Finite Differencing of Computable Expressions

Aj Shankar
It all comes down to laziness

- I’m lazy
- Say I’m making a PowerPoint presentation…
- I realize I need to cover a new topic
- What do I do?
  - A) Do the entire presentation over
  - B) ???
  - C) Profit!!!
Okay, what have I done?

- I used the work I had already done.
- And *incrementally* constructed a new presentation.

That’s pretty much all finite differencing is.

THE END
Okay, okay…

- Things we still need to figure out:
  - What are the benefits of differencing?
  - When is it safe?
    - (and what do we mean by safe?)
    - Can I compute Go positions incrementally?
  - How does it work?
  - Can it be automated?
  - Does the content of this paper really justify its 50 page length?
  - And more…
First, a more relevant example

- Goal: compute the successive sums of each m-element window in an n-element array (with m < n)

\[
\begin{array}{cccccccc}
2 & 1 & 4 & 2 & 7 & 0 & 3 & 5 & 2 \\
\end{array}
\]

n = 9; m = 4
The simple way

for (int i = 0 ; i < n-m ; i++) {
    for (int j = i ; j < m ; j++) {
        sum[i] += ary[j];
    }
}

- Sum up each window independently
Using finite differencing

- Compute a running sum

```c
for (int j = 0 ; j < m ; j++) {
    sum[0] += ary[j];
}
for (int i = 1 ; i < n-m ; i++) {
    sum[i] = sum[i-1] - ary[i-1] + ary[i+m-1];
}
```
Benefits of differencing

- Speedups
  - Possibly in asymptotic complexity
    - (What happened in the example?)
- If done automatically, simple code can become efficient
  - Can stick to “the simple way”
  - No need to uglify it yourself
Sounds unbelievably good

- This is going to revolutionize computing
- We took that $O(n^2)$ algo to $O(n)$…
- Let’s difference that $O(n)$ algorithm to get an $O(1)$ algorithm!
- The fun never ends
- (Sanity check…)
When is it safe to difference?

- Must guarantee the transformation is *semantics preserving*
  - Just like any other optimization
  - Stay tuned…
Finite differencing overview

- Derivative
  - The building block of differencing
- Chain rule
  - Stringing differential expressions together
- Tricks for initialization
Computable derivatives

- “Differencing”: figuring out the difference between $f(x)$ and $f(y)$
- **Derivative**: how $f$ changes with respect to $x$
- We extend this notion to code
Computable derivatives

- Let $E = f(x_1, \ldots, x_n)$
  - $E$ is the incremental replacement for $f$
- Let $dx_i$ be an update to $x_i$, e.g.

  $E = f(\text{list}) = \text{length(list)}$;

  ```
  while (*) {
    list = item :: list;  // $dx_0$
  }
  ```
Derivative example

- Here, \( E = f(\text{list}) = \text{length(\text{list})}; \)
- \( \text{dllist is list = item :: list; } \)
- Then \( dE(\text{dllist}) = E += 1 \)

- What should we expect of the derivative of \( E \)? \( dE(dx_0) \)
  - What properties must hold?
Derivative: formal definition

- Derivative is code blocks \([B1, B2]\):
  
  \[
  B1 \\
  \text{d}x_i \\
  B2
  \]

  \(\{\text{Differenced code}\}\)

- With properties:
  - \(B1\) and \(B2\) only modify locals and \(E\)
  - Semantics of \(\text{d}x_i\) are preserved
  - If \(E = f(x_1, \ldots, x_n)\) before, then \(E = f(x_1, \ldots, x_n)\) afterwards
Derivative questions

- Why is this definition semantics preserving?
- How broad is it?
  - Can it be applied to our sliding window example?
- Why do we need B1 and B2?
- Where do these derivs come from?
Differentiable code

- So we have derivatives of $f$... when can we apply them?
- $E = f()$ is differentiable w.r.t a code block $C$ if:
  - We know the derivatives of $f$ w.r.t. for each $x_i$ updated in $C$
  - $f$ is known, or at least computable, at the start of $C$
- This should intuitively make sense
Doing the differencing

To difference $f$ w.r.t. $C$:
- Replace all $d x_i$ in $C$ by $B_1_i; d x_i; B_2_i$;
- Replace all uses of $f$ with $E$
- Initialize $E$ properly

Does this preserve the semantics of $C$?

... Works for $C_1; C_2$ as well
Example

```plaintext
a := {};
while eof = false
    read(i);
    a with:= i;
end while;
print({x \in a | x \mod 2 == 0});
```
Example

```plaintext
a := {}; E := {}; while eof = false
    read(i);
    if (i mod 2 == 0) then
        E with:= i;
        a with:= i;
    end while;
print(E);
```

```
dE(a := {})
da
E = f(a)
dE(da)
```
Chaining

- So we’ve got the whole program differenced on f
- But what about g, h, etc?
- Just apply them in turn
Chaining, cont’d

- We have $E_1$ and $E_2$. Can chain if
  - $E_1 = f_1$ is differentiable w.r.t. $B$, transforming it to $B'$
  - And $E_2 = f_2$ is differentiable w.r.t. $B'$
- How does this preserve semantics?
- What if all $E_i$ were differentiable w.r.t. just $B$?
Computing speedups

- There’s a lot of stuff in the paper about figuring out when differencing will produce a speedup.
- But: it’s pretty obvious stuff:
  - Initial costs should be relatively low.
  - Derivatives should be faster than recomputing f.
- ...
Initialization tricks

- We have a bunch of differenced code
- But need to initialize $E_i = f_i(\ldots)$ first
  - “setting up the invariant”
- Doing each $E_i$ separately wasteful
- Jamming…
Vertical Jamming

- Want to initialize $c_1$, $c_2$

```latex
\begin{verbatim}
for (x in s) {
  if (k_1(x))
    c_1 with: x;
}

for (x in c_1) {
  if (k_2(x))
    c_2 with: x;
}
\end{verbatim}
```
Vertical Jamming, cont’d

Remember, we’re lazy.
Vertical Jamming, cont’d

\[
\begin{array}{ccccccc}
\text{s} & 2 & 1 & 4 & 2 & 7 & 0 & 3 & 5 & 2 \\
\text{c}_1 & 1 & 4 & 7 & 3 & 5 \\
\text{c}_2 & 1 & 7 & 3 \\
\end{array}
\]

- Wow, that saved a lot of effort.
- Except for the effort it took me to make these slides.
The resulting code

for (x in s) {
    if (k₁(x)) {
        c₁ with: x;
        if (k₂(x))
            c₂ with: x;
    }
}
Horizontal jamming

- I’ll spare you the animations

```plaintext
for (x in s) {
    if (k_1(x))
        c_1 with: x;
}

for (x in s) {
    if (k_2(x))
        c_2 with: x;
}
```
Horizontal jamming, cont’d

- Becomes

```plaintext
for (x in s) {
    if (k_1(x))
        c_1 with: x;
    if (k_2(x_i))
        c_2 with: x;
}
```
Automating the process

- What are the hurdles?
  - Picking an f
  - Coming up with a derivative
    - How comprehensive is the list in the paper?
  - What else?
- “Already implemented a semiautomatic system”
  - “Results reported in near future”
Practical concerns

- Are there any language hurdles?
- What other problems are in the way?
  - (Why isn’t this system used now?)