Principled Strength Reduction

Y. Annie Liu

Computer Science Department
Indiana University

[Liu 1997]: Yanhong A. Liu "Principled strength reduction" In Richard S. Bird and Lambert Meertens, editors, Algorithmic Languages and Calculi, pages 357-381. Chapman & Hall, London, 1997.

Incremental Programs

f' is an <u>incremental version</u> of f under $\oplus$
if f' computes $f(x \oplus y)$ efficiently by making use of $f(x)$.

<u>Examples</u>:

- f is $sort$, x is a list of numbers, $x \oplus i$ is $cons(i, x)$.

  if $sort(x) = r$, then $sort'(x, i, r) = sort(cons(i, x))$.

- f is C compiler, x is C program, $\oplus$ is change to program.

- f is loop body, x is ind. variable, $\oplus$ is increment to ind. var.

Strength Reduction

- example

$$
\begin{array}{lll}
i := 1; & & d := m * k; \\
\textbf{while } i \le n \textbf{ do} & & \\
\quad \vdots & & \\
\quad a := m * i; & a := m * (i + k) & a := a + d; \\
\quad \vdots & \quad = m * i + m * k & \\
\quad i := i + k; & &
\end{array}
$$

- some other compiler optimizations:
  - loop-invariant hoisting
  - data specialization
  - ...

[Bauer, Allen, Cocke, Schwartz, Kennedy, late 60's and 70's]

Incrementalization for Program Optimization

- loop optimization:

  Step 1. form the incrementalization problem

  Step 2. incrementalize

  Step 3. form incrementalized loop

- associated optimizations:

  Step 4. fold initialization

  Step 5. replace termination test

  Step 6. minimize loop body

<u>Cost model</u>: asymptotic time. **<u>Goal</u>**: save 1 time, 2 space, 3 code.

Step 1. Form Incrementalization Problem

- transform a loop into a **while** loop of the form:

$$\begin{array}{ll} s := s_1; & -\text{initialization} \\ \textbf{while } c(s) \textbf{ do} & -\text{termination condition} \\ \quad s := b(s) & -\text{loop body} \end{array}$$

- formulate f and $\oplus$:

$$f(x) = b(x) \qquad\qquad x' = x \oplus y = b(x)$$

Step 1. Form Incrementalization Problem—Example

```
n = input();
m = input();
k = input();
i = 1;
while i ≤ n do
   a = m * i;
   i = i + k;
output(a);
```

assume $n > 0$, $m > 0$, $k > 0$

$< n, m, k, a, i > = < in, in, in,\ default,\ 1 >$;
$\textbf{while } i \le n \textbf{ do}$
$\quad < n, m, k, a, i > = < n, m, k,\ m * i,\ i + k >$

obtain:

$$f(n, m, k, a, i) = < n, m, k,\ m * i,\ i + k >$$

$$\langle n', m', k', a', i' \rangle = \langle n, m, k,\ m * i,\ i + k \rangle$$

Step 2. Incrementalize

- from f and $\oplus$, we obtain

  $\tilde{f}$: extends $f(x)$ with intermediate results and auxiliary info.

  $\tilde{f}'$: computes $\tilde{f}(x \oplus y)$ incrementally

  Π: projects f out of $\tilde{f}$ and $\tilde{f}'$

- if $f(x) = r$, then

$$\Pi(\tilde{f}(x)) = r$$

  if $\tilde{f}(x) = \tilde{r}$, then

$$\tilde{f}'(x, y, \tilde{r}) = \tilde{f}(x \oplus y)$$

Step 2. Incrementalization — Example

$$f(n, m, k, a, i) = < n, m, k,\ m * i,\ i + k >$$

$$\langle n', m', k', a', i' \rangle = \langle n, m, k,\ m * i,\ i + k \rangle$$

incrementalize:

$$m' * i' = m * (i + k) = m * i + m * k$$

obtain:

$$\begin{array}{rcl} \tilde{f}(n, m, k, a, i) & = & < n, m, k,\ m * i,\ i + k,\ m * k > \\ \tilde{f}'(n, m, k, a, i, d) & = & < n, m, k,\ a + d,\ i + k,\ d > \end{array}$$

Step 3. Form Incrementalized Loop

original program

$$\begin{array}{l} s := s_1; \\ \textbf{while } c(s) \textbf{ do} \\ \quad s := b(s) \end{array}$$

$$\tilde{b}(s) = \tilde{f}(s)$$
$$\tilde{b}'(\tilde{s}) = \tilde{f}'(\tilde{s})$$

new program

$$\begin{array}{l} s := s_1; \\ \textbf{if } c(s) \textbf{ then} \\ \quad \tilde{s} := \tilde{b}(s); \\ \quad \textbf{while } c(\Pi(\tilde{s})) \textbf{ do} \\ \quad\quad \tilde{s} := \tilde{b}'(\tilde{s}); \\ \quad s := \Pi(\tilde{s}) \end{array}$$

$$f(x) = b(x)$$
$$x' = x \oplus y = b(x)$$

Step 3. Form Incrementalized Loop — Example

$$\tilde{f}(n,m,k,a,i) \;=\; <n,m,k,\; m*i,\; i+k,\; m*k>$$
$$\tilde{f}'(n,m,k,a,i,d) \;=\; <n,m,k,\; a+d,\; i+k,\; d>$$

obtain:

$$\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ i = 1; \\ \textbf{if } i \le n \textbf{ then} \\ \quad a = m*i; \\ \quad i = i+k; \\ \quad d = m*k; \\ \quad \textbf{while } i \le n \textbf{ do} \\ \quad\quad a = a+d; \\ \quad\quad i = i+k; \end{array}$$

Step 4. Fold Initialization

original program

$$\begin{array}{l} s := s_1; \\ \textbf{while } c(s) \textbf{ do} \\ \quad s := b(s) \end{array} \quad \Longrightarrow$$

new program

$$\begin{array}{l} s := s_1; \\ \textbf{if } c(s) \textbf{ then} \\ \quad \tilde{s} := \tilde{b}(s); \\ \quad \textbf{while } c(\Pi(\tilde{s})) \textbf{ do} \\ \quad\quad \tilde{s} := \tilde{b}'(\tilde{s}); \\ \quad s := \Pi(\tilde{s}) \end{array}$$

$$\Downarrow$$

look for s_0, s.t.
$b(s_0) = s_1$,
$c(s_0) = true$;
try s_0 be $b^{-1}(s_1)$

$$\begin{array}{l} s := s_0; \\ \textbf{while } c(s) \textbf{ do} \\ \quad s := b(s) \end{array} \quad \Longrightarrow \quad \begin{array}{l} \tilde{s} := \tilde{b}(s_0); \\ \textbf{while } c(\Pi(\tilde{s})) \textbf{ do} \\ \quad \tilde{s} := \tilde{b}'(\tilde{s}); \\ s := \Pi(\tilde{s}) \end{array}$$

Step 4. Fold initialization — Example

$$\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ i = 1; \\ \textbf{if } i \le n \textbf{ then} \\ \quad a = m*i; \\ \quad i = i+k; \\ \quad d = m*k; \\ \quad \textbf{while } i \le n \textbf{ do} \\ \quad\quad a = a+d; \\ \quad\quad i = i+k; \end{array}$$

obtain:

$$\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ \\ \\ a = m; \\ i = 1+k; \\ d = m*k; \\ \textbf{while } i \le n \textbf{ do} \\ \quad a = a+d; \\ \quad i = i+k; \end{array}$$

or:

$$\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ a = m*(1-k); \\ i = 1; \\ d = m*k; \\ \textbf{while } i \le n \textbf{ do} \\ \quad a = a+d; \\ \quad i = i+k; \end{array}$$

$$f(n,m,k,a,i) \;=\; <n,m,k,\; m*i,\; i+k>$$
$$\tilde{f}(n,m,k,a,i) \;=\; <n,m,k,\; m*i,\; i+k,\; m*k>$$

Step 5. Replace Termination Test

- replace $c(\Pi(\tilde{s}))$, which equals $c(s)$, by $\tilde{c}(\tilde{s})$

$$\boxed{\begin{array}{l} \tilde{s} := \tilde{b}(s_0); \\ \textbf{while } c(\Pi(\tilde{s})) \textbf{ do} \\ \quad \tilde{s} := \tilde{b}'(\tilde{s}); \\ s := \Pi(\tilde{s}) \end{array}} \quad \Longrightarrow \quad \boxed{\begin{array}{l} \tilde{s} := \tilde{b}(s_0); \\ \textbf{while } \tilde{c}(\tilde{s}) \textbf{ do} \\ \quad \tilde{s} := \tilde{b}'(\tilde{s}); \\ s := \Pi(\tilde{s}) \end{array}}$$

- generalize induction variable elimination in classical strength reduction to the elimination of all replaceable variables.

Step 5. Replace Termination Test — Example

$$\boxed{\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ a = m*(1-k); \\ i = 1; \\ d = m*k; \\ \textbf{while } i \leq n \textbf{ do} \\ \quad a = a+d; \\ \quad i = i+k; \end{array}} \quad \begin{array}{c} a = m*i \\ i \leq n \quad \Longleftrightarrow \quad a \leq m*n \end{array} \quad \boxed{\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ a = m*(1-k); \\ i = 1; \\ d = m*k; \\ e = m*n; \\ \textbf{while } a \leq e \textbf{ do} \\ \quad a = a+d; \\ \quad i = i+k; \end{array}}$$

Step 6. Minimize Loop Body

1. analyze $\tilde{b}$ to determine retrieval of the final value from the additional values maintained

2. analyze $\tilde{b}'$ to determine all the values needed for the retrieval

3. prune the values that are not needed out of $\tilde{b}$ and $\tilde{b}'$

Step 6. Minimize Loop Body — Example

obtain:

$$\boxed{\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ a = m*(1-k); \\ i = 1; \\ d = m*k; \\ e = m*n; \\ \textbf{while } a \leq e \textbf{ do} \\ \quad a = a+d; \\ \quad i = i+k; \end{array}} \quad \boxed{\begin{array}{l} n = input(); \\ m = input(); \\ k = input(); \\ a = m*(1-k); \\ \\ d = m*k; \\ e = m*n; \\ \textbf{while } a \leq e \textbf{ do} \\ \quad a = a+d; \\ \\ \end{array}}$$

one optimization yet to complete: save more space.
let a, d, e reuse the space of n, m, k.

Summary and Discussion

- – form the problem, incrementalize, form new loop
  – fold init., replace termination test, minimize loop body
- – Steps 1 and 3 can be fully automated
  – Steps 2, 4-6 are systematic but are parameterized by equality reasoning, dependence analysis, cost analysis,
- extensions: multiple loops, arrays, recursive functions
- the principle of the method is language independent.

An example from VLSI design

binary integer square root

```
n := input;  m := 2^(l-1);
for i := l-2 downto 0 do
    p := n - m^2;
    if p > 0 then
        m := m + 2^i
    else if p < 0 then
        m := m - 2^i;
output(m)
```

non-restoring method

```
p := input;  v := 0;  w := 2^(2*(l-1));
while w >= 1 do
    if p > 0 then
        p := p - v - w;  v := v/2 + w;  w := w/4
    else if p < 0 then
        p := p + v - w;  v := v/2 - w;  w := w/4
    else v := v/2;  w := w/4;
output(v)
```


Step 1. Form Incrementalization Problem—Example

```
n := input;  m := 2^(l-1);
i := l-2;
while i >= 0 do
   p := n - m^2;
   if p > 0 then
        m := m + 2^i
   else if p < 0 then
        m := m - 2^i;
   i := i - 1;
output(m)
```

$$< n, m, i > := < input, 2^{l-1}, l-2 >;$$
while $i \geq 0$ **do**
$$< n, m, i > := < n, update(n, m, i), i-1 >$$

$update(n,m,i) =$ **let** $p = n - m^2$ **in**
  if $p > 0$ **then**
    $m + 2^i$
  else if $p < 0$ **then**
    $m - 2^i$
  else m

$$f(n,m,i) = < n,\ \underline{update(n,m,i)},\ i-1 >$$

$$\langle n', m', i' \rangle = \langle n,\ update(n,m,i),\ i-1 \rangle$$

Step 2. Incrementalize — Example (1)

$\Pi = 1st$
$.m$
$.p\ .u\ .v\ .w$

$\widetilde{update}(n,m,i) =$ **let** $p = \underline{n - m^2}$ **in**
  if $p > 0$ **then**
    let $u = \underline{2^i}$ **in** $< m+u,\ p,\ u,\ \underline{2*m*u},\ \underline{u^2} >$
  else if $p < 0$ **then**
    let $u = \underline{2^i}$ **in** $< m-u,\ p,\ u,\ \underline{2*m*u},\ \underline{u^2} >$
  else $< m,\ 0 >$

- p, u: intermediate results
- $2*m*u$, u^2: auxiliary info (for maintaining p, u after updates)

$$\langle n', m', i' \rangle = \langle n, update(n,m,i), i-1 \rangle = \langle n, m \pm u, i-1 \rangle$$

$$p:\ n-(m \pm u)^2 = n - m^2 \mp 2*m*u - u^2; \quad u:\ 2^{(i-1)} = 2^i/2$$

- $\widetilde{update}(n', m', i')$: 3 cases under each of the 3 cases for m'

Step 2. Incrementalize — Example (2)

$\widetilde{update}'(\tilde{r_1}) =$ for $\widetilde{update}(n,m,i) = \tilde{r_1}$

$$
\begin{array}{l}
\textbf{if } \tilde{r_1}.p > 0 \textbf{ then} \\
\quad \textbf{let } p = \tilde{r_1}.p - \tilde{r_1}.v - \tilde{r_1}.w \textbf{ in} \\
\quad \textbf{if } p > 0 \textbf{ then} \\
\quad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } < \tilde{r_1}.m+u,\ p,\ u,\ \tilde{r_1}.v/2+\tilde{r_1}.w,\ \tilde{r_1}.w/4 > \\
\quad \textbf{else if } p < 0 \textbf{ then} \\
\quad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } < \tilde{r_1}.m-u,\ p,\ u,\ \tilde{r_1}.v/2+\tilde{r_1}.w,\ \tilde{r_1}.w/4 > \\
\quad \textbf{else } < \tilde{r_1}.m,\ 0 > \\
\textbf{else if } \tilde{r_1}.p < 0 \textbf{ then} \\
\quad \textbf{let } p = \tilde{r_1}.p + \tilde{r_1}.v - \tilde{r_1}.w \textbf{ in} \\
\quad \textbf{if } p > 0 \textbf{ then} \\
\quad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } < \tilde{r_1}.m+u,\ p,\ u,\ \tilde{r_1}.v/2-\tilde{r_1}.w,\ \tilde{r_1}.w/4 > \\
\quad \textbf{else if } p < 0 \textbf{ then} \\
\quad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } < \tilde{r_1}.m-u,\ p,\ u,\ \tilde{r_1}.v/2-\tilde{r_1}.w,\ \tilde{r_1}.w/4 > \\
\quad \textbf{else } < \tilde{r_1}.m,\ 0 > \\
\textbf{else } < \tilde{r_1}.m,\ 0 >
\end{array}
$$

Step 3. Form Incrementalized Loop — Example

$$\tilde{f}(n,m,i) = \ <n,\ \widetilde{update}(n,m,i),\ i-1>$$

$$\tilde{f}'(n,\tilde{r_1},i) = \ <n,\ \widetilde{update}'(\tilde{r_1}),\ i-1>$$

$$\Pi(<n,\tilde{r_1},i>) = \ <n,\ 1st(\tilde{r_1}),\ i>$$

multiplications, exponentials $\longrightarrow$ additions, shifts

Step 4. Fold Initialization — Example

$$s_1 = <input,\ 2^{l-1},\ l-2>$$

$b(s_0) = s_1$
$c(s_0) = true$ i.e., $l-1 \geq 0$

$$s_0 = <input,\ 0,\ l-1>$$

$\tilde{s_1} = \tilde{b}(s_0)$

$$\tilde{s_1} = <input, <2^{l-1}, \underline{input}, 2^{l-1}, \underline{0}, \underline{2^{2*(l-1)}}>, l-1>$$

Step 5. Replace Termination Test — Example

$$
\begin{array}{ll}
\widetilde{update}_1(n,m,i) = & \textbf{let } p = n - m^2 \textbf{ in} \\
& \textbf{if } p > 0 \textbf{ then} \\
& \quad \textbf{let } u = 2^i \textbf{ in } < m+u, p, u, 2*m*u, u^2 > \\
& \textbf{else if } p < 0 \textbf{ then} \\
& \quad \textbf{let } u = 2^i \textbf{ in } < m-u, p, u, 2*m*u, u^2 > \\
& \textbf{else let } u = 2^i \textbf{ in } < m, 0, \underline{u}, \underline{2*m*u}, \underline{u^2} >
\end{array}
$$

- test $i' \geq 0$ after assignments by $\tilde{b}$: $u = 2^i$, $w = u^2 = 2^{2*i}$, $i' = i-1$

  $i' \geq 0$ iff $i \geq 1$ iff $u \geq 2$ iff $u^2 \geq 4$ iff $\underline{w \geq 4}$

- eliminate the maintenance of i

Step 6. Minimize Loop Body — Example (1)

$$
\begin{array}{ll}
\widetilde{update}_1(n,m,i) = & \textbf{let } \underline{p} = n - m^2 \textbf{ in} \\
& \textbf{if } p > 0 \textbf{ then} \\
& \qquad \textbf{let } \underline{u} = 2^i \textbf{ in } < \underline{m+u}, p, u, 2{*}m{*}u, u^2 > \\
& \textbf{else if } p < 0 \textbf{ then} \\
& \qquad \textbf{let } \underline{u} = 2^i \textbf{ in } < \underline{m-u}, p, u, 2{*}m{*}u, u^2 > \\
& \textbf{else let } \underline{u} = 2^i \textbf{ in } < \underline{m}, 0, u, 2{*}m{*}u, u^2 >
\end{array}
$$

∨

- at loop exit: $i' = -1$ iff $i = 0$ iff $u = 1$; so, $m = v/2$. retrieve final return value from additional values maintained:

$$\textbf{if } p > 0 \textbf{ then } v/2{+}1 \textbf{ else if } p < 0 \textbf{ then } v/2{-}1 \textbf{ else } v/2$$

Step 6. Minimize Loop Body — Example (2)

$$
\begin{array}{l}
\widetilde{update}_1{}'(\tilde{r_1}) = \\
\quad \textbf{if } \underline{\tilde{r_1}.p > 0} \textbf{ then} \\
\qquad \textbf{let } \underline{p} = \underline{\tilde{r_1}.p - \tilde{r_1}.v - \tilde{r_1}.w} \textbf{ in} \\
\qquad \textbf{if } p > 0 \textbf{ then} \\
\qquad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } <\tilde{r_1}.m{+}u,\ \underline{p},\ u,\ \underline{\tilde{r_1}.v/2{+}\tilde{r_1}.w},\ \underline{\tilde{r_1}.w/4}> \\
\qquad \textbf{else if } p < 0 \textbf{ then} \\
\qquad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } <\tilde{r_1}.m{-}u,\ \underline{p},\ u,\ \underline{\tilde{r_1}.v/2{+}\tilde{r_1}.w},\ \underline{\tilde{r_1}.w/4}> \\
\qquad \textbf{else let } u = \tilde{r_1}.u/2 \textbf{ in } <\tilde{r_1}.m,\ \underline{0},\ u,\ \underline{\tilde{r_1}.v/2{+}\tilde{r_1}.w},\ \underline{\tilde{r_1}.w/4}> \\
\quad \textbf{else if } \underline{\tilde{r_1}.p < 0} \textbf{ then} \\
\qquad \textbf{let } \underline{p} = \underline{\tilde{r_1}.p + \tilde{r_1}.v - \tilde{r_1}.w} \textbf{ in} \\
\qquad \textbf{if } p > 0 \textbf{ then} \\
\qquad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } <\tilde{r_1}.m{+}u,\ \underline{p},\ u,\ \underline{\tilde{r_1}.v/2{-}\tilde{r_1}.w},\ \underline{\tilde{r_1}.w/4}> \\
\qquad \textbf{else if } p < 0 \textbf{ then} \\
\qquad\quad \textbf{let } u = \tilde{r_1}.u/2 \textbf{ in } <\tilde{r_1}.m{-}u,\ \underline{p},\ u,\ \underline{\tilde{r_1}.v/2{-}\tilde{r_1}.w},\ \underline{\tilde{r_1}.w/4}> \\
\qquad \textbf{else let } u = \tilde{r_1}.u/2 \textbf{ in } <\tilde{r_1}.m,\ \underline{0},\ u,\ \underline{\tilde{r_1}.v/2 - \tilde{r_1}.w},\ \underline{\tilde{r_1}.w/4}> \\
\quad \textbf{else let } u = \tilde{r_1}.u/2 \textbf{ in } <\tilde{r_1}.m,\ \underline{0},\ u,\ \underline{\tilde{r_1}.v/2},\ \underline{\tilde{r_1}.w/4}>
\end{array}
$$

Step 6. Minimize Loop Body — Example (3)

$$
\begin{array}{ll}
\widetilde{update}_2(n,m,i) = & \textbf{let } p = n - m^2 \textbf{ in} \\
& \textbf{let } u = 2^i \textbf{ in } < p, 2{*}m{*}u, u^2 >
\end{array}
$$

$$
\begin{array}{ll}
\widetilde{update}_2{}'(\tilde{r_2}) = & \textbf{if } \tilde{r_2}.p > 0 \textbf{ then} \\
& \qquad < \tilde{r_2}.p - \tilde{r_2}.v - \tilde{r_2}.w,\ \tilde{r_2}.v/2 + \tilde{r_2}.w,\ \tilde{r_2}.w/4 > \\
& \textbf{else if } \tilde{r_2}.p < 0 \textbf{ then} \\
& \qquad < \tilde{r_2}.p + \tilde{r_2}.v - \tilde{r_2}.w,\ \tilde{r_2}.v/2 - \tilde{r_2}.w,\ \tilde{r_2}.w/4 > \\
& \textbf{else } < \tilde{r_2}.p,\ \tilde{r_2}.v/2,\ \tilde{r_2}.w/4 >
\end{array}
$$

More Examples

problem	original	optimized
non-restoring bin. int. sqrt.	$k^2, 2^i$	$+, -, {*}2, /2$
other integer/real division/sqrt	$k^2, 2^i$	$+, -, {*}2, /2$
local neighborhood prob.	$O(n^2m^2)$	$O(n^2)$
minimum sum section prob.	$O(n^3)$	$O(n)$
Fibonacci function	$O(2^n)$	$O(n)$
path sequence prob. in dag	$O(2^n)$	$O(n^2)$

Related Work

- – strength reduction: syntactic and structurally restricted, works on induction vars and region constants [ACK81,CK77]
  – composite hoisting-strength reduction [JD82]
  – optimal code motion adopted to strength-reduction [SKR90]

  cheaper but not as powerful

- – finite differencing: works on set-based expressions [PK82]
  – high-level iterators [E76]
  – inductively computable constructs [FU76,F77,F79]

  powerful but do not exploit conditions, functions, arrays

- – transformations based on general strategies
  – maintaining and strengthening loop invariants [D76,G81,G84,R81]

  general but do not propose systematic procedures

Conclusion

- incrementalization unifies methods of program optimization: strength reduction, code motion, data specialization, ...

- We've given a principled approach for program optimization based on incrementalization.

- Improvements:

  - combine different techniques for optimization
  - merge techniques for different language features
  - equality reasoning, dependence analysis, cost analysis
  - implementation: Cachet and beyond