

The New Simplification Algorithm

John H. Williams, Edward L. Wimmers
IBM Research - Almaden Research Center
K53-803
650 Harry Road
San Jose, CA 95120-6099

February 7, 1990

1 Justification for the Simplification Algorithm

An obscure note: $\text{Func}(\mathbf{e})$ is an expression that is guaranteed to be a function that is SEF. Our names are divided into two classes: the function names and the names that can stand for any value. The names that can stand for any value will usually be denoted by x . During the Func processing, Func may assume that a function name is a function but may only assume that one of the other names is a SEF expression.

Thruout this note, it assumed that the expression $\lambda(x).\mathbf{e}$ does NOT coerce the $\mathbf{e}$ to $\text{Func}(\mathbf{e})$. This language restriction (when adopted) will be taken care of by the parser. This is a technical advantage to the simplifier since it will not have to explicitly do the coercion. For example, not having this restriction allows the simplifier to recurse more easily since not all subexpressions of a function expression need be functions (e.g. ~ 3).

Theorem 1.1

Assume that pattern P makes the definitions $\{\text{def } p_1 \equiv e_1 \dots \text{def } p_n \equiv e_n\}$. Then
$$\lambda(P)E \equiv (\lambda(x).E[p_1:x/p_1, \dots, p_n:x/p_n]) \text{ where}$$
$$\{\text{def } p_1 \equiv \gamma:\text{Func}(e_1) \dots \text{def } p_n \equiv \gamma:\text{Func}(e_n)\}$$

Proof: This seems likely to be true. $\square$

Note: it is assumed that $U:f:t$ produces $\perp_{\text{err}}$ if t is not a pair.

Recall that $C = GC:2$.

Lemma 1.2

Let $F_1, \dots, F_m, G_1, \dots, G_n$ be arbitrary expressions with all the expressions $F_1, \dots, F_m$ having no occurrences of any one of the function names $g_1, \dots, g_n$. Then

$$\begin{aligned} & E \text{ where } \{\text{def } f_1 \equiv F_1 \dots \text{def } f_m \equiv F_m \text{ def } g_1 \equiv G_1 \dots \text{def } g_n \equiv G_n\} \\ \equiv & (E \text{ where } \{\text{def } g_1 \equiv G_1 \dots \text{def } g_n \equiv G_n\}) \text{ where } \{\text{def } f_1 \equiv F_1 \dots \text{def } f_m \equiv F_m\} \end{aligned}$$

Proof: This follows easily by computing the meanings of the two expressions. $\square$

Lemma 1.3

Let $F_1, \dots, F_m$ be arbitrary expressions none of which have any occurrence of the name x . Then

$$\begin{aligned} & \lambda(x).(E \text{ where } \{\text{def } f_1 \equiv F_1 \dots \text{def } f_m \equiv F_m\}) \\ \equiv & (\lambda(x).E) \text{ where } \{\text{def } f_1 \equiv F_1 \dots \text{def } f_m \equiv F_m\} \end{aligned}$$

Proof: This follows easily by computing the meanings of the two expressions. $\square$

Lemma 1.4

Let $G_1, \dots, G_n$ be arbitrary expressions. Then

$$\begin{aligned}
 & \lambda(x).(E \text{ where } \{\text{def } g_1 \equiv G_1 \quad \dots \quad \text{def } g_n \equiv G_n\}) \\
 \equiv & \quad (\lambda(x).E[g_1:x/g_1, \dots, g_n:x/g_n]) \quad \text{where} \\
 & \quad \{ \\
 & \quad \quad \text{def } g_1 \equiv \text{delay}:(\lambda(x).(\text{Func}(G_1)[g_1:x/g_1, \dots, g_n:x/g_n])) \\
 & \quad \quad \dots \\
 & \quad \quad \text{def } g_n \equiv \text{delay}:(\lambda(x).(\text{Func}(G_n)[g_1:x/g_1, \dots, g_n:x/g_n])) \\
 & \quad \}
 \end{aligned}$$

Furthermore, in the RHS expression, note that $g_i:x$ is an FSEF for any SEF x provided this does not cause delay to be optimized away.

Proof: This lemma is too complicated to possibly be false. $\square$

As Alex noted, the `delay` is really needed in the previous lemma. Here is a counterexample if the `delay` is not included.

$\lambda(x).(g \text{ where } \{\text{def } g \equiv g\}) \equiv \sim \text{nt}$
 but $(\lambda(x).(g:x)) \text{ where } \{\text{def } g \equiv \lambda(x).(g:x)\} \equiv \text{nt}.$

Theorem 1.5

Let $F_1, \dots, F_m, G_1, \dots, G_n$ be arbitrary expressions with all the expressions $F_1, \dots, F_m$ having no occurrences of any one of the function names $x, g_1, \dots, g_n$. Then

$$\begin{aligned}
 & \lambda(x.) (E \text{ where} \\
 & \quad \{ \\
 & \quad \quad \text{def } f_1 \equiv F_1 \quad \dots \quad \text{def } f_m \equiv F_m \\
 & \quad \quad \text{def } g_1 \equiv G_1 \quad \dots \quad \text{def } g_n \equiv G_n \\
 & \quad \}) \\
 \equiv & \quad (\lambda(x.) E[g_1:x/g_1, \dots, g_n:x/g_n]) \quad \text{where} \\
 & \quad \{ \\
 & \quad \quad \text{def } f_1 \equiv F_1 \quad \dots \quad \text{def } f_m \equiv F_m \\
 & \quad \quad \text{def } g_1 \equiv \text{delay}:(\lambda(x.) (\text{Func}(G_1)[g_1:x/g_1, \dots, g_n:x/g_n])) \\
 & \quad \quad \dots \\
 & \quad \quad \text{def } g_n \equiv \text{delay}:(\lambda(x.) (\text{Func}(G_n)[g_1:x/g_1, \dots, g_n:x/g_n])) \\
 & \quad \}
 \end{aligned}$$

Furthermore, in the RHS expression, note that $g_i:x$ is an FSEF for any SEF x provided this does not cause delay to be optimized away.

Proof: Easily follows from the previous Lemmas. $\square$

The following conventions are used

a is a SEF and that does not depend on **x**

b is a SEF for any SEF **x**

f is a FSEF and that does not depend on **x**

g is a FSEF for any SEF **x**

X is a pattern giving names to **x**

y is an arbitrary expression (not even assumed to be SEF)

$\lambda_s(\mathbf{X})y$ produces $\perp_{err}$ if the pattern **X** produces a non-true answer

Here are some definitions. It is assumed that $\perp_{err}$ is produced if the argument does not have the indicated form.

$$\mathbf{U}:g:\langle t_1, t_2 \rangle \equiv g:t_1:t_2$$

$$\mathbf{GU}:i:g:\langle t_1, \dots, t_i \rangle \equiv g:\langle t_1, \dots, t_{i-1} \rangle:t_i$$

$$\mathbf{GC}:i:g:t_1:\dots:t_i \equiv g:\langle t_1, \dots, t_i \rangle \text{ for } i \geq 1$$

$$\mathbf{C} \equiv \mathbf{GC}:2$$

$$\mathbf{ra}:i:\langle t_1, \dots, t_i \rangle \equiv \langle \langle t_1, \dots, t_{i-1} \rangle, t_i \rangle$$

Definition 1.6 *A function **f** is **n**-distributive iff $f:\langle g_1, \dots, g_n \rangle \circ \mathbf{Safe}:g_0 \equiv f:\langle g_1 \circ \mathbf{Safe}:g_0, \dots, g_n \circ \mathbf{Safe}:g_0 \rangle$ for all functions $g_1, \dots, g_n, g_0$.*

*A function **f** is distributive iff it is **n**-distributive for all **n**.*

Note that **cons** is distributive, **cond** is 3 -distributive, and **f'** is distributive for all functions **f**.

Here are some interesting facts.

Lemma 1.7

1. $U:(C:g) \triangleright g$
2. $C:(U:g) \equiv \text{delay}:g$
3. $GU:i:(C:g) \equiv g \circ \text{ra}:i$
4. $GC:i:(C:g) \equiv GC:(i+1):(g \circ \text{ra}:(i+1))$ for $i \geq 2$
5. $U:g \circ \text{ra}:i \equiv GU:i:g$
6. $(\lambda_s(\llbracket X_1, \dots, X_i \rrbracket)y) \circ \text{Safe}:s1 \circ \text{ra}:(i+1) \equiv \lambda_s(\llbracket X_1, \dots, X_i, tt \rrbracket)y$
7. $(\lambda_s(\llbracket X_1, \dots, X_{i-1}, X_i \rrbracket)y) \circ \text{ra}:i \equiv \lambda_s(\llbracket X_1, \dots, X_{i-1}, X_i \rrbracket)y$
8. $U:(GC:i:g) \triangleright GC:(i-1):(g \circ \text{Safe}:(a1 \circ \llbracket s1, s2 \rrbracket, t1 \circ t1)))$ for $i \geq 3$
9. if f is n -distributive, then

$$\lambda_s(X)(f:\langle g_1, \dots, g_n \rangle) \triangleright C:(f:\langle U:(\lambda_s(X)g_1), \dots, U:(\lambda_s(X)g_n) \rangle)$$
10. $\lambda_s(X)(f \circ g) \triangleright C:(f \circ U:(\lambda_s(X)g))$
11. $\lambda_s(X)(g_1 \circ g_2) \triangleright C:(((\lambda_s(X)g_1) \circ \text{Safe}:s1) : ' (U:(\lambda_s(X)g_2)))$
12. $\lambda_s(X)(K:b) \triangleright C:(\lambda_s(X)b \circ (\text{Safe}:s1))$

Proof: All cases except case 9 are left to the reader. Note that since g_i is FSEF, it follows that $\lambda_s(X)g_i \triangleright \text{delay}:(\lambda_s(X)g_i)$. Thus, the following calculation holds for any value t .

$$\begin{aligned}
 & (\lambda_s(X)(f:\langle g_1, \dots, g_n \rangle):t) \\
 & \equiv f:\langle (\lambda_s(X)g_1):t, \dots, (\lambda_s(X)g_n):t \rangle \\
 & \triangleright f:\langle \text{delay}:(\lambda_s(X)g_1):t, \dots, \text{delay}:(\lambda_s(X)g_n):t \rangle \\
 & \equiv f:\langle C:(U:(\lambda_s(X)g_1)):t, \dots, C:(U:(\lambda_s(X)g_n)):t \rangle \\
 & \equiv f:\langle U:(\lambda_s(X)g_1) \circ [\sim t, \text{id}], \dots, U:(\lambda_s(X)g_n) \circ [\sim t, \text{id}] \rangle \\
 & \equiv f:\langle U:(\lambda_s(X)g_1), \dots, U:(\lambda_s(X)g_n) \rangle \circ [\sim t, \text{id}] \\
 & \equiv C:(f:\langle U:(\lambda_s(X)g_1), \dots, U:(\lambda_s(X)g_n) \rangle):t
 \end{aligned}$$

From this it follows that $\lambda_s(X)(f:\langle g_1, \dots, g_n \rangle) \equiv C:(f:\langle U:(\lambda_s(X)g_1), \dots, U:(\lambda_s(X)g_n) \rangle)$. $\square$

Using Lemma 1.7, the following facts can be established for all $i \geq 2$.

if $f = \text{cond}$ and $n = 3$ or $f = \text{cons}$ or f has the form $\text{lift}:f_0$, then

$$\begin{aligned} \lambda_s(X)(f:\langle g_1, \dots, g_n \rangle) &\triangleright C:(f:\langle U:(\lambda_s(X)g_1), \dots, U:(\lambda_s(X)g_n) \rangle) \\ GC:i:(\lambda_s(X)(f:\langle g_1, \dots, g_n \rangle)) &\triangleright \\ GC:(i+1):(f:\langle GU:(i+1):(\lambda_s(X)g_1), \dots, GU:(i+1):(\lambda_s(X)g_n) \rangle) & \\ U:(\lambda_s(X)(f:\langle g_1, \dots, g_n \rangle)) &\triangleright f:\langle U:(\lambda_s(X)g_1), \dots, U:(\lambda_s(X)g_n) \rangle \\ GU:i:(\lambda_s(X)(f:\langle g_1, \dots, g_n \rangle)) &\triangleright f:\langle GU:i:(\lambda_s(X)g_1), \dots, GU:i:(\lambda_s(X)g_n) \rangle \end{aligned}$$

The special case for composition is as follows:

$$\begin{aligned} \lambda_s(X)(f \circ g) &\triangleright C:(f \circ U:(\lambda_s(X)g)) \\ GC:i:(\lambda_s(X)(f \circ g)) &\triangleright GC:(i+1):(f \circ GU:(i+1):(\lambda_s(X)g)) \\ U:(\lambda_s(X)(f \circ g)) &\triangleright f \circ U:(\lambda_s(X)g) \\ GU:i:(\lambda_s(X)(f \circ g)) &\triangleright f \circ GU:i:(\lambda_s(X)g) \end{aligned}$$

if X_0 has the form $\llbracket X_1, \dots, X_{i-1} \rrbracket$, then

$$\begin{aligned} \lambda_s(X)(g_1 \circ g_2) &\triangleright C:(\lambda_s(\llbracket X, tt \rrbracket)g_1) : ' (U:(\lambda_s(X)g_2))) \\ GC:i:(\lambda_s(\llbracket X_1, \dots, X_i \rrbracket)(g_1 \circ g_2)) & \\ \triangleright GC:(i+1):(\lambda_s(\llbracket X_1, \dots, X_i, tt \rrbracket)g_1) : ' (GU:(i+1):(\lambda_s(X)g_2)) &) \\ U:(\lambda_s(X)(g_1 \circ g_2)) &\triangleright (\lambda_s(\llbracket X, tt \rrbracket)g_1) : ' (U:(\lambda_s(X)g_2)) \\ GU:i:(\lambda_s(X_0)(g_1 \circ g_2)) &\triangleright (\lambda_s(X_0)g_1) : ' (GU:i:(\lambda_s(X_0)g_2)) \end{aligned}$$

Here is the general case for K

$$\begin{aligned} \lambda_s(X)(K:b) &\triangleright C:(\lambda_s(\llbracket X, tt \rrbracket)b) \\ GC:i:(\lambda_s(\llbracket X_1, \dots, X_i \rrbracket)(K:b)) &\triangleright GC:(i+1):((\lambda_s(\llbracket X_1, \dots, X_i, tt \rrbracket))b) \\ U:(\lambda_s(X)(K:b)) &\triangleright \lambda_s(\llbracket X, tt \rrbracket)b \\ GU:i:(\lambda_s(\llbracket X_1, \dots, X_{i-1} \rrbracket)(K:b)) &\triangleright \lambda_s(\llbracket X_1, \dots, X_{i-1}, tt \rrbracket)b \end{aligned}$$

Here are some optimizations that the Abstractor can use in the general case.

1. $(\lambda_s(\mathbf{X})y) \circ \text{Safe:si} \circ \text{Safe}:(\Delta:(\text{islen:}j)) \equiv \lambda_s(\llbracket \text{tt}, \dots, \text{tt}, \mathbf{X}, \text{tt}, \dots, \text{tt} \rrbracket)y$
where $\mathbf{X}$ occurs in position i
2. $U:(K:g) \equiv g \circ \text{Safe:s2} \circ \text{Safe}:(\Delta:\text{ispair})$
3. $GU:i:(K:g) \equiv g \circ \text{Safe:si} \circ \text{Safe}:(\Delta:(\text{islen:i}))$
4. $U:(g_1:g_2) \equiv (g_1 \circ \text{Safe:s1} \circ \text{Safe}:(\Delta:\text{ispair}))$
 $:\text{' } (g_2 \circ \text{Safe:s1} \circ \text{Safe}:(\Delta:\text{ispair})) :\text{' Safe:s2}$
5. $GU:i:(g_1:g_2) \equiv (g_1 \circ \text{Safe:s1} \circ \text{ra:i}) :\text{' } (g_2 \circ \text{Safe:s1} \circ \text{ra:i}) :\text{' Safe:si}$

Here are some preliminary results that help in the general case of the Abstractor.

1. $\lambda'_s(\mathbf{X}_1) \dots \lambda'_s(\mathbf{X}_{n-1}) \lambda_s(\mathbf{X}_n)y \triangleright GC:n:(\lambda_s(\llbracket \mathbf{X}_1, \dots, \mathbf{X}_n \rrbracket)y)$ for $n \geq 2$
2. $\lambda'_s(\mathbf{X}_1) \dots \lambda'_s(\mathbf{X}_{n-1})(\text{delay}:(\lambda_s(\mathbf{X}_n)y))$
 $\triangleright GC:(n+1):(GU:(n+1):(\lambda_s(\llbracket \mathbf{X}_1, \dots, \mathbf{X}_n \rrbracket)y))$ for $n \geq 2$
3. $\text{delay}:(\lambda_s(\mathbf{X})y) \equiv C:(U:(\lambda_s(\mathbf{X})y))$
4. $\lambda_s(\mathbf{X})(y_1:y_2) \equiv (\lambda_s(\mathbf{X})y_1):\text{' } (\lambda_s(\mathbf{X})y_2)$
5. $\lambda_s(\mathbf{X})(\langle y_1, \dots, y_n \rangle) \equiv [\lambda_s(\mathbf{X})y_1, \dots, \lambda_s(\mathbf{X})y_n]$
6. $\lambda_s(\mathbf{X})(a) \triangleright K:a$
7. $\lambda_s(\llbracket \mathbf{X}_1, \dots, \mathbf{X}_n \rrbracket)(x) \triangleright (\lambda_s(\mathbf{X}_i)(x)) \circ \text{Safe:si}$ if $\mathbf{X}_i$ gives a definition to x
8. $\lambda_s(x).x \equiv \text{id}$

Here is how the general case for handling the Right-hand-sides. In these rules, $\lambda'_s(\mathbf{x})y$ stands for either $\lambda_s(\mathbf{x})y$ or $\text{delay}:(\lambda_s(\mathbf{x})y)$.

$$\lambda'_s(\mathbf{x}_1) \dots \lambda'_s(\mathbf{x}_{n-1}) \lambda_s(\mathbf{x}_n) y \triangleright \text{GC:n}:(\lambda_s(\llbracket \mathbf{x}_1, \dots, \mathbf{x}_n \rrbracket) y) \quad \text{for } n \geq 2$$

$$\lambda'_s(\mathbf{x}_1) \dots \lambda'_s(\mathbf{x}_{n-1}) (\text{delay}:(\lambda_s(\mathbf{x}_n) y)) \triangleright \text{GC:(n+1)}:(\text{GU:(n+1)}:(\lambda_s(\llbracket \mathbf{x}_1, \dots, \mathbf{x}_n \rrbracket) y)) \quad \text{for } n \geq 2$$

$$\text{delay}:(\lambda_s(\mathbf{x}) y) \equiv \text{C:}(\text{U:}(\lambda_s(\mathbf{x}) y))$$

Here is general case for the Abstractor for $i \geq 2$.

Lambda

$$\lambda_s(X_1)\lambda_s(X_2)y \equiv C:(\lambda_s(\llbracket X_1, X_2 \rrbracket)y)$$

$$GC:i:(\lambda_s(\llbracket X_1, \dots, X_i \rrbracket)\lambda_s(X_0)y) \equiv GC:(i+1):(\lambda_s(\llbracket X_1, \dots, X_i, X_0 \rrbracket)y)$$

$$U:(\lambda_s(X_1)\lambda_s(X_2)y) \equiv \lambda_s(\llbracket X_1, X_2 \rrbracket)y$$

$$GU:i:(\lambda_s(\llbracket X_1, \dots, X_{i-1} \rrbracket)\lambda_s(X_0)y) \equiv \lambda_s(\llbracket X_1, \dots, X_{i-1}, X_0 \rrbracket)y$$

Application (no special optimization for GC)

$$\lambda_s(X)(y_1:y_2) \equiv (\lambda_s(X)y_1):'(\lambda_s(X)y_2)$$

$$U:(\lambda_s(X)(y_1:y_2)) \equiv (\lambda_s(\llbracket X, tt \rrbracket)y_1) : ' (\lambda_s(\llbracket X, tt \rrbracket)y_2) : ' \text{ Safe:s2}$$

$$GU:i:(\lambda_s(\llbracket X_1, \dots, X_{i-1} \rrbracket)(y_1:y_2)) \equiv \\ (\lambda_s(\llbracket X_1, \dots, X_{i-1}, tt \rrbracket)y_1) : ' (\lambda_s(\llbracket X_1, \dots, X_{i-1}, tt \rrbracket)y_2) : ' \text{ Safe:si}$$

Sequences (No special optimizations)

$$\lambda_s(X)(\langle y_1, \dots, y_n \rangle) \equiv [\lambda_s(X)y_1, \dots, \lambda_s(X)y_n]$$

Atoms – no special optimizations. Free functions have some optimizations (but none for GC).

$$\lambda_s(X)(a) \triangleright K:a$$

$$U:(\lambda_s(X)(f)) \triangleright f \circ \text{Safe:s2}$$

$$GU:i:(\lambda_s(X)(f)) \triangleright f \circ \text{Safe:si}$$

Functions symbols g that depend on $x_1, \dots, x_n$ with $n \geq 1$ have no special optimization for GC.

$$\lambda_s(X)g \equiv (g \circ \lambda_s(X)x_1):'\lambda_s(X)x_2:'\dots\lambda_s(X)x_n$$

$$U:(\lambda_s(X)g) \equiv (g \circ \lambda_s(\llbracket X, tt \rrbracket)x_1) : ' (\lambda_s(\llbracket X, tt \rrbracket)x_2) : ' \dots : ' (\lambda_s(\llbracket X, tt \rrbracket)x_n) \\ : ' \text{ Safe:s2}$$

$$GU:i:(\lambda_s(\llbracket X_1, \dots, X_{i-1} \rrbracket)g) \equiv (g \circ \lambda_s(\llbracket X_1, \dots, X_{i-1}, tt \rrbracket)x_1) : ' \\ (\lambda_s(\llbracket X_1, \dots, X_{i-1}, tt \rrbracket)x_2) : ' \dots : ' (\lambda_s(\llbracket X_1, \dots, X_{i-1}, tt \rrbracket)x_n) \\ : ' \text{ Safe:si}$$

$\lambda_s(\llbracket X_1, \dots, X_n \rrbracket)(x) \triangleright (\lambda_s(X_i)(x)) \circ \text{Safe:si}$ if X_i gives a definition to x

$$\lambda_s(x).x \equiv \text{id}$$

Recall that $\text{split}:i:\langle x_1, \dots, x_j \rangle \equiv \langle \langle x_1, \dots, x_i \rangle, \langle x_{i+1}, \dots, x_j \rangle \rangle$ for $i \leq j$.

Here are some rewrite rules to use in the rewriter:

1. $U:\text{id} \triangleright \text{apply}$
2. $U:g \triangleright (g \circ \text{Safe}:s1):'(\text{Safe}:s2)$
3. $GC:i:(GC:j:g) \equiv GC:(i+j-1):(g \circ \text{Safe}:(\text{al} \circ \text{split}:i))$
4. $\text{Safe}:(s1 \circ s1 \circ \text{split}:j) \triangleright \text{Safe}:s1$
5. $\text{Safe}:(s1 \circ s2 \circ \text{split}:j) \triangleright \text{Safe}:(s:(i+j))$
6. $C:g:b \equiv g \circ [\sim b, \text{id}]$
7. $GC:i:g:b \equiv GC:(i-1):(g \circ \text{Safe}:(\text{al} \circ [\sim b, \text{id}]))$ for $i \geq 3$
8. $C:(U:g) \equiv \text{delay}:g$

Example 1.8

In the course of this example, the effects of Theorem 1.1 are ignored. Given the program

$$\lambda(x_1.p_1)\lambda(x_2.p_2)E$$

the parser produces

$$p_1 \rightarrow \lambda_s(x_1.)(p_2 \rightarrow \lambda_s(x_2.)E; e_2); e_1$$

Simplification proceeds on the true-arm as follows:

$$\begin{aligned} & \lambda_s(x_1.)(p_2 \rightarrow \lambda_s(x_2.)E; e_2) \\ \equiv & C : (U : (\lambda_s(x_1.)p_2) \rightarrow U : (\lambda_s(x_1.)\lambda_s(x_2.)E); U : (\lambda_s(x_1.)e_2)) \\ \equiv & C : (p_2 \circ \text{Safe} : s_2 \rightarrow U : (\lambda_s(x_1.)\lambda_s(x_2.)E); e_2 \circ \text{Safe} : s_2) \\ \equiv & C : (p_2 \circ \text{Safe} : s_2 \rightarrow \lambda_s(\llbracket x_1., x_2. \rrbracket)E; e_2 \circ \text{Safe} : s_2) \end{aligned}$$

Thus the original program

$$\lambda(x_1.p_1)\lambda(x_2.p_2)E$$

is turned into the program

$$p_1 \rightarrow C : (p_2 \circ \text{Safe} : s_2 \rightarrow \lambda_s(\llbracket x_1., x_2. \rrbracket)E; e_2 \circ \text{Safe} : s_2); e_1$$

whereas the old simplification algorithm produced

$$p_1 \rightarrow (\sim p_2 \rightarrow' \lambda_s(x_1.)\lambda_s(x_2.)E; \sim e_2); e_1$$

The new version is simpler in two ways. First of all, the new version uses the Curry and selection functions whereas the old version uses the K function and the prime operator. Thus, the new version localizes the higher-order effects to the occurrences of the Curry function in contrast to the old version which tended to put a prime or tilde (and sometimes more than one) on practically everything. The second improvement is perhaps even more important. Lambda appears in the new version as the single lambda $\lambda_s(\llbracket x_1., x_2. \rrbracket)$ whereas it appears as the double lambda $\lambda_s(x_1.)\lambda_s(x_2.)$ in the old version. This double (and, in general, multiple) occurrence of lambda in the old version meant there was a potential for a large blow-up in the size of the final program. The single (albeit more complex) occurrence of lambda in the new version greatly limits the potential for blow-up.

Notice that the new form rewrites easily. For example, if $p_2 \equiv tt$, it is an easy task (two steps) for the rewriter to simplify

$p_1 \rightarrow C : (p_2 \circ \text{Safe} : s_2 \rightarrow \lambda_s(\llbracket x_1., x_2. \rrbracket)E; e_2 \circ \text{Safe} : s_2); e_1$ to $p_1 \rightarrow C : (\lambda_s(\llbracket x_1., x_2. \rrbracket)E); e_1$ using already existing rules. However, the old form does not rewrite so easily. For example, either a new rule to express $\rightarrow'$ as a Curried function or a new rule such as $(\sim tt \rightarrow' g_1; g_2) \equiv g_1$ would have to be added, and this rule is valid only if g_1, g_2 are function-valued functions.