On Sharing, Memoization, and Polynomial Time

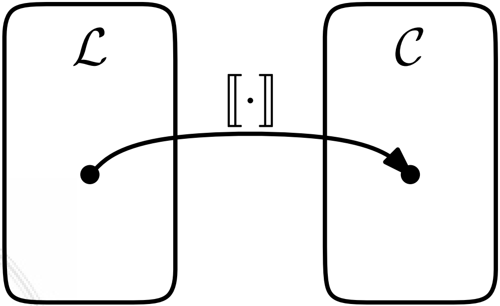
Martin Avanzini[1]
(Joint work with *Ugo Dal Lago*[1])

[1] Università di Bologna & INRIA, Sophia Antipolis

STACS, March 5, 2015

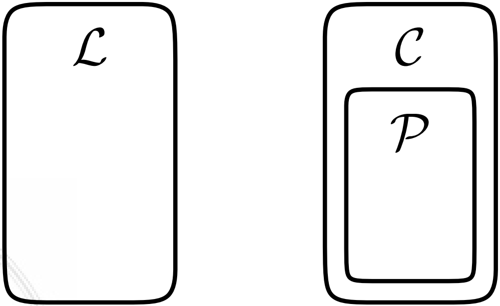
Implicit Computational Complexity

Characterizing Complexity Classes



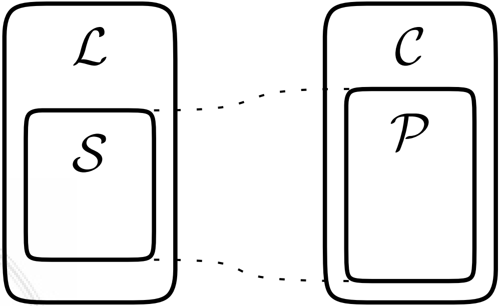
Implicit Computational Complexity

Characterizing Complexity Classes



Implicit Computational Complexity

Characterizing Complexity Classes



General Simultaneous Recursion (GSR)

- let $\mathbb{A}$ be a term algebra formed from constructors $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$
- class of functions definable by **general simultaneous recursion** (**GSR**) is least class of function $\mathtt{f} : \mathbb{A} \times \cdots \times \mathbb{A} \to \mathbb{A}$:
  1. contains **projection** and **constructor** functions
  2. closed under function **composition**
  3. closed under **general simultaneous recursion** (**GSR**)
- functions $\mathtt{f}_1, \ldots, \mathtt{f}_n$ are defined by **GSR** with equations

$$\mathtt{f}_j(\mathbf{c}_i(x_1, \ldots, x_l), \vec{y}) = \mathrm{g}_{i,j}(x_1, \ldots, x_l, \underbrace{\vec{\mathtt{f}}(x_1, \vec{y}), \ldots, \vec{\mathtt{f}}(x_l, \vec{y})}_{n \cdot l \text{ recursive calls}}, \vec{y})$$

where $\vec{\mathtt{f}}(x_i, \vec{y}) = \mathtt{f}_1(x_i, \vec{y}), \ldots, \mathtt{f}_n(x_i, \vec{y})$

General *Ramified* Simultaneous Recursion (GRSR)

ramification **[Leivant, 93; Bellantoni & Cook, 92]**

- take copies $\mathbb{A}_0, \mathbb{A}_1, \mathbb{A}_2, \ldots$ of algebra $\mathbb{A}$
- *ramification* can then be expressed as a typing system

$$\frac{\mathrm{g}_{i,j} \triangleright \mathbb{A}_p^{l} \times \mathbb{A}_q^{n \cdot l} \times \mathbf{A} \to \mathbb{A}_q \qquad p > q}{\mathrm{f}_j \triangleright \mathbb{A}_p \times \mathbf{A} \to \mathbb{A}_q} \;(\mathtt{SimRec})$$

- functions $\mathrm{f}_1, \ldots, \mathrm{f}_n$ are defined by **GSR** with equations

$$\mathrm{f}_j(\mathbf{c}_i(x_1, \ldots, x_l), \vec{y}) = \mathrm{g}_{i,j}(x_1, \ldots, x_l, \underbrace{\vec{\mathrm{f}}(x_1, \vec{y}), \ldots, \vec{\mathrm{f}}(x_l, \vec{y})}_{n \cdot l \text{ recursive calls}}, \vec{y})$$

where $\vec{\mathrm{f}}(x_i, \vec{y}) = \mathrm{f}_1(x_i, \vec{y}), \ldots, \mathrm{f}_n(x_i, \vec{y})$

GRSR on Trees

- we can define functions $\texttt{rabbits}_i : \mathbb{N}_{i+1} \to \mathbb{T}_i$ by

$$\texttt{rabbits}_i(\mathbf{0}) = \mathbf{B}_l \qquad \texttt{b}_i(\mathbf{0}) = \mathbf{B}_l \qquad \texttt{m}_i(\mathbf{0}) = \mathbf{M}_l$$

$$\texttt{rabbits}_i(\mathbf{S}(n)) = \texttt{b}_i(n) \quad \texttt{b}_i(\mathbf{S}(n)) = \mathbf{B}(\texttt{m}_i(n)) \quad \texttt{m}_i(\mathbf{S}(n)) = \mathbf{M}(\texttt{m}_i(n), \texttt{b}_i(n))$$

GRSR on Trees

- we can define functions $\texttt{rabbits}_i : \mathbb{N}_{i+1} \to \mathbb{T}_i$ by

$$\texttt{rabbits}_i(\mathbf{0}) = \mathbf{B}_l \qquad \texttt{b}_i(\mathbf{0}) = \mathbf{B}_l \qquad \texttt{m}_i(\mathbf{0}) = \mathbf{M}_l$$
$$\texttt{rabbits}_i(\mathbf{S}(n)) = \texttt{b}_i(n) \quad \texttt{b}_i(\mathbf{S}(n)) = \mathbf{B}(\texttt{m}_i(n)) \quad \texttt{m}_i(\mathbf{S}(n)) = \mathbf{M}(\texttt{m}_i(n), \texttt{b}_i(n))$$

- data tiering prevents us from defining $\#\texttt{leafs} : \mathbb{T}_j \to \mathbb{N}_i$

$$\#\texttt{leafs}(\mathbf{B}_l) = \#\texttt{leafs}(\mathbf{M}_l) = \mathbf{S}(\mathbf{0})$$
$$\#\texttt{leafs}(\mathbf{B}(t)) = \#\texttt{leafs}(t)$$
$$\#\texttt{leafs}(\mathbf{M}(l, r)) = \texttt{add}_\texttt{i}(\#\texttt{leafs}(l), \#\texttt{leafs}(r))$$

and thus from defining the exponential growing function

$$\texttt{fib}(n) = \#\texttt{leafs}(\texttt{rabbits}_i(n))$$

GRSR Characterizes FPTIME

Theorem (Leivant, 93)

The following classes of functions coincide:

1. *function over strings definiable by GRSR* *constructor arity* $\leqslant 1$
2. *class* FPTIME *of polytime computable functions*

GRSR Characterizes FPTIME

Theorem (Leivant, 93)

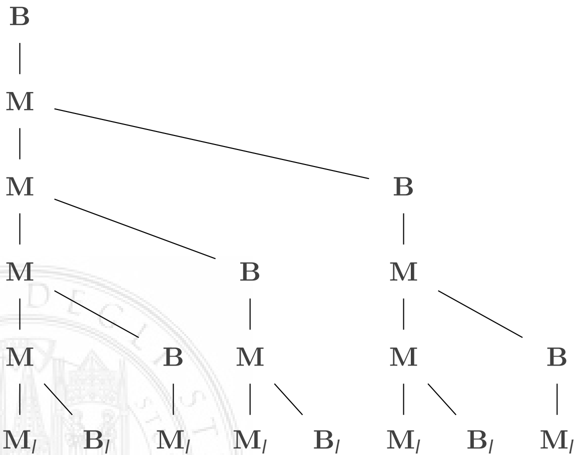
The following classes of functions coincide:

1. *function over strings definiable by GRSR*   *constructor arity* $\leqslant 1$
2. *class* FPTIME *of polytime computable functions*

- expressive power of **GRSR** on trees unknown since $\geqslant 20$ years
- does GRSR lead outside FPTIME in general?

Feasible Evaluation of GRSR Functions

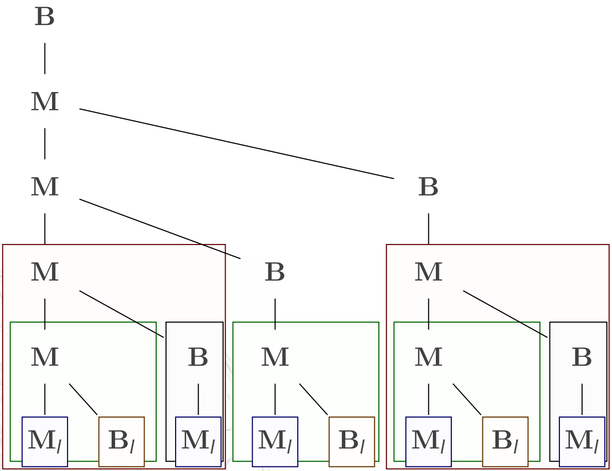
The Need for *Sharing*



(a) result of $\texttt{rabbits}(\mathbf{6})$.

Feasible Evaluation of GRSR Functions

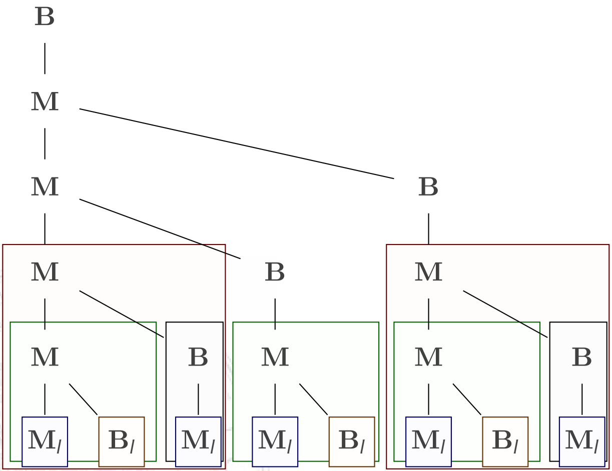
The Need for *Sharing*



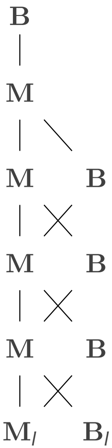
(a) result of `rabbits`$(\mathbf{6})$.

Feasible Evaluation of GRSR Functions

The Need for *Sharing*



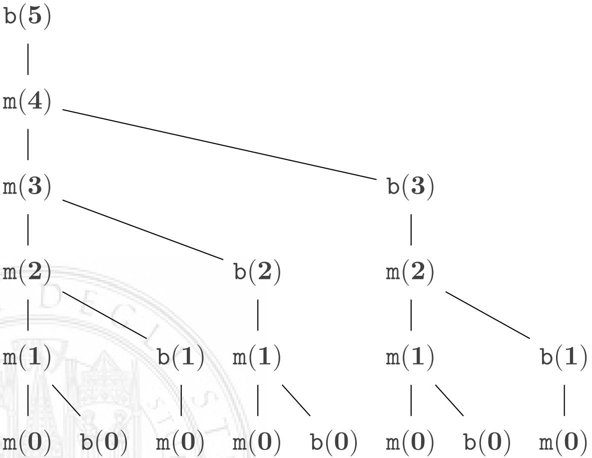
(a) result of `rabbits`$(\mathbf{6})$.



(b) shared.

Feasible Evaluation of GRSR Functions

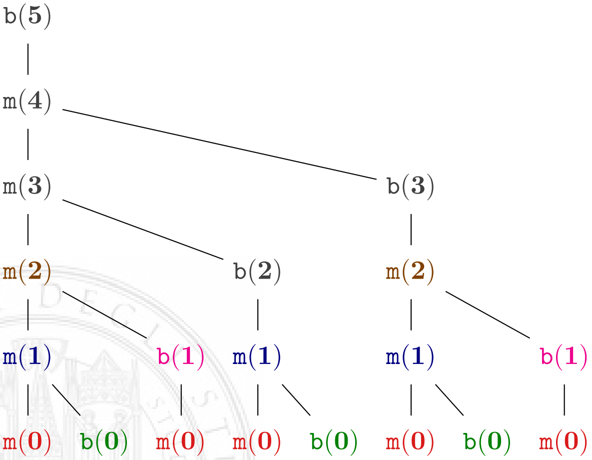
The Need for *Memoisation*



(a) call tree of rabbits(6).

Feasible Evaluation of GRSR Functions

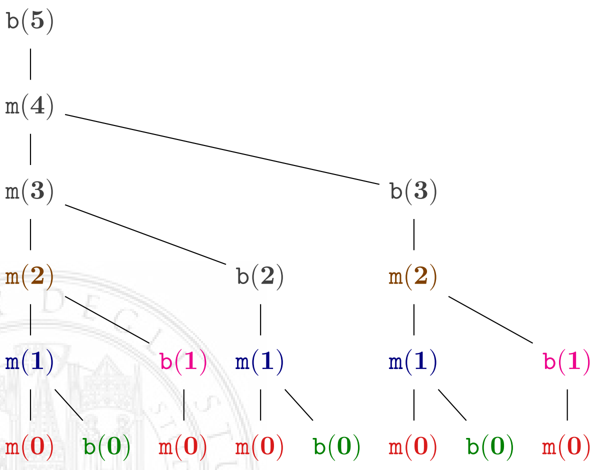
The Need for *Memoisation*



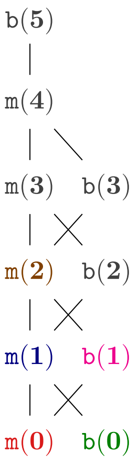
(a) call tree of rabbits(6).

Feasible Evaluation of GRSR Functions

The Need for *Memoisation*



(a) call tree of `rabbits`(6).



(b) memoized.

Key Observations

Let `f` be defined by GRSR.

Suppose $\mathtt{f}(v_1, \ldots, v_k)$ evaluates to u.

Then we can bind by a **polynomial** in the (shared) size of arguments $v_1, \ldots, v_k$:

1. the shared size of result u

   *values can always be represented as a **compact DAG***

2. number of *distinct* function calls in evaluation of $\mathtt{f}(v_1, \ldots, v_k)$

   *reduction with **memoization** feasible*

Definition

shared size of value $v :=$ number of ***distinct* subterms** in value v

Memoization & Sharing, Reconsiled

Call-by-value Memoizing Semantics

- **configuration** is tuple (e, C)
  - e is expression
  - C is cache, mapping calls $\mathtt{f}(\vec{v})$ to results u

- semantics are given as **statements**

$$(\mathtt{f}(\vec{v}), C) \Downarrow_m (u, D)$$

Call-by-value Memoizing Semantics

- **configuration** is tuple (e, C)
  - e is expression
  - C is cache, mapping calls $\mathtt{f}(\vec{v})$ to results u

- semantics are given as **statements**

$$(\mathtt{f}(\vec{v}), C) \Downarrow_m (u, D)$$

Example

$$\frac{(\mathtt{f}(\vec{v}), u) \in C}{(\mathtt{f}(\vec{v}), C) \Downarrow_0 (u, C)} \ (\mathrm{Read})$$

$$\frac{(\mathtt{f}(\vec{v}), u') \notin C \quad \mathtt{f}(\vec{p}) = r \in \mathcal{E} \quad \mathtt{f}(\vec{p})\sigma = \mathtt{f}(\vec{v}) \quad (r\sigma, C) \Downarrow_m (u, D)}{(\mathtt{f}(\vec{v}), C) \Downarrow_{m+1} (u, D \cup \{(\mathtt{f}(\vec{v}), u)\})} \ (\mathrm{Update})$$

Call-by-value Memoizing Semantics

- **configuration** is tuple (e, C)
  - e is expression
  - C is cache, mapping calls $\mathtt{f}(\vec{v})$ to results u
- semantics are given as **statements**

$$(\mathtt{f}(\vec{v}), C) \Downarrow_m (u, D)$$

Example

$$\frac{(\mathtt{f}(\vec{v}), u) \in C}{(\mathtt{f}(\vec{v}), C) \Downarrow_0 (u, C)} \ (\mathrm{Read})$$

$$\frac{(\mathtt{f}(\vec{v}), u') \notin C \quad \mathtt{f}(\vec{p}) = r \in \mathcal{E} \quad \mathtt{f}(\vec{p})\sigma = \mathtt{f}(\vec{v}) \quad (r\sigma, C) \Downarrow_m (u, D)}{(\mathtt{f}(\vec{v}), C) \Downarrow_{m+1} (u, D \cup \{(\mathtt{f}(\vec{v}), u)\})} \ (\mathrm{Update})$$

- gives rist to a **cost model**, where re-occurring calls are free

memoized cost

Integrating Sharing

crucial, one can now define an **implementation** such that:

1. each reduction step is atomic
   - **no copying** of arbitrary large data
   - data is stored on a **heap** H
2. overheads are *“small”*

Integrating Sharing

crucial, one can now define an **implementation** such that:

1. each reduction step is atomic
   - **no copying** of arbitrary large data
   - data is stored on a **heap** H
2. overheads are *“small”*

implementation is given as reduction relation $\rightarrow_{\mathtt{Rrsm}}$ on configurations

$$(e, H, C)$$

- H is **heap**
- e is **expression**
- C is **cache**

(e and C:) contain references to heap

Polynomial Invariance of Memoized Cost Model

Theorem

$(\mathtt{f}(\vec{v}), \varnothing) \Downarrow_m (u, C)$ *if and only if* $(\mathtt{f}(\vec{v}), \varnothing, \varnothing) \rightarrow^{n}_{\mathtt{Rrsm}} (\ell, H, C')$ *where*

- *result* u *is stored in final heap* H *at location* ℓ
- $n \leqslant \delta \cdot m + \mathsf{size}(\vec{v})$ *for* $\delta \in \mathbb{N}$
- $\mathsf{size}((\ell, H, C)) \leqslant \Delta \cdot m + \mathsf{size}(\vec{v})$ *for* $\Delta \in \mathbb{N}$

Polynomial Invariance of Memoized Cost Model

Theorem

$(\mathtt{f}(\vec{v}), \varnothing) \Downarrow_m (u, C)$ *if and only if* $(\mathtt{f}(\vec{v}), \varnothing, \varnothing) \rightarrow^n_{\mathtt{Rrsm}} (\ell, H, C')$ *where*

- *result* u *is stored in final heap* H *at location* ℓ
- $n \leqslant \delta \cdot m + \mathsf{size}(\vec{v})$ *for* $\delta \in \mathbb{N}$
- $\mathsf{size}((\ell, H, C)) \leqslant \Delta \cdot m + \mathsf{size}(\vec{v})$ *for* $\Delta \in \mathbb{N}$

Corollary (Polynomial Invariance of Memoized Cost Model)

There exists a polynomial $p_{\mathtt{f}} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ *such that for* $(\mathtt{f}(\vec{v}), \varnothing) \Downarrow_m (u, C)$, *the value* u *represented as DAG is computable from arguments* $\vec{v}$ *in time* $p_{\mathtt{f}}(\mathsf{size}(\vec{v}), m)$.

GRSR is Sound for Polynomial Time

Theorem

Let $\mathtt{f} : \mathbf{A} \to \mathbb{A}_m$ *be a function defined by* ***GRSR****.*

For all inputs $\vec{v}$*, a DAG representation of* $\mathtt{f}(\vec{v})$ *is* ***computable in time polynomial*** *in the sizes of the inputs.*

GRSR is Sound for Polynomial Time

Theorem

Let $\mathtt{f} : \mathbf{A} \to \mathbb{A}_m$ *be a function defined by* ***GRSR****.*

For all inputs $\vec{v}$*, a DAG representation of* $\mathtt{f}(\vec{v})$ *is* ***computable in time polynomial*** *in the sizes of the inputs.*

Outline.

- by observation on number of distinct function calls during evaluation

$$(\mathtt{f}(\vec{v}), \varnothing) \Downarrow_m (u, C) \quad \Longrightarrow \quad m \leqslant p_{\mathtt{f}}(\mathsf{size}(\vec{v}))$$

for a polynomial $p_{\mathtt{f}}$

GRSR is Sound for Polynomial Time

Theorem

Let $\mathtt{f} : \mathbf{A} \to \mathbb{A}_m$ *be a function defined by* ***GRSR***.

For all inputs $\vec{v}$, *a DAG representation of* $\mathtt{f}(\vec{v})$ *is* ***computable in time polynomial*** *in the sizes of the inputs.*

Outline.

- by observation on number of distinct function calls during evaluation

$$(\mathtt{f}(\vec{v}), \varnothing) \Downarrow_m (u, C) \quad \Longrightarrow \quad m \leqslant p_{\mathtt{f}}(\mathsf{size}(\vec{v}))$$

  for a polynomial $p_{\mathtt{f}}$

- the theorem then follows from polynomial invariance of memoized cost model

□

Conclusion

1. memoized cost gives rise to notion of **memoized runtime complexity**, this cost model is **polynomial invariant** if we allow sharing
2. **general simultaneous ramified recursion** is sound for **polynomial time**
   - extensions, such as parameter substitution, lead immediately outside polynomial time

Thanks!

Cost Annotated Memoizing Semantics

$$\frac{(\mathtt{f}(\vec{v}), v) \in C}{(\mathtt{f}(\vec{v}), C) \Downarrow (v, C)} \ (\mathtt{Read})$$

$$\frac{(\mathtt{f}(\vec{v}), u') \notin C \quad \mathtt{f}(\vec{p}) = r \in \mathcal{E} \quad \mathtt{f}(\vec{p})\sigma = \mathtt{f}(\vec{v}) \quad (r\sigma, C) \Downarrow (u, D)}{(\mathtt{f}(\vec{v}), C) \Downarrow \quad (u, D \cup \{(\mathtt{f}(\vec{v}), u)\})} \ (\mathtt{Update})$$

$$\frac{\mathrm{f} \in \mathcal{F} \quad (t_i, C_{i-1}) \Downarrow (v_i, C_i) \quad (\mathtt{f}(\vec{v}), C_k) \Downarrow (v, C_{k+1})}{(\mathtt{f}(t_1, \dots, t_k), C_0) \Downarrow \quad (v, C_{k+1})} \ (\mathtt{Split})$$

$$\frac{\mathbf{c} \in \mathcal{C} \quad (t_i, C_{i-1}) \Downarrow (v_i, C_i)}{(\mathbf{c}(t_1, \dots, t_k), C_0) \Downarrow \quad (\mathbf{c}(\vec{v}), C_k)} \ (\mathtt{Con})$$

Cost Annotated Memoizing Semantics

$$\frac{(\mathtt{f}(\vec{v}), v) \in C}{(\mathtt{f}(\vec{v}), C) \Downarrow_0 (v, C)} \ (\mathtt{Read})$$

$$\frac{(\mathtt{f}(\vec{v}), u') \notin C \quad \mathtt{f}(\vec{p}) = r \in \mathcal{E} \quad \mathtt{f}(\vec{p})\sigma = \mathtt{f}(\vec{v}) \quad (r\sigma, C) \Downarrow_m (u, D)}{(\mathtt{f}(\vec{v}), C) \Downarrow_{m+1} (u, D \cup \{(\mathtt{f}(\vec{v}), u)\})} \ (\mathtt{Update})$$

$$\frac{\mathrm{f} \in \mathcal{F} \quad (t_i, C_{i-1}) \Downarrow_{n_i} (v_i, C_i) \quad (\mathtt{f}(\vec{v}), C_k) \Downarrow_n (v, C_{k+1})}{(\mathtt{f}(t_1, \ldots, t_k), C_0) \Downarrow_{n + \sum_{i=1}^{k} n_i} (v, C_{k+1})} \ (\mathtt{Split})$$

$$\frac{\mathbf{c} \in \mathcal{C} \quad (t_i, C_{i-1}) \Downarrow_{n_i} (v_i, C_i)}{(\mathbf{c}(t_1, \ldots, t_k), C_0) \Downarrow_{\sum_{i=1}^{k} n_i} (\mathbf{c}(\vec{v}), C_k)} \ (\mathtt{Con})$$

Small Step Semantics

Memoization and Sharing Reconsiled

$$\frac{(\mathtt{f}(\vec{\ell}), \ell) \in C}{(E[\mathtt{f}(\vec{\ell})], H, C) \to_{\mathtt{r}} (E[\ell], H, C)} \quad (\mathtt{read})$$

$$\frac{(\mathtt{f}(\vec{\ell}), \ell) \notin C \qquad \mathtt{f}(\vec{p}) = r \in \mathcal{E} \qquad \text{“}\mathtt{f}(\vec{\ell}) \text{ matches } \mathtt{f}(\vec{p}) \text{ with } \sigma : \mathcal{V} \to \mathrm{Loc}_H\text{”}}{(E[\mathtt{f}(\vec{\ell})], H, C) \to_{\mathtt{R}} (E[\langle \mathtt{f}(\vec{\ell}), r\sigma \rangle], H, C)} \quad (\mathtt{rule})$$

$$\frac{}{(E[\langle \mathtt{f}(\vec{\ell}), \ell \rangle], H, C) \to_{\mathtt{s}} (E[\ell], H, C \cup \{(\mathtt{f}(\vec{\ell}), \ell)\})} \quad (\mathtt{store})$$

$$\frac{(H', \ell) = \mathrm{merge}(H, \mathbf{c}(\vec{\ell}))}{(E[\mathbf{c}(\vec{\ell})], H, C) \to_{\mathtt{m}} (E[\ell], H', C)} \quad (\mathtt{merge})$$