

- 1) unfold the application $f(e'_1, \dots, e'_n)$ to get e' ;
- 2) incrementalize e' with information set I' , cache set C' , and definition set D' to get e'' ;
- 3) eliminate dead parameters of f' , defined by $f'(v_1, \dots, v_k) = e''$, in computing $f'(v_1, \dots, v_k)$.

Note that the second step uses the function \mathcal{Inc} , which may use $\mathcal{IncApply}$ recursively for function applications. After the third step, if we obtain $f'(v_{i_1}, \dots, v_{i_j})$ and, if f' is fully defined, $t(f'(v_{i_1}\theta, \dots, v_{i_j}\theta)) \leq t(f(e_1, \dots, e_n))$, then we replace $f(e_1, \dots, e_n)$ by $f'(v_{i_1}\theta, \dots, v_{i_j}\theta)$. The set D is changed as a side effect.

Dead Parameter Elimination. After the second step above, $f'(v_1, \dots, v_k)$ computes $f(e'_1, \dots, e'_n)$ and is defined as e'' . Since e'' is obtained by replacing some subcomputations of $f(e'_1, \dots, e'_n)$ depending on x by computations depending on the current cache parameter, those parameters of f' on which the replaced computations depend may become dead.

Dead code elimination is a traditional optimization [1, 24]. We assume a subroutine \mathcal{Elim} is given so that $\mathcal{Elim}[[f'(v_1, \dots, v_k)]D'']$, where $f'(v_1, \dots, v_k)$ is defined as e'' in D'' , returns the pair $\langle f'(v_{i_1}, \dots, v_{i_j}), D''' \rangle$, where $1 \leq i_1 < \dots < i_j \leq k$ and $f'(v_{i_1}, \dots, v_{i_j})$ is defined as some e''' in D''' after dead parameter elimination, and $f'(v_{i_1}, \dots, v_{i_j})$ returns a value if and only if $f'(v_1, \dots, v_k)$ returns the same value.

Example. Consider our running example. For the application $row(car(C), insert(i, a, R))$ in the false branch of (6), we introduce a new function row' as in (17). To obtain a definition of row' , we first unfold $row(c, insert(i, a, R))$ to get

$$\begin{aligned} & \mathbf{if} \text{ null}(insert(i, a, R)) \mathbf{then} \text{ nil} \\ & \mathbf{else} \text{ cons}(c * car(insert(i, a, R)), row(c, cdr(insert(i, a, R)))) \end{aligned} \quad (20)$$

Then, we incrementalize (20) using $row(c, R) = r_1$ as given by the cache set in (17). The incrementalization is sketched as follows. It is easy to see that $insert(i, a, R)$ in the condition can be unfolded and the condition simplified to true, and thus (20) is reduced to

$$\text{cons}(c * car(insert(i, a, R)), row(c, cdr(insert(i, a, R)))) \quad (21)$$

The first occurrence of $insert(i, a, R)$ in (21) can be unfolded, conditions in the unfolded application lifted, and car of $cons$ applications simplified. Thus, (21) becomes

$$\begin{aligned} & \mathbf{if} \ i \leq 1 \mathbf{then} \text{ cons}(c * a, row(c, cdr(insert(i, a, R)))) \\ & \mathbf{else if} \ \text{null}(R) \mathbf{then} \text{ cons}(c * a, row(c, cdr(insert(i, a, R)))) \\ & \mathbf{else} \text{ cons}(c * car(R), row(c, cdr(insert(i, a, R)))) \end{aligned} \quad (22)$$

The three occurrences of $insert(i, a, R)$ in (22) can be specialized under their corresponding contexts, unfolded, and then cdr of $cons$ applications simplified. Thus, (22) becomes

$$\begin{aligned} & \mathbf{if} \ i \leq 1 \mathbf{then} \text{ cons}(c * a, row(c, R)) \\ & \mathbf{else if} \ \text{null}(R) \mathbf{then} \text{ cons}(c * a, row(c, nil)) \\ & \mathbf{else} \text{ cons}(c * car(R), row(c, insert(i-1, a, cdr(R)))) \end{aligned} \quad (23)$$

In the first branch of (23), $row(c, R)$ can be directly replaced by r_1 . In the second branch, $row(c, nil)$ can be specialized and unfolded to nil . For the third branch, we have $\text{null}(R) \leftrightarrow F$; thus $row(c, R)$ is specialized to $\text{cons}(c * car(R), row(c, cdr(R)))$ and the cache set is extended so that

$$c * car(R) = car(r_1) \quad \text{and} \quad row(c, cdr(R)) = cdr(r_1).$$

