delta$.

## 4. Abstract Interpretation

In Figure 3 we show the abstract interpretation rules for the most relevant *Core-Safe* expressions. There, an atom $a$ represents either a variable $x$ or a constant $c$, and $|e|$ denotes the function obtained by the size analysis for expression $e$. We can assume that the abstract syntax tree is decorated with such information.

$$[\![c]\!]\ \Sigma\ \Gamma\ td = ([\,]_f, 0, 1) \quad [Lit]$$

$$[\![x]\!]\ \Sigma\ \Gamma\ td = ([\,]_f, 0, 1) \quad [Var]$$

$$[\![a_1 \oplus a_2]\!]\ \Sigma\ \Gamma\ td = ([\,]_f, 0, 2) \quad [Primop]$$

$$[\![C\ \overline{a_i}^n\ @\ r]\!]\ \Sigma\ \Gamma\ td = ([\Gamma\ r \mapsto 1], 1, 1) \quad [Cons]$$

$$\frac{\begin{array}{lll}\Sigma\ g = (\Delta_g, \mu_g, \sigma_g) & G\ \overline{x}^n \equiv \forall i \in \{1..l\}.|a_i|\ \overline{x}^n \neq -\infty & \theta = unify\ \Gamma\ g\ \overline{r_j}^q \\ \mu = \lambda\overline{x}^n.[G\ \overline{x}^n \to \mu_g\ (\overline{|a_i|\ \overline{x}^n}^l)] & \sigma = \lambda\overline{x}^n.[G\ \overline{x}^n \to \sigma_g\ (\overline{|a_i|\ \overline{x}^n}^l)] & \Delta = \lambda\rho \in R_f.\lambda\overline{x}^n.[G\ \overline{x}^n \to \theta\downarrow^{\rho}_{\overline{|a_i|\ \overline{x}^n}^l}\ \Delta_g]\end{array}}{[\![g\ \overline{a_i}^l\ @\ \overline{r_j}^q]\!]\ \Sigma\ \Gamma\ td = (\Delta, \mu, \sqcup\{l+q, \sigma - \ td + l + q\})}\ [App]$$

$$\frac{[\![e_1]\!]\ \Sigma\ \Gamma\ 0 = (\Delta_1, \mu_1, \sigma_1) \quad [\![e_2]\!]\ \Sigma\ \Gamma\ (td+1) = (\Delta_2, \mu_2, \sigma_2)}{[\![\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2]\!]\ \Sigma\ \Gamma\ td = (\Delta_1 + \Delta_2, \sqcup\{\mu_1, |\Delta_1| + \mu_2\}, \sqcup\{2 + \sigma_1, 1 + \sigma_2\})}\ [Let]$$

$$\frac{(\forall i)\ [\![e_i]\!]\ \Sigma\ \Gamma\ (td + n_i) = (\Delta_i, \mu_i, \sigma_i)}{[\![\textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n]\!]\ \Sigma\ \Gamma\ td = (\bigsqcup_{i=1}^n \Delta_i, \bigsqcup_{i=1}^n \mu_i, \bigsqcup_{i=1}^n (\sigma_i + n_i))}\ [Case]$$

Figure 3: Space inference rules for expressions with non-recursive applications

When inferring an expression $e$, we assume it belongs to the body of a function definition $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$, that we will call the *context* function, which we assume well-typed according to the type system in [12]. We consider available a global type environment $\Gamma$ which contains the types of all variables in $e_f$. This has been obtained using the corresponding inference algorithm. It allows to type $e_f$ and also all its subexpressions.

In this section we restrict to non-recursive functions and we will assume it is available a global environment $\Sigma$ giving, for each *Safe* function $g$ in scope, its signature $(\Delta_g, \mu_g, \sigma_g)$.

The abstract interpretation $[\![e]\!]\ \Sigma\ \Gamma\ td$ gives a triple $(\Delta, \mu, \sigma)$ representing the space costs and needs of expression $e$. The statically determined value $td$ occurring as an argument of the interpretation and used in rule *App* is the size of the top part of the environment used when compiling the expression $g\ \overline{a_i}^l\ @\ \overline{r_j}^q$. This size is also an argument of the operational semantics. See [11] for more details.

Rules $[Lit]$, $[Var]$, $[Primop]$ and $[Cons]$ exactly reflect the corresponding resource-aware semantic rules shown in Section 3. When a function application $g\ \overline{a_i}^l\ @\ \overline{r_j}^q$ is found, its signature $\Sigma\ g$ is applied to the sizes of the actual arguments, $\overline{|a_i|\ \overline{x_j}^n}^l$ which have the $\overline{x_j}^n$ as free variables. Due to the application, some different region types of $g$ may instantiate to the same actual region type of $f$. That means that we must accumulate the memory consumed in some formal regions of $g$ in order to get the charge to an actual region of $f$. In Figure 3, $unify\ \Gamma\ g\ \overline{r_j}^q$ computes a substitution $\theta$ from $g$'s region types to $f$'s region types or $\rho^f_{self}$. If $\theta\ \rho_g = \rho_f$, this means that the generic $g$'s region type $\rho_g$ is instantiated to the $f$'s actual region type $\rho_f$. Formally, $\theta :: R_g \to R_f \cup \{\rho^f_{self}\}$ is total.

**Definition 2.** *Given a type environment $\Gamma$, a function $g$ and the sequence $\overline{r_j}^q$, we say that $\theta = unify\ \Gamma\ g\ \overline{r_j}^q$ where $\Gamma\ g = \forall\overline{\alpha}.\overline{t_i}^l \to \overline{\rho_j}^q \to t$ iff*

$$\forall j \in \{1 \ldots q\}.\theta\ \rho_j = \Gamma\ r_j$$

As an example, let us assume $g :: t \to \rho^g_2 \to \rho^g_4 \to \rho^g_5 \to t'$ and consider the application $g\ p\ @\ r_2\ r_1\ r_1$ where $r_1 :: \rho^f_1$ and $r_2 :: \rho^f_2$. The resulting substitution would be:

$$\theta = [\rho^g_2 \mapsto \rho^f_2, \rho^g_4 \mapsto \rho^f_1, \rho^g_5 \mapsto \rho^f_1]$$

The function $\theta \downarrow^{\rho}_{\eta_i\ \overline{\overline{x}^n}^l}\ \Delta_g$ converts an abstract heap for $g$ into an abstract heap for $f$. It is defined as follows:

$$\theta \downarrow^{\rho}_{\eta_i\ \overline{\overline{x_j}^n}^l}\ \Delta_g = \sum_{\substack{\rho' \in R_g \\ \theta\ \rho' = \rho}} \Delta_g\ \rho'\ \overline{\eta_i\ \overline{x_j}^n}^l$$

where $\rho \in R_f \cup \{\rho_{self}\}, \eta_i \in \mathbb{F}$.

Notice that if any of the sizes is not defined we cannot apply $\Delta_g$, as $-\infty$ is not in its domain, so the result is not defined. We use a guarded notation

$$[G\ \overline{x}^n \rightarrow f\ \overline{x}^n]$$

which is equivalent to

$$\begin{cases} -\infty & \text{if } \neg G\ \overline{x}^n \\ f\ \overline{x}^n & \text{if } G\ \overline{x}^n \end{cases}$$

Later we will use a more general notation with several non-overlapping guards

$$[G_1\ \overline{x}^n \rightarrow f_1\ \overline{x}^n\ \ \square \ldots\ \ \square\ \ G_k\ \overline{x}^n \rightarrow f_k\ \overline{x}^n]$$

which is equivalent to

$$\begin{cases} -\infty & \text{if } \wedge_{j=1}^{k} \neg G_j\ \overline{x}^n \\ f_1\ \overline{x}^n & \text{if } G_1\ \overline{x}^n \\ \ldots & \\ f_k\ \overline{x}^n & \text{if } G_k\ \overline{x}^n \end{cases}$$

When defining functions with guards like this we have to ensure that the resulting function is in $\mathbb{F}$, i.e. it is positive (or $-\infty$) and monotone. For example, the following function

$$\lambda x. \begin{cases} x & \text{if } \lfloor x \rfloor \bmod 2 = 0 \\ -\infty & \text{otherwise} \end{cases}$$

is not monotone. It is easy to see that $\Delta$, $\mu$ and $\sigma$ defined in rule $[App]$ are monotone. If for any $i \in \{1..l\}$, $|a_i| = -\infty$ then $\Delta$, $\mu$ and $\sigma$ return $-\infty$, which guarantees monotonicity because its is the smallest value in the domain. For the rest of the arguments monotonicity is guaranteed by monotonicity of $\Delta_g$, $\mu_g$ and $\sigma_g$.

In the example, we have:

$$\Delta\ \rho_2^f = \lambda \overline{x}^n. \begin{cases} -\infty & \text{if } \exists i \in \{1..l\}.|a_i|\ \overline{x}^n = -\infty \\ \Delta_g\ \rho_2^g\ \overline{(|a_i|\ \overline{x}^n)}^l & \text{otherwise} \end{cases}$$

$$\Delta\ \rho_1^f = \lambda \overline{x}^n. \begin{cases} -\infty & \text{if } \exists i \in \{1..l\}.|a_i|\ \overline{x}^n = -\infty \\ \Delta_g\ \rho_4^g\ \overline{(|a_i|\ \overline{x}^n)}^l + \Delta_g\ \rho_5^g\ \overline{(|a_i|\ \ \overline{x}^n)}^l & \text{otherwise} \end{cases}$$

Rule $[Let]$ reflects the corresponding resource-aware semantic rule. Rule $[Case]$ uses the least upper bound operators $\bigsqcup$ in order to obtain an upper bound to the charge costs and needs of the alternatives.

## 5. Correctness of the Abstract Interpretation

Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$, be the *context* function and $\Gamma$ the inferred global type environment.Let us assume an execution of $e_f$ under some $E_0, h_0, k_0$ and $td_0$:

$$E_0 \vdash h_0, k_0, td_0, e_f \Downarrow h_f, k_0, v_f, (\delta_0, m_0, s_0) \tag{2}$$

We will consider only *valid* judgements in which the regions in the domain of $E_0$ are $\{r_j \mid j \in \{1..m\}\} \cup \{self\}$ and:

$$\forall j \in \{1..m\}.E_0\ r_j \in \{0..k_0 - 1\} \wedge E_0\ self = k_0$$

In the following, all $\Downarrow$–judgements corresponding to a given sub-expression of $e_f$ will be assumed to belong to the derivation of (2). It is easy to see by inspection of the operational semantic rules that those judgements are valid if (2) is valid. When a function application is evaluated the environment is built so that the validity is preserved for the new context function.

The correctness argument is split into three parts. First, we shall define a notion of *correct signature* which formalises the intuition of the inferred $(\Delta, \mu, \sigma)$ being an upper bound of the actual $(\delta, m, s)$. Then we prove that the inference rules of Figure 3 are correct, assuming that the signatures given by $\Sigma$ are correct and that the size analysis is correct. Finally, the correctness of the signature inference algorithm is proved, in particular when the function being inferred is recursive.

We need to provide the following definitions:

1. A notion of size of a data structure, so that we can refer to the sizes of the arguments of a function.
2. A notion of correct size analysis, whose results are used in the abstract interpretation.
3. A definition for a correct function signature.

First, we define different sizes of a data structure.

**Definition 3.** *Given a pointer $p$ belonging to a heap $h$, the function size returns the number of cells in $h$ of the data structure starting at $p$:*

$$size(h[p \mapsto (j, C\ \overline{v_i}^n)], p) = 1 + \sum_{i \in RecPos(C)} size(h, v_i)$$

*where $RecPos(C)$ denotes the recursive positions of constructor $C$. In case $p \notin dom(h)$ we define $size(h, p) = 0$.*

For example, if $p$ points to the first cons cell of the list $[1, 2, 3]$ in the heap $h$ then $size(h, p) = 4$. We assume that $size(h, b) = 0$ for every heap $h$ and boolean constant $b$, and $size(h, n) = n$ for every numeric constant $n$.

**Definition 4** (Correct size analysis)**.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ be the context function. The size analysis $|\cdot|$ is correct if given any initial environment $E_0$ and heap $h_0$ such that judgement (2) is derivable, then for all subexpressions $e$ of its body such that the judgement:*

$$E \vdash h, k_0, td, e \Downarrow h', k_0, v, (\delta, m, s)$$

*belongs to the derivation of (2) it holds that*

$$\forall x \in dom\ E :\ |x|\ \overline{s_i}^n \geq size(h, E\ x)\ \text{where}\ s_i = size(h_0, E_0\ x_i)\ \text{for each}\ i \in \{1 \ldots n\}$$

In the following we will assume that a correct size analysis is available.

In order to define the notion of correct signature, we consider *region instantiations*, denoted by $\phi, \phi', \ldots$, which are partial mappings from region types $\rho$ to natural numbers $i$. Region instantiations are needed to specify the actual region $i$ to which every $\rho$ is instantiated at a given execution point. An instantiation $\phi : R_f \cup \{\rho^f_{self}\} \to \mathbb{N}$ is *consistent* with an environment $E$ and a type environment $\Gamma$ if $\phi$ does not contradict the region instantiation obtained at runtime from $E$ and $\Gamma$, i.e. common type region variables are bound to the same actual region:

$$E\ r = i \wedge \Gamma\ r = \rho \Rightarrow \phi\ \rho = i$$

In case $\Gamma$ is inyective then $\phi = E \cdot \Gamma^{-1}|_{R_f \cup \{\rho^f_{self}\}}$ is the only instantiation which is consistent with $E$ and $\Gamma$. This will be the case for function bodies, in which the type inference algorithm gives a different type to each region argument. Henceforth we will abbreviate $\phi = E \cdot \Gamma^{-1}$.

Notice that given a valid judgement (2) and an inyective $\Gamma$ then $\phi = E_0 \cdot \Gamma^{-1}|_{R_f \cup \{\rho^f_{self}\}}$ then:

$$\forall j \in \{1..m\}.\phi\ \rho_j \in \{0..k_0 - 1\} \wedge \phi\ \rho^f_{self} = k_0$$

A formal definition of consistency can be found in [12], where we also proved that if a function is well-typed, consistency of region instantiations is preserved along its execution.

**Definition 5.** *Given a sequence of sizes $\overline{s_i}^n$ for the input parameters, a number $k$ of regions and a region instantiation $\phi$, we say that*

- *$\Delta$ is an upper bound for $\delta$ in the context of $\overline{s_i}^n$, $k$ and $\phi$, denoted by $\Delta \succeq_{\overline{s_i}^n,k,\phi} \delta$ iff*

$$\forall j \in \{0 \ldots k\} : \sum_{\phi\ \rho=j} \Delta\ \rho\ \overline{s_i}^n \geq \delta\ j$$

- *$\mu$ is an upper bound for $m$, denoted $\mu \succeq_{\overline{s_i}^n} m$, iff $\mu\ \overline{s_i}^n \geq m$; and*
- *$\sigma$ is an upper bound for $s$, denoted $\sigma \succeq_{\overline{s_i}^n} s$, iff $\sigma\ \overline{s_i}^n \geq s$.*

A signature $(\Delta_g, \mu_g, \sigma_g)$ for a function $g$ is said to be *correct* if the components $(\Delta_g, \mu_g, \sigma_g)$ are upper bounds to the actual $(\delta, m, s)$ obtained from any execution of $g$. This is formalised in the following definition.

**Definition 6** (Correct signature)**.** *Let $(\Delta_g, \mu_g, \sigma_g)$ the signature of a function definition $g\ \overline{y_i}^l\ @\ \overline{r_j}^q = e_g$ and $\Gamma_g$ the type environment inferred for $e_g$. This signature is said to be* correct *iff for all $h$, $h'$, $k$, $\overline{v_i}^l$, $\overline{i_j}^q$, $v$, $\delta$, $m$, $s$, $t$, $\overline{s_i}^n$ such that:*

1. *$E_g \vdash h, k+1, l+q, e_g \Downarrow h', k+1, v, (\delta, m, s)$*
   *where $E_g = [\overline{y_i \mapsto v_i}^l, \overline{r_j \mapsto i_j}^q, self \mapsto k+1]$*
2. *$\forall i \in \{1 \ldots l\} : s_i = size(h, v_i)$*

*then $\Delta_g \succeq_{\overline{s_i}^l,k,\phi} \delta|_k\ \wedge\ \mu_g \succeq_{\overline{s_i}^l} m\ \wedge\ \sigma_g \succeq_{\overline{s_i}^l} s$ for the consistent region instantiation $\phi$ determined by $E_g$ and $\Gamma_g$, i.e. $\phi = E_g \cdot \Gamma_g^{-1}$.*

The correctness of the abstract interpretation rules in Fig. 3 can be proven provided the type signatures in $\Sigma$ are correct and the size analysis is also correct.

**Theorem 1** (Correctness of the abstract interpretation)**.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ a non-recursive context function, $\Gamma$ the inferred global type environment for $e_f$, $\Sigma$ containing correct signatures for all the functions called from $e_f$, an initial environment $E_0$ and a heap $h_0$ such that judgement (2) is derivable. For each subexpression $e$ of $e_f$ and $E$, $td$, $\Delta$, $\mu$, $\sigma$, $h$, $,h'$, $v$, $,t$, $\delta$, $m$, $s$ such that:*

1. *$[\![e]\!]\ \Sigma\ \Gamma\ td = (\Delta, \mu, \sigma)$, where every occurrence of $|x|$ in its derivation has been inferred with a correct size analysis.*
2. *$E \vdash h, k_0, td, e \Downarrow h', k_0, v, (\delta, m, s)$, belongs to the derivation of (2)*

*then $\Delta \succeq_{\overline{s_i}^n, k_0, \phi} \delta$, $\mu \succeq_{\overline{s_i}^n} m$ and $\sigma \succeq_{\overline{s_i}^n} s$, where $s_i = size(h, E_0\ x_i)$ for each $i \in \{1 \dots n\}$, and the consistent region instantiation $\phi$ determined by $E$ and $\Gamma$.*

*Proof.* By structural induction on $e$. We show here the function application case. The rest are shown in Appendix A. In the following we shall leave out the $\overline{s_i}^n$ and $k_0$ subscripts in the $\succeq$ relations for a better readability.

$e \equiv g\ \overline{a_i}^l\ @\ \overline{r'_j}^q$ We shall assume that $\Sigma\ g \equiv g\ \overline{y_i}^l\ @\ \overline{r''_j}^q = e_g$ and, by using the corresponding rule:

$$E_g \vdash h, k_0 + 1, l + q, e_g \Downarrow h', k_0 + 1, v, (\delta_g, m_g, s_g)$$
$$\text{where } E_g = \left[\overline{y_i \mapsto E\ a_i}^l, \overline{r''_j \mapsto E\ r'_j}^q, self \mapsto k_0 + 1\right]$$

We can assume signature $(\Delta_g, \mu_g, \sigma_g)$ for function $g$ is correct. Function $g$ is well-typed and if $\Gamma\ g = \forall \overline{\alpha}\ \overline{\rho}.\overline{t_i}^l \to \overline{\rho_j}^q \to t$ then the global type environment $\Gamma_g = \Gamma' + [\overline{y_i : t_i}^l, \overline{r''_j : \rho_j}^q, self : \rho_{self}]$ has been inferred for $e_g$. On the other hand, if $s_{i,g}$ denote the size of the $i$-th actual argument before evaluating the function's body (i.e. $\forall i \in \{1 \dots l\} : s_{i,g} = size(h, E_g\ y_i)$) then:

$$\Delta_g \succeq_{\overline{s_{i,g}}^l, k_0, \phi'} \delta_g|_{k_0} \qquad \mu_g \succeq_{\overline{s_{i,g}}^l} m_g \qquad \sigma_g \succeq_{\overline{s_{i,g}}^l} s$$

where $\phi' = E_g \cdot \Gamma_g^{-1}$. Now we prove:

1. $\Delta \succeq_{\overline{s_i}^n, k_0, \phi} \delta$. Let $i \in \{0 \dots k_0\}$. By Definition 4 we get for each $i \in \{1 \dots l\}$
$$|a_i|\ \overline{s_i}^n \geq size(h, E\ a_i) = size(h, E_g\ y_i) = s_{i,g} \tag{3}$$
which is always positive, i.e. $|a_i|\ \overline{s_i}^n \neq -\infty$. So, by the definition of $\Delta$:
$$\sum_{\phi\ \rho=i} \Delta\ \rho\ \overline{s_i}^n = \sum_{\phi\ \rho=i}\ \sum_{\theta\ \rho'=\rho} \Delta_g\ \rho'\ \overline{|a_i|\ \overline{s_i}^n}^l$$
where $\theta = unify\ \Gamma\ g\ \overline{r'_j}^q$.
Because of the monotonicity of $\Delta_g\ \rho$ for every $\rho$:
$$\sum_{\phi\ \rho=i} \Delta\ \rho\ \overline{s_i}^n \geq \sum_{\phi\ \rho=i}\ \sum_{\theta\ \rho'=\rho} \Delta_g\ \rho'\ \overline{s_{i,g}}^l = \sum_{(\phi\cdot\theta)\ \rho'=i} \Delta_g\ \rho'\ \overline{s_{i,g}}^l$$
By definition of $\Delta_g \succeq_{h,k_0,(\phi\cdot\theta)} \delta_g|_{k_0}$ and because of the fact that $i \neq k_0 + 1$, we can get the desired result:
$$\sum_{\phi\ \rho=i} \Delta\ \rho\ \overline{s_i}^n \geq \delta_g|_{k_0}\ i = \delta\ i$$

provided the involved region instantiation $\phi \cdot \theta = \phi'$. For each $j \in \{1..q\}$:

$$\begin{array}{rcll} \phi\ (\theta\ \rho_j) & = & E\ (\Gamma^{-1}\ (\Gamma\ r'_j)) & \{\text{by definition of } \phi \text{ and } \theta\} \\ & = & E\ r'_j & \\ & = & E_g\ r''_j & \{\text{by definition of } E_g\} \\ & = & E_g\ (\Gamma_g^{-1}\ \rho_j) & \{\text{by definition of } \Gamma_g\} \end{array}$$

2. $\mu \succeq m$. We get:

$$\begin{array}{rcll} \mu\ \overline{s_i}^n & = & \mu_g\ \overline{|a_i|\ \overline{s_i}^n}^l & \\ & \geq & \mu_g\ \overline{s_{i,g}}^l & \{\text{because of (3) and monotonicity of } \mu_g\} \\ & \geq & m_g & \{\text{since } \mu_g \succeq_{\overline{s_{i,g}}^l} m_g\} \\ & = & m & \end{array}$$

3. $\sigma \succeq s$. Similarly, for $\sigma_g$ being monotonic:

$$\begin{array}{rcl} \sigma\ \overline{s_i}^n & = & \sqcup\ \{l+q, \sigma_g\ \overline{(|a_i|\ \overline{s_j}^n)}^l - td + l + q\} \\ & \geq & \sqcup\ \{l+q, \sigma_g\ \overline{s_{j,g}}^l - td + l + q\} \\ & \geq & \sqcup\ \{l+q, \sigma_g\ s_1 - td + l + q\} \\ & = & s \end{array}$$

□

In order to prove the correctness of the algorithms shown in the following section for recursive functions we need the abstract interpretation to be monotonic with respect to function signatures.

**Lemma 2.** *Let $f$ be a context function. Given $\Sigma_1, \Sigma_2$, $\Gamma$, and $td$ such that $\Sigma_1 \sqsubseteq \Sigma_2$, then $[\![e]\!]\ \Sigma_1\ \Gamma\ td \sqsubseteq [\![e]\!]\ \Sigma_2\ \Gamma\ td$.*

*Proof.* By structural induction on $e$, because $+$ and $\sqcup$ are monotonic. □

## 6. Space Inference Algorithms

Given a recursive function $f$ with $n+m$ arguments, the algorithms for inferring $\Delta_f$ and $\sigma_f$ do not depend on each other, while the algorithm for inferring $\mu_f$ needs a correct value for $\Delta_f$. We will assume that $\mu_f$, $\sigma_f$, and the cost functions in $\Delta_f$, do only depend on arguments of $f$ non-increasing in size. The consequence of this restriction is that the costs charged to regions, or to the stack, by the most external call to $f$ are safe upper bounds to the costs charged by all the lower level internal calls. This restriction holds for the majority of programs occurring in the literature. Of course, it is always possible to design an example where the charges grow as we progress towards the leafs of the call-tree.

We assume that, for every recursive function $f$, there has been an analysis giving the following information as functions of the argument sizes $\overline{x_i}^n$:

1. $nr_f$, an upper bound to the number of calls to $f$ invoking $f$ again. It corresponds to the internal nodes of $f$'s call tree.

2. $nb_f$, an upper bound to the number of *basic* calls to $f$. It corresponds to the leaves of $f$'s call tree.

$$
\begin{array}{l}
\mathit{splitExp}_f\ e = (e, \#) \qquad \text{if } e = c, x, C\ \overline{a_i}^n\ @\ r, a_1 \oplus a_2, \text{ or } g\ \overline{a_i}^n\ @\ \overline{r_j}^m \text{ with } g \neq f \\
\mathit{splitExp}_f\ (f\ \overline{a_i}^n\ @\ \overline{r_j}^m) = (\#, f\ \overline{a_i}^n\ @\ \overline{r_j}^m) \\
\mathit{splitExp}_f\ (\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2) = (e_b, e_r) \\
\quad \textbf{where} \quad (e_{1b}, e_{1r}) = \mathit{splitExp}_f\ e_1 \\
\qquad\qquad (e_{2b}, e_{2r}) = \mathit{splitExp}_f\ e_2 \\
\qquad\qquad e_b = \begin{cases} \# & \text{if } e_{1b} = \# \text{ or } e_{2b} = \# \\ \textbf{let}\ x_1 = e_{1b}\ \textbf{in}\ e_{2b} & \text{otherwise} \end{cases} \\
\qquad\qquad e_r = \begin{cases} \# & \text{if } e_{1r} = \# \text{ and } e_{2r} = \# \\ \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2r} & \text{if } e_{1r} = \# \text{ and } e_{2r} \neq \# \\ \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2 & \text{if } e_{1r} \neq \# \text{ and } e_{2r} = \# \\ & \text{or } e_{1r} \neq \# \text{ and } e_{2r} \neq \# \text{ and } e_{1b} = \# \\ \sqcup \left\{ \begin{array}{l} \textbf{let}\ x_1 = e_{1b}\ \textbf{in}\ e_{2r}, \\ \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2 \end{array} \right\} & \text{otherwise} \end{cases} \\
\mathit{splitExp}_f\ (\textbf{case}\ x\ \textbf{of}\ \overline{alt_i}^n) = (e_b, e_r) \\
\quad \textbf{where} \quad (\overline{alt_{ib}}^n, \overline{alt_{ir}}^n) = \mathit{unzip}\ (\mathit{map}\ \mathit{splitAlt}_f\ \overline{alt_i}^n) \\
\qquad\qquad \overline{falt_{ib}}^m = \mathit{filter}(\neq \#)\ \overline{alt_{ib}}^n \\
\qquad\qquad \overline{falt_{ir}}^l = \mathit{filter}(\neq \#)\ \overline{alt_{ir}}^n \\
\qquad\qquad e_b = \begin{cases} \# & \text{if } \overline{falt_{ib}} = [\,] \\ \textbf{case}\ x\ \textbf{of}\ \overline{falt_{ib}}^m & \text{otherwise} \end{cases} \\
\qquad\qquad e_r = \begin{cases} \# & \text{if } \overline{falt_{ir}} = [\,] \\ \textbf{case}\ x\ \textbf{of}\ \overline{falt_{ir}}^l & \text{otherwise} \end{cases} \\
\mathit{splitAlt}_f\ (C\ \overline{x_j}^n \to e) = (alt_b, alt_r) \\
\quad \textbf{where} \quad (e_b, e_r) = \mathit{splitExp}_f\ e \\
\qquad\qquad alt_b = \begin{cases} \# & \text{if } e_b = \# \\ C\ \overline{x_j}^n \to e_b & \text{otherwise} \end{cases} \\
\qquad\qquad alt_r = \begin{cases} \# & \text{if } e_r = \# \\ C\ \overline{x_j}^n \to e_r & \text{otherwise} \end{cases}
\end{array}
$$

Figure 4: Function splitting a *Core-Safe* expression into its base and recursive cases

3. $len_f$, an upper bound to the maximum length of $f$'s call chains. It corresponds to the height of $f$'s call tree.

In general, these functions are not independent of each other. For instance, with linear recursion we have $nr_f = len_f - 1$ and $nb_f = 1$. However, we will not assume a fixed relation between them. If this relation exists, it has been already used to compute them. We will only assume that each function is a correct upper bound to its corresponding runtime figure. As a running example, let us consider the `splitAt` definition in Fig. 6(a). We would assume $nr_{\texttt{splitAt}} = \lambda n\ x.\min\{n, x-1\}$, $nb_{\texttt{splitAt}} = \lambda n\ x.1$ and $len_{\texttt{splitAt}} = \lambda n\ x.\min\{n+1, x\}$.

### *6.1. Splitting Core-Safe expressions*

In order to do a more precise analysis, we separately analyse the base and the recursive cases of a *Core-Safe* function definition. Fig. 4 describes the functions $\mathit{splitExp}_f$ and $\mathit{splitAlt}_f$ which, given a *Safe* expression $e$ return the part of its body contributing to the base cases $e_b$ and the part contributing to the recursive cases $e_r$. We introduce an expression # to represent that the recursive or the base case of an expression is empty. They cannot be simultaneously empty. There is no rule in the operational semantics for an empty expression. However it is convenient to define its abstract interpretation as:

$$[\![\#]\!]\ \Sigma\ \Gamma\ td = ([\,]_f, 0, 0) \quad [\mathit{Empty}]$$

$$
\begin{array}{l}
\mathit{splitBA}_f\ e = [\,] \qquad \text{if } e = \#, c, x, C\ \overline{a_i}^n\ @\ r, a_1 \oplus a_2 \text{ or } g\ \overline{a_i}^n\ @\ \overline{r_j}^m \text{ with } g \neq f \\
\mathit{splitBA}_f\ (\sqcup_{i=1}^n e_i) = \mathit{concat}\left[\mathit{splitBA}_f\ e_i \mid i \in \{1 \ldots n\}\right] \\
\mathit{splitBA}_f\ (f\ \overline{a_i}^n\ @\ \overline{r_j}^m) = [(f\ \overline{a_i}^n\ @\ \overline{r_j}^m, \Box)] \\
\mathit{splitBA}_f\ (\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2) = A +\!\!+ B \\
\qquad \textbf{where} \quad (e_{1b}, e_{1r}) = \mathit{splitExp}_f\ e_1 \\
\qquad\qquad (e_{2b}, e_{2r}) = \mathit{splitExp}_f\ e_2 \\
\qquad\qquad es_{1r} = \mathit{splitBA}_f\ e_{1r} \\
\qquad\qquad es_{2r} = \mathit{splitBA}_f\ e_{2r} \\
\qquad\qquad A = [(\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2rb}, \\
\qquad\qquad\qquad\qquad \mathit{collapse}\ (\textbf{let}\ x_1 = \Box\ \textbf{in}\ e_{2ra})) \mid (e_{2rb}, e_{2ra}) \ \in\ es_{2r}] \\
\qquad\qquad B = \begin{cases} [\,] & \text{if } e_{2b} = \# \\ [(\textbf{let}\ x_1 = e_{1rb}\ \textbf{in}\ \Box, \\ \qquad \textbf{let}\ x_1 = e_{1ra}\ \textbf{in}\ e_{2b}) \mid (e_{1rb}, e_{1ra}) \ \in\ es_{1r}] & \text{otherwise} \end{cases} \\
\mathit{splitBA}_f\ (\textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n) = \\
\qquad\qquad \left[\left(\textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{ibj}}^{m_i}}^n, \mathit{collapse}\ (\textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{iaj}}^{m_i}}^n)\right)\right] \\
\textbf{where}\ \ \overline{(e_{ibj}, e_{iaj})}^{m_i} = \mathit{splitBA}_f\ e_i
\end{array}
$$

$$
\mathit{collapse}\ e = \begin{cases} \Box & \text{if } e \in \{\textbf{let}\ x_1 = \Box\ \textbf{in}\ \Box,\ \textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to \bigvee \overline{\Box}^{m_i}}^n\} \\ e & \text{otherwise} \end{cases}
$$

Figure 5: Function splitting a *Core-Safe* expression into its parts executing before and after the last direct recursive call

Since it might be not possible to split a expression into a single pair with the base and recursive cases, we introduce expressions of the form $\sqcup\, e_i$ to express different possibilities. The following non-deterministic rule captures this intuition:

$$\frac{\exists j\ .\ E \vdash h, k, td, e_j \Downarrow h', k, v, (\delta, m, s)}{E \vdash h, k, td, \sqcup_i e_i \Downarrow h', k, v, (\delta, m, s)}\ [Lub]$$

Its abstract interpretation is the least upper bound of the interpretations of the $e_i$:

$$[\![\sqcup\, e_i]\!]\ \Sigma\ \Gamma\ td = \sqcup [\![e_i]\!]\ \Sigma\ \Gamma\ td \quad [Lub]$$

It will also be useful to define another function which splits a *Core-Safe* expression into those parts that execute before and including the last direct recursive call, and those executed after the last direct recursive call, In Fig. 5 we define such function, called $\mathit{splitBA}_f$. There we use $\Box$ in order not to lose the structure of the original expression. We use *es* to denote a list of expressions.

Function $\mathit{splitBA}_f$ is applied to the recursive part of an expression, denoted as $e_r$, which is returned by function $\mathit{splitExp}_f$. This means that $e_r$ may be any Core-Safe normal expression, # or a least upper bound of expressions. So, if $e_f$ is $f$'s body, in the following we will assume $(e_r, e_b) = \mathit{splitExp}_f\ e_f$ and $(e_{bef}, e_{aft}) = (\sqcup_i\ e^i_{bef}, \sqcup_i\ e^i_{aft})$, where $\overline{(e^i_{bef}, e^i_{aft})}^n = \mathit{splitBA}_f\ e_r$.

In the *case* expression we use $\bigvee$ to denote an ordered least upper bound. This is used not to lose the correspondence between the before and after components of each pair. In the pair semantics the r-th component is selected in each list to be executed.

It is easy to see that given an expression $e_r$, the number of pairs in $\mathit{splitBA}_f\ e_r$ is proportional to the size of $e_r$. Only $\sqcup$ of expressions and **let** expressions may generate more than one pair. In the first case the pairs of

```
splitAt 0 xs     = ([],xs)
splitAt n []     = ([],[])
splitAt n (x:xs) = (x:xs1,xs2)
   where (xs1,xs2) = split (n-1) xs
```

(a) *Full-Safe* version

```
splitAt n xs @ r1 r2 r3 =
 case n of
  _ -> case xs of
       (: y1 y2) ->
          let y3 = let x6 = - n 1 in
                   splitAt x6 y2 @ r1 r2 r3 in #
```

(b) *Core-Safe* up to the last call

```
splitAt n xs @ r1 r2 r3 =
 case n of
  _ -> case xs of
       (: y1 y2) ->
          let y3  = let x6 = - n 1 in
                    splitAt x6 y2 @ r1 r2 r3 in
          let xs1 = case y3 of (y4,y5) -> y4 in
          let xs2 = case y3 of (y6,y7) -> y7 in
          let x7  = (: y1 xs1) @ r2 in
          let x8  = (x7,xs2) @ r3 in x8
```

(d) *Core-Safe* recursive cases

```
splitAt n xs @ r1 r2 r3 =
 case n of
  0 -> let  x1 = [] @ r2 in
       let  x2 = (x1,xs) @ r3 in x2
  _ -> case xs of
        [] -> let x4 = [] @ r2 in
              let x3 = [] @ r1 in
              let x5 = (x4,x3) @ r3 in x5
```

(c) *Core-Safe* base cases

```
splitAt n xs @ r1 r2 r3 =
 case n of
  _ -> case xs of
       (: y1 y2) ->
          let y3  = # in
          let xs1 = case y3 of (y4,y5) -> y4 in
          let xs2 = case y3 of (y6,y7) -> y7 in
          let x7  = (: y1 xs1) @ r2 in
          let x8  = (x7,xs2) @ r3 in x8
```

(e) *Core-Safe* after the last call

Figure 6: Splitting a *Core-Safe* definition

all the expressions are joined. In a **let** $x_1 = e_1$ **in** $e_2$ expression there may be at most as many pairs as those pairs from $e_1$ (set $B$) and $e_2$ (set $A$) together.

In Fig. 6 we show a *Full-Safe* definition for a function `splitAt` splitting a list, and the result of applying $splitExp_f$ and $splitBA_f$ to its *Core-Safe* version.

*6.1.1. Correctness and properties of $splitExp_f$*

In the following lemmas $e$ is a Core-safe expression, not containing # nor $\sqcup$. If $(e_b, e_r) = splitExp_f\ e$ , from the definition we can easily see that:

- $e_b$ may be # or a Core-safe expression not containing #,
- $e_r$ may be # or a Core-safe expression containing $\sqcup$ but not #.

**Lemma 3.** *Let $(e_b, e_r) = splitExp_f\ e$. Then, $e_b = \#$ implies $e_r \neq \#$ (and so $e_r = \#$ implies $e_b \neq \#$).*

*Proof.* By structural induction on $e$. □

**Lemma 4.** *Let $(e_b, e_r) = splitExp_f\ e$. Then, $e_b \neq \#$ and $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta, m, s)$ if and only if $E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s)$ such that there are no calls to $f$ in this derivation.*

*Proof.* Both implications can be proved by induction on the depth of the $\Downarrow$-derivation, see Appendix A. □

**Lemma 5.** *Let $(e_b, e_r) = splitExp_f\ e$. Then, $e_r \neq \#$ and $E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s)$ such that there is at least one direct call to $f$ in this derivation if and only if $E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s)$ such that there is at least one direct call to $f$ in this derivation.*

*Proof.* Both implications can be proved by induction on the depth of the $\Downarrow$-derivation. We show here only one of the cases, the rest can be found at Appendix A.

- Case **let**

  (⇐) Let $e = \textbf{let } x_1 = e_1 \textbf{ in } e_2$. By the operational semantics, we get:

$$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1) \tag{4}$$

  and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h, k, v, (\delta_2, m_2, s_2) \tag{5}$$

  with $\delta = \delta_1 + \delta_2$, $m = \max\{m_1, |\delta_1| + m_2\}$ and $s = \max\{2 + s_1, 1 + s_2\}$. Let $(e_{1b}, e_{1r}) = \mathit{splitExp}_f\ e_1$ and $(e_{2b}, e_{2r}) = \mathit{splitExp}_f\ e_2$. We know that in the derivations of either (4), or (5), or both, there are direct calls to $f$. Let us distinguish these three cases:

  1. There are calls in (4). By the induction hypothesis we get $e_{1r} \neq \#$ and:

$$E \vdash h, k, 0, e_{1r} \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

     with at least a direct call to $f$. As $e_{1r}$ is non-empty, $\mathit{splitExp}_f\ e$ gives either $e_r = \textbf{let } x_1 = e_{1r} \textbf{ in } e_2$ or:

$$e_r = \sqcup\{\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2r},\ \textbf{let } x_1 = e_{1r} \textbf{ in } e_2\}$$

     In both cases we get $e_r \neq \#$ and $E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s)$ with at least a direct call to $f$.
  2. There are calls in (5) but not in (4). By the induction hypothesis $e_{2r} \neq \#$ . The reasoning is symmetrical to the previous case.

  (⇒) Let $e_r = \textbf{let } x_1 = e_{1r} \textbf{ in } e_{2r}$. As $e_r \neq \#$, we have to distinguish two cases.

  $e_{1r} = \#$, $e_{2r} \neq \#$ In this case $e_{1r} = e_1$ and $(\_, e_{2r}) = \mathit{splitExp}_f\ e_2$. By hypothesis on $e_1$ and induction hypothesis on $e_{2r}$ we prove this implication in a similar way to (⇐).

  $e_{1r} \neq \#$, $e_{2r} = \#$ In this case $e_{2r} = e_2$ and $(\_, e_{1r}) = \mathit{splitExp}_f\ e_1$. The reasoning is symmetrical to the previous case.

□

Function $\mathit{splitExp}_f$ is idempotent. We need to define

$$\mathit{splitExp}_f\ (\sqcup\{e_1, e_2\}) = \sqcup\{\mathit{splitExp}_f\ e_1, \mathit{splitExp}_f\ e_2\}$$

**Lemma 6.** *For any expression $e$, if $\mathit{splitExp}_f\ e = (e_b, e_r)$, then the following properties hold:*

1. *If $e_b \neq \#$ then $\mathit{splitExp}_f\ e_b = (e_b, \#)$.*
2. *If $e_r \neq \#$ then $\mathit{splitExp}_f\ e_r = (\#, e_r)$*

*Proof.* By structural induction on $e$, see Appendix A. □

In the following, if $[\![e]\!]\ \Sigma\ \Gamma\ td = (\Delta, \mu, \sigma)$ we will use $[\![e]\!]_\Delta\ \Sigma\ \Gamma\ td$, $[\![e]\!]_\mu\ \Sigma\ \Gamma\ td$ and $[\![e]\!]_\sigma\ \Sigma\ \Gamma\ td$ to denote the corresponding individual component of the result of the abstract interpretation.

**Lemma 7.** *For any expression $e$, and for any $\Sigma$, $\Gamma$ and $td$, if $splitExp_f\ e = (e_b, e_r)$, then*

$$\begin{array}{rcl} [\![e]\!]_\Delta\ \Sigma\ \Gamma\ td & = & ([\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td) \\ [\![e]\!]_\sigma\ \Sigma\ \Gamma\ td & = & ([\![e_b]\!]_\sigma\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\sigma\ \Sigma\ \Gamma\ td) \end{array}$$

*Proof.* By structural induction on $e$. We show here a relevant case, for the rest of the proof see Appendix A.

We show the case when $e_{1b} \neq \#$, $e_{2b} \neq \#$, $e_{1r} \neq \#$ and $e_{2r} \neq \#$. In this case $[\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td = ([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ td) + ([\![e_{2b}]\!]_\Delta\ \Sigma\ \Gamma\ td)$ and $e_r = \sqcup\{\mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2r}, \mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2\}$.

$$\begin{array}{ll} & ([\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td) \\ = & \sqcup\{([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_{2b}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)), \\ & \quad \sqcup\{([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)), ([\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1))\}\} \\ = & \sqcup\{([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + \sqcup\{[\![e_{2b}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1), [\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)\}, \\ & \quad ([\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1))\} \\ = & \sqcup\{([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)), \\ & \quad ([\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1))\} & \{\text{by i.h.}\} \\ = & \sqcup\{[\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0, [\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ 0\} + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) \\ = & ([\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) & \{\text{by i.h.}\} \\ = & [\![e]\!]_\Delta\ \Sigma\ \Gamma\ td \end{array}$$

□

*6.1.2. Correctness of $splitBA_f$*

The $splitBA_f$ function, given the recursive part of an expression, returns a list of $n$ pairs $\overline{(e_b, e_a)}^n$. Before showing the equivalence between these pairs and the original expression we have to give a meaning to the execution of these pairs. For this reason we shall extend the semantics in Figure 1 in order to consider the execution of pairs $(e_b, e_a)$ as follows: both expressions are evaluated simultaneously until one of the components is $\square$. From this point, we can evaluate the nonempty component under the semantics of Figure 1.

Another key aspect to the correctness of $splitBA_f$ is the fact that every $e_b$ expression is executed *before* its $e_a$, that is, there is no interleaving between $e_b$ and $e_a$. We shall formalize this property by *labelling* each pair being executed. Labels are elements from a set of marks $\mathcal{L} = \{\mathbf{B}, \mathbf{M}, \mathbf{A}\}$, whose intended meanings are as follows:

- **A** specifies that only *after* parts of the pair are executed.
- **B** specifies that only *before* parts of the pair are executed.
- **M** specifies that the execution of the pair contains both *before* and *after* parts, and in this execution the former will be finished and the latter started.

Therefore we shall define the execution of labelled pairs $(e_b, e_a)_X$ with $X \in \mathcal{L}$. Figure 7 shows the semantic rules for pairs. Rules [*Before*] and [*After*] control the execution when one of the pair components is $\square$. In rule *LetP* we have to ensure that the *before* part is actually executed before the *after* part. Thus we assume that there is an order relation between the elements of $\mathcal{L}$ as follows: $\mathbf{B} < \mathbf{M} < \mathbf{A}$.

**Lemma 8.** *Given an expression $e$, let $e_r = \pi_2(splitExp_f\ e)$ and let $(e_b, e_a) \in splitBA_f\ e_r$. Then:*

- $e_b \neq \square$.

$$\frac{E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta, m, s)}{E \vdash h, k, td, (e_b, \square)_{\mathbf{B}} \Downarrow h', k, v, (\delta/0, m/0, s/0)}\ [Before]$$

$$\frac{E \vdash h, k, td, e_a \Downarrow h', k, v, (\delta, m, s)}{E \vdash h, k, td, (\square, e_a)_{\mathbf{A}} \Downarrow h', k, v, (0/\delta, 0/m, 0/s)}\ [After]$$

$$\frac{\begin{array}{c} E \vdash h, k, 0, (e_{1b}, e_{1a})_X \Downarrow h_1, k, v_1, (\delta_{1b}/\delta_{1a}, m_{1b}/m_{1a}, s_{1b}/s_{1a}) \\ E \cup [x_1 \mapsto v_1] \vdash h_1, k, td+1, (e_{2b}, e_{2a})_Y \Downarrow h', k, v, (\delta_{2b}/\delta_{2a}, m_{2b}/m_{2a}, s_{2b}/s_{2a}) \\ X < Y \quad s_b = \max\{2 + s_{1b}, 1 + s_{2b}\} \quad s_a = \max\{2 + s_{1a}, 1 + s_{2a}\} \\ \delta_b = \delta_{1b} + \delta_{2b} \quad \delta_a = \delta_{1a} + \delta_{2a} \\ m_b = \max\{m_{1b}, |\delta_{1b}| + m_{2b}\} \quad m_a = \max\{m_{1a}, |\delta_{1a}| + m_{2a}\} \end{array}}{E \vdash h, k, td, \left( \begin{array}{l} \textbf{let } x_1 = e_{1b} \textbf{ in } e_{2b}, \\ \textbf{let } x_1 = e_{1a} \textbf{ in } e_{2a} \end{array} \right)_{\mathbf{M}} \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)}\ [LetP]$$

$$\frac{l \in \{1..m_r\} \quad E \cup [\overline{x_i \mapsto v_i}^{n_r}] \vdash h, k, td + n_r, (e_{rbl}, e_{ral})_X \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)}{E[x \mapsto p] \vdash h[p \mapsto (j, C_r\ \overline{v_i}^{n_r})], k, td, \left( \begin{array}{l} \textbf{case } x \textbf{ of } \overline{C_i\ \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{ibj}}^{m_i}}^{n}, \\ \textbf{case } x \textbf{ of } \overline{C_i\ \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{iaj}}^{m_i}}^{n} \end{array} \right)_{\mathbf{M}} \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b + n_r/s_a + n_r)}\ [CaseP]$$

Figure 7: Semantic rules for labelled pairs $(e_b, e_a)$

- *If* $e_a = \square$ *then*

$$E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b) \Rightarrow E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta_b, m_b, s_b)$$

*Proof.* By structural induction on $e$, see Appendix A. □

**Lemma 9.** *Given* $(e_b, e_a) \in splitBA_f\ e_r$ *such that* $E \vdash h, k, td, (e_b, e_a)_X \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$*, then*

- $X \neq \boldsymbol{A}$
- $X = \boldsymbol{B} \Rightarrow e_a = \square \wedge \delta_a = m_a = s_a = 0 \wedge E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b)$
- $X = \boldsymbol{M} \Rightarrow e_a \neq \square$

*Proof.* By inspection of the pair semantics rules, see Appendix A. □

We now prove that this pair semantics is equivalent to the operational standard semantics. When reasoning by induction on the depth of a derivation for a labelled pair, we only consider the rules for pairs, but not those corresponding to the standard semantics. From that point of view rules [*Before*] and [*After*] are axioms.

**Theorem 2.** *Let* $e_r, E, h, h', k, td, v, X \in \{\boldsymbol{B}, \boldsymbol{M}\}, (e_b, e_a) \in splitBA_f\ e_r, \delta_b, \delta_a, m_b, m_a, s_b, s_a$ *such that*

$$E \vdash h, k, td, (e_b, e_a)_X \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$$

*Then,*

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s)$$

*where* (1) $\delta = \delta_b + \delta_a$, (2) $m = \max\{m_b, |\delta_b| + m_a\}$, (3) $s = \max\{s_b, s_a\}$

*Proof.* By induction on the depth $n$ of the derivation for $(e_b, e_a)_X$ and by cases on the last rule applied, see Appendix A. □

**Theorem 3.** *Let* $e_r, E, h, h', k, td, v, \delta, m, s$ *such that*

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s) \quad (*)$$

*If there are direct calls to* $f$ *in derivation (*) then there exist* $X \in \{\boldsymbol{B}, \boldsymbol{M}\}$, $(e_b, e_a) \in splitBA_f\ e_r$, $\delta_b$, $\delta_a$, $m_b$, $m_a$, $s_b$, $s_a$ *such that*

$$E \vdash h, k, td, (e_b, e_a)_X \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$$

*and* (1) $\delta = \delta_b + \delta_a$, (2) $m = \max\{m_b, |\delta_b| + m_a\}$, (3) $s = \max\{s_b, s_a\}$.

*Proof.* By induction on the depth $n$ of the derivation for $e_r$ and by cases on the last rule applied. We show here only one case, the rest of the proof can be found at Appendix A.

$n > 0, [Let]$ Now $e_r =$ **let** $x_1 = e_1$ **in** $e_2$:

$$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1) \tag{6}$$

and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h', k, v, (\delta_2, m_2, s_2) \tag{7}$$

where $\delta = \delta_1 + \delta_2$, $m = \max\{m_1, |\delta_1| + m_2\}$ and $s = \max\{2 + s_1, 1 + s_2\}$. We distinguish two cases:

- There are no direct calls to $f$ in the derivation 7. As there is at least a direct call to $f$ in the derivation of $e_r$, then it necessarily appears in the derivation (6). Consequently, by Lemma 5, $e_{1r} = \pi_2(splitExp_f\ e_1) \neq \#$ and

  $$E \vdash h, k, 0, e_{1r} \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

  containing a direct call to $f$.
  By induction hypothesis, there exist $Z \in \{\mathbf{B}, \mathbf{M}\}$, $(e_{1rb}, e_{1ra}) \in splitBA_f\ e_{1r}$, $\delta_{1b}$, $\delta_{1a}$, $m_{1b}$, $m_{1a}$, $s_{1b}$, $s_{1a}$ such that

  $$E \vdash h, k, 0, (e_{1rb}, e_{1ra})_Z \Downarrow h_1, k, v_1, (\delta_{1b}/\delta_{1a}, m_{1b}/m_{1a}, s_{1b}/s_{1a}) \tag{8}$$

  where $\delta_1 = \delta_{1b} + \delta_{1a}$, $m_1 = \max\{m_{1b}, |\delta_{1b}| + m_{1a}\}$ and $s_1 = \max\{s_{1b}, s_{1a}\}$.
  By Lemma 4, $e_{2b} = \pi_1(splitExp_f\ e_2) \neq \#$ and

  $$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2b} \Downarrow h', k, v, (\delta_2, m_2, s_2)$$

  and by rule $[After]$

  $$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, (\Box, e_{2b})_{\mathbf{A}} \Downarrow h', k, v, (0/\delta_2, 0/m_2, 0/s_2) \tag{9}$$

  As $Z < \mathbf{A}$, by applying rule $[LetP]$ to (8) and (9) we obtain for pair $p =$ (**let** $x_1 = e_{1r,b}$ **in** $\Box$, **let** $x_1 = e_{1r,a}$ **in** $e_{2b}$) $\in splitBA_f\ e_r$:

  $$E \vdash h, k, td, p_{\mathbf{M}} \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$$

  where $\delta_b = \delta_{1b}$, $\delta_a = \delta_{1a} + \delta_2$, $m_b = \max\{m_{1b}, |\delta_{1b}|\}$, $m_a = \max\{m_{1a}, |\delta_{1a}| + m_2\}$, $s_b = \max\{2 + s_{1b}, 1\} = 2 + s_{1b}$ and $s_a = \max\{2 + s_{1a}, 1 + s_2\}$. Then:

  $$\begin{aligned} \delta_b + \delta_a &= \delta_{1b} + \delta_{1a} + \delta_2 \\ &= \delta_1 + \delta_2 \\ &= \delta \end{aligned}$$

Also

$$\begin{array}{rcl}\max\{m_b, |\delta_b| + m_a\} & = & \max\{\max\{m_{1b}, |\delta_{1b}|\}, |\delta_{1b}| + \max\{m_{1a}, |\delta_{1a}| + m_2\}\} \\ & = & \max\{m_{1b}, |\delta_{1b}|, |\delta_{1b}| + m_{1a}, |\delta_{1b}| + |\delta_{1a}| + m_2\} \\ & = & \max\{\max\{m_{1b}, |\delta_{1b}| + m_{1a}\}, |\delta_1| + m_2\} \\ & = & \max\{m_1, |\delta_1| + m_2\} \\ & = & m\end{array}$$

and

$$\begin{array}{rcl}\max\{s_b, s_a\} & = & \max\{2 + s_{1b}, \max\{2 + s_{1a}, 1 + s_2\}\} \\ & = & \max\{2 + \max\{s_{1b}, s_{1a}\}, 1 + s_2\} \\ & = & \max\{2 + s_1, 1 + s_2\} \\ & = & s\end{array}$$

- There is a direct call to $f$ in the derivation (7). By Lemma 5, $e_{2r} = \pi_2(\mathit{splitExp}_f\ e_2) \neq \#$ and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2r} \Downarrow h', k, v, (\delta_2, m_2, s_2)$$

containing a direct call to $f$. By induction hypothesis, there exist $Z \in \{\mathbf{B}, \mathbf{M}\}$, $(e_{2rb}, e_{2ra}) \in \mathit{splitBA}_f\ e_{2r}$, $\delta_{2b}$, $\delta_{2a}$, $m_{2b}$, $m_{2a}$, $s_{2b}$ and $s_{2a}$ such that

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, (e_{2rb}, e_{2ra})_Z \Downarrow h', k, v, (\delta_{2b}/\delta_{2a}, m_{2b}/m_{2a}, s_{2b}/s_{2a}) \tag{10}$$

where $\delta_2 = \delta_{2b} + \delta_{2a}$, $m_2 = \max\{m_{2b}, |\delta_{2b}| + m_{2a}\}$, and $s_2 = \max\{s_{2b}, s_{2a}\}$.
By applying rule [*Before*] to (6) we obtain

$$E \vdash h, k, 0, (e_1, \Box)_{\mathbf{B}} \Downarrow h_1, k, v_1, (\delta_1/0, m_1/0, s_1/0) \tag{11}$$

We distinguish two cases:

$Z = \mathbf{M}$ By Lemma 9, $e_{2ra} \neq \Box$. The pair we are looking for is $(\mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_{2rb}, \mathbf{let}\ x_1 = \Box\ \mathbf{in}\ e_{2ra})_{\mathbf{M}} \in \mathit{splitBA}_f\ e_r$. By applying rule [*LetP*] to (11) and (10) we obtain $\delta_b = \delta_1 + \delta_{2b}$, $\delta_a = \delta_{2a}$, $m_b = \max\{m_1, |\delta_1| + m_{2b}\}$, $m_a = m_{2a}$, $s_b = \max\{2 + s_1, 1 + s_{2b}\}$, $s_a = \max\{2, 1 + s_{2a}\}$.

$Z = \mathbf{B}$ By Lemma 9, then $e_{2ra} = \Box$, $\delta_{2a} = m_{2a} = s_{2a} = 0$ and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2rb} \Downarrow h', k, v, (\delta_{2b}, m_{2b}, s_{2b}) \tag{12}$$

The pair we are looking for is $(\mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_{2rb}, \Box)_{\mathbf{B}} \in \mathit{splitBA}_f\ e_r$. By applying rule [*Let*] to (6) and (12), and then rule [*Before*], we obtain $\delta_b = \delta_1 + \delta_{2b}$, $\delta_a = 0$, $m_b = \max\{m_1, |\delta_1| + m_{2b}\}$, $m_a = 0$, $s_b = \max\{2 + s_1, 1 + s_{2b}\}$, $s_a = 0$.

This cases distinction is reflected by function *collapse* in the definition of $\mathit{splitBA}_f$. In both cases it is easy to prove that $\delta = \delta_b + \delta_a$, $m = \max\{m_b, |\delta_b| + m_a\}$ and $s = \max\{s_b, s_a\}$.

□

*6.2. Algorithm for computing $\Delta_f$*

The idea here is to separately compute the charges to regions of the recursive and non-recursive parts of $f$'s body, and then multiply these charges by respectively the number of internal and leaf nodes of $f$'s call-tree. Charges to the region *self* (of type $\rho_{self}$) have to be excluded from the result, since they are not taken into account when calling $f$ from another function. We use $\lfloor\Delta\rfloor$ to denote the restriction of $\Delta$ to $dom(\Delta) - \{\rho_{self}\}$.

1. Set $\Sigma\ f = ([\,]_f, 0, 0)$.
2. Let $\Delta_r = [\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ (n+m)$
3. Let $\Delta_b = [\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ (n+m)$
4. Then, $\Delta_f \stackrel{\text{def}}{=} (\lfloor\Delta_r\rfloor \cdot nr_f + \lfloor\Delta_b\rfloor \cdot nb_f) \sqcup \lfloor\Delta_b\rfloor$.

   We will require that $nr_f \sqsupseteq 0$ and $nb_f \sqsupseteq 1$. So, $\lfloor\Delta_b\rfloor \cdot nb_f \sqsupseteq \lfloor\Delta_b\rfloor$. Then, the expression $\lfloor\Delta_r\rfloor \cdot nr_f + \lfloor\Delta_b\rfloor \cdot nb_f$ will in general dominate $\lfloor\Delta_b\rfloor$ except in those points $\overline{x_i}^n$ where $\lfloor\Delta_r\rfloor\ \rho\ \overline{x_i}^n = -\infty$. This usually happens for small values of $\overline{x_i}^n$ due to the guards in the *App* rule.

Notice that if $nr_f, nb_f$, and all the size functions belong to $\mathbb{F}$, then all functions in $\Delta_f$ belong to $\mathbb{F}$ because it is closed by the operations $\{+, \sqcup, *\}$.

If we apply the abstract interpretation rules for the base cases of our `splitAt` example in Fig. 6(b) we get $\Delta_b = [\rho \mapsto \lambda n\ x.1 \mid \rho \in \{\rho_1, \rho_2, \rho_3\}]$. If we apply them to the recursive case in Fig. 6(d) we get $\Delta_r = [\rho \mapsto \lambda n\ x.1 \mid \rho \in \{\rho_1, \rho_2\}]$. The resulting $\Delta_{\texttt{splitAt}}$ is shown in Fig. 12.

*6.2.1. Correctness*

**Definition 7.** *Two expressions $e$ and $e'$ are equivalent up to memory consumption if for each $E$, $h$, $h'$, $td$, $k$, $v$:*

$$\begin{array}{c} \exists(\delta, m, s)\ E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s) \\ \Updownarrow \\ \exists(\delta', m', s')\ E \vdash h, k, td, e' \Downarrow h', k, v, (\delta', m', s') \end{array}$$

**Lemma 10** (Additivity for $\delta$)**.** *Let us assume, in a heap with $k$ regions, the execution of an expression $e$ with a resource vector $(\delta, m, s)$ under a function signature $\Sigma$, such that there are $p \geq 0$ direct calls to a function $g \in dom\ \Sigma$, each with an associated resource vector $(\delta_i, m_i, s_i)$, $i \in \{1 \ldots p\}$. If $e$ is executed under an different function environment $\Sigma'$, in which $g$ is replaced with an equivalent definition (up to memory consumption), obtaining a resource vector $(\delta', m', s')$, the following equation holds:*

$$\delta' = \delta + \sum_{i=1}^{p} \delta_i'|_k - \sum_{i=1}^{p} \delta_i|_k$$

*where the $\delta_i'$ correspond to the calls to $g$ in the latter execution of $e$.*

*Proof.* By induction on the structure of $e$, see Appendix A. □

**Lemma 11.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ be a typeable function definition under type environment $\Gamma$ such that:*

$$E \vdash h, k+1, n+m, e_f \Downarrow h', k+1, v, (\delta, m, s), (n_t, n_b, l)_f \tag{13}$$

*where $E \equiv_{def} \left[\overline{x_i \mapsto v_i}^n, \overline{r_j \mapsto i_j}^m, self \mapsto k+1\right]$. Then $(\lfloor\Delta_b\rfloor \cdot n_b + \lfloor\Delta_r\rfloor \cdot (n_t - n_b)) \sqcup \lfloor\Delta_b\rfloor \succeq_{\overline{s_i}^n, k, \phi} \delta$, where $\Delta_b$ and $\Delta_r$ are as defined in the algorithm, $s_i = size(h, v_i)$ for each $i \in \{1 \ldots n\}$ and $\phi$ is the region instantiation consistent with $E$ and $\Gamma$.*

*Proof.* By induction on the number of calls $n_t$.

$n_t = 1$ There are no direct recursive calls to $f$ in (A.33) and, by Lemma 4, we can substitute $e_b$ for $e_f$ in this derivation, obtaining the same result and memory consumption. $\Delta_b$ is computed by applying the abstract interpretation rules to $e_b$ and by Theorem 1 we get $\Delta_b \succeq_{\overline{s_i}^n,k,\phi} \delta$, which is equivalent to $\lfloor \Delta_b \rfloor \succeq_{\overline{s_i}^n,k,\phi} \delta$, since $\phi\ \rho^f_{self} = k+1$ and in the definition of $\succeq_{\overline{s_i}^n,k,\phi}$ we only consider regions ranging from 0 to $k$. Then we get:

$$(\lfloor \Delta_b \rfloor \cdot n_b + \lfloor \Delta_r \rfloor \cdot (n_t - n_b)) \sqcup \lfloor \Delta_b \rfloor \sqsupseteq \lfloor \Delta_b \rfloor \succeq_{\overline{s_i}^n,k,\phi} \delta.$$

$n_t > 1$ We can assume there are $p \geq 1$ direct recursive calls to $f$ in (A.33). Let us denote by $(\delta_i, m_i, s_i)$ and $(n_{t,i}, n_{b,i}, l_i)$ the resource vector and number of calls of the $i$-th recursive call.

Firstly we prove that $\Delta_r$ safely approximates the actual $\delta$ without taking the $\delta_i$'s into account. We have to replace the recursive calls in (A.33) with an expression that produces the same result, but with an empty $\delta_i$. Let **rmask** $e$ be a new construct with the following semantics:

$$\frac{E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s)}{E \vdash h, k, td, \textbf{rmask}\ e \Downarrow h', k, v, ([\,], 0, 0)}\ [\text{RMask}]$$

Let $\Sigma'$ be a function environment in which the body $e_f$ of $f$ is replaced with **rmask** $e_f$. Under this $\Sigma'$ we get the following derivation:

$$E \vdash h, k+1, n+m, e_f \Downarrow h', k+1, v, (\delta', m', s'), (n_t, n_b, l)_f \tag{14}$$

where, by Lemma 10:

$$\delta' = \delta - \sum_{i=1}^{p} \delta_i|_k \tag{15}$$

Now we can apply the correctness proof of the abstract interpretation (Theorem 1), since the abstract signature $\Sigma^{\#}[f \mapsto ([\,]_f, 0, 0)]$ is correct with respect to the above mentioned $\Sigma'$ and since (according to Definition 6), we can substitute $e_r$ for $e_f$ in (A.34) by Lemma 5. Hence we get $\Delta_r \succeq_{\overline{s_i}^n,k,\phi} \delta'$, which is equivalent to:

$$\lfloor \Delta_r \rfloor \succeq_{\overline{s_i}^n,k,\phi} \delta' \tag{16}$$

since we are only considering regions from 0 to $k$. Moreover, every region type variable $\rho$ in the domain of $\lfloor \Delta_r \rfloor$ must satisfy $\lfloor \Delta_r \rfloor\ \rho \neq -\infty$, as $\lfloor \Delta_r \rfloor$ is an upper bound to a $\delta'|_k$ in which every charge is strictly greater than $-\infty$. Now we can prove the required result:

$$
\begin{array}{l}
seqs_f :: Exp \to [[Exp]] \\
seqs_f\ e = [[\,]] \qquad\qquad \text{-- if } e \in \{c, x, a_1 \oplus a_2, g\ \overline{a}\ @\ \overline{r}, C\ \overline{a}\} \\
seqs_f\ (f\ \overline{a}\ @\ \overline{r}) = [[f\ \overline{a}\ @\ \overline{r}]] \\
seqs_f\ (\textbf{case of}\ alts) = concat\ [seqs_f\ e \mid (C\ \overline{x} \to e) \in alts] \\
seqs_f\ (\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2) = [s_1 +\!\!+ s_2 \mid s_1 \in seqs_f\ e_1,\ s_2 \in seqs_f\ e_2]
\end{array}
$$

Figure 8: Extracting the sequences of internal calls to $f$ of a *Core-Safe* expression

$$
\begin{array}{lll}
 & (\lfloor\Delta_b\rfloor \cdot n_b + \lfloor\Delta_r\rfloor \cdot (n_t - n_b)) \sqcup \lfloor\Delta_b\rfloor & \\
\sqsupseteq & & \\
 & \lfloor\Delta_b\rfloor \cdot n_b + \lfloor\Delta_r\rfloor \cdot (n_t - n_b) & \\
= & \quad \{\text{by Lemma 1}\} & \\
 & \lfloor\Delta_b\rfloor \cdot \sum_{i=1}^{p} n_{b,i} + \lfloor\Delta_r\rfloor \cdot (1 + \sum_{i=1}^{p} n_{t,i} - \sum_{i=1}^{p} n_{b,i}) & \\
= & & \\
 & \sum_{i=1}^{p} (\lfloor\Delta_b\rfloor \cdot n_{b,i}) + \sum_{i=1}^{p} (\lfloor\Delta_r\rfloor \cdot (n_{t,i} - n_{b,i})) + \lfloor\Delta_r\rfloor & \\
= & & \\
 & \sum_{i=1}^{p} (\lfloor\Delta_b\rfloor \cdot n_{b,i} + \lfloor\Delta_r\rfloor \cdot (n_{t,i} - n_{b,i})) + \lfloor\Delta_r\rfloor & \\
= & \quad \{\text{since } n_{b,i} \geq 1 \text{ and } n_{t,i} \geq n_{b,i} \text{ for every } i \in \{1 \ldots p\}\} & \\
 & \sum_{i=1}^{p} ((\lfloor\Delta_b\rfloor \cdot n_{b,i} + \lfloor\Delta_r\rfloor \cdot (n_{t,i} - n_{b,i})) \sqcup \lfloor\Delta_b\rfloor) + \lfloor\Delta_r\rfloor & \\
\succeq_{\overline{s_i}^n,k,\phi} & \quad \{\text{by i.h.}\} & \\
 & \sum_{i=1}^{p} \delta_i|_k + \lfloor\Delta_r\rfloor & \\
\succeq_{\overline{s_i}^n,k,\phi} & \quad \{\text{by (A.35) and (A.36)}\} & \\
 & \sum_{i=1}^{p} \delta_i|_k + \delta - \sum_{i=1}^{p} \delta_i|_k & \\
= & & \\
 & \delta &
\end{array}
$$

Notice that we can safely substitute $\delta_i$ for $\delta_i|_k$ in each application of the induction hypothesis, since we only take into account regions from 0 to $k$.

□

**Lemma 12.** *$\Delta_f$ is a correct abstract heap for $f$.*

*Proof.* The proof is a direct consequence of $nr_f$, $nb_f$, and all the size functions being upper bounds of their respective runtime figures, and of Lemma 11, see Appendix A for more details. □

*6.2.2. Reductivity of the $\Delta$ interpretation*

Let us call $\mathbb{I}_\Delta : \mathbb{D} \to \mathbb{D}$ to an iteration of the interpretation function, i.e. $\mathbb{I}_\Delta\ \Delta_1 = \Delta_2$, being $\Delta_2$ the abstract heap obtained by initially setting $\Sigma\ f = (\Delta_1, 0, 0)$, then computing $(\Delta, \_, \_) = [\![e_f]\!]\ \Sigma\ \Gamma\ (n+m)$, where $e_f$ is $f$'s body, and then defining $\Delta_2 = \lfloor\Delta\rfloor$. By Theorem 1 we know that, if $\Delta_f$ is a correct upper bound for $f$'s heap charges, then $\mathbb{I}_\Delta\ \Delta_f$ is also a correct one. We have observed that very frequently, but not always, $\mathbb{I}_\Delta\ \Delta_f \sqsubseteq \Delta_f$. By monotonicity of $\mathbb{I}_\Delta$ and transitivity of $\sqsubseteq$ we have $\mathbb{I}_\Delta^n\ \Delta_f \sqsubseteq \Delta_f$ for any $n \geq 0$. This is a good thing because it means that by iterating the interpretation we may achieve tighter upper bounds. We are then interested in finding sufficient conditions under which $\mathbb{I}_\Delta$ is reductive in the lattice $\mathbb{D}$.

In Fig. 8 we show a function $seqs_f$ which, when applied to $f$'s body, produces a list of lists of *Core-Safe* expressions, representing the different paths through $f$'s body which an evaluation may follow. Each internal

list consists of a sequence of zero or more of recursive calls to $f$ which might execute consecutively in a run. Calls belonging to different sequences will never execute together. Let $n_r$ be the number of recursive sequences, and $n_j$ be the number of recursive calls of sequence $j$. Given $1 \leq j \leq n_r,\ 1 \leq i \leq n_j$, $\overline{a_{ji}}$ denotes the arguments of the internal call $i$ of sequence $j$. The following lemma relates the result of the $\Delta$ interpretation for $f$'s body with the assumption $\Delta_f$ made for the internal calls. The lemma does not need to assume that $\Delta_f$ is correct.

**Lemma 13.** *Let $e_f$ be $f$'s body expression, let $\Delta_f \in \mathbb{D}$, and let $d$ be any natural number. Then, it holds that:*

$$([\![e_f]\!]_\Delta\ \Sigma[f \mapsto \Delta_f]\ \Gamma\ d)(\overline{x}) \sqsubseteq \left( \bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \Delta_r(\overline{x}) \right) \sqcup \Delta_b(\overline{x})$$

*where $\Delta_r$ and $\Delta_b$ are defined as in Sec. 6.2, and $\Delta_f^G(\overline{a_{ji}})$ is an abbreviation for the guarded expression $[|\overline{a_{ji}}| \neq -\infty \to \Delta_f\ \ |\overline{a_{ji}}|]$.*

*Proof.* By structural induction on expression $e_f$. □

Let us assume now that $\Delta_f$ is a correct upper bound to function $f$'s heap charges. By Lemma 13, and by deleting $f$'s *self* region, a sufficient condition for $\mathbb{I}_\Delta$ to be reductive at $\Delta_f$ is:

$$\left( \bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \lfloor \Delta_r \rfloor(\overline{x}) \right) \sqcup \lfloor \Delta_b \rfloor(\overline{x}) \sqsubseteq \Delta_f(\overline{x})$$

We will prove that, if $\Delta_f$ is defined as in Sec. 6.2, a sufficient condition for that is some well behaviour of the functions $nr_f$ and $nb_f$.

**Definition 8.** *Let $\overline{a_{ji}},\ 1 \leq j \leq n_r,\ 1 \leq j \leq n_j$ be defined as in Lemma 13. We say that $nr_f$ and $nb_f$ are* well-behaved *in a domain $D$ if for all $\overline{x} \in D$:*

$$\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} nr_f(\overline{|a_{ji}|\ \ \overline{x}}) + 1 \sqsubseteq nr_f(\overline{x}) \quad \text{and} \quad \bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} nb_f(\overline{|a_{ji}|\ \ \overline{x}}) \sqsubseteq nb_f(\overline{x})$$

If $nr_f$ and $nb_f$ were exact bounds to the number of internal and leaf nodes of $f$'s call trees, these properties amount to say that the number of the tree internal nodes is one more that the sum of the number of internal nodes of the subtrees, and that the number of the tree leaf nodes is the sum of the number of leaf nodes of the subtrees.

**Lemma 14.** *If $\Delta_f = (\lfloor \Delta_r \rfloor \cdot nr_f + \lfloor \Delta_b \rfloor \cdot nb_f) \sqcup \lfloor \Delta_b \rfloor$, and $nr_f$ and $nb_f$ are well-behaved in dom $\lfloor \Delta_r \rfloor$, then $\mathbb{I}_\Delta\ \Delta_f \sqsubseteq \Delta_f$.*

*Proof.* We distinguish two cases. In the first place, we show the reductivity for those points in $D = \{\overline{x} \mid \forall ji\,.\, \lfloor \Delta_r^G \rfloor(\overline{a_{ji}}) \neq -\infty\}$. Notice that this is included in the bigger domain $\{\overline{x} \mid \forall ji\,.\, |\overline{a_{ji}}|\ \ \overline{x} \neq -\infty\}$. In the second place, we consider the complementary domain $\overline{D} = (\mathbb{R}^+ \cup \{+\infty\})^n - D$ which of course includes the domain $\{\overline{x} \mid \exists ji\,.\, |\overline{a_{ji}}|\ \ \overline{x} = -\infty\}$. Here, for these particular $j$ and $i$, $\lfloor \Delta_f^G \rfloor\ (\overline{a_{ji}}) = -\infty$. Each case is proved by equational reasoning. □

If $\mathbb{I}_\Delta$ is reductive at $\Delta_f$, as the algorithm for $\mu_f$ critically depends on how good is the result for $\Delta_f$, it is advisable to spend some time iterating the interpretation $\mathbb{I}_\Delta$ in order to get better results for $\mu_f$.

*6.3. Algorithm for computing $\mu_f$*

Let $(e_b, e_r) = splitExp_f\ e_f$ and assume $(e_{bef}, e_{aft}) = splitBA_f\ e_r$. In case $splitBA_f\ e_r = [\,]$ we consider $\mu_f = \mu_b$.

We separately infer the part $\Delta_{self}$ of $\Delta_r$ due to space charges to the *self* region of $f$. As the *self* regions for $f$ are stacked, this part only depends on the longest $f$'s call chain, i.e. on the height of the call-tree.

1. Let $\mu_b = [\![e_b]\!]_\mu\ \Sigma\ \Gamma\ (n+m)$, i.e. the heap peak of the non-recursive part of $f's$ body.
2. Let $\Delta_{self} = ([\![e_{bef}]\!]_\Delta\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ (n+m))\ \rho_{self}$, i.e. the charges to $\rho_{self}$ made by the part of $f's$ body before (and including) the last direct recursive call. Notice that including in the environemt $\Sigma$ a correct signature for $f$'s heap charges has no effect since calls to $f$ produces no charge to $\rho_{self}$ given that we do not allow polymorphic recursion. Notice also that $\Delta_{self} \in \mathbb{F}$ rather than to $\mathbb{D}$.
3. Let $\Delta_{bef} = \lfloor([\![e_{bef}]\!]_\Delta\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ (n+m))\rfloor$, i.e. the charges to regions other than $\rho_{self}$ made by the part of $f's$ body before (and including) the last direct recursive call. Then $\Delta_{bef} \uplus \{\rho_{self} \mapsto \Delta_{self}\}$ is an upper bound to the memory charged to the heap before the last direct call to $f$ in $f$'s body.
4. Let $\mu_{bef} = ([\![e_{bef}]\!]_\mu\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ (n+m)) - \Delta_{self}$. It represents the heap peak of $f's$ body before the last direct recursive call, without considering the $\rho_{self}$ region.
5. Let $\mu_{aft} = [\![e_{aft}]\!]_\mu\ \Sigma\ \Gamma\ (n+m)$, i.e. the heap peak of $f's$ body after the last direct recursive call.
6. Then, $\mu_f \stackrel{\text{def}}{=} (\Delta_{self} \cdot (len_f - 1) + |\,\Delta_{bef}\,| + \sqcup\{\mu_{bef}, \mu_{aft}, \mu_b\}) \sqcup \mu_b$.

The intuitive idea is that the charges to $\rho_{self}$ are stacked, and those to regions other than $\rho_{self}$ are considered as done in the worst possible case: from the last but one call to $f$ of the longest chain call. The final supremum with $\mu_b$ ensures that $\mu_f$ does not collapse to $-\infty$ for small values of the argument sizes $\overline{x_i}^n$.

In our example, if we take as $e_b, e_{bef}$ and $e_{aft}$ the definitions of Fig. 6, we get $\mu_{self} = 0$, $\mu_b = 3$, $\mu_{bef} = 0$, and $\mu_{aft} = 2$. Hence $\mu_f = \lambda n\ x.2\min(n, x-1) + 6$.

*6.3.1. Correctness*

**Lemma 15.** *If the functions in $\Delta_f$, $len_f$, and the size functions belong to $\mathbb{F}$, then $\mu_f$ belongs to $\mathbb{F}$.*

*Proof.* This is a consequence of $\mathbb{F}$ being closed by the operations $\{+, \sqcup, \cdot\}$ and $len_f \sqsupseteq 1$. □

**Theorem 4.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$, be the* context *function, $\Gamma$ the inferred global type environment, an initial environment $E_0$ and a heap $h_0$ such that judgement (2) is derivable under an environment $\Sigma'$ in which $f$'s body $e_f$ is replaced with a definition* **hmask** *$e_f$ producing the same final heap and value than $e_f$, but resetting the second component of the resource vector, obtaining $(\delta, 0, s)$.*

*Let $e$ be a subexpression of $e_f$, and let $(e_{bef}, e_{aft}) = splitBA_f\ (\pi_2\ (splitExp_f\ e))$. In case $splitBA_f\ (\pi_2\ (splitExp_f\ e)) = [\,]$ we will consider $(e_{bef}, e_{aft}) = (e, \square)$ or $(\square, e)$ as convenient.*

*Then for any pair $(e_b, e_a) \in splitBA_f\ (\pi_2\ (splitExp_f\ e))$ and execution under $\Sigma'$*

$$E \vdash h, k_0, td, (e_b, e_a)_X \Downarrow h', k_0, v, (\delta_b/\delta_a, m'_b/m'_a, s_b/s_a)$$

*such that $E \vdash h, k_0, td, e \Downarrow h', k_0, v, (\delta, m', s)$, belongs to the derivation of (2), it holds that*

1. $[\![e_{bef}]\!]_\Delta\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \succeq_{\overline{s_i}^l, k_0, \phi} \delta_b$, *where $E\ x_i = v_i$ and $s_i = size(h, v_i)$ for each $i \in \{1 \ldots n\}$ and $\phi$ is the region instantiation consistent with $E$ and $\Gamma$.*

2. $[\![e_{bef}]\!]_\mu\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \succeq_{\overline{s_i}^l} m'_b$

3. $[\![e_{aft}]\!]_\mu\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \succeq_{\overline{s_i}^l} m'_a$

*Proof.* By structural induction on $e$, see Appendix A □

If we consider an evaluation of a subexpression $e$ of $f$'s body $e_f$ producing a resource vector $(\delta, m, s)$ with at least one recursive call to $f$, by Lemma 5, then the execution of $e_r = \pi_2\ (splitExp_f\ e)$ produces the same resource vector. Then, by Theorem 3 there is an equivalent pair $(e_b, e_a) \in splitBA_f\ e_r$ producing resources $(\delta_b/\delta_a, m_b/m_a, s_b/s_a)$ meeting the equations shown there. In the following lemma we will consider that if there are no direct calls to $f$ in the execution of $e$, we can choose as equivalent pair both $(e_b, e_a) = (e, \square)$ and $(e_b, e_a) = (\square, e)$.

**Lemma 16.** *Let us consider an evaluation of a subexpression $e$ of $f$'s body $e_f$ producing a resource vector $(\delta, m, s)$, such that there are $p \geq 0$ direct calls to $f \in dom\ \Sigma$, each one producing an associated resource vector $(\delta_j, m_j, s_j)$, $j \in P = \{1 \ldots p\}$. We execute now, with the pair semantics of Fig. 7, the equivalent pair $(e_b, e_a)$ under another environment $\Sigma'$, in which $f$'s body $e_f$ is replaced with a definition* **hmask** *$e_f$ producing the same final heap and value than $e_f$, but resetting the second component of the resource vector, obtaining $(\delta_j, 0, s_j)$ instead of $(\delta_j, m_j, s_j)$. Assume that we obtain $(\delta_b/\delta_a, m'_b/m'_a, s_b/s_a)$. Then, the following equation holds:*

$$m \leq \sqcup\ \{m'_b,\ \sqcup_{j \in P}\ (m_j - |\,\delta_j\,|) + |\,\delta_b\,|,\ \ |\,\delta_b\,| + m'_a\}$$

*Proof.* By structural induction on $e$. See Appendix A. □

**Definition 9.** *Assume $splitBA_f\ (\pi_2\ (splitExp_f\ e_f)) \neq [\ ]$. We will consider that $\Delta_b$ is a correct upper bound to the memory charged to the heap before the last direct call to $f$ in $f$'s body if for all $(e_b, e_a) \in splitBA_f\ (\pi_2\ (splitExp_f\ e_f))$, $h$, $h'$,$k$, $\overline{v_i}^l$, $\overline{i_j}^q$, $v$, $\delta_b$, $\delta_a$, $m_b$, $m_a$, $s_b$, $s_a$, $\overline{s_i}^n$ such that:*

1. $E \vdash h, k+1, l+q, (e_b, e_a) \Downarrow h', k+1, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$
   *where* $E = [\overline{x_i \mapsto v_i}^l, \overline{r_j \mapsto i_j}^q, self \mapsto k+1]$

2. $\forall i \in \{1 \ldots l\} : s_i = size(h, v_i)$

*it holds that $\Delta_b \succeq_{\overline{s_i}^l, k+1, \phi} \delta_b$ for the consistent region instatiation $\phi$ determined by $E$ and $\Gamma$.*

**Lemma 17.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ be a typeable function definition such that $splitBA_f\ (\pi_2\ (splitExp_f\ e_f)) \neq [\,]$ and*

$$E \vdash h, k+1, n+m, e_f \Downarrow h', k+1, v, (\delta, m, s), (n_t, n_b, l)_f \tag{17}$$

*where $E \equiv_{def} \left[\overline{x_i \mapsto v_i}^n, \overline{r_j \mapsto i_j}^m, self \mapsto k+1\right]$. Let $\Delta_b$ be any correct upper bound to the memory charged to the heap before the last direct call to $f$ in $f$'s body and $\Delta'_b = \lfloor \Delta_b \rfloor$. Then*

$$(\Delta_{self} \cdot (l-1) + |\,\Delta'_b\,| + \sqcup\ \{\mu_{bef}, \mu_b, \mu_{aft}\}) \sqcup \mu_b \succeq_{\overline{s_i}^n} m$$

*where $s_i = size(h, v_i)$ for each $i \in \{1 \ldots n\}$.*

*Proof.* By induction on the maximum length $l$ of $f$'s call chains. In the following, we shall abbreviate $|\,\Delta\,|$ by just $\Delta$.

$l = 1$ There are no calls to $f$ during $e_f$'s evaluation. By Lemma 4 we have

$$E \vdash h, k+1, n+m, e_b \Downarrow h', k+1, v, (\delta, m, s)$$

and by Theorem 1 we have $\mu_b \succeq_{\overline{s_i}^n} m$, and then

$$(\Delta_{self} \cdot 0 + \Delta'_b + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}) \sqcup \mu_b \succeq_{\overline{s_i}^n} m$$

$l > 1$ Let us assume that there are $p > 0$ direct calls to $f$ during $e_f$'s evaluation, and let $P = \{1 \dots p\}$. Every call $j \in P$ is done with environment and initial heap $E_j, h_j$, actual arguments $\overline{a_{ji}}$ with sizes $\overline{s_{ji}} = size(h_j, E_j\ \overline{a_{ji}})$, resource consumption $(\delta_j, m_j, s_j)$, and maximum chain length $l_j$, such that $\sqcup_{j \in P}\{l_j\} = l - 1$. By induction hypothesis we have for all $j$:

$$(\Delta_{self} \cdot (l_j - 1) + \Delta'_b + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}) \sqcup \mu_b \succeq_{\overline{s_{ji}}} m_j$$

Assume that we modify the definition of $f$ in the environment $\Sigma$ and replace expression $e_f$ by **hmask** $e_f$ and then we evaluate the pair $(e_b, e_a)$, equivalent to $e_f$ and guaranteed by Lemma 5 and Theorem 3, with the modified $\Sigma$ under the pair semantics, obtaining a resource consumption of $(\delta_b/\delta_a, m'_b/m'_a, s_b/s_a)$. The following property, proved in Lemma 16 above holds:

$$m \leq \sqcup\ \{m'_b,\ \ \sqcup_{j \in P}\ (m_j - \delta_j) + \delta_b,\ \ \delta_b + m'_a\}$$

It is then enough to prove:

$$\begin{array}{ll} & (\Delta_{self} \cdot (l-1) + \Delta'_b + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}) \sqcup \mu_b \\ \succeq_{\overline{s_i}^n} & \sqcup\{m'_b,\ \ \sqcup_{j \in P}\ (m_j - \delta_j) + \delta_b,\ \ \delta_b + m'_a\} \end{array} \qquad (*)$$

We have the following equalities:

$$\begin{array}{ll} & [(\Delta_{self} \cdot (l-1) + \Delta'_b + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}) \sqcup \mu_b]\ \ \overline{s_i} \\ = & [(\Delta_{self} \cdot (\sqcup_{j \in P}\{l_j\}) + \Delta'_b + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}) \sqcup \mu_b]\ \ \overline{s_i} \\ = & [(\sqcup_{j \in P}(\Delta_{self} \cdot (l_j - 1) + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}) + \Delta_{self} + \Delta'_b) \sqcup \mu_b]\ \ \overline{s_i} \\ = & [\sqcup_{j \in P}(\Delta_{self} \cdot (l_j - 1) + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\})\ \overline{s_i} + (\Delta_{self} + \Delta'_b)\ \overline{s_i}] \sqcup (\mu_b\ \overline{s_i}) \end{array}$$

Now, we distinguish two cases for each $l_j$:

- If $l_j = 1$ then, as we have seen above, it holds that $\mu_b\ \overline{s_{ji}} \geq m_j$. Also, as $l > 1$, by Theorem 4 we know that $\Delta_{self}\ \overline{s_i} \geq 0$, i.e. $\Delta_{self}\ \overline{s_i} \neq -\infty$. By $\overline{s_i} \geq \overline{s_{ji}}$ and monotonicity we get:

$$\begin{array}{ll} & (\Delta_{self} \cdot (l_j - 1) + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\})\ \overline{s_i} \\ \geq & (\Delta_{self}\ \overline{s_i}) \cdot 0 + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}\ \overline{s_{ji}} \\ = & \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}\ \overline{s_{ji}} \\ \geq & \mu_b\ \overline{s_{ji}} \\ \geq & m_j \\ \geq & m_j - \delta_j \end{array}$$

- If $l_j > 1$ then $\delta_{jbef} \geq 0$ is defined by Th. 3. Then by hypothesis about $\Delta'_b$ and Theorem 4, it must hold $(\Delta_{self} + \Delta'_b)\ \overline{s_{ji}} \geq \delta_{jbef} \geq 0$, i.e. $(\Delta_{self} + \Delta'_b)\ \overline{s_{ji}} \neq -\infty$. Consequently:

$$(\Delta_{self} \cdot (l_j - 1) + \Delta'_b + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\})\ \overline{s_{ji}} \geq \mu_b\ \overline{s_{ji}}$$

and from the i.h.

$$(\Delta_{self} \cdot (l_j - 1) + \Delta'_b + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\})\ \overline{s_{ji}} \geq m_j$$

If this is true for an arbitrary upper bound $\Delta'_b\ \overline{s_{ji}} \geq \lfloor \delta_{jbef} \rfloor$, in particular it holds that:

$$((\Delta_{self}\ \overline{s_{ji}}) \cdot (l_j - 1) + \lfloor \delta_{jbef} \rfloor + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\}\ \overline{s_{ji}}) \geq m_j$$

By knowing that $\delta_j \geq \lfloor \delta_{jbef} \rfloor$ and $\overline{s_i} \geq \overline{s_{ji}}$ we get:

$$\begin{array}{ll} & (\Delta_{self} \cdot (l_j - 1) + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\})\ \overline{s_i} \\ \geq & (\Delta_{self} \cdot (l_j - 1) + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\})\ \overline{s_{ji}} + \lfloor \delta_{jbef} \rfloor - \delta_j \\ \geq & m_j - \delta_j \end{array}$$

In both cases we obtain the same result. So, we get:

$$\sqcup_{j \in P}(\Delta_{self} \cdot (l_j - 1) + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\})\ \overline{s_i} \geq \sqcup_{j \in P}(m_j - \delta_j)$$

On the other hand, we have:

1. $(\Delta_{self} + \Delta'_b)\ \overline{s_i} \geq \delta_b$, by Theorem 4 and hypothesis about $\Delta'_b$. Then, equation $(*)$ above holds for $\sqcup_{j \in P}(m_j - \delta_j) + \delta_b$.
2. $(\Delta_{self} + \mu_{bef})\ \overline{s_i} = (\llbracket e_{bef} \rrbracket_\mu\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ (n+m))\ \overline{s_i} \geq m'_b$, by Theorem 4. Then, equation $(*)$ holds for $m'_b$.
3. $\mu_{aft}\ \overline{s_i} = (\llbracket e_{aft} \rrbracket_\mu\ \Sigma\ \Gamma\ (n+m))\ \overline{s_i} \geq m'_a$, by Theorem 4. Then, together with (1), equation $(*)$ holds for $\delta_b + m'_a$.

Then, the required property holds.

□

**Corollary 1.** *$\mu_f$ is a safe upper bound to the heap peak of function $f$.*

*Proof.* It immediately follows from Theorem 4, Lemma 17 and $len_f$ being a safe upper bound to the maximum length of call chains to $f$. □

*6.3.2. Reductivity of the $\mu$ interpretation*

As in the case of $\Delta_f$, we can define an interpretation $\mathbb{I}_\mu$ taking any upper bound $\mu_1$ as input, and producing a better one $\mu_2 = \mathbb{I}_\mu(\mu_1)$ as output.

**Lemma 18.** *For all $n$, $\mathbb{I}^n_\mu(\mu_f)$ is a safe upper bound for $f$'s heap peak.*

*Proof.* This is a consequence of $\mathbb{F}$ being a complete lattice, $\mathbb{I}_\mu$ being monotonic in $\mathbb{F}$, and $\mathbb{I}_\mu$ being reductive at $\mu_f$. We prove now that $\mathbb{I}_\mu$ is reductive, i.e. $\mathbb{I}_\mu(\mu_f) \sqsubseteq \mu_f$. Let us assume that there are $p > 0$ direct recursive calls to $f$ in the text of $e_r$. Let $P = \{1 \ldots p\}$ and $\mu^j, \Delta^j, \ldots, j \in P$ denote the corresponding functions $\mu, \Delta, \ldots$ applied to the arguments $\overline{a_{ji}}^n(\overline{x})$ of the recursive call $j$. Then, we get:

$$
\begin{array}{lll}
 & \mathbb{I}_\mu(\mu_f) & \\
= & \pi_2\left([\![e_r]\!]\ \Sigma[f \mapsto (\Delta_f, \mu_f, \_)]\ \Gamma\ (n+m)\right) & \text{-- by definition of } \mathbb{I}_\mu \\
\sqsubseteq & |\Delta_{self}| + \sqcup\{\mu_{bef}, |\Delta_{bef}| + \mu_{aft}, |\Delta_{bef}| + \sqcup_{j\in P}\{\mu_f^j - |\Delta_f^j|\}\} & \text{-- by Lemma 19 below} \\
= & |\Delta_{self}| + \sqcup\{\mu_{bef}, |\Delta_{bef}| + \mu_{aft}, |\Delta_{bef}| + & \\
 & \sqcup_{j\in P}\{|\Delta_{self}^j| \times (len_f^j - 1) + |\Delta_{bef}^j| + \sqcup\{\mu_{bef}^j, \mu_b^j, \mu_{aft}^j\} - |\Delta_f^j|\} & \text{-- by definition of } \mu_f \\
\overset{(a)}{=} & |\Delta_{self}| + |\Delta_{bef}| + \sqcup_{j\in P}\{|\Delta_{self}^j| \times (len_f^j - 1) + |\Delta_{bef}^j| + & \\
 & \sqcup\{\mu_{bef}^j, \mu_b^j, \mu_{aft}^j\} - |\Delta_f^j|\} & \text{-- case } (a)\text{, the lub is the third one} \\
\sqsubseteq & |\Delta_{self}| \times \sqcup_{j\in P}\, len_f^j + |\Delta_{bef}| + & \\
 & \sqcup_{j\in P}\{|\Delta_{bef}^j - \Delta_f^j| + \sqcup\{\mu_{bef}^j, \mu_b^j, \mu_{aft}^j\}\} & \text{-- For all } j,\ \Delta_{self}^j \sqsubseteq \Delta_{self} \\
\sqsubseteq & |\Delta_{self}| \times (len_f - 1) + |\Delta_{bef}| + \sqcup_{j\in P} \sqcup\{\mu_{bef}^j, \mu_b^j, \mu_{aft}^j\} & \text{-- For all } j,\ len_f^j \sqsubseteq len_f - 1 \text{ and } \Delta_{bef}^j \sqsubseteq \Delta_f^j \\
\sqsubseteq & |\Delta_{self}| \times (len_f - 1) + |\Delta_{bef}| + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\} & \text{-- } \overline{a_{ji}}^n \sqsubseteq \overline{x_i}^n \text{ and } \mu_{bef}, \mu_b, \mu_{aft} \text{ monotonic} \\
= & \mu_f & \\
\overset{(b)}{=} & |\Delta_{self}| + \sqcup\{\mu_{bef}, |\Delta_{bef}| + \mu_{aft}\} & \text{-- case } (b)\text{, the lub is not the third one} \\
\sqsubseteq & |\Delta_{self}| + |\Delta_{bef}| + \sqcup\{\mu_{bef}, \mu_{aft}, \mu_b\} & \text{-- mathematics} \\
\sqsubseteq & |\Delta_{self}| \times (len_f - 1) + |\Delta_{bef}| + \sqcup\{\mu_{bef}, \mu_b, \mu_{aft}\} & \text{-- at least a recursive call, then } len_f \geq 2 \\
= & \mu_f &
\end{array}
$$

Notice the assumption on well-behaviour of $len_f$. □

**Lemma 19.** *Let us assume two functions $\mu_0 \in \mathbb{F}$ and $\Delta_f \in \mathbb{D}$, and an expression $e$ such that $(\_, e_r) = splitExp_f(e)$ and $e_r \neq \#$. Let $\mu_r = [\![e_r]\!]_\mu\ \Sigma[f \mapsto (\Delta_f, \mu_0, \_)]\ \Gamma\ (n+m)$, let $P \neq \emptyset$ be the set of textual recursive calls to $f$ in $e_r$, and let $\mu^j, \Delta^j, \ldots, j \in P$ denote the corresponding functions $\mu, \Delta, \ldots$ applied to the arguments $\overline{a_{ji}}^n(\overline{x})$ of the recursive call $j \in P$. Let $(e_{rb}, e_{ra}) = splitBA_f(e_r)$ (it is not empty because $P \neq \emptyset$), and let $\Delta_{rself}$, $\Delta_{rbef}$, $\mu_{rbef}$, and $\mu_{raft}$ computed as explained at the beginning of Sec. 6.3. Then it holds that:*

$$\mu_r \sqsubseteq |\Delta_{rself}| + \sqcup\{\mu_{rbef}, |\Delta_{rbef}| + \mu_{raft}, |\Delta_{rbef}| + \sqcup_{j\in P}\{\mu_0^j - |\Delta_f^j|\}\}$$

*Proof.* By structural induction on $e$. The cases are:

$\boxed{e \in \{c,\ x,\ x!,\ a_1 \oplus a_2,\ x@r, g\ \overline{a_i}\ @\ \overline{r_j}, \#\}}$ , $g \neq f$. These cases are not possible because $e_r = \#$.

$\boxed{e \equiv f\ \overline{a_i}\ @\ \overline{r_j}}$ Then, $e_r = f\ \overline{a_i}\ @\ \overline{r_j}$, $splitBA_f(e_r) = (f\ \overline{a_i}\ @\ \overline{r_j}, \Box)$, $\mu_r = \mu_0(\overline{a_i})$, $\Delta_{rself} = 0$, $\mu_{rbef} = \mu_{raft} = 0$, $\Delta_{rbef} = \Delta_f(\overline{a_i})$. We get:

$$\mu_r \sqsubseteq |\Delta_{rself}| + \sqcup\{\mu_{rbef}, |\Delta_{rbef}| + \mu_{raft}, |\Delta_{rbef}| + \mu_0(\overline{a_i}) - |\Delta_f(\overline{a_i})|\}$$

$\boxed{e \equiv \textbf{case}\ x\ \textbf{of}\ \overline{C\ \overline{x_{ij}} \to e_i}^n}$ Then, $e_r = \textbf{case}\ x\ \textbf{of}\ \overline{C\ \overline{x_{ij}} \to e_{ir}}^{n'}$, where $e_{ir} = splitExp_f(e_i)$ are the $n' \leq n$ branches different from # (at least one exists because $P \neq \emptyset$). Then,

$$splitBA_f(e_r) = [\overline{(\textbf{case}\ x\ \textbf{of}\ \overline{C\ \overline{x_{ij}} \to e_{irb}},\ \textbf{case}\ x\ \textbf{of}\ \overline{C\ \overline{x_{ij}} \to e_{ira}})}^m]$$

where $(e_{irb}, e_{ira}) \in splitBA_f(e_{ir})$ for all $i$. As there are textual calls to $f$ in all the $e_{ir}$, we have $e_{ir} \neq \#$ and $splitBA_f(e_{ir}) \neq [\,]$. Then, the $e_i$ satisfy the hypothesis of the lemma. By induction hypothesis on $e_i$ we have:

$$\mu_{ir} \sqsubseteq\ |\Delta_{irself}| + \sqcup\{\mu_{irbef}, |\Delta_{irbef}| + \mu_{iraft}, |\Delta_{irbef}| + \sqcup_{j\in P_i}\{\mu_0^j - |\Delta_f^j|\}\}$$

for all $i$, being $P_i$ the set of textual calls in $e_i$. Then, we get:

$$\begin{array}{lll} & \mu_r & \\ = & \sqcup_i \mu_{ir} & \text{-- by the a.i. rules} \\ \sqsubseteq & \sqcup_i(|\Delta_{irself}| + \sqcup\{\mu_{irbef}, |\Delta_{irbef}| + \mu_{iraft}, |\Delta_{irbef}| + \sqcup_{j\in P_i}\{\mu_0^j - |\Delta_f^j|\}\}) & \text{-- by induction hypothesis} \\ \sqsubseteq & \sqcup_i |\Delta_{irself}| + \sqcup\{\sqcup_i \mu_{irbef}, \sqcup_i |\Delta_{irbef}| + \sqcup_i \mu_{iraft}, & \\ & \sqcup_i |\Delta_{irbef}| + \sqcup_i \sqcup_{j\in P_i}\{\mu_0^j - |\Delta_f^j|\}\} & \text{-- by mathematics} \\ = & |\Delta_{rself}| + \sqcup\{\mu_{rbef}, |\Delta_{rbef}| + \mu_{raft}, |\Delta_{rbef}| + \sqcup_{j\in P}\{\mu_0^j - |\Delta_f^j|\}\} & \text{-- by the a.i. rules} \end{array}$$

$\boxed{e \equiv \textbf{let } x_1 = e_1 \textbf{ in } e_2}$ Some of the abstract functions disregard region $\rho_{self}$. For brevity, if $\eta$ is an abstract function, $\eta^*$ denotes $\eta$ restricted to those regions different from $\rho_{self}$. Also for brevity, we remove the size notation $|\Delta|$ applied to abstract heaps. Let $(\_, e_r) = splitExp_f(e)$. Let us call

$$(e_{1b}, e_{1r}) = splitExp_f(e_1), \qquad (e_{2b}, e_{2r}) = splitExp_f(e_2)$$

As $e_r \neq \#$ by hypothesis, we distinguish the following cases according to $e_r$'s shape (see Fig. 4):

$\boxed{e_r \equiv \textbf{let } x_1 = e_1 \textbf{ in } e_{2r},}$ if $e_{1r} = \# \wedge e_{2r} \neq \#$. According to Fig. 5 we have $splitBA_f(e_{1r}) = [\,], B = [\,]$, and $splitBA_f(e_{2r}) = [\overline{(e^i_{2rb}, e^i_{2ra})}^m]$. Then,

$$splitBA_f(e_r) = [\overline{(\textbf{let } x_1 = e_1 \textbf{ in } e^i_{2rb},\ collapse(\textbf{let } x_1 = \Box \textbf{ in } e^i_{2ra}))}^m]$$

As $e_1$ has no textual calls to $f$, then all the calls of $P$ are contained in $e_{2r}$. By induction hypothesis on $e_2$ and by definition, we have the following facts:

$$\begin{array}{l} \mu_{2r} \sqsubseteq \Delta_{2rself} + \sqcup\{\mu_{2rbef}, \Delta_{2rbef} + \mu_{2raft}, \Delta_{2rbef} + \sqcup_{j\in P}\{\mu_0^j - \Delta_f^j\}\} \\ \mu_r = \sqcup\{\mu_1, \Delta_1 + \mu_{2r}\} \\ \Delta_{rself} = \Delta_{1self} + \Delta_{2rself} \\ \Delta_{rbef} = \sqcup_i\{\Delta_1^* + \Delta^i_{2rb}\} = \Delta_1^* + \sqcup_i \Delta^i_{2rb} = \Delta_1^* + \Delta_{2rbef} \\ \mu_{rbef} = \sqcup_i \sqcup\{\mu_1^*, \Delta_1^* + \mu^i_{2rb}\} = \sqcup\{\mu_1^*, \Delta_1^* + \sqcup_i \mu^i_{2rb}\} = \sqcup\{\mu_1^*, \Delta_1^* + \mu_{2rbef}\} \\ \mu_{raft} = \sqcup_i \sqcup\{0, 0 + \mu^i_{2ra}\} = \sqcup_i \mu^i_{2ra} = \mu_{2raft} \end{array}$$

We must prove:

$$\sqcup\{\mu_1, \Delta_1 + \mu_{2r}\} \sqsubseteq \Delta_{rself} + \sqcup\{\mu_1^*, \Delta_1^* + \mu_{2rbef}, \Delta_1^* + \Delta_{2rbef} + \mu_{2raft}, \Delta_1^* + \Delta_{2rbef} + \sqcup_{j\in P}\{\mu_0^j - \Delta_f^j\}\}$$

This is true iff every element of the *lhs* is dominated by some element of the *rhs*.

- $\mu_1$ is dominated by $\Delta_{1self} + \mu_1^*$ and so by $\Delta_{rself} + \mu_1^*$
- $\Delta_1$ is dominated by $\Delta_{1self} + \Delta_1^*$
- $\mu_{2r}$ is dominated either by $\Delta_{2rself} + \mu_{2rbef}$, or by $\Delta_{2rself} + \Delta_{2rbef} + \mu_{2raft}$, or by $\Delta_{2rself} + \Delta_{2rbef} + \sqcup_{j\in P}\{\mu_0^j - \Delta_f^j\}$

So, the inequality holds.

$\boxed{e_r \equiv \textbf{let } x_1 = e_{1r} \textbf{ in } e_2,}$ if $e_{1r} \neq \# \wedge e_{2r} = \#$ or $e_{1r} \neq \# \wedge e_{2r} \neq \# \wedge e_{1b} = \#$.

$\boxed{e_r \equiv \sqcup\{\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2r},\ \textbf{let } x_1 = e_{1r} \textbf{ in } e_2\},}$ if $e_{1r} \neq \# \wedge e_{2r} \neq \# \wedge e_{1b} \neq \#\#$.

□

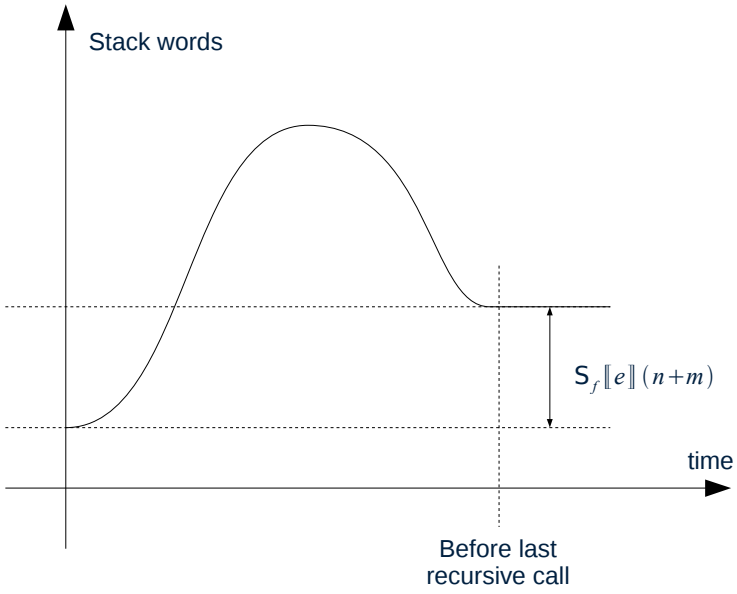


Figure 9: Intuitive meaning of the $\mathcal{S}_f$ function

*6.4. Algorithm for computing $\sigma_f$*

The algorithm for inferring $\mu_f$ traverses $f$'s body from left to right because the abstract interpretation rules for $\mu$ need the charges to the previous heaps. For inferring $\sigma_f$ we can do it better because its rules are symmetrical. The main idea is to count only once the stack peak due to calling to external functions.

1. Let $\sigma = [\![e]\!]_\sigma\ \Sigma[f \mapsto (\_, \_, 0)]\ \Gamma\ (n+m)$.

2. We define the following function $\mathcal{S}_f$ returning a natural number. Intuitively it computes an upper bound to the difference in words between the initial stack in a call to $f$ and the stack just before $e$ is about to jump to $f$ again (Fig. 9):

$$\begin{array}{lcll} \mathcal{S}_f\ (f\ \overline{a_i}^n\ @\ \overline{r_j}^m)\ td & = & n+m-td & \\ \mathcal{S}_f\ (\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2)\ td & = & \sqcup\{2+\mathcal{S}_f\ e_1\ 0, 1+\mathcal{S}_f\ e_2\ (td+1)\} & \\ \mathcal{S}_f\ (\textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n)\ td & = & \sqcup_{r=1}^n (n_r + \mathcal{S}_f\ e_r\ (td+n_r)) & \\ \mathcal{S}_f\ e\ td & = & -\infty & \text{otherwise} \end{array}$$

3. Then, $\sigma_f = \max\{0, \mathcal{S}_f\ e\ (n+m)\} * (len_f - 1) + \sigma$

In our example, if we denote by $e^{\texttt{splitAt}}$ the result of translating the definition of Fig. 6(a) into *Core-Safe* we get $\mathcal{S}_f\ [\![e^{\texttt{splitAt}}]\!]\ (2+3) = 9$ and, by applying the abstract interpretation rules we obtain $\sigma = \lambda n\ xs.10$. Hence $\sigma_f = \lambda n\ xs.9 \min\{n, xs-1\} + 10 = \lambda n\ xs.9 \min\{n+1, xs\} + 1$.

$$E \vdash h, k, (td, s_0), c \Downarrow h, k, c, ([\,], 0, 1)\ [Lit]$$

$$E[x \mapsto v] \vdash h, k, (td, s_0), x \Downarrow h, k, v, ([\,], 0, 1)\ [Var]$$

$$\frac{j \leq k \quad fresh(p)}{E \vdash h, k, (td, s_0), C\ \overline{a_i}^n\ @\ r \Downarrow h \uplus [p \mapsto (j, C\ \overline{v_i}^n)], k, p, ([j \mapsto 1], 1, 1)}\ [Cons]$$

$$\frac{(f\ \overline{x_i}^n\ @\ \overline{r_j}^l = e) \in \Sigma \quad [\overline{x_i \mapsto E(a_i)}^n, \overline{r_j \mapsto E(r'_j)}^l, self \mapsto k+1] \vdash h, k+1, (n+l, s_0+n+l-td), e \Downarrow h', k+1, v, (\delta, m, s)}{E \vdash h, k, (td, s_0), f\ \overline{a_i}^n\ @\ \overline{r'_j}^l \Downarrow h'|_k, k, v, (\delta|_k, m, \max\{n+l, s+n+l-td\})}\ [App]$$

$$\frac{\begin{array}{c} E \vdash h, k, (0, s_0+2), e_1 \Downarrow h', k, v_1, (\delta_1, m_1, s_1) \\ E \cup [x_1 \mapsto v_1] \vdash h', k, (td+1, s_0+1), e_2 \Downarrow h'', k, v, (\delta_2, m_2, s_2) \end{array}}{E \vdash h, k, (td, s_0), \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2 \Downarrow h'', k, v, (\delta_1+\delta_2, \max\{m_1, |\delta_1|+m_2\}, \max\{2+s_1, 1+s_2\})}\ [Let]$$

$$\frac{C = C_r \quad E \cup [\overline{x_{r_i} \mapsto v_i}^{n_r}] \vdash h, k, (td+n_r, s_0+n_r), e_r \Downarrow h', k, v, (\delta, m, s)}{E[x \mapsto p] \vdash h[p \mapsto (j, C\ \overline{v_i}^n)], k, (td, s_0), \mathbf{case}\ x\ \mathbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n \Downarrow h', k, v, (\delta, m, s+n_r)}\ [Case]$$

Figure 10: Operational semantics with stack size information

*6.4.1. Correctness*

**Lemma 20.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ be a function definition. Then $\mathcal{S}_f\ e\ td \geq n + m - td$ for all $td$ and every subexpression $e$ of $e_f$ in which there is a recursive call to $f$.*

*Proof.* By induction on the structure of $e$. If $e$ is a recursive call to $f$ the property holds trivially. We consider the remaining cases:

$e \equiv \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$ If there is a call to $f$ in $e_2$ then:

$$\mathcal{S}_f\ e\ td \geq 1 + \mathcal{S}_f\ e_2\ (td+1) \geq 1 + n + m - (td-1) = n + m - td$$

If there is no call to $f$ in $e_2$, there must be at least one call in $e_1$ and hence:

$$\mathcal{S}_f\ e\ td \geq 2 + \mathcal{S}_f\ e_1\ 0 \geq 2 + n + m - 0 \geq n + m - td$$

$e \equiv \mathbf{case}\ x\ \mathbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n$ Let us assume that there is a call to $f$ in the $r$-th branch of the **case**. Then:

$$\mathcal{S}_f\ e\ td \geq n_r + \mathcal{S}_f\ e_r\ (td+n_r) \geq n_r + n + m - (td - n_r) = n + m - td$$

□

**Lemma 21.** *If $len_f$ and all the size functions belong to $\mathbb{F}$, then $\sigma_f$ belongs to $\mathbb{F}$.*

*Proof.* It follows trivially from the fact that $len_f \sqsupseteq 1$ (hence $len_f - 1 \in \mathbb{F}$) and that monotonicity is preserved by $+$, $\sqcup$ and substraction by a constant function. It is also easy to prove (by simple inspection of the abstract interpretation rules) that $\sigma$ is always nonnegative.

□

**Lemma 22.** *Let us assume the execution of an expression under an environment $\Sigma$ containing the function definition $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ and that there are $p$ direct calls to this function (and hence $p$ derivations of $e_f$). For each $i \in \{1 \dots p\}$, we denote the initial stack size (resp. stack cost) of the $i$-th derivation by $s_0^{(i)}$ (resp. $s^{(i)}$):*

$$\frac{\dots (td^{(1)}, s_0^{(1)}), e_f \Downarrow \dots (\delta^{(1)}, m^{(1)}, s^{(1)}) \qquad \dots (td^{(p)}, s_0^{(p)}), e_f \Downarrow \dots (\delta^{(p)}, m^{(p)}, s^{(p)})}{\vdots \qquad\qquad \dots \qquad\qquad \vdots}$$
$$\dots (td, s_0), e \Downarrow \dots (\delta, m, s)$$

*In addition, we assume the execution of the same expression $e$ under an environment $\Sigma'$ in which the definition of $f$ is replaced by $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = \mathbf{rmask}\ e_f$, obtaining $s'$ as the stack cost. Then the following relation between $s$ and $s'$ holds:*

$$s = \max\left(\{s'\} \cup \{s_0^{(i)} - s_0 + s^{(i)} \mid i = 1 \dots p\}\right)$$

*Proof.* By induction on the structure of $e$.

$e \in \{\#, c, x, a_1 \oplus a_2, C\ \overline{a_i}^n\ @\ r, g\ \overline{a_i}^l\ @\ \overline{r_j}^s \text{ with } g \neq f\}$

This is trivial, since $p = 0$ and both executions with $\Sigma$ and with $\Sigma'$ produce the same stack cost. Hence $s = s' = \max\{s'\}$

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r'_j}^m$

There is a single recursive call ($p = 1$):

$$\frac{\dots (n + m, s_0 + n + m - td), [\mathbf{rmask}]\ e_f \Downarrow \dots (\_, \_, s^{(1)})}{\dots (td, s_0), f\ \overline{a_i}^n\ @\ \overline{r'_j}^m \Downarrow \dots (\_, \_, \max\{n + m, s^{(1)} + n + m - td\})}$$

Under $\Sigma'$ we get $s^{(1)} = 0$ and hence $s' = \max\{n + m, 0 + n + m - td\} = n + m$. Since $s_0^{(1)} - s_0 = (s_0 + n + m - td) - s_0 = n + m - td$, we get:

$$s = \max\{n + m, s^{(1)} + n + m - td\} = \max\{s', s_0^{(1)} - s_0 + s^{(1)}\}$$

$e \equiv \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$

We assume that there are $q$ calls to $f$ in the execution of $e_1$ and $p - q$ calls in the execution of $e_2$, with $0 \leq q \leq p$. We have got the following situation:

$$\frac{\dfrac{(q \text{ calls to } f)}{\dots, (0, s_0 + 2), e_1 \Downarrow \dots (\delta_1, m_1, s_1)} \quad \dfrac{(q - p \text{ calls to } f)}{\dots, (td + 1, s_0 + 1), e_2 \Downarrow \dots (\delta_2, m_2, s_2)}}{\dots, (td, s_0), \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2 \Downarrow \dots (\delta, m, s)}$$

If we denote by $s'$ (resp. $s'_1$, $s'_2$) the costs of $e$ (resp. $e_1$, $e_2$) when evaluated under $\Sigma'$, we get:

$$\begin{array}{rcll} s & = & \max\{2+s_1, 1+s_2\} & \\ & = & \max\Big\{2+\max\Big(\{s'_1\}\cup\{s_0^{(i)}-(s_0+2)+s^{(i)} \mid i\in\{1\dots q\}\}\Big), & \\ & & \quad 1+\max\Big(\{s'_2\}\cup\{s_0^{(j)}-(s_0+1)+s^{(j)} \mid j\in\{q+1\dots p\}\}\Big)\Big\} & \text{– by I.H} \\ & = & \max\Big\{\max\{2+s'_1, 1+s'_2\}, \max\{s_0^{(i)}-s_0+s^{(i)} \mid i\in\{1\dots q\}\}, & \\ & & \quad \max\{s_0^{(j)}-s_0+s^{(j)} \mid j\in\{q+1\dots p\}\}\Big\} & \text{– properties of max} \\ & = & \max\Big(\{s'\}\cup\{s_0^{(i)}-s_0+s^{(i)} \mid i\in\{1\dots p\}\}\Big) & \end{array}$$

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

We assume that the $r$-th branch is executed and we denote by $s_r$ and $s'_r$ the costs associated to the derivations with $\Sigma$ and $\Sigma'$.

$$\begin{array}{rcll} s & = & n_r + s_r & \\ & = & n_r + \max\Big(\{s'_r\}\cup\{s_0^{(i)}-(s_0+n_r)+s^{(i)} \mid i\in\{1\dots p\}\}\Big) & \text{– by I.H.} \\ & = & \max\Big(\{n_r+s'_r\}\cup\{s_0^{(i)}-s_0+s^{(i)} \mid i\in\{1\dots p\}\}\Big) & \text{– properties of max} \\ & = & \max\Big(\{s'\}\cup\{s_0^{(i)}-s_0+s^{(i)} \mid i\in\{1\dots p\}\}\Big) & \end{array}$$

□

**Lemma 23** (Correctness lemma for $\mathcal{S}_f$)**.** *We assume the following execution $E \vdash h, k, (td, s_0), e \Downarrow h', k, v, (\delta, m, s)$ in which there are $p \geq 1$ direct calls to a function $f$. For a given $i \in \{1\dots p\}$, we denote by $s_0^{(i)}$ the stack size before executing the $e_f$ body corresponding to the $i$-th call of $f$:*

$$\frac{\dots \qquad \dfrac{E' \vdash h', k, (td', s_0^{(i)}), e_f \Downarrow h'', k, v', (\delta', m', s')}{\vdots} \qquad \dots}{E \vdash h, k, (td, s_0), e \Downarrow h', k, v, (\delta, m, s)}$$

*Then: $\mathcal{S}_f\ e\ td \geq s_0^{(i)} - s_0$*

*Proof.* By induction on the size of the $\Downarrow$-derivation. Since the derivation contains a call to $f$, we assume that $e$ contains a subexpression $f\ \overline{a_i}\ @\ \overline{r_j}$. This rules out the cases $e \equiv c, x, a_1 \oplus a_2, C\ \overline{a_i}^n\ @\ r$ and $g\ \overline{b_i}^q\ @\ \overline{s_j}^q$ with $g \neq f$. We distinguish the remaining cases:

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r_j}^m$

We get $s_0^{(i)} = s_0 + n + m - td$, and hence $\mathcal{S}_f\ e\ td = n + m - td = s_0^{(i)} - s_0$

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

If the $i$-th call to $f$ is in $e_2$ we get:

$$\begin{array}{rcll} \mathcal{S}_f\ e\ td & \geq & 1 + (\mathcal{S}_f\ e_2\ (td+1)) & \text{– definition of } \mathcal{S}_f \\ & \geq & 1 + s_0^{(i)} - (s_0+1) & \text{– I.H.} \\ & = & s_0^{(i)} - s_0 & \end{array}$$

If the $i$-th call to $f$ is in $e_1$ we get:

$$\begin{array}{rcll} \mathcal{S}_f\ e\ td & \geq & 2 + (\mathcal{S}_f\ e_1\ 0) & \text{– definition of } \mathcal{S}_f \\ & \geq & 2 + s_0^{(i)} - (s_0 + 2) & \text{– I.H.} \\ & = & s_0^{(i)} - s_0 & \end{array}$$

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

We assume the $r$-th branch being executed ($1 \leq r \leq n$). Therefore:

$$\begin{array}{rcll} \mathcal{S}_f\ e\ td & \geq & n_r + (\mathcal{S}_f\ e_r\ (td + n_r)) & \text{– definition of } \mathcal{S}_f \\ & \geq & n_r + s_0^{(i)} - (s_0 + n_r) & \text{– I.H.} \\ & = & s_0^{(i)} - s_0 & \end{array}$$

□

**Lemma 24.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ be a function definition and the following execution of $f$ applied to some $\overline{v_i}^n$ and $\overline{i_j}^m$.*

$$\left[\overline{x_i \mapsto v_i}^n, \overline{r_j \mapsto i_j}^m, self \mapsto k+1\right] \vdash h, k+1, (td, s_0), n+m, e_f \Downarrow h', k+1, v, (\delta, m, s), (n_t, n_b, l)_f \qquad (18)$$

*Then $\max\{0, \mathcal{S}_f\ e_f\ (n+m)\} * (l-1) + \sigma \succeq_{\overline{s_i}^n} s$, where $\sigma$ is as defined in the algorithm and $s_i = size(h, v_i)$ for all $i \in \{1 \ldots n\}$.*

*Proof.* By induction on the number of nested recursive calls $l$.

$l = 1$ This implies that there are not direct calls to $f$ in the derivation of (18), which allows us (by Lemma 4) to substitute the $e_f$ for $e_b$ in it, being $(e_b, \_) = splitExp_f\ e_f$. If we denote by $\sigma_b$ the result of $[\![e_b]\!]_\sigma\ \Sigma[f \mapsto (\_, \_, 0)]\ \Gamma\ (n+m)$, we get (by Lemma 7) $\sigma \sqsupseteq \sigma_b$.

$$\max\{0, \mathcal{S}_f\ e_f\ (n+m)\} * 0 + \sigma = \sigma \sqsupseteq \sigma_b \succeq_{\overline{s_i}^n} s$$

with the last step justified by Theorem 1.

$l > 1$ We assume that there are $p$ direct calls to $f$ in (18), with their respective $l^{(i)}$ and $s^{(i)}$, being $i \in \{1 \ldots p\}$. We denote by $s_{ij}$ the actual size of the $j$-th argument in the $i$-th recursive call. Moreover, let $s'$ be the stack cost of executing the main call to $f$ under a signature $\Sigma'$, in which the definition of $f$ is replaced in $\Sigma$ by $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = \textbf{rmask}\ e_f$. Since the abstract signature $\Sigma^\#[f \mapsto (\_, \_, 0)]$ is correct with respect to $\Sigma'$, we can establish $\sigma \succeq_{\overline{s_i}^n} s'$ by Theorem 1.

We also note that in this case we can apply Lemma 20 since we have got at least one call to $f$ in the definition of $e_f$, so we prove the required result as follows:

$$
\begin{array}{ll}
 & \max\{0, \mathcal{S}_f\ e_f\ (n+m)\} * (l-1) + \sigma\ \overline{s_i}^n \\
= & \quad \{ \text{ by Lemma 20 we get } \mathcal{S}_f\ e_f\ (n+m) \geq 0 \ \} \\
 & (\mathcal{S}_f\ e_f\ (n+m)) * (l-1) + \sigma\ \overline{s_i}^n \\
= & \quad \{ \text{ by Lemma 1 } \} \\
 & (\mathcal{S}_f\ e_f\ (n+m)) * (1 + \max_{i\in\{1\ldots p\}}\{l^{(i)}\} - 1) + \sigma\ \overline{s_i}^n \\
= & \\
 & (\mathcal{S}_f\ e_f\ (n+m)) + (\mathcal{S}_f\ e\ (n+m)) * (\max_{i\in\{1\ldots p\}}\{l^{(i)} - 1\}) + \sigma\ \overline{s_i}^n \\
= & \quad \{ \text{ properties of max } \} \\
 & \max_{i\in\{1\ldots p\}}\{(\mathcal{S}_f\ e_f\ (n+m)) + (\mathcal{S}_f\ e_f\ (n+m)) * (l^{(i)} - 1) + \sigma\ \overline{s_i}^n\} \\
= & \quad \{ \text{ properties of max } \} \\
 & \max\left((\mathcal{S}_f\ e_f\ (n+m)) + (\mathcal{S}_f\ e_f\ (n+m)) * (l^{(i)} - 1) + \sigma\ \overline{s_i}^n \mid i \in \{1\ldots p\}\} \cup \{\sigma\ \overline{s_i}^n\}\right) \\
\geq & \quad \{ \text{ monotonicity of } \sigma \text{ and } \overline{s_{ij}}^n \leq \overline{s_i}^n \ \} \\
 & \max\left((\mathcal{S}_f\ e_f\ (n+m)) + (\mathcal{S}_f\ e_f\ (n+m)) * (l^{(i)} - 1) + \sigma\ \overline{s_{ij}}^n \mid i \in \{1\ldots p\}\} \cup \{\sigma\ \overline{s_i}^n\}\right) \\
\geq & \quad \{ \text{ by I.H. and } \sigma \succeq_{\overline{s_i}^n} s' \ \} \\
 & \max\left(\{(\mathcal{S}_f\ e_f\ (n+m)) + s^{(i)} \mid i \in \{1\ldots p\}\} \cup \{s'\}\right) \\
\geq & \quad \{ \text{ by Lemma 23 } \} \\
 & \max\left(\{s_0^{(i)} - s_0 + s^{(i)} \mid i \in \{1\ldots p\}\} \cup \{s'\}\right) \\
= & \quad \{ \text{ by Lemma 22 } \} \\
 & s
\end{array}
$$

□

**Lemma 25.** *$\sigma_f$ is a safe upper bound for $f$'s stack needs.*

*Proof.* This is a consequence of Lemma 24 and of $len_f$ being an upper bound to $f$'s call-tree height.

$$\sigma_f\ \overline{s_i}^n = \max\{0, \mathcal{S}_f\ e\ (n+m)\} * (len_f\ \overline{s_i}^n - 1) + \sigma\ \overline{s_i}^n \geq \max\{0, \mathcal{S}_f\ e\ (n+m)\} * (l-1) + \sigma\ \overline{s_i}^n \geq s$$

□

Also in this case, it makes sense iterating the interpretation as we did with $\Delta_f$ and $\mu_f$, since it holds that $\mathbb{I}_\sigma(\sigma_f) \sqsubseteq \sigma_f$.

**Lemma 26.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ be the context function definition, $e$ an expression which contains $p$ recursive calls to $f$ and $\sigma' \in \mathbb{F}$ a stack cost function. For each recursive call $k \in \{1\ldots p\}$ we denote by $\sigma'_k$ the stack cost function $\lambda\overline{x}^n.\sigma'\ (\overline{|a_{ki}|\ \overline{x}^n}^n)$, where $|a_{ki}|$ is the size function of its $i$-th argument, $(i \in \{1\ldots n\})$. If we define:*

$$
\begin{array}{rcl}
\sigma & = & [\![e]\!]\ \Sigma\ [f \mapsto (\_, \_, \sigma')]\ \ \Gamma\ \ td \\
\sigma_r & = & [\![e]\!]\ \Sigma\ [f \mapsto (\_, \_, 0)]\ \ \Gamma\ \ td
\end{array}
$$

*then the following relation between $\sigma$, $\sigma_r$ and $\sigma'$ holds:*

$$\sigma \sqsubseteq \sqcup(\{\sigma_r\} \cup \{\mathcal{S}_f\ e\ td + \sigma'_k \mid k \in \{1\ldots p\}\})$$

*Proof.* By induction on the structure of $e$.

$e \in \{\#, c, x, C\ \overline{a_i}^n\ @\ r, g\ \overline{a_i}^l\ @\ \overline{r_j}^s \text{ with } g \neq f\}$

In this case we get $p = 0$ and $\sigma = \sigma_r$, from which the required result follows trivially.

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r_j}^m$

We get $p = 1$, $\sigma = \sigma'_1 - td + n + m$ and $\sigma_r = n + m - td$ and hence:

$$\sigma = n + m - td + \sigma'_1 = \sqcup\{n + m - td, n + m - td + \sigma'_1\} = \sqcup\{\sigma_r, \mathcal{S}_f\ e\ td + \sigma'_1\}$$

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

We assume that there are $q \in \{0 \ldots p\}$ calls to $f$ in $e_1$ and $q - p$ calls to $f$ in $e_2$. If we denote by $\sigma_1$ and $\sigma_2$ the stack costs obtained in $e_1$ and $e_2$, we can apply the induction hypothesis as follows:

$$\begin{array}{lcl} \sigma_1 & \sqsubseteq & \sqcup(\{\sigma_{1,r}\} \cup \{\mathcal{S}_f\ e_1\ 0 + \sigma'_k \mid k \in \{1 \ldots q\}\}) \\ \sigma_2 & \sqsubseteq & \sqcup(\{\sigma_{2,r}\} \cup \{\mathcal{S}_f\ e_2\ (td + 1) + \sigma'_k \mid k \in \{q + 1 \ldots p\}\}) \end{array}$$

Therefore,

$$\begin{array}{lcll} \sigma & = & \sqcup\{2 + \sigma_1, 1 + \sigma_2\} & \{\text{by rules of Fig. 3}\} \\ & \sqsubseteq & \sqcup\{2 + \sqcup(\{\sigma_{1,r}\} \cup \{\mathcal{S}_f\ e_1\ 0 + \sigma'_k \mid k \in \{1 \ldots q\}\}), & \\ & & \quad 1 + \sqcup(\{\sigma_{2,r}\} \cup \{\mathcal{S}_f\ e_2\ (td + 1) + \sigma'_k \mid k \in \{q + 1 \ldots p\}\})\} & \{\text{by I.H.}\} \\ & = & \sqcup\{\sqcup\{2 + \sigma_{1,r}, 1 + \sigma_{2,r}\}, \sqcup\{2 + \mathcal{S}_f\ e_1\ 0 + \sigma'_k \mid k \in \{1 \ldots q\}\}, & \\ & & \quad \sqcup\{1 + \mathcal{S}_f\ e_2\ (td + 1) + \sigma'_k \mid k \in \{q + 1 \ldots p\}\}\} & \{\text{by properties of } \sqcup\} \\ & \sqsubseteq & \sqcup\{\sigma_r, \sqcup\{\mathcal{S}_f\ e\ td + \sigma'_k \mid k \in \{1 \ldots q\}\}, & \\ & & \quad \sqcup\{\mathcal{S}_f\ e\ (td) + \sigma'_k \mid k \in \{q + 1 \ldots p\}\}\} & \{\text{by definition of } \mathcal{S}_f \text{ and } \sigma_r\} \\ & = & \sqcup\{\sigma_r, \sqcup\{\mathcal{S}_f\ e\ td + \sigma'_k \mid k \in \{1 \ldots p\}\}\} & \{\text{by properties of } \sqcup\} \\ & = & \sqcup(\{\sigma_r\} \cup \{\mathcal{S}_f\ e\ td + \sigma'_k \mid k \in \{1 \ldots p\}\}) & \{\text{by properties of } \sqcup\} \end{array}$$

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

The $p$ recursive calls are distributed among the $e_i$, so we assume a sequence $1 = p_0 \leq p_1 \leq \ldots \leq p_n = p$ such that there are $p_j - p_{j-1}$ recursive calls in the $j$-th branch. If we denote by $\sigma_j$ ($j \in \{1 \ldots n\}$) the result of applying the abstract interpretation rules to the $j$-th branch, we get:

$$\begin{array}{lcll} \sigma & = & \sqcup_{j=1}^n\{n_j + \sigma_j\} & \{\text{by rules of Fig.3}\} \\ & \sqsubseteq & \sqcup_{j=1}^n\{n_j + \sqcup(\{\sigma_{j,r}\} \cup \{\mathcal{S}_f\ e_j\ (td + n_j) + \sigma'_k \mid k \in \{p_{j-1} \ldots p_j\}\})\} & \{\text{by I.H.}\} \\ & = & \sqcup\{\sqcup_{j=1}^n\{n_j + \sigma_{j,r}\}, \sqcup_{j=1}^n\{n_j + \mathcal{S}_f\ e_j\ (td + n_j) + \sigma'_k \mid k \in \{p_{j-1} \ldots p_j\}\}\} & \{\text{by properties of } \sqcup\} \\ & \sqsubseteq & \sqcup\{\sigma_r, \sqcup_{j=1}^n\{\mathcal{S}_f\ e\ td + \sigma'_k \mid k \in \{p_{j-1} \ldots p_j\}\}\} & \{\text{by defs. of } \mathcal{S}_f \text{ and } \sigma_r\} \\ & = & \sqcup(\{\sigma_r\} \cup \{\mathcal{S}_f\ e\ td + \sigma'_k \mid k \in \{1 \ldots p\}\}) & \{\text{by properties of } \sqcup\} \end{array}$$

□

**Lemma 27.** *If $\sigma_f$ is as computed in the algorithm above, then $\mathbb{I}_\sigma(\sigma_f) \sqsubseteq \sigma_f$*

*Proof.* If $f$ is not recursive, we get:

$$\mathbb{I}_\sigma(\sigma_f) = [\![e]\!]\ \Sigma[f \mapsto (\_, \_, \sigma)]\ \Gamma\ (n + m) = [\![e]\!]\ \Sigma[f \mapsto (\_, \_, 0)]\ \Gamma\ (n + m) = \sigma \sqsubseteq \sigma_f$$

```
length []     = 0
length (x:xs) = 1 + length xs

merge []     ys = ys
merge (x:xs) [] = x : xs
merge (x:xs) (y:ys)
  | x <= y = x : merge xs (y:ys)
  | x > y  = y : merge (x:xs) ys
```

$$
\begin{array}{ll}
splitAt & :: Int \to [a]@\rho_1 \to \rho_1 \to \rho_2 \to \rho_3 \to ([a]@\rho_2, [a]@\rho_1)@\rho_3 \\
length & :: [a]@\rho_1 \to Int \\
merge & :: [a]@\rho_1 \to [a]@\rho_1 \to \rho_1 \to [a]@\rho_1 \\
msort & :: [a]@\rho_1 \to \rho_1 \to \rho_2 \to [a]@\rho_2
\end{array}
$$

```
msort []     = []
msort (x:[]) = x:[]
msort xs     = merge (msort xs1) (msort xs2)
               where (xs1,xs2) = splitAt (length xs / 2) xs
```

Figure 11: *Full-Safe* mergesort program

Let us assume that there are $p \geq 1$ recursive calls in the function definition. For each $k \in \{1 \dots p\}$ we define $\sigma_f^{(k)}$, $len_f^{(k)}$ and $\sigma_f^{(k)}$ as follows.

$$\sigma_f^{(k)} \stackrel{\text{def}}{=} \lambda \overline{x}^n.\sigma_f\ (\overline{|a_{ki}|\ \overline{x}^n}^n) \qquad len_f^{(k)} \stackrel{\text{def}}{=} \lambda \overline{x}^n.len_f\ (\overline{|a_{ki}|\ \overline{x}^n}^n) \qquad \sigma_f^{(k)} \stackrel{\text{def}}{=} \lambda \overline{x}^n.\sigma_f\ (\overline{|a_{ki}|\ \overline{x}^n}^n)$$

where the $|a_{ki}|$ are size functions previously inferred for each parameter in each recursive call. Then:

$$
\begin{array}{ll}
 & \mathbb{I}_\sigma(\sigma_f) \\
\sqsubseteq & \quad \{ \text{ by Lemma 26 } \} \\
 & \sqcup \Big( \{\sigma\} \cup \{\mathcal{S}_f\ e\ (n+m) + \sigma_f^{(k)} \mid k \in \{1 \dots p\}\} \Big) \\
= & \quad \{ \text{ by Lemma 20, } \mathcal{S}_f\ e\ (n+m) \geq 0 \ \} \\
 & \sqcup \Big( \{\sigma\} \cup \{\max\{0, \mathcal{S}_f\ e\ (n+m)\} + \sigma_f^{(k)} \mid k \in \{1 \dots p\}\} \Big) \\
= & \quad \{ \text{ by definition of } \sigma_f \ \} \\
 & \sqcup \Big( \{\sigma\} \cup \{\max\{0, \mathcal{S}_f\ e\ (n+m)\} + \max\{0, \mathcal{S}_f\ e\ (n+m)\} * (len_f^{(k)} - 1) + \sigma^{(k)} \mid k \in \{1 \dots p\}\} \Big) \\
= & \\
 & \sqcup \Big( \{\sigma\} \cup \{\max\{0, \mathcal{S}_f\ e\ (n+m)\} * len_f^{(k)} + \sigma^{(k)} \mid k \in \{1 \dots p\}\} \Big) \\
\sqsubseteq & \quad \{ \text{ since } 1 + len_f^{(k)} \sqsubseteq len_f \text{ for each } k \in \{1 \dots p\} \ \} \\
 & \sqcup \big( \{\sigma\} \cup \{\max\{0, \mathcal{S}_f\ e\ (n+m)\} * (len_f - 1) + \sigma^{(k)} \mid k \in \{1 \dots p\}\} \big) \\
\sqsubseteq & \quad \{ \text{ since } \sigma^{(k)} \sqsubseteq \sigma \text{ for each } k \in \{1 \dots p\} \ \} \\
 & \sqcup \big( \{\sigma\} \cup \{\max\{0, \mathcal{S}_f\ e\ (n+m)\} * (len_f - 1) + \sigma \mid k \in \{1 \dots p\}\} \big) \\
= & \quad \{ \text{ by properties of } \sqcup \ \} \\
 & \max\{0, \mathcal{S}_f\ e\ (n+m)\} * (len_f - 1) + \sigma \\
= & \quad \{ \text{ by definition of } \sigma_f \ \} \\
 & \sigma_f
\end{array}
$$

□

## 7. Case Studies

In Fig. 11 we show a *Full-Safe* version of the mergesort algorithm (the code for `splitAt` was presented in Fig. 6) with the types inferred by the compiler. Region $\rho_1$ is used inside `msort` for the internal call `splitAt n' xs @ r1 r1 self`, while region $\rho_2$ receives the charges made by `merge`. Notice that some charges to `msort`'s *self* region are made by `splitAt`. In Fig. 12 we show the results of our interpretation for this program as functions of the argument sizes. Remember that the size of a list (the number of its cells) is the list length plus one. The

| Function | Heap charges $\Delta$ | Heap needs $\mu$ | Stack needs $\sigma$ |
|---|---|---|---|
| $length(x)$ | $[\,]$ | $0$ | $5x-4$ |
| $splitAt(n,x)$ | $\left[\begin{array}{l}\rho_1 \mapsto 1\\ \rho_2 \mapsto \min(n,x-1)+1\\ \rho_3 \mapsto \min(n,x-1)+1\end{array}\right]$ | $2\min(n,x-1)+6$ | $9\min(n,x-1)+4$ |
| $merge(x,y)$ | $\left[\rho_1 \mapsto \max(1,2x+2y-5)\right]$ | $\max(1,2x+2y-5)$ | $11(x+y-4)+20$ |
| $msort^1(x)$ | $\left[\begin{array}{l}\rho_1 \mapsto \frac{x^2}{2}-\frac{1}{2}\\ \rho_2 \mapsto 2x^2-3x+3\end{array}\right]$ | $0.31x^2+0.25x\log(x+1)+14.3x$ $+0.75\log(x+1)+10.3$ | $\max(80,13x-10)$ |
| $msort^2(x)$ | $\left[\begin{array}{l}\rho_1 \mapsto \frac{x^2}{4}+x-\frac{1}{4}\\ \rho_2 \mapsto x^2+x+1\end{array}\right]$ | $0.31x^2+8.38x+13.31$ | $\max(80,11x-25)$ |
| $msort^3(x)$ | $\left[\begin{array}{l}\rho_1 \mapsto \frac{x^2}{8}+\frac{7x}{4}+\frac{9}{8}\\ \rho_2 \mapsto \frac{x^2}{2}+4x+\frac{1}{2}\end{array}\right]$ | $0.31x^2+8.38x+13.31$ | $\max(80,11x-25)$ |

Figure 12: Cost results for the mergesort program

| Function | Heap needs $\mu$ | Stack needs $\sigma$ |
|---|---|---|
| $partition(p,x)$ | $3x-1$ | $9x-5$ |
| $append(x,y)$ | $x-1$ | $\max(8,7x-6)$ |
| $quicksort(x)$ | $3x^2-20x+76$ | $\max(40,20x-27)$ |
| $insertD(e,x)$ | $1$ | $9x-1$ |
| $insertTD(x,t)$ | $2$ | $\frac{11}{2}t+\frac{7}{2}$ |
| $fib(n)$ | $2^n+2^{n-3}+2^{n-4}-3$ | $\max(10,7n-11)$ |
| $sum(n)$ | $0$ | $3n+6$ |
| $sumT(a,n)$ | $0$ | $5$ |

Figure 13: Cost results for miscellaneous *Safe* functions

functions shown have been simplified with the help of a computer algebra tool. We show the fixpoints framed in grey. The upper bounds obtained for `length`, `splitAt`, and `merge` are exact and they are, as expected, fixpoints of the inference algorithm. For `msort` we show three iterations for $\Delta$ and $\sigma$, and another three for $\mu$ by using the last $\Delta$. The upper bounds for $\Delta$ and $\mu$ are clearly over-approximated, since a term in $x^2$ arises which is beyond the actual space complexity class $O(x \log x)$ of this function. Let us note that the quadratic term's coefficient quickly decreases at each iteration in the inference of $\Delta$. Also, $\mu$ and $\sigma$ decrease in the second iteration but not in the third. This confirms the predictions of lemmas **??** and 18.

We have tried some more examples and the results inferred for $\mu$ and $\sigma$ after a maximum of three iterations are shown in Fig. 13, where the fixpoints are also framed in grey. There is a `quicksort` function using two auxiliary functions `partition` and `append` respectively classifying the list elements into those lower (or equal) and greater than the pivot, and appending two lists. We also show the destructive `insertD` function of Sec. 2, and a destructive version of the insertion in a search tree (its code is shown in Fig. 14). Both consume constant heap space. The next one shown is the usual Fibonacci function with exponential time cost, and using a constructed integer in order to show that an exponential heap space is inferred. Finally, we show two simple summation functions (its code also appears in Fig. 14), the first one being non-tail recursive, and the second being tail-recursive. Our abstract machine consumes constant stack space in the second case (see [11]). It can

```
sum 0 = 0
sum n = n + sum (n - 1)

sumT acc 0 = acc
sumT acc n = sumT (acc + n) (n - 1)
```

```
insertTD x Empty! = Node (Empty) x (Empty)
insertTD x (Node lt y rt)!
  | x == y = Node lt! y rt!
  | x > y  = Node lt! y (insertTD x rt)
  | x < y  = Node (insertTD x lt) y rt!
```

Figure 14: Two summation functions and a destructive tree insertion function

be seen that our stack inference algorithm is able to detect this fact.

## 8. Related and Future Work

Hughes and Pareto developed in [7] a type system and a type-checking algorithm which guarantees safe memory upper bounds in a region-based first-order functional language. Unfortunately, the approach requires the programmer to provide detailed consumption annotations, and it is limited to linear bounds. Hofmann and Jost's work [6] presents a type system and a type inference algorithm which, in case of success, guarantees linear heap upper bounds for a first-order functional language, and it does not require programmer annotations.

The national project AHA [15] aims at inferring amortised costs for heap space by using a variant of sized-types [8] in which the annotations are polynomials of any degree. They address two novel problems: polynomials are not necessarily monotonic and they are *exact* bounds, as opposed to approximate upper bounds. Type-checking is undecidable in this system and in [14, 16] they propose an inference algorithm for a list-based functional language with severe restrictions in which a combination of testing and type-checking is done. The algorithm does not terminate if the input-output size relation is not polynomial.

In [2], the authors directly analyse Java bytecode and compute safe upper bounds for the heap allocation made by a program. The approach uses the results of [1], and consists of combining a code transformation to an intermediate representation, a cost relations inference step, and a cost relations solving step. The second one combines ranking functions inference and partial evaluation. The results are impressive and go far beyond linear bounds. The authors claim to deal with data structures such as lists and trees, as well as arrays. Two drawbacks compared to our results are that the second step performs a global program analysis (so, it lacks modularity), and that only the allocated memory (as opposed to the live memory) is analysed. Very recently [3] they have added an escape analysis to each method in order to infer live memory upper bounds. The new results are very promising.

The strengths of our approach can be summarised as follows: (a) It scales well to large programs as each *Safe* function is separately inferred. The relevant information about the called functions is recorded in the signature environment; (b) We can deal with any user-defined algebraic datatype. Most of other approaches are limited to lists; (c) We get upper bounds for the *live* memory, as the inference algorithms take into account the deallocation of dead regions made at function termination; (d) We can get bounds of virtually any complexity class; and (e) It is to our knowledge the only approach in which the upper bounds can be easily improved just by iterating the inference algorithm.

The weak points that still require more work are the restrictions we have imposed to our functions: they must be non-negative and monotonic. This exclude some interesting functions such as those that destroy more memory than they consume, or those whose output size decreases as the input size increases. Another limitation is that the arguments of recursive *Safe* functions related to heap or stack consumption must be non-increasing. This limitation could be removed in the future by an analysis similar to that done in [1] in which they maximise the argument sizes across a call-tree by using linear programming tools. Of course, this could only be done if the size relations are linear.

Another open problem is inferring *Safe* functions with region-polymorphic recursion. Our region inference algorithm [13] frequently infers such functions, where the regions used in an internal call may differ from those used in the external one. This feature is very convenient for maximising garbage (i.e. allocations to the *self* region) but it makes more difficult the attribution of costs to regions.

**References**


[1] E. Albert, P. Arenas, S. Genaim, and G. Puebla. Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis. In *Static Analysis Symposium, SAS'08*, pages 221–237. LNCS 5079, Springer, 2008.

[2] E. Albert, S. Genaim, and M. Gómez-Zamalloa. Heap Space Analysis for Java Bytecode. In *Proc. Int. Symp. on Memory Management, ISMM'07, Montreal, Canada*, pages 105–116. ACM, 2007.

[3] E. Albert, S. Genaim, and M. Gómez-Zamalloa. Live Heap Space Analysis for Languages with Garbage Collection. In *Proc. Int. Symp. on Memory Management, ISMM'09, Dublin, Ireland*, pages 129–138. ACM, 2009.

[4] J. de Dios and R. Peña. A Certified Implementation on top of the Java Virtual Machine. In *Formal Method in Industrial Critical Systems, FMICS'09, Eindhoven (The Netherlands)*, pages 1–16. LNCS (to appear), Springer, November 2009.

[5] J. de Dios and R. Peña. Formal Certification of a Resource-Aware Language Implementation. In *Int. Conf. on Theorem Proving in Higher Order Logics, TPHOL'09, Munich (Germany)*, pages 1–15. LNCS 5674 (to appear), Springer, August 2009.

[6] M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. In *Proc. 30th ACM Symp. on Principles of Programming Languages, POPL'03*, pages 185–197. ACM Press, 2003.

[7] R. J. M. Hughes and L. Pareto. Recursion and Dynamic Data-Structures in Bounded Space; Towards Embedded ML Programming. In *Proc. Int. Conf. on Functional Programming, ICFP'99, Paris*, pages 70–81. ACM Press, Sept. 1999.

[8] R. J. M. Hughes, L. Pareto, and A. Sabry. Proving the Correctness of Reactive Systems Using Sized Types. In *Conference Record of POPL '96: The* 23rd *ACM SIGPLAN-SIGACT*, pages 410–423, 1996.

[9] T. Lindholm and F. Yellin. *The Java Virtual Machine Sepecification Second Edition*. The Java Series. Addison-Wesley, 1999.

[10] S. Lucas and R. Peña. Rewriting Techniques for Analysing Termination and Complexity Bounds of SAFE Programs. In *Proc. Logic-Based Program Synthesis and Transformation, LOPSTR'08, Valencia, Spain*, pages 43–57, July 2008.

[11] M. Montenegro, R. Peña, and C. Segura. A Resource-Aware Semantics and Abstract Machine for a Functional Language with Explicit Deallocation. In *Workshop on Functional and (Constraint) Logic Programming, WFLP'08, Siena, Italy July, 2008 (to appear in ENTCS)*, pages 47–61, 2008.

[12] 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.

[13] M. Montenegro, R. Peña, and C. Segura. A simple region inference algorithm for a first-order functional language. In S. Escobar, editor, *Int. Work. on Functional and (Constraint) Logic Programming, WFLP 2009, Brasilia*, pages 63–77, 2009.

[14] A. Tamalet, O. Shkaravska, and M. van Eekelen. Size Analysis of Algebraic Data Types. In Peter Achten, Pieter Koopman, and Marco T. Morazán, editors, *Trends in Functional Programming Volume 9 (TFP'08)*, pages 33–48. Intellect, 2009.

[15] M. van Eekelen, O. Shkaravska, R. van Kesteren, B. Jacobs, E. Poll, and S. Smetsers. AHA: Amortized Space Usage Analysis. In *Selected Papers Trends in Functional Programming, TFP'07, New York*, pages 36–53. Intellect, 2008.

[16] R. van Kesteren, O. Shkaravska, and M. van Eekelen. Inferring static non-monotonically sized types through testing. In *Proc. Work. on Functional and (Constraint) Logic Programming, WFLP'07, Paris, France*. ENTCS, Elsevier, 2007.

## Appendix A. Proofs

**Theorem 1** (Correctness of the abstract interpretation)**.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ a non-recursive context function, $\Gamma$ the inferred global type environment for $e_f$, $\Sigma$ containing correct signatures for all the functions called from $e_f$, an initial environment $E_0$ and a heap $h_0$ such that judgement (2) is derivable. For each subexpression $e$ of $e_f$ and $E$, $td$, $\Delta$, $\mu$, $\sigma$, $h$, $,h'$, $v$, $,t$, $\delta$, $m$, $s$ such that:*

1. *$[\![e]\!]\ \Sigma\ \Gamma\ td = (\Delta, \mu, \sigma)$, where every occurrence of $|x|$ in its derivation has been inferred with a correct size analysis.*
2. *$E \vdash h, k_0, td, e \Downarrow h', k_0, v, (\delta, m, s)$, belongs to the derivation of (2)*

*then $\Delta \succeq_{\overline{s_i}^n, k_0, \phi} \delta$, $\mu \succeq_{\overline{s_i}^n} m$ and $\sigma \succeq_{\overline{s_i}^n} s$, where $s_i = size(h, E_0\ x_i)$ for each $i \in \{1 \dots n\}$, and the consistent region instantiation $\phi$ determined by $E$ and $\Gamma$, i.e. $\phi = E \cdot \Gamma^{-1}$.*

*Proof.* By structural induction on $e$. In the following we shall leave out the $\overline{s_i}^n$ and $k_0$ subscripts in the $\succeq$ relations for a better readability.

$e \equiv c, x$ We get $\Delta = [\,]_f = \lambda\rho.\lambda\overline{x_i}^n.0$, $\mu = \lambda\overline{x_i}^n.0$ and $\sigma = \lambda\overline{x_i}^n.1$. We prove:

1. $\Delta \succeq \delta$
   Since for every $i \in \{0 \dots k_0\}$ we get:
   $$\sum_{\phi\ \rho=i} \Delta\ \rho\ \overline{s_i}^n = 0 = \delta\ i$$
2. $\mu \succeq m$, since $\mu\ \overline{s_i}^n = 0 = m$
3. $\sigma \succeq s$, since $\sigma\ \overline{s_i}^n = 1 = s$

$e \equiv C\overline{a_i}^l @ r, a_1 \oplus a_2$ These cases are also trivial.

$e \equiv g\ \overline{a_i}^l\ @\ \overline{r'_j}^q$ We shall assume that $\Sigma\ g \equiv g\ \overline{y_i}^l\ @\ \overline{r''_j}^q = e_g$ and, by using the corresponding rule:

$$E_g \vdash h, k_0 + 1, l + q, e_g \Downarrow h', k_0 + 1, v, (\delta_g, m_g, s_g)$$
$$\text{where } E_g = \left[\overline{y_i \mapsto E\ a_i}^l, \overline{r''_j \mapsto E\ r'_j}^q, self \mapsto k_0 + 1\right]$$

We can assume signature $(\Delta_g, \mu_g, \sigma_g)$ for function $g$ is correct. Function $g$ is well-typed and if $\Gamma\ g = \forall\overline{\alpha}\ \overline{\rho}.\overline{t_i}^l \to \overline{\rho_j}^q \to t$ then the global type environment $\Gamma_g = \Gamma' + [\overline{y_i : t_i}^l, \overline{r''_j : \rho_j}^q, self : \rho_{self}]$ has been inferred for $e_g$. On the other hand, if $s_{i,g}$ denote the size of the $i$-th actual argument before evaluating the function's body (i.e. $\forall i \in \{1 \dots l\} : s_{i,g} = size(h, E_g\ y_i)$) then:

$$\Delta_g \succeq_{\overline{s_{i,g}}^l, k_0, \phi'} \delta_g|_{k_0} \qquad \mu_g \succeq_{\overline{s_{i,g}}^l} m_g \qquad \sigma_g \succeq_{\overline{s_{i,g}}^l} s$$

where $\phi' = E_g \cdot \Gamma_g^{-1}$. Now we prove:

1. $\Delta \succeq_{\overline{s_i}^n,k_0,\phi} \delta$. Let $i \in \{0 \ldots k_0\}$. By Definition 4 we get for each $i \in \{1 \ldots l\}$

$$|a_i|\ \overline{s_i}^n \geq size(h, E\ a_i) = size(h, E_g\ y_i) = s_{i,g} \tag{A.1}$$

which is always positive, i.e. $|a_i|\ \overline{s_i}^n \neq -\infty$. So, by the definition of $\Delta$:

$$\sum_{\phi\ \rho=i} \Delta\ \rho\ \overline{s_i}^n = \sum_{\phi\ \rho=i}\ \sum_{\theta\ \rho'=\rho} \Delta_g\ \rho'\ \overline{|a_i|\ \overline{s_i}^n}^l$$

where $\theta = unify\ \Gamma\ g\ \overline{r'_j}^q$.
Because of the monotonicity of $\Delta_g\ \rho$ for every $\rho$:

$$\sum_{\phi\ \rho=i} \Delta\ \rho\ \overline{s_i}^n \geq \sum_{\phi\ \rho=i}\ \sum_{\theta\ \rho'=\rho} \Delta_g\ \rho'\ \overline{s_{i,g}}^l = \sum_{(\phi\cdot\theta)\ \rho'=i} \Delta_g\ \rho'\ \overline{s_{i,g}}^l$$

By definition of $\Delta_g \succeq_{h,k_0,(\phi\cdot\theta)} \delta_g|_{k_0}$ and because of the fact that $i \neq k_0 + 1$, we can get the desired result:

$$\sum_{\phi\ \rho=i} \Delta\ \rho\ \overline{s_i}^n \geq \delta_g|_{k_0}\ i = \delta\ i$$

provided the involved region instantiation $\phi \cdot \theta = \phi'$. For each $j \in \{1..q\}$:

$$\begin{array}{rcll} \phi\ (\theta\ \rho_j) & = & E\ (\Gamma^{-1}\ (\Gamma\ r'_j)) & \{\text{by definition of } \phi \text{ and } \theta\} \\ & = & E\ r'_j & \\ & = & E_g\ r''_j & \{\text{by definition of } E_g\} \\ & = & E_g\ (\Gamma_g^{-1}\ \rho_j) & \{\text{by definition of } \Gamma_g\} \end{array}$$

$$\begin{array}{rcll} \mu\ \overline{s_i}^n & = & \mu_g\ \overline{|a_i|\ \overline{s_i}^n}^l & \\ & \geq & \mu_g\ \overline{s_{i,g}}^l & \{\text{because of (A.1) and monotonicity of } \mu_g\} \\ & \geq & m_g & \{\text{since } \mu_g \succeq_{\overline{s_{i,g}}^l} m_g\} \\ & = & m & \end{array}$$

2. $\sigma \succeq s$. Similarly, for $\sigma_g$ being monotonic:

$$\begin{array}{rcl} \sigma\ \overline{s_i}^n & = & \sqcup\ \{l + q, \sigma_g\ \overline{(|a_i|\ \overline{s_j}^n)}^l - td + l + q\} \\ & \geq & \sqcup\ \{l + q, \sigma_g\ \overline{s_{j,g}}^l - td + l + q\} \\ & \geq & \sqcup\ \{l + q, \sigma_g\ s_1 - td + l + q\} \\ & = & s \end{array}$$

$e \equiv$ **let** $x_1 = e_1$ **in** $e_2$ Let us assume that, by the corresponding rules, we get $E \vdash h, k_0, 0, e_1 \Downarrow h_2, k_0, v_1, (\delta_1, m_1, s_1)$ and $[\![e_1]\!]\ \Sigma\ \Gamma\ 0 = (\Delta_1, \mu_1, \sigma_1)$ for some $\delta_1$, $m_1$, $s_1$, $\Delta_1$, $\mu_1$ and $\sigma_1$. In this case the induction hypothesis can be applied on $e_1$, so as to get:

$$\Delta_1 \succeq_{\overline{s_i}^n,k_0,\phi} \delta_1 \qquad \mu_1 \succeq_{\overline{s_i}^n} m_1 \qquad \sigma_1 \succeq_{\overline{s_i}^n} s_1 \tag{A.2}$$

where $\phi = E \cdot \Gamma^{-1}$.

Similarly, we apply the induction hypothesis on $e_2$, in order to obtain:

$$\Delta_2 \succeq_{\overline{s_i}^n,k_0,\phi'} \delta_2 \qquad \mu_2 \succeq_{\overline{s_i}^n} m_2 \qquad \sigma_2 \succeq_{\overline{s_i}^n} s_2 \tag{A.3}$$

where $\phi' = E_2 \cdot \Gamma^{-1} = \phi$.

Now the results in (A.2) and (A.3) are combined in order to get the desired result:

1. $\Delta \succeq \delta$. For each $i \in \{0 \ldots k_0\}$:

$$\begin{array}{rcl} \sum\limits_{\phi\ \rho=i} (\Delta_1 + \Delta_2)\ \rho\ \overline{s_i}^n & = & \sum\limits_{\phi\ \rho=i} (\Delta_1\ \rho\ \overline{s_i}^n + \Delta_2\ \rho\ \overline{s_i}^n) \\ & = & \sum\limits_{\phi\ \rho=i} (\Delta_1\ \rho\ \overline{s_i}^n) + \sum\limits_{\phi\ \rho=i} (\Delta_2\ \rho\ \overline{s_i}^n) \\ & \geq & (\delta_1\ i) + (\delta_2\ i) \\ & = & \delta\ i \end{array}$$

2. $\mu \succeq m$. For every $\rho \in dom\ \Delta_1$ there exists an $i \in \{0 \ldots k_0\}$ such that $\phi\ \rho = i$. This allows us to establish:

$$|\Delta_1|\ \overline{s_i}^n = \sum_{\rho \in dom\ \Delta_1} \Delta_1\ \rho\ \overline{s_i}^n = \sum_{i=0}^{k_0} \sum_{\phi\ \rho=i} \Delta_1\ \rho\ \overline{s_i}^n \geq \sum_{i=0}^{k_0} \delta_1\ i = |\delta_1|$$

   Therefore:

$$\begin{array}{rcl} \mu\ \overline{s_i}^n & = & \sqcup\{\mu_1\ \overline{s_i}^n, |\Delta_1|\ \overline{s_i}^n + \mu_2\ \overline{s_i}^n\} \\ & \geq & \sqcup\{m_1, |\delta_1| + m_2\} \\ & = & m \end{array}$$

3. $\sigma \succeq s$. It follows trivially from the induction hypothesis:

$$\sigma\ \overline{s_i}^n = \sqcup\{2 + \sigma_1\ \overline{s_i}^n, 1 + \sigma_2\ \overline{s_i}^n\} \geq \sqcup\{2 + s_1, 1 + s_2\} = s$$

$e \equiv \textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \rightarrow e_i}^l$ We shall assume that the $r$-th branch is executed, that is, $h\ (E\ x) = (j, C_r\ \overline{v_i}^{n_r})$ for some $j$, $v_1$, …, $v_{n_r}$ and $r \in \{1 \ldots l\}$. Therefore the following judgements hold:

$$\begin{array}{l} [\![e_r]\!]\ \Sigma\ \Gamma\ (td + n_r) = (\Delta_r, \mu_r, \sigma_r) \\ E_r \vdash h, k_0, td + n_r, e_r \Downarrow h', k_0, v, (\delta_r, m_r, s_r) \end{array}$$

for some $\Delta_r$, $\mu_r$, $\sigma_r$, $\delta_r$, $m_r$, $s_r$ and where $E_r$ denote the extended environment:

$$E_r \quad = \quad E \cup [\overline{x_{rj} \mapsto v_j}^{n_r}]$$

From the induction hypothesis it follows that $\Delta_r \succeq_{h,k_0,\phi} \delta_r$, $\mu_r \succeq m_r$ and $\sigma_r \succeq s_r$, because $E_r \cdot \Gamma^{-1} = E \cdot \Gamma^{-1} = \phi$, which allows us to prove:

1. $\Delta \succeq \delta$. Let $i \in \{0 \ldots k_0\}$

$$\begin{array}{rcl} \sum\limits_{\phi\ \rho=i} (\Delta\ \rho\ \overline{s_i}^n) & = & \sum\limits_{\phi\ \rho=i} \left(\left(\sqcup_{i=1}^{l} \Delta_i\right)\ \rho\ \overline{s_i}^n\right) \\ & = & \sum\limits_{\phi\ \rho=i} \max\{\Delta_i\ \rho\ \overline{s_i}^n \mid 1 \leq i \leq l\} \\ & \geq & \sum\limits_{\phi\ \rho=i} \Delta_r\ \rho\ \overline{s_i}^n \\ & \geq & \delta_r\ i \\ & = & \delta\ i \end{array}$$

2. $\mu \succeq m$, since:

$$\begin{array}{rcl} \mu\ \overline{s_i}^n & = & \sqcup_{i=1}^{l} \mu_i\ \overline{s_i}^n \\ & = & \max\{\mu_i\ \overline{s_i}^n \mid 1 \leq i \leq l\} \\ & \geq & \mu_r\ \overline{s_i}^n \\ & \geq & m_r \\ & = & m \end{array}$$

3. $\sigma \succeq s$, since:

$$\begin{array}{rcl} \sigma\ \overline{s_i}^n & = & \sqcup_{i=1}^{l} (\sigma_i + n_i)\ \overline{s_i}^n \\ & = & \max\{\sigma_i\ \overline{s_i}^n + n_i \mid 1 \leq i \leq l\} \\ & \geq & \sigma_r\ \overline{s_i}^n + n_r \\ & \geq & s_r + n_r \\ & = & s \end{array}$$

□

**Lemma 4.** *Let* $(e_b, e_r) = \mathit{splitExp}_f\ e$. *Then,* $e_b \neq \#$ *and* $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta, m, s)$ *if and only if* $E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s)$ *such that there are no calls to* $f$ *in this derivation.*

*Proof.* Both implications can be proved by induction on the depth of the $\Downarrow$-derivation. We distinguish cases according to the structure of $e$ for ($\Leftarrow$) and $e_b$ for ($\Rightarrow$).

- Cases $c$, $x$, and $C\ \overline{a_i}^n @ r$ and $a_1 \oplus a_2$

  Both implications hold trivially by hypothesis, by applying the same operational semantics rule since $e = e_b$ in all these cases.

- Case $g\ \overline{a_i}^n\ @\ \overline{r_j}^m$

  ($\Leftarrow$) The absence of calls to $f$ in the whole $\Downarrow$-derivation forces $g$ to be distinct from $f$ and in this case the implication holds trivially by hypothesis, since $e = e_b$.

  ($\Rightarrow$) As $e_b \neq \#$, by definition of $\mathit{splitExp}_f$ again $g \neq f$ and $e_b = e$, so the implication holds by hypothesis and because there is not mutual recursion in the language.

- Case **let**

  ($\Leftarrow$) Let $e =$ **let** $x_1 = e_1$ **in** $e_2$. We get:

  $$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1) \tag{A.4}$$

  and

  $$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h, k, v, (\delta_2, m_2, s_2) \tag{A.5}$$

  with $\delta = \delta_1 + \delta_2$, $m = \max\{m_1, |\delta_1| + m_2\}$ and $s = \max\{2 + s_1, 1 + s_2\}$. We know that in the derivations of both (A.4) and (A.5) there are no calls to $f$. Let $(e_{1b}, e_{1r}) = \mathit{splitExp}_f\ e_1$ and $(e_{2b}, e_{2r}) = \mathit{splitExp}_f\ e_2$. By induction hypothesis $e_{1b} \neq \#$, $e_{2b} \neq \#$,

  $$E \vdash h, k, 0, e_{1b} \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1) \tag{A.6}$$

  and

  $$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2b} \Downarrow h, k, v, (\delta_2, m_2, s_2) \tag{A.7}$$

  Since both $e_{1b}$ and $e_{2b}$ are nonempty we get $e_b =$ **let** $x_1 = e_{1b}$ **in** $e_{2b} \neq \#$, and from the judgements (A.6) and (A.7) we can derive $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta, m, s)$.

  ($\Rightarrow$) Let $e_b =$ **let** $x_1 = e_{1b}$ **in** $e_{2b}$. By definition of $\mathit{splitExp}_f$, $e =$ **let** $x_1 = e_1$ **in** $e_2$ where $(e_{1b}, \_) = \mathit{splitExp}_f\ e_1$ and $(e_{2b}, \_) = \mathit{splitExp}_f\ e_2$, and $e_{1b}, e_{2b} \neq \#$. Similarly to the proof of ($\Leftarrow$), this implication holds by applying induction hypothesis.

- Case **case**

  ($\Leftarrow$) Let $e =$ **case** $x$ **of** $\overline{alt_i}^n$, where $alt_i = C_i\ \overline{x_{ij}}^{n_i} \rightarrow e_i$. Assume $E(x) = p$ and $h(p) = (j, C_r\ \overline{v_j}^{n_r})$ for some $r \in \{1 \ldots n\}$. By the rule [*Case*] we get:

  $$E \cup [\overline{x_{rj} \mapsto v_j}^{n_r}] \vdash h_r, k, td + n_r, e_r \Downarrow h', k, v, (\delta_r, m_r, s_r)$$

where the relationships between $h$, $\delta$, $m$, $s$ and $h_r$, $\delta_r$, $m_r$, $s_r$ are given by the corresponding rule $[Case]$. Let $(e_{rb}, e_{rr}) = splitExp_f\ e_r$. Since in the derivation above for $e_r$ there is no call to $f$, we can apply the induction hypothesis in order to ensure that $e_{rb} \neq \#$ and that:

$$E \cup [\overline{x_{rj} \mapsto v_j}^{n_r}] \vdash h_r, k, td + n_r, e_{rb} \Downarrow h', k, v, (\delta_r, m_r, s_r)$$

Moreover, and since $e_{rb} \neq \#$ we get $e_b = \mathbf{case}\ x\ \mathbf{of}\ \overline{alt_{ib}}^n \neq \#$ and we can derive $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta, m, s)$ by applying the same rule $[Case]$.

**(⇒)** Let $e_b = \mathbf{case}\ x\ \mathbf{of}\ \overline{alt_{ib}}^n$, where $alt_{ib} = C_i\ \overline{x_{ij}}^{n_i} \rightarrow e_{ib}$. By definition of $splitExp_f$, $e = \mathbf{case}\ x\ \mathbf{of}\ \overline{alt_i}^n$ such that $(alt_{ib}, \_) = splitAlt_f\ alt_i$ for each $i \in \{1..n\}$ and there exists at least one $s \in \{1..n\}$ such that $alt_{sb} \neq \#$.

By rule $[Case]$, there exists $r \in \{1..n\}$ such that:

$$E \cup [\overline{x_{rj} \mapsto v_j}^{n_r}] \vdash h_r, k, td + n_r, e_{rb} \Downarrow h', k, v, (\delta_r, m_r, s_r)$$

There is no operational rule for an empty expression, which implies that $e_{rb}$ must be non-empty. By applying induction hypothesis on alternative $r$ we get the desired implication, in a similar way to (⇐).

□

**Lemma 5.** *Let* $(e_b, e_r) = \mathit{splitExp}_f\ e$. *Then,* $e_r \neq \#$ *and* $E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s)$ *such that there is at least one direct call to* $f$ *in this derivation if and only if* $E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s)$ *such that there is at least one direct call to* $f$ *in this derivation.*

*Proof.* Both implications can be proved by induction on the depth of the $\Downarrow$-derivation. We distinguish cases according to the structure of $e$ for ($\Leftarrow$) and $e_r$ for ($\Rightarrow$). For the proof of ($\Rightarrow$), we use the fact that the structure of $e_r$ is the same as the structure of $e$ with the exception of the $\sqcup$ case. But in this case we know that it always correspond to a **let** expression.

Cases $c$, $x$, $C\ \overline{a_i}^n @ r$, $a_1 \oplus a_2$ and $g\ \overline{a_i}^n\ @\ \overline{r_j}^m$ with $g \neq f$

These cases are trivial in both directions as the corresponding hypotheses are false.

Case $f\ \overline{a_i}^n\ @\ \overline{r_j}^m$

Both implications hold trivially by hypothesis, since $e = e_r$.

Case **let**

($\Leftarrow$) Let $e = \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$. By the operational semantics, we get:

$$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1) \tag{A.8}$$

and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h, k, v, (\delta_2, m_2, s_2) \tag{A.9}$$

with $\delta = \delta_1 + \delta_2$, $m = \max\{m_1, |\delta_1| + m_2\}$ and $s = \max\{2 + s_1, 1 + s_2\}$. Let $(e_{1b}, e_{1r}) = \mathit{splitExp}_f\ e_1$ and $(e_{2b}, e_{2r}) = \mathit{splitExp}_f\ e_2$. We know that in the derivations of either (A.8), or (A.9), or both, there are direct calls to $f$. Let us distinguish these three cases:

1. There are calls in (A.8). By the induction hypothesis we get $e_{1r} \neq \#$ and:

   $$E \vdash h, k, 0, e_{1r} \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

   with at least a direct call to $f$. As $e_{1r}$ is non-empty, $\mathit{splitExp}_f\ e$ gives either $e_r = \mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2$ or:

   $$e_r = \sqcup\{\mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2r},\ \mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2\}$$

   In both cases we get $e_r \neq \#$ and $E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s)$ with at least a direct call to $f$.
2. There are calls in (A.9) but not in (A.8). By the induction hypothesis $e_{2r} \neq \#$ . The reasoning is symmetrical to the previous case.

($\Rightarrow$) Let $e_r = \mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_{2r}$. As $e_r \neq \#$, we have to distinguish two cases.

$e_{1r} = \#$, $e_{2r} \neq \#$ In this case $e_{1r} = e_1$ and $(\_, e_{2r}) = \mathit{splitExp}_f\ e_2$. By hypothesis on $e_1$ and induction hypothesis on $e_{2r}$ we prove this implication in a similar way to ($\Leftarrow$).

$e_{1r} \neq \#$, $e_{2r} = \#$ In this case $e_{2r} = e_2$ and $(\_, e_{1r}) = \mathit{splitExp}_f\ e_1$. The reasoning is symmetrical to the previous case.

Case **case**

($\Leftarrow$) Let $e = \textbf{case}\ x\ \textbf{of}\ \overline{alt_i}^n$, where $alt_i = C_i\ \overline{x_{ij}}^{n_i} \to e_i$.
We assume $E(x) = p$ and $h(p) = (j, C_l\ \overline{v_j}^{n_r})$ for some $l \in \{1 \ldots n\}$. By the rule $[Case]$ we get:

$$E \cup [\overline{x_{lj} \mapsto v_j}^{n_l}] \vdash h_l, k, td + n_l, e_l \Downarrow h', k, v, (\delta_l, m_l, s_l)$$

where the relationships between $h$, $\delta$, $m$, $s$ and $h_l$, $\delta_l$, $m_l$, $s_l$ are given by the corresponding rule $[Case]$. Let $(e_{lb}, e_{lr}) = splitExp_f\ e_l$. Since in the derivation above for $e_l$ there are calls to $f$, we can apply the induction hypothesis on $e_l$ and get $e_{lr} \neq \#$ and:

$$E \cup [\overline{x_{lj} \mapsto v_j}^{n_l}] \vdash h_l, k, td + n_l, e_{lr} \Downarrow h', k, v, (\delta_l, m_l, s_l)$$

Moreover, and since $e_{lr} \neq \#$, by the definition of $splitExp_f$, we get $e_r = \textbf{case}\ x\ \textbf{of}\ \overline{alt_{ir}}^n$ and we can derive $E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s)$ by applying the same rule ($[Case]$.

($\Rightarrow$) Let $e_r = \textbf{case}\ x\ \textbf{of}\ \overline{alt_{ir}}^n$, where $alt_{ir} = C_i\ \overline{x_{ij}}^{n_i} \to e_{ir}$. By definition of $splitExp_f$, there exists $e = \textbf{case}\ x\ \textbf{of}\ \overline{alt_i}^n$ such that $(\_, alt_{ir}) = splitAlt_f\ alt_i$ for each $i \in \{1..n\}$ and there exists at least one $s \in \{1..n\}$ such that $alt_{sr} \neq \#$.
By rule $[Case]$, there exists $l \in \{1..n\}$ such that:

$$E \cup [\overline{x_{lj} \mapsto v_j}^{n_l}] \vdash h_l, k, td + n_l, e_{lr} \Downarrow h', k, v, (\delta_l, m_l, s_l)$$

There is no operational rule for an empty expression, which implies that $e_{lr}$ must be non-empty. By applying induction hypothesis on alternative $r$ we get the desired implication, in a similar way to ($\Leftarrow$).

**Case** $e_r = \sqcup \left\{ \begin{array}{l} \textbf{let}\ x_1 = e_{1b}\ \textbf{in}\ e_{2r} \\ \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2 \end{array} \right\}$

This case has no sense for ($\Leftarrow$). In this case $e = \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2$ where $(e_{1b}, e_{1r}) = splitExp_f\ e_1$, $(e_{2b}, e_{2r}) = splitExp_f\ e_2$ and both $e_{1r}$ and $e_{2r}$ are non-empty. By rule $[Lub]$

$$E \vdash h, k, td, \textbf{let}\ x_1 = e_{1b}\ \textbf{in}\ e_{2r} \Downarrow h', k, v, (\delta, m, s) \tag{A.10}$$

or

$$E \vdash h, k, td, \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2 \Downarrow h', k, v, (\delta, m, s) \tag{A.11}$$

Consider first the case when (A.10) holds. Then

$$E \vdash h, k, 0, e_{1b} \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2r} \Downarrow h, k, v, (\delta_2, m_2, s_2)$$

with $\delta = \delta_1 + \delta_2$, $m = \max\{m_1, |\delta_1| + m_2\}$ and $s = \max\{2 + s_1, 1 + s_2\}$. As there is no rule for an empty expression, $e_{1b}$ must be non-empty, so by Lemma 4:

$$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

As $e_{2r}$ is non-empty, by induction hypothesis

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h, k, v, (\delta_2, m_2, s_2)$$

and there is a call to $f$ in this derivation. So we can derive:

$$E \vdash h, k, td, \textbf{let } x_1 = e_1 \textbf{ in } e_2 \Downarrow h', k, v, (\delta, m, s)$$

and there is a call to $f$ in this derivation.
If (A.11) holds, the reasoning is similar. The difference is that we reason by induction on $e_{1r} \neq \#$ and by hypothesis on $e_2$. In this case we do not need Lemma 4.

□

**Lemma 6.** *For any expression $e$, if $splitExp_f\ e = (e_b, e_r)$, then the following properties hold:*

1. *If $e_b \neq \#$ then $splitExp_f\ e_b = (e_b, \#)$.*
2. *If $e_r \neq \#$ then $splitExp_f\ e_r = (\#, e_r)$*

*Proof.* By structural induction on $e$:

$e \equiv c, x,\ \overline{a_i}^n\ @\ r, a_1 \oplus a_2, g\ \overline{a_i}^n\ @\ \overline{r_j}^l$ with $g \neq f$ In these cases $splitExp_f\ e = (e, \#)$, so (1) trivially holds because $e_b = e$ and (2) holds because the premise is false.

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r_j}^l$ In this case $splitExp_f\ e = (\#, e)$, so (1) holds because the premise is false, and (2) holds trivially as $e_r = e$.

$e \equiv \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$ Assume $splitExp_f\ e_1 = (e_{1b}, e_{1r})$ and $splitExp_f\ e_2 = (e_{2b}, e_{2r})$.

We prove first (1). We have to distinguish two cases:

$e_{1b} = \#$ **or** $e_{2b} = \#$ By definition of $splitExp_f$, then $e_b = \#$ and the property holds because the premise does not hold.

$e_{1b} \neq \#$ **and** $e_{2b} \neq \#$ Then $e_b = \mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2b}$. By i.h. $splitExp_f\ e_{1b} = (e_{1b}, \#)$ and $splitExp_f\ e_{2b} = (e_{2b}, \#)$, so by definition we have $splitExp_f\ e_b = (e_b, \#)$.

Now we prove (2). We have to distinguish four cases:

$e_{1r} = \#$ **and** $e_{2r} = \#$ In this case $e_r = \#$ so the property holds because the premise does not hold.

$e_{1r} = \#$ **and** $e_{2r} \neq \#$ In this case $e_r = \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_{2r}$. By i.h. $splitExp_f\ e_{2r} = (\#, e_{2r})$. Consequently, we are in the first case of $splitExp_f$ for the base case and the second one for the recursive one, so $splitExp_f\ e_r = (\#, e_r)$.

$e_{1r} \neq \#$ **and** $e_{2r} = \#$**, or** $e_{1r} \neq \#$ **and** $e_{2r} \neq \#$ **and** $e_{1b} = \#$ The case $e_{1r} \neq \#$ and $e_{2r} = \#$ is symmetrical to the previous one.

If $e_{1r} \neq \#$ and $e_{2r} \neq \#$ and $e_{1b} = \#$. Then $e_r = \mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2$. By i.h. $splitExp_f\ e_{1r} = (\#, e_{1r})$, so we are in the first case of the base case and the third case of the recursive one, which implies that $splitExp_f\ e_r = (\#, e_r)$.

$e_{1r} \neq \#$ **and** $e_{2r} \neq \#$ **and** $e_{1b} \neq \#$ Now $e_r = \bigsqcup\{\mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2r}, \mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2\}$, so $splitExp_f\ e_r = \bigsqcup\{splitExp_f\ (\mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2r}), splitExp_f\ (\mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2)\}$.
By i.h. $splitExp_f\ e_{1r} = (\#, e_{1r})$ and $splitExp_f\ e_{2r} = (\#, e_{2r})$. By (1) $splitExp_f\ e_{1b} = (e_{1b}, \#)$. Consequently, by definition:

$$splitExp_f\ (\mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2r}) = (\#, \mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2r})$$

and

$$splitExp_f\ (\mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2) = (\#, \mathbf{let}\ x_1 = e_{1r}\ \mathbf{in}\ e_2)$$

which implies that $splitExp_f\ e_r = e_r$.

$e \equiv \mathbf{case}\ x\ \mathbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^n$ If $e_b \neq \#$ then by definition of $splitExp_f$ there is at least an alternative $i$ such that if $(e_{ib}, e_{ir}) = splitExp_f\ e_i$ then $e_{ib} \neq \#$. So the property holds by applying i.h. to all the alternatives of $e_b$. The same happens with $e_r$.

□

**Lemma 7.** *For any expression $e$, and for any $\Sigma$, $\Gamma$ and $td$, if $splitExp_f\ e = (e_b, e_r)$, then*

$$\begin{array}{rcl} [\![e]\!]_\Delta\ \Sigma\ \Gamma\ td & = & ([\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td) \\ [\![e]\!]_\sigma\ \Sigma\ \Gamma\ td & = & ([\![e_b]\!]_\sigma\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\sigma\ \Sigma\ \Gamma\ td) \end{array}$$

*Proof.* By structural induction on $e$.

$e \equiv c, x,\ \overline{a_i}^n\ @\ r, a_1 \oplus a_2, g\ \overline{a_i}^n\ @\ \overline{r_j}^l$ with $g \neq f$ In these cases $e_b = e$ and $e_r = \#$ so

$$\begin{array}{l} ([\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td) = ([\![e]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup [\,]_f = [\![e]\!]_\Delta\ \Sigma\ \Gamma\ td \\ ([\![e_b]\!]_\sigma\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\sigma\ \Sigma\ \Gamma\ td) = ([\![e]\!]_\sigma\ \Sigma\ \Gamma\ td) \sqcup 0 = [\![e]\!]_\sigma\ \Sigma\ \Gamma\ td \end{array}$$

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r_j}^l$ Now $e_b = \#$ and $e_r = e$, so again the properties hold trivially.

$e \equiv \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$ Assume $splitExp_f\ e_1 = (e_{1b}, e_{1r})$ and $splitExp_f\ e_2 = (e_{2b}, e_{2r})$. We have to distinguish several cases:

$e_{1b} = \#$ **or** $e_{2b} = \#$ In this case $e_b = \#$ so $[\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td = [\,]_f$ and $[\![e_b]\!]_\sigma\ \Sigma\ \Gamma\ td = 0$. As we have previously said, it is impossible that both $e_b$ and $e_r$ be $\#$, so we have the following three cases:

$e_{1r} = \#$ **and** $e_{2r} \neq \#$ This implies that $e_{1b} \neq \#$ and consequently $e_{2b} = \#$. For all $td$, by i.h.

$$\begin{array}{rcll} [\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ td & = & ([\![e_{2b}]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup ([\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ td) & \\ & = & [\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ td & \end{array} \tag{A.12}$$

$$\begin{array}{rcll} [\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ td & = & ([\![e_{2b}]\!]_\sigma\ \Sigma\ \Gamma\ td) \sqcup ([\![e_{2r}]\!]_\sigma\ \Sigma\ \Gamma\ td) & \\ & = & [\![e_{2r}]\!]_\sigma\ \Sigma\ \Gamma\ td & \end{array} \tag{A.13}$$

In this case $e_r = \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_{2r}$, so

$$\begin{array}{rcll} [\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td & = & ([\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) & \\ & = & ([\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) & \{\text{by (A.12)}\} \\ & = & [\![e]\!]_\Delta\ \Sigma\ \Gamma\ td & \end{array}$$

$$\begin{array}{rcll} [\![e_r]\!]_\sigma\ \Sigma\ \Gamma\ td & = & \sqcup\{2 + ([\![e_1]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_{2r}]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\} & \\ & = & \sqcup\{2 + ([\![e_1]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\} & \{\text{by (A.13)}\} \\ & = & [\![e]\!]_\sigma\ \Sigma\ \Gamma\ td & \end{array}$$

which implies the property.

$e_{1r} \neq \#$ **and** $e_{2r} = \#$**, or** $e_{1r} \neq \#$ **and** $e_{2r} \neq \#$ **and** $e_{1b} = \#$ In both cases $[\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ td = [\,]_f$ and $[\![e_{1b}]\!]_\sigma\ \Sigma\ \Gamma\ td = 0$ so

$$\begin{array}{rcl} [\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ td & = & ([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup ([\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ td) \\ & = & [\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ td \\ [\![e_1]\!]_\sigma\ \Sigma\ \Gamma\ td & = & ([\![e_{1b}]\!]_\sigma\ \Sigma\ \Gamma\ td) \sqcup ([\![e_{1r}]\!]_\sigma\ \Sigma\ \Gamma\ td) \\ & = & [\![e_{1r}]\!]_\sigma\ \Sigma\ \Gamma\ td \end{array}$$

and the proof is symmetrical to the previous one.

$e_{1r} \neq \#$ **and** $e_{2r} \neq \#$ **and** $e_{1b} \neq \#$ Now $e_r = \sqcup\{\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2r}, \textbf{let } x_1 = e_{1r} \textbf{ in } e_2\}$. Necessarily $e_{2b} = \#$ so by i.h. we get for all $td$:

$$[\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ td = [\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ td \qquad [\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ td = [\![e_{2r}]\!]_\sigma\ \Sigma\ \Gamma\ td \tag{A.14}$$

Then

$$\begin{array}{ll}
& [\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td \\
= & \sqcup\{([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)), ([\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1))\} \\
= & \sqcup\{([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)), ([\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1))\} & \{\text{by (A.14)}\} \\
= & \sqcup\{[\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0, [\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ 0\} + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) \\
= & ([\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) & \{\text{by i.h.}\} \\
= & [\![e]\!]_\Delta\ \Sigma\ \Gamma\ td
\end{array}$$

$$\begin{array}{ll}
& [\![e_r]\!]_\sigma\ \Sigma\ \Gamma\ td \\
= & \sqcup\{\sqcup\{2 + ([\![e_{1b}]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_{2r}]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\}, \\
& \quad \sqcup\{2 + [\![e_{1r}]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\}\} \\
= & \sqcup\{\sqcup\{2 + ([\![e_{1b}]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\}, \\
& \quad \sqcup\{2 + [\![e_{1r}]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\}\} & \{\text{by (A.14)}\} \\
= & \sqcup\{2 + \sqcup\{([\![e_{1b}]\!]_\sigma\ \Sigma\ \Gamma\ 0), ([\![e_{1r}]\!]_\sigma\ \Sigma\ \Gamma\ 0)\}, 1 + ([\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\} \\
= & \sqcup\{2 + ([\![e_1]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\} & \{\text{by i.h.}\} \\
= & [\![e]\!]_\sigma\ \Sigma\ \Gamma\ td
\end{array}$$

$e_{1b} \neq \#$ **and** $e_{2b} \neq \#$ Now $[\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td = ([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_{2b}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1))$. We have the following cases:

$e_{1r} = e_{2r} = \#$ In this case $[\![e_{1r}]\!]_\Delta\ \Sigma\ \Gamma\ td = [\![e_{2r}]\!]_\Delta\ \Sigma\ \Gamma\ td = [\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td = [\,]_f$ so by i.h.

$$[\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ td = [\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ td$$

and

$$[\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ td = [\![e_{2b}]\!]_\Delta\ \Sigma\ \Gamma\ td$$

The same applies for $\sigma$. So

$$\begin{array}{l}
([\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\Delta\ \Sigma\ \Gamma\ td) \\
= [\![e_b]\!]_\Delta\ \Sigma\ \Gamma\ td \\
= ([\![e_{1b}]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_{2b}]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) \\
= ([\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ 0) + ([\![e_2]\!]_\Delta\ \Sigma\ \Gamma\ (td+1)) \\
= [\![e]\!]_\Delta\ \Sigma\ \Gamma\ td
\end{array}$$

$$\begin{array}{l}
([\![e_b]\!]_\sigma\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_\sigma\ \Sigma\ \Gamma\ td) \\
= [\![e_b]\!]_\sigma\ \Sigma\ \Gamma\ td \\
= \sqcup\{2 + ([\![e_{1b}]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_{2b}]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\} \\
= \sqcup\{2 + ([\![e_1]\!]_\sigma\ \Sigma\ \Gamma\ 0), 1 + ([\![e_2]\!]_\sigma\ \Sigma\ \Gamma\ (td+1))\} \\
= [\![e]\!]_\sigma\ \Sigma\ \Gamma\ td
\end{array}$$

$e_{1r} = \#$ **and** $e_{2r} \neq \#$ In this case $e_r = \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2r}$, so

$$
\begin{aligned}
&(\llbracket e_b \rrbracket_\Delta\ \Sigma\ \Gamma\ td) \sqcup (\llbracket e_r \rrbracket_\Delta\ \Sigma\ \Gamma\ td) \\
&= \sqcup\{(\llbracket e_{1b} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_{2b} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)), (\llbracket e_1 \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_{2r} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1))\} \\
&= \sqcup\{(\llbracket e_1 \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_{2b} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)), (\llbracket e_1 \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_{2r} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1))\} \\
&\quad \{\text{by i.h. and } \llbracket e_{1r} \rrbracket_\Delta\ \Sigma\ \Gamma\ td = [\,]_f\} \\
&= (\llbracket e_1 \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + \sqcup\{\llbracket e_{2b} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1), \llbracket e_{2r} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)\} \\
&= (\llbracket e_1 \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_2 \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)) \qquad \{\text{by i.h.}\} \\
&= \llbracket e \rrbracket_\Delta\ \Sigma\ \Gamma\ td
\end{aligned}
$$

$$
\begin{aligned}
&(\llbracket e_b \rrbracket_\sigma\ \Sigma\ \Gamma\ td) \sqcup (\llbracket e_r \rrbracket_\sigma\ \Sigma\ \Gamma\ td) \\
&= \sqcup\{\sqcup\{2 + (\llbracket e_{1b} \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_{2b} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}, \\
&\qquad \sqcup\{2 + (\llbracket e_1 \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_{2r} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}\} \\
&= \sqcup\{\sqcup\{2 + (\llbracket e_1 \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_{2b} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}, \\
&\qquad \sqcup\{2 + (\llbracket e_1 \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_{2r} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}\} \\
&= \sqcup\{2 + (\llbracket e_1 \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + \sqcup\{(\llbracket e_{2b} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1)), 1 + (\llbracket e_{2r} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}\} \\
&= \sqcup\{2 + (\llbracket e_1 \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_2 \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\} \qquad \{\text{by i.h.}\} \\
&= \llbracket e \rrbracket_\sigma\ \Sigma\ \Gamma\ td
\end{aligned}
$$

$e_{1r} \neq \#$ **and** $e_{2r} = \#$ This case is symmetrical to the previous one. Notice that the case $e_{1r} \neq \#$, $e_{2r} \neq \#$, $e_{1b} = \#$ was already proved previously.

$e_{1r} \neq \#$ **and** $e_{2r} = \#$ Now $e_r = \sqcup\{\textbf{let}\ x_1 = e_{1b}\ \textbf{in}\ e_{2r}, \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2\}$.

$$
\begin{aligned}
&(\llbracket e_b \rrbracket_\Delta\ \Sigma\ \Gamma\ td) \sqcup (\llbracket e_r \rrbracket_\Delta\ \Sigma\ \Gamma\ td) \\
&= \sqcup\{(\llbracket e_{1b} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_{2b} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)), \\
&\qquad \sqcup\{(\llbracket e_{1b} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_{2r} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)), (\llbracket e_{1r} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_2 \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1))\}\} \\
&= \sqcup\{(\llbracket e_{1b} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + \sqcup\{\llbracket e_{2b} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1), \llbracket e_{2r} \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)\}, \\
&\qquad (\llbracket e_{1r} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_2 \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1))\} \\
&= \sqcup\{(\llbracket e_{1b} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_2 \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)), \\
&\qquad (\llbracket e_{1r} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_2 \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1))\} \qquad \{\text{by i.h.}\} \\
&= \sqcup\{\llbracket e_{1b} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0, \llbracket e_{1r} \rrbracket_\Delta\ \Sigma\ \Gamma\ 0\} + (\llbracket e_2 \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)) \\
&= (\llbracket e_1 \rrbracket_\Delta\ \Sigma\ \Gamma\ 0) + (\llbracket e_2 \rrbracket_\Delta\ \Sigma\ \Gamma\ (td+1)) \qquad \{\text{by i.h.}\} \\
&= \llbracket e \rrbracket_\Delta\ \Sigma\ \Gamma\ td
\end{aligned}
$$

$$
\begin{aligned}
&(\llbracket e_b \rrbracket_\sigma\ \Sigma\ \Gamma\ td) \sqcup (\llbracket e_r \rrbracket_\sigma\ \Sigma\ \Gamma\ td) \\
&= \sqcup\{\sqcup\{2 + (\llbracket e_{1b} \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_{2b} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}, \\
&\qquad \sqcup\{\sqcup\{2 + (\llbracket e_{1b} \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_{2r} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}, \\
&\qquad\quad \sqcup\{2 + (\llbracket e_{1r} \rrbracket_\sigma\ \Sigma\ \Gamma\ 0), 1 + (\llbracket e_2 \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1))\}\}\} \\
&= \sqcup\{2 + \sqcup\{\llbracket e_{1b} \rrbracket_\sigma\ \Sigma\ \Gamma\ 0, \llbracket e_{1r} \rrbracket_\sigma\ \Sigma\ \Gamma\ 0\}, \\
&\qquad 1 + \sqcup\{\llbracket e_{2b} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1), \llbracket e_{2r} \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1)\}, 1 + \llbracket e_2 \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1)\} \\
&= \sqcup\{2 + \llbracket e_1 \rrbracket_\sigma\ \Sigma\ \Gamma\ 0, 1 + \llbracket e_2 \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1), 1 + \llbracket e_2 \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1)\} \qquad \{\text{by i.h.}\} \\
&= \sqcup\{2 + \llbracket e_1 \rrbracket_\sigma\ \Sigma\ \Gamma\ 0, 1 + \llbracket e_2 \rrbracket_\sigma\ \Sigma\ \Gamma\ (td+1)\} \\
&= \llbracket e \rrbracket_\sigma\ \Sigma\ \Gamma\ td
\end{aligned}
$$

$e \equiv \textbf{case}\ x\ \textbf{of}\ \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^{n}$ This can easily be proved by applying i.h. to the alternatives. Notice that some

alternatives may be lost in $e_b$ and $e_r$ if the corresponding component is #:

$$
\begin{array}{ll}
[\![e]\!]_{\Delta}\ \Sigma\ \Gamma\ td & \\
= \bigsqcup_{i=1}^{n}([\![e_i]\!]_{\Delta}\ \Sigma\ \Gamma\ (td+n_i)) & \\
= \bigsqcup_{i=1}^{n}(([\![e_{ib}]\!]_{\Delta}\ \Sigma\ \Gamma\ (td+n_i)) \sqcup ([\![e_{ir}]\!]_{\Delta}\ \Sigma\ \Gamma\ (td+n_i))) & \{\text{by i.h.}\} \\
= (\bigsqcup_{i=1}^{n}([\![e_{ib}]\!]_{\Delta}\ \Sigma\ \Gamma\ (td+n_i))) \sqcup (\bigsqcup_{i=1}^{n}([\![e_{ir}]\!]_{\Delta}\ \Sigma\ \Gamma\ (td+n_i))) & \\
= (\bigsqcup_{i=1}^{n}\{[\![e_{ib}]\!]_{\Delta}\ \Sigma\ \Gamma\ (td+n_i) \mid e_{ib} \neq \#\}) \sqcup (\bigsqcup_{i=1}^{n}\{[\![e_{ir}]\!]_{\Delta}\ \Sigma\ \Gamma\ (td+n_i) \mid e_{ir} \neq \#\})) & \{\text{as } [\![\#]\!]_{\Delta}\ \Sigma\ \Gamma\ td = [\,]_f\} \\
= ([\![e_b]\!]_{\Delta}\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_{\Delta}\ \Sigma\ \Gamma\ td) &
\end{array}
$$

$$
\begin{array}{ll}
[\![e]\!]_{\sigma}\ \Sigma\ \Gamma\ td & \\
= \bigsqcup_{i=1}^{n}(n_i + ([\![e_i]\!]_{\sigma}\ \Sigma\ \Gamma\ (td+n_i))) & \\
= \bigsqcup_{i=1}^{n}(n_i + (([\![e_{ib}]\!]_{\sigma}\ \Sigma\ \Gamma\ (td+n_i)) \sqcup ([\![e_{ir}]\!]_{\sigma}\ \Sigma\ \Gamma\ (td+n_i)))) & \{\text{by i.h.}\} \\
= \bigsqcup_{i=1}^{n}((n_i + ([\![e_{ib}]\!]_{\sigma}\ \Sigma\ \Gamma\ (td+n_i))) \sqcup (n_i + ([\![e_{ir}]\!]_{\sigma}\ \Sigma\ \Gamma\ (td+n_i)))) & \\
= (\bigsqcup_{i=1}^{n}([\![e_{ib}]\!]_{\sigma}\ \Sigma\ \Gamma\ td)) \sqcup (\bigsqcup_{i=1}^{n}([\![e_{ir}]\!]_{\sigma}\ \Sigma\ \Gamma\ td)) & \\
= (\bigsqcup_{i=1}^{n}\{[\![e_{ib}]\!]_{\sigma}\ \Sigma\ \Gamma\ td \mid e_{ib} \neq \#\}) \sqcup (\bigsqcup_{i=1}^{n}\{[\![e_{ir}]\!]_{\sigma}\ \Sigma\ \Gamma\ td \mid e_{ir} \neq \#\})) & \{\text{as } [\![\#]\!]_{\sigma}\ \Sigma\ \Gamma\ td = 0\} \\
= ([\![e_b]\!]_{\sigma}\ \Sigma\ \Gamma\ td) \sqcup ([\![e_r]\!]_{\sigma}\ \Sigma\ \Gamma\ td) &
\end{array}
$$

□

**Lemma 8.** *Given an expression $e$, let $e_r = \pi_2(splitExp_f\ e)$ and let $(e_b, e_a) \in splitBA_f\ e_r$. Then:*

- $e_b \neq \square$.
- *If $e_a = \square$ then*

$$E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b) \Rightarrow E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta_b, m_b, s_b)$$

*Proof.* First item can be easily proved from the definition of $splitBA_f$. We prove the second item by structural induction on $e$.

$e \equiv c, x, C\ \overline{a_i}^n\ @\ r, a_1 \oplus a_2, g\ \overline{a_i}^n\ @\ \overline{r_j}^l$ with $g \neq f$ In these cases $e_r = \#$ and $splitBA_f\ e_r = []$, so the property holds trivially.

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r_j}^l$ Now, $e_r = f\ \overline{a_i}^n\ @\ \overline{r_j}^l$ and the unique pair in $splitBA_f\ e_r$ is $(f\ \overline{a_i}^n\ @\ \overline{r_j}^l, \square)$, so $e_r = e_b$ and the property holds trivially.

$e \equiv \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_2$ Assume $splitExp_f\ e_1 = (e_{1b}, e_{1r})$ and $splitExp_f\ e_2 = (e_{2b}, e_{2r})$.

We distinguish the following cases:

$e_{1r} = \#$ **and** $e_{2r} = \#$ In this case $e_r = \#$ and $splitBA_f\ e_r = [\,]$, so the property holds trivially.

$e_{1r} = \#$ **and** $e_{2r} \neq \#$ In this case $e_r = \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2r}$. By Lemma 6 $splitExp_f\ e_{2r} = (\#, e_{2r})$. As $e_{1r} = \#$, list $B$ is empty, so $splitBA_f\ e_r$ only contains pairs of list $A$, i.e. $e_b = \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2rb}$ where $(e_{2rb}, e_{2ra}) \in splitBA_f\ e_{2r}$. If $e_a = \square$, by definition of *collapse* then $e_{2ra} = \square$.
Assume $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b)$. By rule [*Let*], then

$$E \vdash h, k, 0, e_1 \Downarrow h', k, v_1, (\delta_1, m_1, s_1) \tag{A.15}$$

and

$$E \cup [x_1 \mapsto v_1] \vdash h', k, td + 1, e_{2rb} \Downarrow h'', k, v, (\delta_2, m_2, s_2) \tag{A.16}$$

where $(\delta_b, m_b, s_b) = (\delta_1 + \delta_2, \max\{m_1, |\delta_1| + m_2\}, \max\{2 + s_1, 1 + s_2\})$.
By i.h. and (A.16) we have that

$$E \cup [x_1 \mapsto v_1] \vdash h', k, td + 1, e_{2r} \Downarrow h'', k, v, (\delta_2, m_2, s_2) \tag{A.17}$$

So, applying rule [*Let*] to (A.15) and (A.17) we have that

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta_b, m_b, s_b)$$

$e_{1r} \neq \#$ **and** $e_{2r} = \#$**, or** $e_{1r} \neq \#$, $e_{2r} \neq \#$ **and** $e_{1b} = \#$ Consider first the case $e_{1r} \neq \#$ and $e_{2r} = \#$. In this case $e_r = \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2$. By Lemma 6 $splitExp_f\ e_{1r} = (\#, e_{1r})$. As $e_{2r} = \#$, list $A$ is empty, so $splitBA_f\ e_r$ only contains pairs of list $B$. But then $e_a \neq \square$ so the property holds trivially.
If $e_{1r} \neq \#$, $e_{2r} \neq \#$ and $e_{1b} = \#$, again $e_r = \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2$, but now both lists $A$ and $B$ are non-empty. However the only pairs whose second component may be $\square$ are those belonging to $A$, and the proof is the same as the previous item.

$e_{1r} \neq \#$ **and** $e_{2r} \neq \#$ **and** $e_{1b} \neq \#$ Now $e_r = \sqcup\{\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2r}, \textbf{let } x_1 = e_{1r} \textbf{ in } e_2\}$. By definition:

$$\begin{array}{l} \mathit{splitBA}_f\ e_r = \mathit{concat}\ [l_1, l_2] \\ \quad \text{where} \\ \quad l_1 = \mathit{splitBA}_f\ (\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2r}) \\ \quad l_2 = \mathit{splitBA}_f\ (\textbf{let } x_1 = e_{1r} \textbf{ in } e_2) \end{array}$$

So we have to distinguish two cases:

- If the pair belongs to $l_1$. By Lemma 6 we know that $\mathit{splitExp}_f\ e_{1b} = (e_{1b}, \#)$ and $\mathit{splitExp}_f\ e_{2r} = (\#, e_{2r})$. So $l_1$ only contains pairs of list $A$, i.e. $e_b = \textbf{let } x_1 = e_{1b} \textbf{ in } e_{2rb}$ and $e_{2ra} = \square$. Assume $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b)$. By rule $[\mathit{Let}]$, then

$$E \vdash h, k, 0, e_{1b} \Downarrow h', k, v_1, (\delta_1, m_1, s_1) \tag{A.18}$$

and

$$E \cup [x_1 \mapsto v_1] \vdash h', k, td + 1, e_{2rb} \Downarrow h'', k, v, (\delta_2, m_2, s_2) \tag{A.19}$$

where $(\delta_b, m_b, s_b) = (\delta_1 + \delta_2, \max\{m_1, |\delta_1| + m_2\}, \max\{2 + s_1, 1 + s_2\})$.
By i.h. and (A.19) we have that

$$E \cup [x_1 \mapsto v_1] \vdash h', k, td + 1, e_{2r} \Downarrow h'', k, v, (\delta_2, m_2, s_2) \tag{A.20}$$

So, applying rule $[\mathit{Let}]$ to (A.18) and (A.20) and then rule $[\mathit{Lub}]$ we have that:

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta_b, m_b, s_b)$$

- If the pair belongs to $l_2$. By Lemma 6 we know that $\mathit{splitExp}_f\ e_{1r} = (\#, e_{1r})$. In this case $l_2$ may contain pairs from $A$ and $B$ but the only ones whose second component may be $\square$ are those from $A$, so $e_b = \textbf{let } x_1 = e_{1r} \textbf{ in } e_{2rb}$ and $e_{2ra} = \square$.
The rest of the proof is similar to the previous case.

$e \equiv \textbf{case } x \textbf{ of } \overline{C_i\ \overline{x_{ij}}^{n_i} \to e_i}^{n}$ Here $e_r = \textbf{case } x \textbf{ of } \overline{C_k\ \overline{x_{kj}}^{n_k} \to e_{kr}}^{S}$, where only those alternatives such that $e_{kr} = \pi_2(\mathit{splitExp}_f\ e_k) \neq \#$ appear, i.e. $k \in S \subseteq \{1..n\}$. Then $e_b = \textbf{case } x \textbf{ of } \overline{C_k\ \overline{x_{kj}}^{n_k} \to \bigvee \overline{e_{krb}}}^{S}$ where $\overline{(e_{krb}, e_{kra})} = \mathit{splitBA}_f\ e_{kr}$

By definition of $\mathit{splitBA}_f$, if $e_a = \square$ then for all $k \in S$, and for any pair $(e_{krb}, e_{kra}) \in \mathit{splitBA}_f\ e_{kr}$, $e_{kra} = \square$.

Assume Assume $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b)$. By rule $[\mathit{Case}]$, if $E(x) = p$ and $h(p) = (j, C\ \overline{v_i}^{n})$ where $C = C_l$, $l \in S$, then

$$E \cup [\overline{x_{l_i} \mapsto v_i}^{n_l}] \vdash h, k, td + n_l, e_{lrb} \Downarrow h', k, v, (\delta_b, m_b, s)$$

where $s_b = s + n_l$.

As $e_{lra} = \square$, by i.h. we have that

$$E \cup [\overline{x_{l_i} \mapsto v_i}^{n_l}] \vdash h, k, td + n_l, e_{lr} \Downarrow h', k, v, (\delta_b, m_b, s) \tag{A.21}$$

By applying rule $[\mathit{Case}]$ to (A.21) we obtain $E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta_b, m_b, s_b)$.

$\square$

**Lemma 9.** *Given* $(e_b, e_a) \in splitBA_f\ e_r$ *such that* $E \vdash h, k, td, (e_b, e_a)_X \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$*, then*

- $X \neq \boldsymbol{A}$
- $X = \boldsymbol{B} \Rightarrow e_a = \square \wedge \delta_a = m_a = s_a = 0 \wedge E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b)$
- $X = \boldsymbol{M} \Rightarrow e_a \neq \square$

*Proof.* We prove first that $X \neq \mathbf{A}$. By inspection of the rules, the only one where $X = \mathbf{A}$ is rule [*After*]. However, by Lemma 8, if $(e_b, e_a) \in splitBA_f\ e_r$ then $e_b \neq \square$ so such rule cannot be applied.

Now we prove second item. By inspection of the rules, the only one where $X = \mathbf{B}$ is rule [*Before*], which can only be applied if $e_a = \square$ and $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b)$. Then $\delta_a = m_a = s_a = 0$.

Rules where $X = \mathbf{M}$ are [*LetP*] and [*CaseP*] and trivially $e_a$ is a **let** or a **case** expression but not a $\square$.

$\square$

**Theorem 2.** *Let* $e_r, E, h, h', k, td, v, X \in \{\boldsymbol{B}, \boldsymbol{M}\}, (e_b, e_a) \in \mathit{splitBA}_f\ e_r, \delta_b, \delta_a, m_b, m_a, s_b, s_a$ *such that*

$$E \vdash h, k, td, (e_b, e_a)_X \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$$

*Then,*

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s)$$

*where* (1) $\delta = \delta_b + \delta_a$, (2) $m = \max\{m_b, |\delta_b| + m_a\}$, (3) $s = \max\{s_b, s_a\}$

*Proof.* By induction on the depth $n$ of the derivation for $(e_b, e_a)_X$ and by cases on the last rule applied:

$n = 0, [\mathit{Before}]$ From the premise of the rule $E \vdash h, k, td, e_b \Downarrow h', k, v, (\delta_b, m_b, s_b)$. By Lemma 8 then $E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta_b, m_b, s_b)$. As $\delta_a = m_a = s_a = 0$ and $m_b \geq |\delta_b|$, we have $\delta_b = \delta$, $m_b = m$ and $s_b = s$.

$n = 0, [\mathit{After}]$ This case is not possible as $X \neq \mathbf{A}$.

$n > 0, [\mathit{LetP}]$ In this case $e_r = \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$. Assume $e_b = \mathbf{let}\ x_1 = e_{1b}\ \mathbf{in}\ e_{2b}$ and $e_a = \mathbf{let}\ x_1 = e_{1a}\ \mathbf{in}\ e_{2a}$. From the rule, it holds that there exist $X < Y$ such that:

$$E \vdash h, k, 0, (e_{1b}, e_{1a})_X \Downarrow h_1, k, v_1, (\delta_{1b}/\delta_{1a}, m_{1b}/m_{1a}, s_{1b}/s_{1a}) \tag{A.22}$$

and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, (e_{2b}, e_{2a})_Y \Downarrow h', k, v, (\delta_{2b}/\delta_{2a}, m_{2b}/m_{2a}, s_{2b}/s_{2a}) \tag{A.23}$$

where $\delta_b = \delta_{1b} + \delta_{2b}$, $\delta_a = \delta_{1a} + \delta_{2a}$, $m_b = \max\{m_{1b}, |\delta_{1b}| + m_{2b}\}$, $m_a = \max\{m_{1a}, |\delta_{1a}| + m_{2a}\}$, $s_b = \max\{2 + s_{1b}, 1 + s_{2b}\}$ and $s_a = \max\{2 + s_{1a}, 1 + s_{2a}\}$.

We distinguish two cases:

$e_{2b} \neq \square$ By definition of $\mathit{splitBA}_f$, pair $(e_b, e_a)$ belongs to set $A$ in the definition of $\mathit{splitBA}_f$, so $e_{1b} = e_1$, $e_{1a} = \square$, $(e_{2b}, e_{2a}) \in \mathit{splitBA}_f\ e_{2r}$ where $e_{2r} = \pi_2(\mathit{splitExp}_f\ e_2) \neq \#$. Then, $X = \mathbf{B}$, and $\delta_{1a} = m_{1a} = s_{1a} = 0$. By Lemma 9, $Y \neq \mathbf{A}$. Consequently, from (A.22) we obtain:

$$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_{1b}, m_{1b}, s_{1b})$$

and by induction hypothesis on (A.23) we obtain:

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2r} \Downarrow h', k, v, (\delta_2, m_2, s_2)$$

where $\delta_2 = \delta_{2b} + \delta_{2a}$, $m_2 = \max\{m_{2b}, |\delta_{2b}| + m_{2a}\}$ and $s_2 = \max\{s_{2b}, s_{2a}\}$. By Lemma 5

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h', k, v, (\delta_2, m_2, s_2)$$

By rule $[\mathit{Let}]$ then

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta', m', s')$$

where $\delta' = \delta_{1b} + \delta_2$, $m' = \max\{m_{1b}, |\delta_{1b}| + m_2\}$ and $s' = \max\{2 + s_{1b}, 1 + s_2\}$. Consequently:

$$\begin{array}{rcll} \delta' & = & \delta_{1b} + \delta_2 & \\ & = & \delta_{1b} + (\delta_{2b} + \delta_{2a}) & \\ & = & \delta_b + \delta_{2a} & \\ & = & \delta_b + \delta_a & \{\text{as } \delta_{1a} = 0\} \end{array}$$

Also

$$\begin{array}{rcll} m' & = & \max\{m_{1b}, |\delta_{1b}| + m_2\} & \\ & = & \max\{m_{1b}, |\delta_{1b}| + \max\{m_{2b}, |\delta_{2b}| + m_{2a}\}\} & \\ & = & \max\{m_{1b}, |\delta_{1b}| + m_{2b}, |\delta_{1b}| + |\delta_{2b}| + m_{2a}\} & \\ & = & \max\{\max\{m_{1b}, |\delta_{1b}| + m_{2b}\}, |\delta_{1b}| + |\delta_{2b}| + m_{2a}\} & \\ & = & \max\{m_b, |\delta_{1b}| + |\delta_{2b}| + \max\{m_{1a}, |\delta_{1a}| + m_{2a}\}\} & \\ & & \{\text{as } \delta_{1a} = m_{1a} = 0 \text{ and } m_{2a} \geq 0\} & \\ & = & \max\{m_b, |\delta_b| + m_a\} & \end{array}$$

and

$$\begin{array}{rcll} s' & = & \max\{2 + s_{1b}, 1 + s_2\} & \\ & = & \max\{2 + s_{1b}, 1 + \max\{s_{2b}, s_{2a}\}\} & \\ & = & \max\{2 + s_{1b}, 1 + s_{2b}, 1 + s_{2a}\} & \\ & = & \max\{2 + s_{1b}, 1 + s_{2b}, 2 + s_{1a}, 1 + s_{2a}\} & \{\text{as } s_{1a} = 0 \text{ and } s_{1b} \geq 0\} \\ & = & \max\{\max\{2 + s_{1b}, 1 + s_{2b}\}, \max\{2 + s_{1a}, 1 + s_{2a}\}\} & \\ & = & \max\{s_b, s_a\} & \end{array}$$

$e_{2b} = \square$ By definition of $splitBA_f$ and Lemma 8, pair $(e_b, e_a)$ belongs to set $B$ in the definition of $splitBA_f$, so $(e_{1b}, e_{1a}) \in splitBA_f\ e_{1r}$ where $e_{1r} = \pi_2(splitExp_f\ e_1) \neq \#$, $Y = \mathbf{A}$, $\delta_{2b} = m_{2b} = s_{2b} = 0$, and $e_{2a} = \pi_1(splitExp_f\ e_2) \neq \#$.
As $X < Y = \mathbf{A}$, then $X \in \{\mathbf{B}, \mathbf{M}\}$. So by induction hypothesis on (A.22) we obtain

$$E \vdash h, k, 0, e_{1r} \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

where $\delta_1 = \delta_{1b} + \delta_{1a}$, $m_1 = \max\{m_{1b}, |\delta_{1b}| + m_{1a}\}$ and $s_1 = \max\{s_{1b}, s_{1a}\}$. By Lemma 5

$$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

From (A.23) we obtain

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2a} \Downarrow h', k, v, (\delta_{2a}, m_{2a}, s_{2a})$$

and by Lemma 4

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h', k, v, (\delta_{2a}, m_{2a}, s_{2a})$$

By rule [*Let*] then

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta', m', s')$$

where $\delta' = \delta_1 + \delta_{2a}$, $m' = \max\{m_1, |\delta_1| + m_{2a}\}$ and $s' = \max\{2 + s_1, 1 + s_{2a}\}$. Consequenlty:

$$\begin{array}{rcll} \delta' & = & \delta_1 + \delta_{2a} & \\ & = & \delta_{1b} + \delta_{1a} + \delta_{2a} & \\ & = & \delta_{1b} + \delta_a & \\ & = & \delta_b + \delta_a & \{\text{as } \delta_{2b} = 0\} \end{array}$$

Also

$$\begin{aligned} m' &= \max\{m_1, |\delta_1| + m_{2a}\} \\ &= \max\{\max\{m_{1b}, |\delta_{1b}| + m_{1a}\}, |\delta_{1b}| + |\delta_{1a}| + m_{2a}\} \\ &= \max\{m_{1b}, |\delta_{1b}| + m_{1a}, |\delta_{1b}| + |\delta_{1a}| + m_{2a}\} \\ &= \max\{\max\{m_{1b}, |\delta_{1b}| + m_{2b}\}, |\delta_{1b}| + \max\{m_{1a}, |\delta_{1a}| + m_{2a}\}\} \\ &\quad \{\text{as } m_{2b} = 0 \text{ and } m_{1b} \geq |\delta_{1b}|\} \\ &= \max\{m_b, |\delta_{1b}| + |\delta_{2b}| + \max\{m_{1a}, |\delta_{1a}| + m_{2a}\}\} \\ &\quad \{\text{as } \delta_{2b} = 0\} \\ &= \max\{m_b, |\delta_b| + m_a\} \end{aligned}$$

and

$$\begin{aligned} s' &= \max\{2 + s_1, 1 + s_{2a}\} \\ &= \max\{2 + \max\{s_{1b}, s_{1a}\}, 1 + s_{2a}\} \\ &= \max\{2 + s_{1b}, 2 + s_{1a}, 1 + s_{2b}, 1 + s_{2a}\} && \{\text{as } s_{2b} = 0\} \\ &= \max\{\max\{2 + s_{1b}, 1 + s_{2b}\}, \max\{2 + s_{1a}, 1 + s_{2a}\}\} \\ &= \max\{s_b, s_a\} \end{aligned}$$

$n > 0, [CaseP]$ This is trivial by induction hypothesis.

□

**Theorem 3.** *Let* $e_r, E, h, h', k, td, v, \delta, m, s$ *such that*

$$E \vdash h, k, td, e_r \Downarrow h', k, v, (\delta, m, s) \quad (*)$$

*If there are direct calls to* $f$ *in derivation (*) then there exist* $X \in \{\boldsymbol{B}, \boldsymbol{M}\}$, $(e_b, e_a) \in \mathit{splitBA}_f\ e_r$, $\delta_b$, $\delta_a$, $m_b$, $m_a$, $s_b$, $s_a$ *such that*

$$E \vdash h, k, td, (e_b, e_a)_X \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$$

*and* (1) $\delta = \delta_b + \delta_a$, (2) $m = \max\{m_b, |\delta_b| + m_a\}$, (3) $s = \max\{s_b, s_a\}$.

*Proof.* By induction on the depth $n$ of the derivation for $e_r$ and by cases on the last rule applied:

$n = 0$ These are rules [*Lit*] and [*Var*]. In these cases there are no direct calls to $f$ in the derivation, so the property is trivially true.

$n > 0, [\mathit{Lub}]$ In this case $e_r = \sqcup_{i=1}^{n} e_i$ and there exists $j \in \{1..n\}$

$$E \vdash h, k, td, e_j \Downarrow h', k, v, (\delta, m, s)$$

This case is trivial by induction hypothesis.

$n > 0, [\mathit{App}]$ If $e_r = g\ \overline{a_i}^n\ @\ \overline{r_j}^m$ with $g \neq f$, then there are no direct calls to $f$ in the derivation and the property is trivially true.

If $e_r = f\ \overline{a_i}^n\ @\ \overline{r_j}^m$ then $\mathit{splitBA}_f\ e_r = [(f\ \overline{a_i}^n\ @\ \overline{r_j}^m, \Box)]$. So, taking this unique pair, $X = \mathbf{B}$, $\delta_b = \delta$, $m_b = m$, $s_b = s$, and $\delta_a = m_a = s_a = 0$ we apply rule [*Before*] and obtain the desired property.

$n > 0, [\mathit{Let}]$ Now $e_r = \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$:

$$E \vdash h, k, 0, e_1 \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1) \tag{A.24}$$

and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_2 \Downarrow h', k, v, (\delta_2, m_2, s_2) \tag{A.25}$$

where $\delta = \delta_1 + \delta_2$, $m = \max\{m_1, |\delta_1| + m_2\}$ and $s = \max\{2 + s_1, 1 + s_2\}$. We distinguish two cases:

- There are no direct calls to $f$ in the derivation A.25. As there is at least a direct call to $f$ in the derivation of $e_r$, then it necessarily appears in the derivation (A.24). Consequently, by Lemma 5, $e_{1r} = \pi_2(\mathit{splitExp}_f\ e_1) \neq \#$ and

  $$E \vdash h, k, 0, e_{1r} \Downarrow h_1, k, v_1, (\delta_1, m_1, s_1)$$

  containing a direct call to $f$.

  By induction hypothesis, there exist $Z \in \{\mathbf{B}, \mathbf{M}\}$, $(e_{1rb}, e_{1ra}) \in \mathit{splitBA}_f\ e_{1r}$, $\delta_{1b}$, $\delta_{1a}$, $m_{1b}$, $m_{1a}$, $s_{1b}$, $s_{1a}$ such that

  $$E \vdash h, k, 0, (e_{1rb}, e_{1ra})_Z \Downarrow h_1, k, v_1, (\delta_{1b}/\delta_{1a}, m_{1b}/m_{1a}, s_{1b}/s_{1a}) \tag{A.26}$$

  where $\delta_1 = \delta_{1b} + \delta_{1a}$, $m_1 = \max\{m_{1b}, |\delta_{1b}| + m_{1a}\}$ and $s_1 = \max\{s_{1b}, s_{1a}\}$.
  By Lemma 4, $e_{2b} = \pi_1(\mathit{splitExp}_f\ e_2) \neq \#$ and

  $$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2b} \Downarrow h', k, v, (\delta_2, m_2, s_2)$$

and by rule [*After*]

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td+1, (\Box, e_{2b})_{\mathbf{A}} \Downarrow h', k, v, (0/\delta_2, 0/m_2, 0/s_2) \tag{A.27}$$

As $Z < \mathbf{A}$, by applying rule [*LetP*] to (A.26) and (A.27) we obtain for pair $p = (\textbf{let } x_1 = e_{1r,b} \textbf{ in } \Box, \textbf{let } x_1 = e_{1r,a} \textbf{ in } e_{2b}) \in \mathit{splitBA}_f\ e_r$:

$$E \vdash h, k, td, p_{\mathbf{M}} \Downarrow h', k, v, (\delta_b/\delta_a, m_b/m_a, s_b/s_a)$$

where $\delta_b = \delta_{1b}$, $\delta_a = \delta_{1a} + \delta_2$, $m_b = \max\{m_{1b}, |\delta_{1b}|\}$, $m_a = \max\{m_{1a}, |\delta_{1a}| + m_2\}$, $s_b = \max\{2 + s_{1b}, 1\} = 2 + s_{1b}$ and $s_a = \max\{2 + s_{1a}, 1 + s_2\}$. Then:

$$\begin{array}{rcl} \delta_b + \delta_a & = & \delta_{1b} + \delta_{1a} + \delta_2 \\ & = & \delta_1 + \delta_2 \\ & = & \delta \end{array}$$

Also

$$\begin{array}{rcl} \max\{m_b, |\delta_b| + m_a\} & = & \max\{\max\{m_{1b}, |\delta_{1b}|\}, |\delta_{1b}| + \max\{m_{1a}, |\delta_{1a}| + m_2\}\} \\ & = & \max\{m_{1b}, |\delta_{1b}|, |\delta_{1b}| + m_{1a}, |\delta_{1b}| + |\delta_{1a}| + m_2\} \\ & = & \max\{\max\{m_{1b}, |\delta_{1b}| + m_{1a}\}, |\delta_1| + m_2\} \\ & & \{\text{as } |\delta_{1a}| \leq m_{1a}\} \\ & = & \max\{m_1, |\delta_1| + m_2\} \\ & = & m \end{array}$$

and

$$\begin{array}{rcl} \max\{s_b, s_a\} & = & \max\{2 + s_{1b}, \max\{2 + s_{1a}, 1 + s_2\}\} \\ & = & \max\{2 + \max\{s_{1b}, s_{1a}\}, 1 + s_2\} \\ & = & \max\{2 + s_1, 1 + s_2\} \\ & = & s \end{array}$$

- There is a direct call to $f$ in the derivation (A.25). By Lemma 5, $e_{2r} = \pi_2(\mathit{splitExp}_f\ e_2) \neq \#$ and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td+1, e_{2r} \Downarrow h', k, v, (\delta_2, m_2, s_2)$$

containing a direct call to $f$. By induction hypothesis, there exist $Z \in \{\mathbf{B}, \mathbf{M}\}$, $(e_{2rb}, e_{2ra}) \in \mathit{splitBA}_f\ e_{2r}$, $\delta_{2b}$, $\delta_{2a}$, $m_{2b}$, $m_{2a}$, $s_{2b}$ and $s_{2a}$ such that

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td+1, (e_{2rb}, e_{2ra})_Z \Downarrow h', k, v, (\delta_{2b}/\delta_{2a}, m_{2b}/m_{2a}, s_{2b}/s_{2a}) \tag{A.28}$$

where $\delta_2 = \delta_{2b} + \delta_{2a}$, $m_2 = \max\{m_{2b}, |\delta_{2b}| + m_{2a}\}$, and $s_2 = \max\{s_{2b}, s_{2a}\}$.
By applying rule [*Before*] to (A.24) we obtain

$$E \vdash h, k, 0, (e_1, \Box)_{\mathbf{B}} \Downarrow h_1, k, v_1, (\delta_1/0, m_1/0, s_1/0) \tag{A.29}$$

We distinguish two cases:

$Z = \mathbf{M}$ By Lemma 9, $e_{2ra} \neq \Box$. The pair we are looking for is $(\textbf{let } x_1 = e_1 \textbf{ in } e_{2rb}, \textbf{let } x_1 = \Box \textbf{ in } e_{2ra})_{\mathbf{M}} \in \mathit{splitBA}_f\ e_r$. By applying rule [*LetP*] to (A.29) and (A.28) we obtain $\delta_b = \delta_1 + \delta_{2b}$, $\delta_a = \delta_{2a}$, $m_b = \max\{m_1, |\delta_1| + m_{2b}\}$, $m_a = m_{2a}$, $s_b = \max\{2 + s_1, 1 + s_{2b}\}$, $s_a = \max\{2, 1 + s_{2a}\}$.

$Z = \mathbf{B}$ By Lemma 9, then $e_{2ra} = \square$, $\delta_{2a} = m_{2a} = s_{2a} = 0$ and

$$E \cup [x_1 \mapsto v_1] \vdash h_1, k, td + 1, e_{2rb} \Downarrow h', k, v, (\delta_{2b}, m_{2b}, s_{2b}) \tag{A.30}$$

The pair we are looking for is $(\textbf{let } x_1 = e_1 \textbf{ in } e_{2rb}, \square)_{\mathbf{B}} \in \mathit{splitBA}_f\ e_r$. By applying rule [*Let*] to (A.24) and (A.30), and then rule [*Before*], we obtain $\delta_b = \delta_1 + \delta_{2b}$, $\delta_a = 0$, $m_b = \max\{m_1, |\delta_1| + m_{2b}\}$, $m_a = 0$, $s_b = \max\{2 + s_1, 1 + s_{2b}\}$, $s_a = 0$.

This cases distinction is reflected by function *collapse* in the definition of $\mathit{splitBA}_f$. In both cases it is easy to prove that $\delta = \delta_b + \delta_a$, $m = \max\{m_b, |\delta_b| + m_a\}$ and $s = \max\{s_b, s_a\}$.

$n > 0, [\mathit{Case}]$ Now $e_r = \textbf{case } x \textbf{ of } \overline{C_i \overline{x_{ij}}^{n_i} \to e_i}^{n}$ and:

$$E \cup [\overline{x_{ki} \mapsto v_i}^{n_k}] \vdash h, k, td + n_k, e_k \Downarrow h', k, v, (\delta, m, s_k)$$

where $s = s_k + n_k$. By induction hypothesis there exists $X \in \{\mathbf{B}, \mathbf{M}\}$, $(e_{kbl}, e_{kal}) \in \mathit{splitBA}_f\ e_k$ such that

$$E \cup [\overline{x_{ki} \mapsto v_i}^{n_k}] \vdash h, k, td + n_k, (e_{kbl}, e_{kal})_X \Downarrow h', k, v, (\delta_{kbl}/\delta_{kal}, m_{kbl}/m_{kal}, s_{kbl}/s_{kal}) \tag{A.31}$$

where $\delta = \delta_{kbl} + \delta_{kal}$, $m = \max\{m_{kbl}, |\delta_{kbl}| + m_{kal}\}$, and $s_k = \max\{s_{kbl}, s_{kal}\}$.

We distinguish two cases:

$X = \mathbf{M}$ By Lemma 9, $e_{kal} \neq \square$, so the pair

$$(\textbf{case } x \textbf{ of } \overline{C_i \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{ibj}}^{m_i}}^{n}, \textbf{case } x \textbf{ of } \overline{C_i \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{iaj}}^{m_i}}^{n})_{\mathbf{M}}$$

where $\overline{(e_{ibj}, e_{iaj})}^{m_i} = \mathit{splitBA}_f\ e_i$, belongs to $\mathit{splitBA}_f\ e_r$ and applying rule [*CaseP*] to (A.31), it holds that $\delta_b = \delta_{kbl}$, $\delta_a = \delta_{kal}$, $m_b = m_{kbl}$, $m_a = m_{kal}$, $s_b = s_{kbl} + n_k$, $s_a = s_{kal} + n_k$.

$X = \mathbf{B}$ By Lemma 9, $e_{kal} = \square$, $\delta_{kal} = m_{kal} = s_{kal} = 0$ and

$$E \cup [\overline{x_{ki} \mapsto v_i}^{n_k}] \vdash h, k, td + n_k, e_{kbl} \Downarrow h', k, v, (\delta_{kbl}, m_{kbl}, s_{kbl}) \tag{A.32}$$

Consider the following set containing all the non-empty after parts of the alternatives:

$$N = \{e_{iaj} \mid (\_, e_{iaj}) \in \mathit{splitBA}_f\ e_i, e_{iaj} \neq \square, i \in \{1..n\}\}$$

If $N = \emptyset$, we can take the pair $(\textbf{case } x \textbf{ of } \overline{C_i \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{kbj}}^{m_i}}^{n}, \square)_{\mathbf{B}} \in \mathit{splitBA}_f\ e_r$. By applying rule [*Case*] to (A.32) and then rule [*Before*] we obtain $\delta_b = \delta_{kbl}$, $\delta_a = 0$, $m_b = m_{kbl}$, $m_a = 0$, $s_b = s_{kbl} + n_k$, $s_a = 0$.

Otherwise, we can take again the pair

$$(\textbf{case } x \textbf{ of } \overline{C_i \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{ibj}}^{m_i}}^{n}, \textbf{case } x \textbf{ of } \overline{C_i \overline{x_{ij}}^{n_i} \to \bigvee \overline{e_{kiaj}}^{m_i}})_{\mathbf{M}}$$

In this case, by applying rule [*CaseP*] to (A.31), we obtain $\delta_b = \delta_{kbl}$, $\delta_a = \delta_{kal}$, $m_b = m_{kbl}$, $m_a = m_{kal}$, $s_b = s_{kbl} + n_k$, $s_a = s_{kal} + n_k$. This cases distinction is reflected by function *collapse* in the definition of $\mathit{splitBA}_f$.

In both cases it is easy to prove that $\delta = \delta_b + \delta_a$, $m = \max\{m_b, |\delta_b| + m_a\}$ and $s = \max\{s_b, s_a\}$.

□

**Lemma 10** (Additivity for $\delta$)**.** *Let us assume, in a heap with $k$ regions, the execution of an expression $e$ with a resource vector $(\delta, m, s)$ under a function signature $\Sigma$, such that there are $p \geq 0$ direct calls to a function $g \in dom\ \Sigma$, each with an associated resource vector $(\delta_i, m_i, s_i)$, $i \in \{1 \dots p\}$. If $e$ is executed under an different function environment $\Sigma'$, in which $g$ is replaced with an equivalent definition (up to memory consumption), obtaining a resource vector $(\delta', m', s')$, the following equation holds:*

$$\delta' = \delta + \sum_{i=1}^{p} \delta'_i|_k - \sum_{i=1}^{p} \delta_i|_k$$

*where the $\delta'_i$ correspond to the calls to $g$ in the latter execution of $e$.*

*Proof.* By induction on the structure of $e$.

$e \equiv c, x, C\ \overline{a_i}^n\ @\ r, f\ \overline{a_i}^n\ @\ \overline{r_j}^m$ with $f \neq g$ There are no direct calls to $g$ (i.e. $p = 0$) and the result follows trivially, since $\delta = \delta'$.

$e \equiv g\ \overline{a_i}^n\ @\ \overline{r_j}^m$ Let $\delta_g$ (resp. $\delta'_g$) be the result of evaluating $g$'s body in this call under the signature $\Sigma$ (resp. $\Sigma'$). Then $\delta = \delta_g|_k$ and $\delta' = \delta'_g|_k$. Then:

$$\delta' = \delta'_g|_k = \delta_g|_k + \delta'_g|_k - \delta_g|_k = \delta + \delta'_g|_k - \delta_g|_k$$

$e \equiv$ **let** $x_1 = e_1$ **in** $e_2$ From the $p$ recursive calls, let us assume that $q$ calls ($0 \leq q \leq p$) occur during the evaluation of $e_1$ and $p - q$ calls take place in the evaluation of $e_2$. Then:

$$\begin{array}{rcl} \delta' & = & \delta'_1 + \delta'_2 \\ & = & \delta_1 + \sum_{i=1}^{q} \delta'_i|_k - \sum_{i=1}^{q} \delta_i|_k + \delta_2 + \sum_{i=q+1}^{p} \delta'_i|_k - \sum_{i=q+1}^{p} \delta_i|_k \\ & = & \delta + \sum_{i=1}^{p} \delta'_i|_k - \sum_{i=1}^{p} \delta_i|_k \end{array}$$

$e \equiv$ **case** $x$ **of** $\overline{C_i\ \overline{a_{ij}}^{n_i} \rightarrow e_i}^n$ By asumming that the $r$-th branch ($1 \leq r \leq n$) is executed, we get the same $\delta$ as in the evaluation of $e_r$, so the result follows trivially from the induction hypothesis applied to $e_r$.

□

**Lemma 11.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$ be a typeable function definition under type environment $\Gamma$ such that:*

$$E \vdash h, k+1, n+m, e_f \Downarrow h', k+1, v, (\delta, m, s), (n_t, n_b, l)_f \tag{A.33}$$

*where $E \equiv_{def} \left[\overline{x_i \mapsto v_i}^n, \overline{r_j \mapsto i_j}^m, self \mapsto k+1\right]$. Then $(\lfloor\Delta_b\rfloor \cdot n_b + \lfloor\Delta_r\rfloor \cdot (n_t - n_b)) \sqcup \lfloor\Delta_b\rfloor \succeq_{\overline{s_i}^n,k,\phi} \delta$, where $\Delta_b$ and $\Delta_r$ are as defined in the algorithm, $s_i = size(h, v_i)$ for each $i \in \{1 \ldots n\}$ and $\phi$ is the region instantiation consistent with $E$ and $\Gamma$.*

*Proof.* By induction on the number of calls $n_t$.

$n_t = 1$ There are no direct recursive calls to $f$ in (A.33) and, by Lemma 4, we can substitute $e_b$ for $e_f$ in this derivation, obtaining the same result and memory consumption. $\Delta_b$ is computed by applying the abstract interpretation rules to $e_b$ and by Theorem 1 we get $\Delta_b \succeq_{\overline{s_i}^n,k,\phi} \delta$, which is equivalent to $\lfloor\Delta_b\rfloor \succeq_{\overline{s_i}^n,k,\phi} \delta$, since $\phi\ \rho^f_{self} = k+1$ and in the definition of $\succeq_{\overline{s_i}^n,k,\phi}$ we only consider regions ranging from 0 to $k$. Then we get:

$$(\lfloor\Delta_b\rfloor \cdot n_b + \lfloor\Delta_r\rfloor \cdot (n_t - n_b)) \sqcup \lfloor\Delta_b\rfloor \sqsupseteq \lfloor\Delta_b\rfloor \succeq_{\overline{s_i}^n,k,\phi} \delta.$$

$n_t > 1$ We can assume there are $p \geq 1$ direct recursive calls to $f$ in (A.33). Let us denote by $(\delta_i, m_i, s_i)$ and $(n_{t,i}, n_{b,i}, l_i)$ the resource vector and number of calls of the $i$-th recursive call.

Firstly we prove that $\Delta_r$ safely approximates the actual $\delta$ without taking the $\delta_i$'s into account. We have to replace the recursive calls in (A.33) with an expression that produces the same result, but with an empty $\delta_i$. Let **rmask** $e$ be a new construct with the following semantics:

$$\frac{E \vdash h, k, td, e \Downarrow h', k, v, (\delta, m, s)}{E \vdash h, k, td, \textbf{rmask}\ e \Downarrow h', k, v, ([\,], 0, 0)}\ [\text{RMask}]$$

Let $\Sigma'$ be a function environment in which the body $e_f$ of $f$ is replaced with **rmask** $e_f$. Under this $\Sigma'$ we get the following derivation:

$$E \vdash h, k+1, n+m, e_f \Downarrow h', k+1, v, (\delta', m', s'), (n_t, n_b, l)_f \tag{A.34}$$

where, by Lemma 10:

$$\delta' = \delta - \sum_{i=1}^{p} \delta_i|_k \tag{A.35}$$

Now we can apply the correctness proof of the abstract interpretation (Theorem 1), since the abstract signature $\Sigma^{\#}[f \mapsto ([\,]_f, 0, 0)]$ is correct with respect to the above mentioned $\Sigma'$ and since (according to Definition 6), we can substitute $e_r$ for $e_f$ in (A.34) by Lemma 5. Hence we get $\Delta_r \succeq_{\overline{s_i}^n,k,\phi} \delta'$, which is equivalent to:

$$\lfloor\Delta_r\rfloor \succeq_{\overline{s_i}^n,k,\phi} \delta' \tag{A.36}$$

since we are only considering regions from 0 to $k$. Moreover, every region type variable $\rho$ in the domain of $\lfloor\Delta_r\rfloor$ must satisfy $\lfloor\Delta_r\rfloor\ \rho \neq -\infty$, as $\lfloor\Delta_r\rfloor$ is an upper bound to a $\delta'|_k$ in which every charge is strictly greater than $-\infty$. Now we can prove the required result:

$$
\begin{array}{ll}
 & (\lfloor \Delta_b \rfloor \cdot n_b + \lfloor \Delta_r \rfloor \cdot (n_t - n_b)) \sqcup \lfloor \Delta_b \rfloor \\
\sqsupseteq & \\
 & \lfloor \Delta_b \rfloor \cdot n_b + \lfloor \Delta_r \rfloor \cdot (n_t - n_b) \\
= & \qquad \{\text{by Lemma 1}\} \\
 & \lfloor \Delta_b \rfloor \cdot \sum_{i=1}^{p} n_{b,i} + \lfloor \Delta_r \rfloor \cdot (1 + \sum_{i=1}^{p} n_{t,i} - \sum_{i=1}^{p} n_{b,i}) \\
= & \\
 & \sum_{i=1}^{p} (\lfloor \Delta_b \rfloor \cdot n_{b,i}) + \sum_{i=1}^{p} (\lfloor \Delta_r \rfloor \cdot (n_{t,i} - n_{b,i})) + \lfloor \Delta_r \rfloor \\
= & \\
 & \sum_{i=1}^{p} (\lfloor \Delta_b \rfloor \cdot n_{b,i} + \lfloor \Delta_r \rfloor \cdot (n_{t,i} - n_{b,i})) + \lfloor \Delta_r \rfloor \\
= & \qquad \{\text{since } n_{b,i} \geq 1 \text{ and } n_{t,i} \geq n_{b,i} \text{ for every } i \in \{1 \dots p\}\} \\
 & \sum_{i=1}^{p} ((\lfloor \Delta_b \rfloor \cdot n_{b,i} + \lfloor \Delta_r \rfloor \cdot (n_{t,i} - n_{b,i})) \sqcup \lfloor \Delta_b \rfloor) + \lfloor \Delta_r \rfloor \\
\succeq_{\overline{s_i}^n,k,\phi} & \qquad \{\text{by i.h.}\} \\
 & \sum_{i=1}^{p} \delta_i|_k + \lfloor \Delta_r \rfloor \\
\succeq_{\overline{s_i}^n,k,\phi} & \qquad \{\text{by (A.35) and (A.36)}\} \\
 & \sum_{i=1}^{p} \delta_i|_k + \delta - \sum_{i=1}^{p} \delta_i|_k \\
= & \\
 & \delta
\end{array}
$$

Notice that we can safely substitute $\delta_i$ for $\delta_i|_k$ in each application of the induction hypothesis, since we only take into account regions from 0 to $k$.

□

**Lemma 12.** *$\Delta_f$ is a correct abstract heap for $f$.*

*Proof.* We prove that $\Delta_f \succeq_{\overline{s_i}^n,k,\phi} \delta$ for any execution of $e_f$ under an environment $E$ where the size of the data structure bound to the $i$-th parameter is $s_i$, and for the consistent region instantiation $\phi$. The proof is a direct consequence of $nr_f$, $nb_f$, and all the size functions being upper bounds of their respective runtime figures, and of Lemma 11. For any $j \in \{0..k\}$:

$$
\begin{array}{ll}
 & \sum_{\phi\ \rho=j} \Delta_f\ \rho\ \overline{s_i}^n \\
= & \\
 & \sum_{\phi\ \rho=j} ((\lfloor \Delta_b \rfloor\ \rho \cdot nb_f + \lfloor \Delta_r \rfloor\ \rho \cdot nr_f) \sqcup \lfloor \Delta_b \rfloor)\ \overline{s_i}^n \\
\geq & \\
 & \sum_{\phi\ \rho=j} ((\lfloor \Delta_b \rfloor\ \rho \cdot n_b + \lfloor \Delta_r \rfloor\ \rho \cdot (n_t - n_b)) \sqcup \lfloor \Delta_b \rfloor)\ \overline{s_i}^n \\
\geq & \quad \{\text{by Lemma 11}\} \\
 & \delta\ j
\end{array}
$$

□

**Lemma 13.** *Let $e_f$ be $f$'s body expression, let $\Delta_f \in \mathbb{D}$, and let $d$ be any natural number. Then, it holds that:*

$$(\llbracket e_f \rrbracket_\Delta\ \Sigma[f \mapsto \Delta_f]\ \Gamma\ d)(\overline{x}) \sqsubseteq \left( \bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \Delta_r(\overline{x}) \right) \sqcup \Delta_b(\overline{x})$$

*where $\Delta_r$ and $\Delta_b$ are defined as in Sec. 6.2, and $\Delta_f^G(\overline{a_{ji}})$ is an abbreviation for the guarded expression $[|\overline{a_{ji}}| \neq -\infty \to \Delta_f\ \ |\overline{a_{ji}}|]$.*

*Proof.* By structural induction on expression $e_f$. For brevity, let us call $\Delta^*$ to $(\llbracket e_f \rrbracket_\Delta\ \Sigma[f \mapsto \Delta_f]\ \Gamma\ d)(\overline{x})$.

$e_f \equiv c, x, C\ \overline{a_i}^n\ @\ r, g\ \overline{b_i @ r_j}\}$ In these cases we have $\mathit{seqs}_f\ e_f = [[\,]]$, $n_r = 1$, $\mathit{splitExp}_f\ e_f = (e_f, \#)$, and $\Delta_r = [\,]_f$. So, we get $\Delta^* = \Delta_b \sqsubseteq [\,]_f \sqcup \Delta_b$.

$e_f \equiv f\ \overline{a_{11}@r_{11}}$ In this case we have $\mathit{seqs}_f\ e_f = [[f\ \overline{a_{11}@r_{11}}]]$, $n_r = 1$, $\mathit{splitExp}_f\ e_f = (\#, e_f)$, $\Delta_r = \Delta_b = [\,]_f$. So, we get $\Delta^* = \Delta_f^G(\overline{a_{11}}) \sqsubseteq (\Delta_f^G(\overline{a_{11}}) + [\,]_f) \sqcup [\,]_f$.

$e_f \equiv \mathbf{case}\ x\ \mathbf{of}\ \overline{C_k \overline{x_{kj}} \to e_k}$ By induction hypothesis, we have:

$$\Delta_k^* \sqsubseteq \left( \bigsqcup_{j_k=1}^{n_{r_k}} \sum_{i_k=1}^{n_{j_k}} \Delta_f^G(\overline{a_{j_k i_k}}) + \Delta_{r_k}(\overline{x}) \right) \sqcup \Delta_{b_k}(\overline{x})$$

where $n_{r_k} = \mathit{length}\ (\mathit{seqs}_f\ e_k)$, $\mathit{splitExp}_f\ e_k = (e_{bk}, e_{rk})$, and $\mathit{splitExp}_f\ e_f = (\mathbf{case}\ x\ \mathbf{of}\ \overline{C_k \overline{x_{kj}} \to e_{bk}}$, $\mathbf{case}\ x\ \mathbf{of}\ \overline{C_k \overline{x_{kj}} \to e_{rk}})$. For simplicity we will assume that all the branches are present both in the base and in the recursive expressions resulting from $\mathit{splitExp}_f\ e_f$ even if they consist just of the empty expression $\#$. This will not affect the proof because the contribution of $\#$ to the lub operator $\sqcup$ is null. By the $\mathit{seqs}_f$ definition, we have $n_r = \sum_k n_{rk}$. Then, we get:

$$\begin{array}{ll}
& \Delta^* \\
= & \quad \{\text{by the abstract interpretation definition on } e_f\} \\
& \bigsqcup_k \Delta_k^* \\
\sqsubseteq & \quad \{\text{by induction hypothesis}\} \\
& \bigsqcup_k \left( (\bigsqcup_{j_k=1}^{n_{r_k}} \sum_{i_k=1}^{n_{j_k}} \Delta_f^G(\overline{a_{j_k i_k}}) + \Delta_{r_k}(\overline{x})) \sqcup \Delta_{b_k}(\overline{x}) \right) \\
\sqsubseteq & \quad \{\text{by mathematics}\} \\
& \left( \bigsqcup_k \bigsqcup_{j_k=1}^{n_{r_k}} \sum_{i_k=1}^{n_{j_k}} \Delta_f^G(\overline{a_{j_k i_k}}) + \bigsqcup_k \Delta_{r_k}(\overline{x}) \right) \sqcup \bigsqcup_k \Delta_{b_k}(\overline{x}) \\
= & \quad \{\text{by the abstract interpretation definition on } \mathit{splitExp}_f\ e_f\} \\
& \left( \bigsqcup_k \bigsqcup_{j_k=1}^{n_{r_k}} \sum_{i_k=1}^{n_{j_k}} \Delta_f^G(\overline{a_{j_k i_k}}) + \Delta_r(\overline{x}) \right) \sqcup \Delta_b(\overline{x}) \\
= & \quad \{\text{by the } \mathit{seqs}_f \text{ definition}\} \\
& \left( \bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \Delta_r(\overline{x}) \right) \sqcup \Delta_b(\overline{x})
\end{array}$$

$e_f \equiv \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_2$ By induction hypothesis, we have:

$$\Delta_k^* \sqsubseteq \left( \bigsqcup_{j_k=1}^{n_{r_k}} \sum_{i_k=1}^{n_{j_k}} \Delta_f^G(\overline{a_{j_k i_k}}) + \Delta_{r_k}(\overline{x}) \right) \sqcup \Delta_{b_k}(\overline{x})$$

where $n_{r_k} = \mathit{length}\ (\mathit{seqs}_f\ e_k)$, $\mathit{splitExp}_f\ e_k = (e_{b_k}, e_{r_k})$, $k = 1, 2$, and $\mathit{splitExp}_f\ e_f = (e_b, e_r)$. By the $\mathit{seqs}_f$ definition, we have $n_r = n_{r_1} \times n_{r_2}$. Then, we get:

$$
\begin{array}{ll}
 & \Delta^* \\
= & \quad \{\text{by the abstract interpretation definition}\} \\
 & \Delta_1^* + \Delta_2^* \\
\sqsubseteq & \quad \{\text{by induction hypothesis}\} \\
 & \left((\bigsqcup_{j_1=1}^{n_{r_1}} \sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \Delta_{r_1}(\overline{x})) \sqcup \Delta_{b_1}(\overline{x})\right) + \\
 & \left((\bigsqcup_{j_2=1}^{n_{r_2}} \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}}) + \Delta_{r_2}(\overline{x})) \sqcup \Delta_{b_2}(\overline{x})\right) \qquad (*)
\end{array}
$$

Now we distinguish cases according to the result of $\mathit{splitExp}_f\ e_f$ (see Fig.4):

$e_b = \#, e_{b_1} = \#$ So, $\Delta_b = \Delta_{b_1} = [\,]_f$. By Lemma 3 we have $e_r \neq \#$ and $e_{r_1} \neq \#$. It does not matter whether $e_{r_2} = \#$ or $e_{r_2} \neq \#$, since in both cases we get $e_r \equiv \mathbf{let}\ x_1 = e_{r_1}\ \mathbf{in}\ e_2$. Then, $\Delta_r = \Delta_{r_1} + \Delta_2$. On the other hand, expression $(*)$ becomes:

$$
\begin{array}{ll}
 & \left(\bigsqcup_{j_1=1}^{n_{r_1}} \sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \Delta_{r_1}(\overline{x})\right) + \\
 & \left((\bigsqcup_{j_2=1}^{n_{r_2}} \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}}) + \Delta_{r_2}(\overline{x})) \sqcup \Delta_{b_2}(\overline{x})\right) \\
\sqsubseteq & \quad \{\text{by mathematics}\} \\
 & \left(\bigsqcup_{j_1=1}^{n_{r_1}} \sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \bigsqcup_{j_2=1}^{n_{r_2}} \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}})\right) + \\
 & \sqcup\{\Delta_{r_1} + \Delta_{r_2},\ \Delta_{r_1} + \Delta_{b_2}\}(\overline{x}) \\
= & \quad \{\text{by mathematics}\} \\
 & \bigsqcup_{j_1=1, j_2=1}^{n_{r_1}, n_{r_2}} \left(\sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}})\right) + \Delta_{r_1} + \sqcup\{\Delta_{r_2}, \Delta_{b_2}\} \\
= & \quad \{\text{by the } \mathit{seqs}_f \text{ definition, and by Lemma 7}\} \\
 & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}})\right) + (\Delta_{r_1} + \Delta_2)(\overline{x}) \\
= & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \left(\Delta_f^G(\overline{a_{ji}})\right) + \Delta_r(\overline{x})\right) \sqcup [\,]_f
\end{array}
$$

$e_b = \#, e_{b_1} \neq \#, e_{b_2} = \#$ So, $\Delta_b = \Delta_{b_2} = [\,]_f$. By Lemma 3 we have $e_r \neq \#$ and $e_{r_2} \neq \#$. Then, we get $e_r \equiv \mathbf{let}\ x_1 = e_1\ \mathbf{in}\ e_{r_2}$ if $e_{r_1} = \#$, and $e_r \equiv \sqcup\{\mathbf{let}\ x_1 = e_{b_1}\ \mathbf{in}\ e_{r_2},\ \mathbf{let}\ x_1 = e_{r_1}\ \mathbf{in}\ e_2\}$ if $e_{r_1} \neq \#$. Then, either $\Delta_r = \Delta_1 + \Delta_{r_2}$ or $\Delta_r = \sqcup\{\Delta_{b_1} + \Delta_{r_2}, \Delta_{r_1} + \Delta_2\}$. But $\Delta_{b_2} = [\,]_f$, so $\Delta_2 = \Delta_{r_2} \sqcup \Delta_{b_2} = \Delta_{r_2}$. From $\Delta_1 = \Delta_{b_1} \sqcup \Delta_{r_1}$ we get in any case $\Delta_r = \Delta_1 + \Delta_{r_2}$. On the other hand, expression $(*)$ becomes:

$$
\begin{array}{ll}
 & \left(\bigsqcup_{j_1=1}^{n_{r_1}} \sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \Delta_{r_1}(\overline{x})\right) \sqcup \Delta_{b_1}(\overline{x}) + \\
 & \left((\bigsqcup_{j_2=1}^{n_{r_2}} \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}}) + \Delta_{r_2}(\overline{x}))\right) \\
\sqsubseteq & \quad \{\text{by mathematics}\} \\
 & \left(\bigsqcup_{j_1=1}^{n_{r_1}} \sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \bigsqcup_{j_2=1}^{n_{r_2}} \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}})\right) + \\
 & \sqcup\{\Delta_{r_1} + \Delta_{r_2},\ \Delta_{b_1} + \Delta_{r_2}\}(\overline{x}) \\
= & \quad \{\text{by mathematics}\} \\
 & \bigsqcup_{j_1=1, j_2=1}^{n_{r_1}, n_{r_2}} \left(\sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}})\right) + \sqcup\{\Delta_{r_1}, \Delta_{b_1}\} + \Delta_{r_2} \\
= & \quad \{\text{by the } seqs_f \text{ definition, and by Lemma 7}\} \\
 & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}})\right) + (\Delta_1 + \Delta_{r_2})(\overline{x}) \\
= & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \Delta_r(\overline{x})\right) \sqcup [\,]_f
\end{array}
$$

$e_b \neq \#, e_{b_1} \neq \#, e_{b_2} \neq \#$ We assume the most complex case $e_{r_1} \neq \#$ and $e_{r_2} \neq \#$, being the other cases very similar. We have $e_b \equiv \textbf{let } x_1 = e_{b_1} \textbf{ in } e_{b_2}$, and $e_r \equiv \sqcup\{\textbf{let } x_1 = e_{b_1} \textbf{ in } e_{r_2},\ \textbf{let } x_1 = e_{r_1} \textbf{ in } e_2\}$. Then $\Delta_b = \Delta_{b_1} + \Delta_{b_2}$, and $\Delta_r = \sqcup\{\Delta_{b_1} + \Delta_{r_2}, \Delta_{r_1} + \Delta_2\}$. On the other hand, expression $(*)$ becomes:

$$
\begin{array}{ll}
 & \left(\bigsqcup_{j_1=1}^{n_{r_1}} \sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \Delta_{r_1}(\overline{x})\right) \sqcup \Delta_{b_1}(\overline{x}) + \\
 & \left(\bigsqcup_{j_2=1}^{n_{r_2}} \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}}) + \Delta_{r_2}(\overline{x})\right) \sqcup \Delta_{b_2}(\overline{x}) \\
\sqsubseteq & \quad \{\text{by mathematics}\} \\
 & \left(\left(\bigsqcup_{j_1=1}^{n_{r_1}} \sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \bigsqcup_{j_2=1}^{n_{r_2}} \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}})\right) + \right. \\
 & \left. \bigsqcup\{\Delta_{r_1} + \Delta_{r_2}, \Delta_{r_1} + \Delta_{b_2}, \Delta_{b_1} + \Delta_{r_2}\}\right) \bigsqcup(\Delta_{b_1} + \Delta_{b_2}) \\
= & \quad \{\text{by mathematics}\} \\
 & \left(\bigsqcup_{j_1=1, j_2=1}^{n_{r_1}, n_{r_2}} \left(\sum_{i_1=1}^{n_{j_1}} \Delta_f^G(\overline{a_{j_1 i_1}}) + \sum_{i_2=1}^{n_{j_2}} \Delta_f^G(\overline{a_{j_2 i_2}})\right) + \right. \\
 & \left. \bigsqcup\{\Delta_{r_1} + \sqcup\{\Delta_{r_2}, \Delta_{b_2}\}, \Delta_{b_1} + \Delta_{r_2}\}\right) \sqcup \Delta_b \\
= & \quad \{\text{by the } seqs_f \text{ definition, and by Lemma 7}\} \\
 & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \sqcup\{\Delta_{r_1} + \Delta_2, \Delta_{b_1} + \Delta_{r_2}\}\right) \sqcup \Delta_b \\
= & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \Delta_r(\overline{x})\right) \sqcup \Delta_b(\overline{x})
\end{array}
$$

□

**Lemma 14.** *If* $\Delta_f = (\lfloor \Delta_r \rfloor \cdot nr_f + \lfloor \Delta_b \rfloor \cdot nb_f) \sqcup \lfloor \Delta_b \rfloor$*, and* $nr_f$ *and* $nb_f$ *are well-behaved in dom* $\lfloor \Delta_r \rfloor$*, then* $\mathbb{I}_\Delta\ \Delta_f \sqsubseteq \Delta_f$*.*

*Proof.* We distinguish two cases. In the first place, we show the reductivity for those points in $D = \{\overline{x} \mid \forall ji\,.\, \lfloor \Delta_r^G \rfloor(\overline{a_{ji}}) \neq -\infty\}$. Notice that this is included in the bigger domain $\{\overline{x} \mid \forall ji\,.\, |\overline{a_{ji}}|\ \overline{x} \neq -\infty\}$. In this domain $\Delta_f^G\ (\overline{a_{ji}}) = (\lfloor \Delta_r^G \rfloor \cdot nr_f^G + \lfloor \Delta_b^G \rfloor \cdot nb_f^G)\ (\overline{a_{ji}}) = (\lfloor \Delta_r \rfloor \cdot nr_f + \lfloor \Delta_b \rfloor \cdot nb_f)\ (\overline{a_{ji}})$ because all the guards are true, $nb_f^G\ (\overline{a_{ji}}) \sqsupseteq 1$, and $\lfloor \Delta_b^G \rfloor \cdot nb_f^G \sqsupseteq \lfloor \Delta_b^G \rfloor$.

$$
\begin{array}{ll}
 & (\mathbb{I}_\Delta\ \Delta_f)(\overline{x}) \\
= & \quad \{\text{by definition of } \mathbb{I}_\Delta\} \\
 & \lfloor [\![e_f]\!]_\Delta\ \Sigma[f \mapsto \Delta_f]\ \Gamma\ (n+m) \rfloor(\overline{x}) \\
\sqsubseteq & \quad \{\text{by Lemma 13}\} \\
 & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \lfloor \Delta_r \rfloor(\overline{x})\right) \sqcup \lfloor \Delta_b \rfloor(\overline{x}) \\
= & \quad \{\text{by definition of } \Delta_f^G \text{ in } D\} \\
 & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} (\lfloor \Delta_r \rfloor(\overline{a_{ji}}) \cdot nr_f(\overline{a_{ji}}) + \lfloor \Delta_b \rfloor(\overline{a_{ji}}) \cdot nb_f(\overline{a_{ji}})) + \lfloor \Delta_r \rfloor(\overline{x})\right) \sqcup \lfloor \Delta_b \rfloor(\overline{x}) \\
\sqsubseteq & \quad \{\overline{a_{ji}} \sqsubseteq \overline{x} \text{ and } \Delta_r,\ \Delta_b \text{ monotonic}\} \\
 & \left(\lfloor \Delta_r \rfloor(\overline{x}) \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} nr_f(\overline{a_{ji}})\right) + \lfloor \Delta_b \rfloor(\overline{x}) \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} nb_f(\overline{a_{ji}})\right) + \lfloor \Delta_r \rfloor(\overline{x})\right) \sqcup \lfloor \Delta_b \rfloor(\overline{x}) \\
\sqsubseteq & \quad \{nr_f \text{ and } nb_f \text{ well-behaved in } \mathit{dom}\ \lfloor \Delta_r \rfloor\} \\
 & (\lfloor \Delta_r \rfloor(\overline{x}) \cdot nr_f(\overline{x}) + \lfloor \Delta_b \rfloor(\overline{x}) \cdot nb_f(\overline{x})) \sqcup \lfloor \Delta_b \rfloor(\overline{x}) \\
= & \quad \{\text{by definition of } \Delta_f\} \\
 & \Delta_f(\overline{x})
\end{array}
$$

In the second place, we consider the complementary domain $\overline{D} = (\mathbb{R}^+ \cup \{+\infty\})^n - D$ which of course includes the domain $\{\overline{x} \mid \exists ji\,.\, |\overline{a_{ji}}|\ \overline{x} = -\infty\}$. In the second case, for these particular $j$ and $i$, $\lfloor \Delta_f^G \rfloor\ (\overline{a_{ji}}) = -\infty$. If the first case applies but not the second one, then for these particular $j$ and $i$, $\lfloor \Delta_r^G \rfloor\ (\overline{a_{ji}}) = -\infty$. Then, we get:

$$
\begin{array}{ll}
 & (\mathbb{I}_\Delta\ \Delta_f)(\overline{x}) \\
= & \quad \{\text{by definition of } \mathbb{I}_\Delta\} \\
 & \lfloor [\![e_f]\!]_\Delta\ \Sigma[f \mapsto \Delta_f]\ \Gamma\ (n+m) \rfloor(\overline{x}) \\
\sqsubseteq & \quad \{\text{by Lemma 13}\} \\
 & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \Delta_f^G(\overline{a_{ji}}) + \lfloor \Delta_r \rfloor(\overline{x})\right) \sqcup \lfloor \Delta_b \rfloor(\overline{x}) \\
= & \quad \{\text{by definition of } \Delta_f^G\} \\
 & \left(\bigsqcup_{j=1}^{n_r} \sum_{i=1}^{n_j} \left((\lfloor \Delta_r^G \rfloor(\overline{a_{ji}}) \cdot nr_f^G(\overline{a_{ji}}) + \lfloor \Delta_b^G \rfloor(\overline{a_{ji}}) \cdot nb_f^G(\overline{a_{ji}})) \sqcup \lfloor \Delta_b^G \rfloor(\overline{a_{ji}})\right) + \lfloor \Delta_r \rfloor(\overline{x})\right) \\
 & \sqcup \lfloor \Delta_b \rfloor(\overline{x})
\end{array}
$$

Now we distinguish three mutually exclusive cases:

1. Those $\overline{x}$ and $j', i'$ such that $G$ is false, i.e. $|\overline{a_{j'i'}}|\ \overline{x} = -\infty$. In this case, the term $i'$ of the sum $\sum_{i=1}^{n_{j'}}$ becomes $-\infty$, and so does the whole sum. As a result, the supremum $\bigsqcup_{j=1}^{n_r}$ is done with one sequence less, namely the sequence $j'$, for those $\overline{x}$.

2. Those $\overline{x}$ and $j', i'$ such that $G$ is true but $\lfloor \Delta_r^G \rfloor(\overline{a_{j'i'}}) = -\infty$. In this case, the term $i'$ of the sum $\sum_{i=1}^{n_{j'}}$ becomes $\lfloor \Delta_b^G \rfloor(\overline{a_{j'i'}}) = \lfloor \Delta_b \rfloor(\overline{a_{j'i'}}) \sqsubseteq \lfloor \Delta_b \rfloor(\overline{x})$. Then, the 'contribution' of this $\lfloor \Delta_f^G \rfloor(\overline{a_{j'i'}})$ to the global sum is with $nb_f(\overline{a_{j'i'}})$ less copies of $\lfloor \Delta_r \rfloor(\overline{x})$ and $1 \sqsubseteq nb_f(\overline{a_{j'i'}})$ copy of $\lfloor \Delta_b \rfloor(\overline{x})$.

3. Those $\overline{x}$ and $j', i'$ such that $\lfloor \Delta_r^G \rfloor(\overline{a_{j'i'}}) \neq -\infty$. In this case, as it happened in the domain $D$ the term $\sqcup \lfloor \Delta_b \rfloor(\overline{a_{j'i'}})$ in the unfolding of $\lfloor \Delta_f^G \rfloor$ can be safely deleted.

As a result, for all $\overline{x}$, we can write

$$\begin{array}{ll} & (\mathbb{I}_\Delta\ \Delta_f)(\overline{x}) \\ \sqsubseteq & \left(\lfloor \Delta_r \rfloor(\overline{x}) \left(\bigsqcup_{j=1}^{n'_r} \sum_{i=1}^{n'_j} nr_f(\overline{a_{ji}})\right) + \lfloor \Delta_b \rfloor(\overline{x}) \left(\bigsqcup_{j=1}^{n'_r} \sum_{i=1}^{n''_j} nb_f(\overline{a_{ji}})\right) + \lfloor \Delta_r \rfloor(\overline{x})\right) \\ & \sqcup \lfloor \Delta_b \rfloor(\overline{x}) \end{array}$$

for some $n'_r \leq n_r$ and two sets of $n'j$ and $n''_j$ satisfying $n'j \leq n_j$ and $n''j \leq n_j$ for all $j$. As we have shown that this expression is bound by $\Delta_f(\overline{x})$ for the original $n_r$ and $n_j$'s, then we are done. □

**Theorem 4.** *Let $f\ \overline{x_i}^n\ @\ \overline{r_j}^m = e_f$, be the* context *function, $\Gamma$ the inferred global type environment, an initial environment $E_0$ and a heap $h_0$ such that judgement (2) is derivable under an environment $\Sigma'$ in which $f$'s body $e_f$ is replaced with a definition* **hmask** *$e_f$ producing the same final heap and value than $e_f$, but resetting the second component of the resource vector, obtaining $(\delta, 0, s)$.*

*Let $e$ be a subexpression of $e_f$, and let $(e_{bef}, e_{aft}) = splitBA_f\ (\pi_2\ (splitExp_f\ e))$. In case $splitBA_f\ (\pi_2\ (splitExp_f\ e)) = [\,]$ we will consider $(e_{bef}, e_{aft}) = (e, \Box)$ or $(\Box, e)$ as convenient.*

*Then for any pair $(e_b, e_a) \in splitBA_f\ (\pi_2\ (splitExp_f\ e))$ and execution under $\Sigma'$*

$$E \vdash h, k_0, td, (e_b, e_a)_X \Downarrow h', k_0, v, (\delta_b/\delta_a, m'_b/m'_a, s_b/s_a)$$

*such that $E \vdash h, k_0, td, e \Downarrow h', k_0, v, (\delta, m', s)$, belongs to the derivation of (2), it holds that*

1. $[\![e_{bef}]\!]_\Delta\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \succeq_{\overline{s_i}^l, k_0, \phi} \delta_b$, *where $E\ x_i = v_i$ and $s_i = size(h, v_i)$ for each $i \in \{1 \ldots n\}$ and $\phi$ is the region instantiation consistent with $E$ and $\Gamma$.*
2. $[\![e_{bef}]\!]_\mu\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \succeq_{\overline{s_i}^l} m'_b$
3. $[\![e_{aft}]\!]_\mu\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \succeq_{\overline{s_i}^l} m'_a$

*Proof.* By definition

$$[\![e_{bef}]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td = [\![\sqcup_i\ e^i_{bef}]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td$$
$$[\![e_{aft}]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td = [\![\sqcup_i\ e^i_{aft}]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td$$

where $\overline{(e^i_{bef}, e^i_{aft})}^n = splitBA_f\ (\pi_2\ (splitExp_f\ e))$, which means that:

$$[\![e_{bef}]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \geq [\![e_b]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td$$
$$[\![e_{aft}]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td \geq [\![e_a]\!]\ \Sigma[f \mapsto (\Delta_f, 0, 0)]\ \Gamma\ td$$

so we are going to prove the properties for each pair $(e_b, e_a)$. By structural induction on $e$:

$e \equiv c, x, a_1 \oplus a_2, C\ \overline{a_i}^n\ @\ r, g\ \overline{a_i}^n\ @\ \overline{r_j}^l$ In all these cases $e_r = \#$ and $splitBA_f\ e_r = [\,]$, so the properties hold trivially, following proof of Theorem 1.

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r_j}^l$ We have $e_r = e$, $splitBA_f\ e_r = [(f\ \overline{a_i}^n\ @\ \overline{r_j}^l,\ \Box)]$. In this case $e_{bef} = e$ and $e_{aft} = \Box$, and the only pair $(e_b, e_a) = (e, \Box)_{\mathbf{B}}$, so $\delta_b = \delta$, $m'_b = m'_a = 0$. Property (1) hold by Lemma 12, while (2) and (3) hold trivially.

$e \equiv$ **let** $x_1 = e_1$ **in** $e_2$ Let $(e_b, e_a) \in splitBA_f\ (\pi_2\ (splitExp_f\ e))$ and an execution as explained before. Let us use $\Sigma$ instead of $\Sigma[f \mapsto (\Delta_f, 0, 0)]$ in order to simplify. Let $e_r = \pi_2\ (splitExp_f\ e)$, $e_{1r} = \pi_2\ (splitExp_f\ e_1)$ and $e_{2r} = \pi_2\ (splitExp_f\ e_2)$. We considere all the possible pairs belonging to $splitBA_f\ (\pi_2\ (splitExp_f\ e))$:

$e_{1r} = e_{2r} = \#$ The properties hold trivially by Theorem 1.

$e_{1r} = \#$**,** $e_{2r} \neq \#$ In this case $e_r =$ **let** $x_1 = e_1$ **in** $e_{2r}$.

The pairs are:

$$\{(\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2rb}, collapse(\textbf{let}\ x_1 = \Box\ \textbf{in}\ e_{2ra})) \mid (e_{2rb}, e_{2ra}) \in splitBA_f\ e_{2r}\}$$

and $\delta_b = \delta_1 + \delta_{2rb}$, $m'_b = \max\{m'_1, |\delta_1| + m'_{2rb}\}$ and $m'_a = m'_{2ra}$.
Then, we have

$$\begin{array}{l}[\![\textbf{let } x_1 = e_1 \textbf{ in } e_{2rb}]\!]_\Delta\ \Sigma\ \Gamma\ td \\ = [\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ td + [\![e_{2rb}]\!]_\Delta\ \Sigma\ \Gamma\ td \\ \succeq_{\overline{si}^n, k_0, \phi}\ \delta_1 + \delta_{2rb} \qquad \{\text{by i.h.}\} \\ = \delta_b\end{array}$$

$$\begin{array}{l}[\![\textbf{let } x_1 = e_1 \textbf{ in } e_{2rb}]\!]_\mu\ \Sigma\ \Gamma\ td \\ = \sqcup\{[\![e_1]\!]_\mu\ \Sigma\ \Gamma\ td, |[\![e_1]\!]_\Delta\ \Sigma\ \Gamma\ td| + [\![e_{2rb}]\!]_\mu\ \Sigma\ \Gamma\ td\} \\ \succeq_{\overline{si}^n}\ \max\{m'_1, |\delta_1| + m'_{2rb}\} \qquad \{\text{by i.h.}\} \\ = m'_b\end{array}$$

$$\begin{array}{l}[\![\mathit{collapse}(\textbf{let } x_1 = \Box \textbf{ in } e_{2ra})]\!]_\mu\ \Sigma\ \Gamma\ td \\ = [\![e_{2ra}]\!]_\mu\ \Sigma\ \Gamma\ td \\ \succeq_{\overline{si}^n}\ m'_{2ra} \qquad \{\text{by i.h.}\} \\ = m'_a\end{array}$$

**$e_{1r} \neq \#$, $e_{2r} = \#$** In this case $e_r = \textbf{let } x_1 = e_{1r} \textbf{ in } e_2$, the pairs are

$$\{(\textbf{let } x_1 = e_{1rb} \textbf{ in } \Box, \textbf{let } x_1 = e_{1ra} \textbf{ in } e_{2b}) \mid (e_{1rb}, e_{1ra}) \in \mathit{splitBA}_f\ e_{1r}\}$$

where $e_{2b} = \pi_1\ (\mathit{splitExp}_f\ e_2)$. Now $\delta_b = \delta_{1rb}$, $m'_b = \max\{m'_{1rb}, |\delta_{1rb}|\}$ and $m'_a = \max\{m'_{1ra}, |\delta_{1ra}| + m'_{2b}\}$. Then:

$$\begin{array}{l}[\![\textbf{let } x_1 = e_{1rb} \textbf{ in } \Box]\!]_\Delta\ \Sigma\ \Gamma\ td \\ = [\![e_{1rb}]\!]_\Delta\ \Sigma\ \Gamma\ td \\ \succeq_{\overline{si}^n, k_0, \phi}\ \delta_{1rb} \qquad \{\text{by i.h.}\} \\ = \delta_b\end{array}$$

$$\begin{array}{l}[\![\textbf{let } x_1 = e_{1rb} \textbf{ in } \Box]\!]_\mu\ \Sigma\ \Gamma\ td \\ = \sqcup\{[\![e_{1rb}]\!]_\mu\ \Sigma\ \Gamma\ td, |[\![e_{1rb}]\!]_\Delta\ \Sigma\ \Gamma\ td|\} \\ \succeq_{\overline{si}^n, k_0, \phi}\ \max\{m'_{1rb}, |\delta_{1rb}|\} \qquad \{\text{by i.h.}\} \\ = m'_b\end{array}$$

It is easy to prove that as $e_{2r} = \#$ then $e_{2b} = e_2$ so

$$\begin{array}{l}[\![\textbf{let } x_1 = e_{1ra} \textbf{ in } e_{2b}]\!]_\mu\ \Sigma\ \Gamma\ td \\ = \sqcup\{[\![e_{1ra}]\!]_\mu\ \Sigma\ \Gamma\ td, |[\![e_{1ra}]\!]_\Delta\ \Sigma\ \Gamma\ td| + [\![e_{2b}]\!]_\mu\ \Sigma\ \Gamma\ td\} \\ \succeq_{\overline{si}^n, k_0, \phi}\ \max\{m'_{1ra}, |\delta_{1ra}| + m'_{2b}\} \qquad \{\text{by i.h. and Theorem 1}\} \\ = m'_b\end{array}$$

**$e_{1r} \neq \#$, $e_{2r} \neq \#$, $e_{1b} = \#$** In this case

$$e_r = \textbf{let } x_1 = e_{1r} \textbf{ in } e_2$$

The pairs may belong to set $A$:

$$(\textbf{let } x_1 = e_{1r} \textbf{ in in } e_{2rb}, \mathit{collapse}(\textbf{let } x_1 = \Box \textbf{ in } e_{2ra}))$$

or to set $B$ (when $e_{2b} \neq \#$):

$$(\textbf{let } x_1 = e_{1rb} \textbf{ in } \Box, \textbf{let } x_1 = e_{1ra} \textbf{ in } e_{2b})$$

Both cases can be proved similarly by applying induction hypothesis and Theorem **??**.

$e_{1r} \neq \#$, $e_{2r} \neq \#$, $e_{1b} \neq \#$ In this case

$$e_r = \sqcup\{\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2r}, \textbf{let } x_1 = e_{1r} \textbf{ in } e_2\}$$

The pairs may be

$$(\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2rb}, \textbf{let } x_1 = \Box \textbf{ in } e_{2ra})$$

or

$$(\textbf{let } x_1 = e_{1r} \textbf{ in } e_{2rb}, \mathit{collapse}(\textbf{let } x_1 = \Box \textbf{ in } e_{2ra}))$$

or

$$(\textbf{let } x_1 = e_{1rb} \textbf{ in } \Box, \textbf{let } x_1 = e_{1ra} \textbf{ in } e_{2b})$$

All of them are proved similarly by applying induction hypothesis and Theorem **??**.

□

**Lemma 16.** *Let us consider an evaluation of a subexpression $e$ of $f$'s body $e_f$ producing a resource vector $(\delta, m, s)$, such that there are $p \geq 0$ direct calls to $f \in dom\ \Sigma$, each one producing an associated resource vector $(\delta_j, m_j, s_j)$, $j \in P = \{1 \ldots p\}$. We execute now, with the pair semantics of Fig. 7, the equivalent pair $(e_b, e_a)$ under another environment $\Sigma'$, in which $f$'s body $e_f$ is replaced with a definition* **hmask** *$e_f$ producing the same final heap and value than $e_f$, but resetting the second component of the resource vector, obtaining $(\delta_j, 0, s_j)$ instead of $(\delta_j, m_j, s_j)$. Assume that we obtain $(\delta_b/\delta_a, m'_b/m'_a, s_b/s_a)$. Then, the following equation holds:*

$$m \leq \sqcup\ \{m'_b,\ \ \sqcup_{j \in P}\ (m_j - |\,\delta_j\,|) + |\,\delta_b\,|,\ \ |\,\delta_b\,| + m'_a\}$$

*Proof.* By structural induction on $e$. In the following we shall abbreviate $|\,\delta\,|$ by just $\delta$. Also, we will assume $g \neq f$.

$e \equiv c, x, a_1 \oplus a_2, C\ \overline{a_i}^n\ @\ r, g\ \overline{a_i}^n\ @\ \overline{r_j}^l$ In all these cases $P = \emptyset$, $e_r = \#$ and $splitBA_f\ e_r = [\ ]$. So, either $m'_b = m$, $\delta_b = \delta$ and $m'_a = 0$, or $m'_b = \delta_b = 0$ and $m'_a = m$. In both cases, the property holds.

$e \equiv f\ \overline{a_i}^n\ @\ \overline{r_j}^l$ We have $e_r = e$, $splitBA_f\ e_r = [(f\ \overline{a_i}^n\ @\ \overline{r_j}^l,\ \Box)]$, and $P = \{j\}$. So $m = m_j$, $m'_b = m'_a = 0$, $\delta_b = \delta_j$, and we must prove $m_j \leq \sqcup\ \{0, (m_j - \delta_j) + \delta_j, \delta_j + 0\}$, which is obviously true.

$e \equiv$ **case** $x$ **of** $\overline{C_i\ \overline{x_{ij}}^{n_i} \rightarrow e_i}^n$ Let us assume that alternative $i$ is executed. By induction hypothesis we have $m_i \leq \sqcup\ \{m'_{ib},\ \ \sqcup_{j \in P}\ (m_j - \delta_j) + \delta_{ib},\ \ \delta_{ib} + m'_{ia}\}$. Then we have $m = m_i$, $m'_b = m'_{ib}$, $m'_a = m'_{ia}$, $\delta_b = \delta_{ib}$, and the lemma holds by induction hypothesis.

$e \equiv$ **let** $x_1 = e_1$ **in** $e_2$ Assume that the direct calls in $P$ are split into the subset $P_1$ executed in $e_1$, and the subset $P_2$ executed in $e_2$.

If $P = \emptyset$ we can reason as in the first case. Assume $P \neq \emptyset$.

By Lemma 5, if $P \neq \emptyset$, $e$ and $e_r$ produce the same resource consumption. We have different cases for $e_r$ and the pair equivalent to $e$ and $e_r$. Let $(e_{1b}, e_{1r}) = splitExp_f\ e_1$ and $(e_{2b}, e_{2r}) = splitExp_f\ e_2$:

$e_{1r} = e_{2r} = \#$ This case has been already been considered because then $e_r = \#$ and by Lemma 5 necessarily $P = \emptyset$.

$e_{1r} = \#$**,** $e_{2r} \neq \#$ In this case

$$e_r = \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2r}$$

By Lemma 5 necessarily $P_1 = \emptyset$, so $P_2 \neq \emptyset$. By Theorem 3 there is a pair $(e_{2rb}, e_{2ra})_{\mathbf{Z}} \in splitBA_f\ e_{2r}$ equivalent to $e_{2r}$ and by Lemma 5 also to $e_2$. As $P_1 = \emptyset$ we choose $(e_1, \Box)_{\mathbf{B}}$ as equivalent pair. Following proof of Theorem 3, then there are two possible pairs equivalent to $e_r$ according to $\mathbf{Z}$:

$Z = M$ The pair is $(\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2rb}, \textbf{let}\ x_1 = \Box\ \textbf{in}\ e_{2ra})_{\mathbf{M}}$.

$Z = B$ The pair is $(\textbf{let}\ x_1 = e_1\ \textbf{in}\ e_{2rb}, \Box)_{\mathbf{B}}$.

$e_{1r} \neq \#$**,** $e_{2r} = \#$ In this case

$$e_r = \textbf{let}\ x_1 = e_{1r}\ \textbf{in}\ e_2$$

Now, for the same reasons as above, $P_2 = \emptyset$, $P_1 \neq \emptyset$. The equivalent pair is $(\textbf{let}\ x_1 = e_{1rb}\ \textbf{in}\ \Box, \textbf{let}\ x_1 = e_{1ra}\ \textbf{in}\ e_{2b})_{\mathbf{M}}$ where $(e_{1rb}, e_{1ra})_{\mathbf{Z}}$ is the pair equivalent to $e_{1r}$ and $e_1$, and $(\Box, e_{2b})_{\mathbf{A}}$ is the one equivalent to $e_2$.

$e_{1r} \neq \#$, $e_{2r} \neq \#$, $e_{1b} = \#$ Necessarily $P_1 \neq \emptyset$ by Lemma 4. In this case, again

$$e_r = \textbf{let } x_1 = e_{1r} \textbf{ in } e_2$$

but now we have two cases:

$P_2 = \emptyset$ The pair is the same as the previous case.

$P_2 \neq \emptyset$ The pairs can be:

- $(\textbf{let } x_1 = e_{1r} \textbf{ in } e_{2rb}, \textbf{let } x_1 = \Box \textbf{ in } e_{2ra})_{\mathbf{M}}$ if $(e_{1r}, \Box)_{\mathbf{B}}$ and $(e_{2rb}, e_{2ra})_{\mathbf{M}}$ are the pairs equivalent to $e_{1r}$ and $e_{2r}$, and so to $e_1$ and $e_2$.
- $(\textbf{let } x_1 = e_{1r} \textbf{ in } e_{2rb}, \Box)_{\mathbf{B}}$ if $(e_{1r}, \Box)_{\mathbf{B}}$ and $(e_{2rb}, e_{2ra})_{\mathbf{B}}$ are the pairs equivalent to $e_{1r}$ and $e_{2r}$, and so to $e_1$ and $e_2$.

$e_{1r} \neq \#$, $e_{2r} \neq \#$, $e_{1b} \neq \#$ In this case

$$e_r = \sqcup\{\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2r}, \textbf{let } x_1 = e_{1r} \textbf{ in } e_2\}$$

We have the following cases:

$P_1 = \emptyset$ Then, taking pairs $(e_{1b}, \Box)_{\mathbf{B}}$ and $(e_{2rb}, e_{2ra})_{\mathbf{Z}}$ equivalent to $e_1$ and $e_2$ we have pairs

- $(\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2rb}, \textbf{let } x_1 = \Box \textbf{ in } e_{2ra})$ if $Z = M$.
- $(\textbf{let } x_1 = e_{1b} \textbf{ in } e_{2rb}, \Box)$ if $Z = B$.

$P_1 \neq \emptyset$, $P_2 = \emptyset$ In this case the equivalent pair is $(\textbf{let } x_1 = e_{1rb} \textbf{ in } \Box, \textbf{let } x_1 = e_{1ra} \textbf{ in } e_{2b})$, where $(e_{1rb}, e_{1ra})_{\mathbf{Z}}$ and $(\Box, e_{2b})_{\mathbf{A}}$ are equivalent to $e_1$ and $e_2$.

$P_1 \neq \emptyset$, $P_2 \neq \emptyset$ Taking $(e_{1r}, \Box)_{\mathbf{B}}$ and $(e_{2rb}, e_{2ra})_{\mathbf{Z}}$ as equivalent to $e_1$ and $e_2$, we have two cases:

- $(\textbf{let } x_1 = e_{1r} \textbf{ in } e_{2rb}, \textbf{let } x_1 = \Box \textbf{ in } e_{2ra})$ if $Z = M$.
- $(\textbf{let } x_1 = e_{1r} \textbf{ in } e_{2rb}, \Box)$ if $Z = B$.

Recall that $\textbf{let } x_1 = \Box \textbf{ in } \Box$ is collapsed to $\Box$ and we can consider them equivalent. Notice that in all the cases the subexpressions of the components of the pair which is equivalent to $e$ are the respective components of the pairs which are equivalent to $e_1$ and $e_2$, so we can apply i.h. to both subexpressions.

By induction hypothesis we have:

$$\begin{array}{l} m_1 \leq \sqcup\ \{m'_{1b},\ \ \sqcup_{j \in P_1}\ (m_j - \delta_j) + \delta_{1b},\ \ \delta_{1b} + m'_{1a}\} \\ m_2 \leq \sqcup\ \{m'_{2b},\ \ \sqcup_{j \in P_2}\ (m_j - \delta_j) + \delta_{2b},\ \ \delta_{2b} + m'_{2a}\} \end{array}$$

By the operational semantics, we have $m = \max\{m_1, \delta_1 + m_2\}$. By Theorem 3 we know that $\delta_1 = \delta_{1b} + \delta_{1a}$, and by rule *LetP* of Fig. 7 we have $\delta_b = \delta_{1b} + \delta_{2b}$, $m'_b = \max\{m'_{1b}, \delta_{1b} + m'_{2b}\}$, and $m'_a = \max\{m'_{1a}, \delta_{1a} + m'_{2a}\}$. We must prove:

$$m \leq \sqcup\ \{m'_b,\ \ \sqcup_{j \in P}\ (m_j - \delta_j) + \delta_b,\ \ \delta_b + m'_a\}$$

We do it by cases on how $m$ is obtained in the computation $\max\{m_1, \delta_1 + m_2\}$. Remember that $v \leq \sqcup\{v_1, \ldots, v_n\} \equiv \exists i\ .\ v \leq v_i$.

$m = m_1$ We distinguish three subcases:

1. $m_1 \leq m'_{1b}$. Then we get $m = m_1 \leq m'_{1b} \leq m'_b \leq \sqcup\{m'_b, \ldots\}$.
2. $m_1 \leq\ \sqcup_{j \in P_1}\ (m_j - \delta_j) + \delta_{1b}$. Then we get $m = m_1 \leq\ \sqcup_{j \in P_1}\ (m_j - \delta_j) + \delta_{1b} \leq \sqcup_{j \in P}(m_j - \delta_j) + \delta_{1b} \leq\ \sqcup_{j \in P}\ (m_j - \delta_j) + \delta_b$, and the property holds.

3. $m_1 \leq \delta_{1b} + m'_{1a}$. Then we get $m = m_1 \leq \delta_{1b} + m'_{1a} \leq \delta_{1b} + m'_a \leq \delta_b + m'_a$, and the property holds.

$m = \delta_1 + m_2$ We distinguish three subcases:

1. $m_2 \leq m'_{2b}$. Then we get $m = \delta_1 + m_2 \leq \delta_{1b} + \delta_{1a} + m'_{2b}$. We can distinguish two cases according to the proof of Theorem 3:
   (a) $P_2 \neq \emptyset$. In this case we get $\delta_{1a} = 0$. So $m \leq \delta_{1b} + m'_{2b} \leq m'_b \leq \sqcup\{m'_b, \ldots\}$.
   (b) $P_2 = \emptyset$ (so $P_1 \neq \emptyset$). In this case we get $\delta_{2b} = m'_{2b} = 0$. Then, $m_2 = 0$ and $m = \delta_1 \leq m_1$. This case can be dealt with in the same way as $m = m_1$ above.
2. $m_2 \leq \sqcup_{j \in P_2}(m_j - \delta_j) + \delta_{2b}$. Then we get $m = \delta_1 + m_2 \leq \delta_{1a} + \delta_{1b} + \sqcup_{j \in P_2}(m_j - \delta_j) + \delta_{2b} \leq \sqcup_{j \in P}(m_j - \delta_j) + \delta_b + \delta_{1a}$. Again we distinguish two subcases as above:
   (a) $P_2 \neq \emptyset$. In this case we get $\delta_{1a} = 0$. Consequently, $m \leq \sqcup_{j \in P}(m_j - \delta_j) + \delta_b$ and we are done.
   (b) $P_2 = \emptyset$ (so $P_1 \neq \emptyset$). In this case $\delta_{2b} = 0$ so $m_2 \leq \sqcup_{j \in P_2}(m_j - \delta_j) + \delta_{2b} \leq 0 + 0 = 0$. As $m_2 \geq 0$ always holds, we get $m_2 = 0$ and $m = \delta_1 + m_2 = \delta_1 \leq m_1$, and this case can be handled as the subcase $m = m_1$ above.
3. $m_2 \leq \delta_{2b} + m'_{2a}$. Then we get $m = \delta_1 + m_2 \leq \delta_{1a} + \delta_{1b} + \delta_{2b} + m'_{2a} = \delta_b + \delta_{1a} + m'_{2a} \leq \delta_b + m'_a$, and the property holds.

□