

Lifted inference: normalizing loops by evaluation

Oleg Kiselyov
Chung-chieh Shan

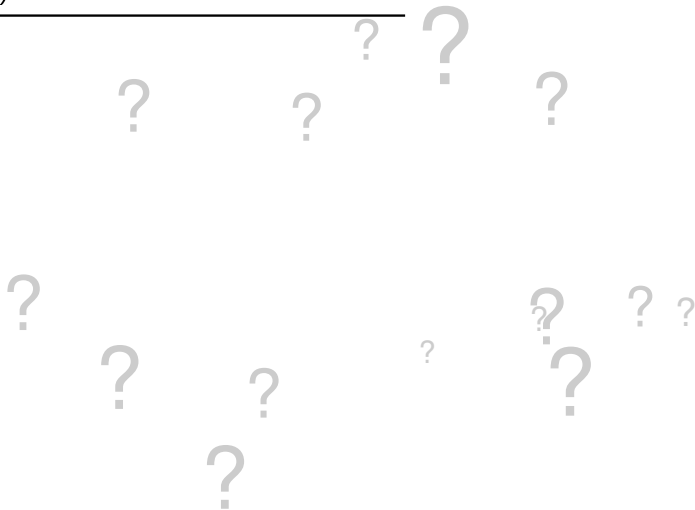
10 April 2012

As the two friends wandered through the snow on their way home, Piglet grinned to himself, thinking how lucky he was to have a best friend like Pooh.



Pooh thought to himself:
"If the pig sneezes,
he's dead."

I	Disease is infectious?
$S(a)$	Alice is sick?
$S(b)$	Bob is sick?
$S(c)$	Carol is sick?



I	Disease is infectious?
$S(a)$	Alice is sick?
$S(b)$	Bob is sick?
$S(c)$	Carol is sick?

$$\frac{\Pr(S(b))}{\Pr(\neg S(b))} ?$$

Probabilities represent uncertainty.

I	Disease is infectious?
$S(a)$	Alice is sick?
$S(b)$	Bob is sick?
$S(c)$	Carol is sick?

$$\frac{\Pr(S(b) \wedge \neg S(a))}{\Pr(\neg S(b) \wedge \neg S(a))}$$

Conditional probabilities represent prior knowledge.

I	f	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t

Possible worlds as in modal logic.

I	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

Possible worlds as in modal logic, but weighted numerically.

I	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

$$\left(\prod_{\neg I \wedge \neg S(x)} 9 \right)$$

Possible worlds as in modal logic, but weighted numerically.

I	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

$$\left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right)$$

Possible worlds as in modal logic, but weighted numerically.

I	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

$$\overbrace{\left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right)}{\text{Weight}}$$

```
let I = dist [(1, true);
              (1, false)]
in let S(x) = dist [(1, true);
                   (if I then 4 else 9, false)]
   in ...
```

Want to express generative model: random choice as side effect.

I	f	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	f	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

$$\overbrace{\left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right)}^{\text{Weight}}$$

```

create table Weights ( $W$ ,  $Weight$ ) as
select  $W$ , (select product(case when  $I$  then 4 else 9 end)
          from Infectious join Sick using ( $W$ )
          where  $W$  = Worlds. $W$  and not  $S$ )
from Worlds

```

Express a wide variety of queries. Typically:

I	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

$$\text{Tabulate } \sum_{S(b) \in \{f, t\}} \sum_{\substack{I \in \{f, t\} \\ S(z) \in \{f, t\} \\ \text{for each } z \neq b}} \overbrace{\left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right)}^{\text{Weight}}$$

```
select S, sum(Weight)
from Weights
      join Sick as B using (W)
where B.Person = 'b'
group by B.S
```

Group worlds and sum weights.

<i>I</i>	f	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t
<i>S(a)</i>	f	f	f	f	t	t	t	t	f	f	f	f	f	t	t	t
<i>S(b)</i>	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
<i>S(c)</i>	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

$$\text{Tabulate } \sum_{S(b) \in \{f,t\}} \sum_{\substack{I \in \{f,t\} \\ S(z) \in \{f,t\} \\ \text{for each } z \neq b}} \overbrace{\left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right)}^{\text{Weight}} \times \begin{cases} \text{if } S(a) \\ \text{then } 0 \\ \text{else } 1 \end{cases}$$

```
select S, sum(Weight)
```

```
from Weights
```

```
    join Sick as B using (W) join Sick as A using (W)
where B.Person = 'b' and A.Person = 'a' and not A.S
group by B.S
```

Group worlds and sum weights, filtered by prior knowledge.

I	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f	t
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4	1

$$\begin{aligned}
 & \text{Tabulate } \sum_{S(b) \in \{f,t\}} \sum_{I \in \{f,t\}} \sum_{\substack{S(z) \in \{f,t\} \\ \text{for each } z \neq b}} \overbrace{\left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right)}^{\text{Weight}} \times \begin{cases} \text{(if } S(a) \\ \text{then 0} \\ \text{else 1)} \end{cases} \\
 \text{Query plan } \left\{ \begin{aligned}
 & = \text{Tabulate } \sum_{S(b) \in \{f,t\}} \sum_{I \in \{f,t\}} \prod_x \text{let } f(S) = \begin{cases} \text{(if } \neg I \wedge \neg S \text{ then 9 else 1)} \\ \cdot \text{(if } I \wedge \neg S \text{ then 4 else 1)} \\ \cdot \text{(if } x = a \wedge S \text{ then 0 else 1)} \end{cases} \\
 & \quad \text{in if } x = b \text{ then } f(S(b)) \text{ else } f(t) + f(f)
 \end{aligned}
 \right.
 \end{aligned}$$

Distributivity turns sum of 2^{n-1} products into product of n sums.

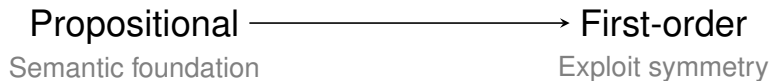
I	f	f	f	f	f	f	f	f	t	t	t	t	t	t	t
$S(a)$	f	f	f	f	t	t	t	t	f	f	f	f	t	t	t
$S(b)$	f	f	t	t	f	f	t	t	f	f	t	t	f	f	t
$S(c)$	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f
	729	81	81	9	81	9	9	1	64	16	16	4	16	4	4

Weight

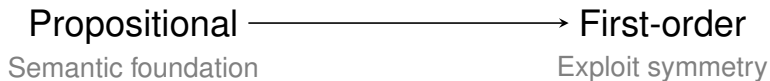
$$\text{Tabulate } \sum_{S(b) \in \{f, t\}} \sum_{I \in \{f, t\}} \sum_{\substack{S(z) \in \{f, t\} \\ \text{for each } z \neq b}} \overbrace{\left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right)}^{\text{Weight}} \times \begin{cases} \text{if } S(a) \\ \text{then } 0 \\ \text{else } 1 \end{cases}$$

$$\text{Query plan } \left\{ \begin{aligned} &= \text{Tabulate } \sum_{S(b) \in \{f, t\}} \sum_{I \in \{f, t\}} \prod_x \text{let } f(S) = \begin{aligned} &(\text{if } \neg I \wedge \neg S \text{ then } 9 \text{ else } 1) \\ &\cdot (\text{if } I \wedge \neg S \text{ then } 4 \text{ else } 1) \\ &\cdot (\text{if } x = a \wedge S \text{ then } 0 \text{ else } 1) \end{aligned} \\ &\quad \underbrace{\text{in if } x = b \text{ then } f(S(b)) \text{ else } f(t) + f(f)}_{\text{Boring loop}} \end{aligned} \right.$$

Lift probabilities *generally*?



Lift probabilities *generally*?



Reasoning system?

Inference algorithm?

Query plan? Yes!

Outline

► **MapReduce loops in sublinear time**

Normalizing loop bodies by evaluation

Nested loops

Loops over tuples and subsets

MapReduce loops in sublinear time

```
["alice"; "bob"; "carol"; ...; "Äijö"]
```

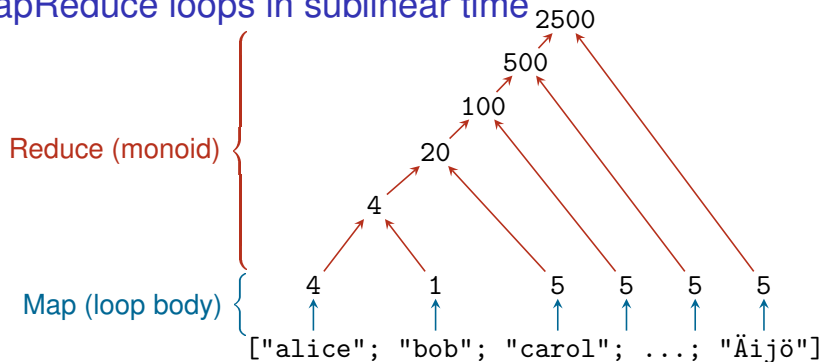
MapReduce loops in sublinear time

Map (loop body) {

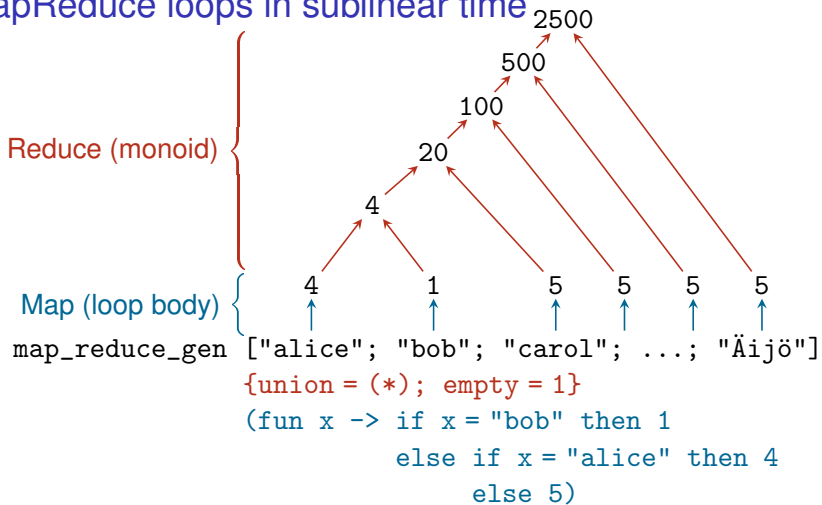
	4	1	5	5	5	5
	↑	↑	↑	↑	↑	↑
	["alice";	"bob";	"carol";	...	"Äijö"]	

}

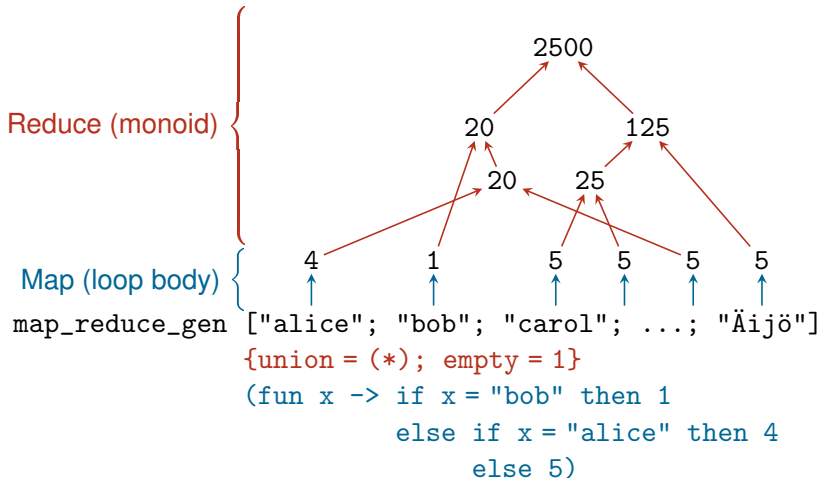
MapReduce loops in sublinear time



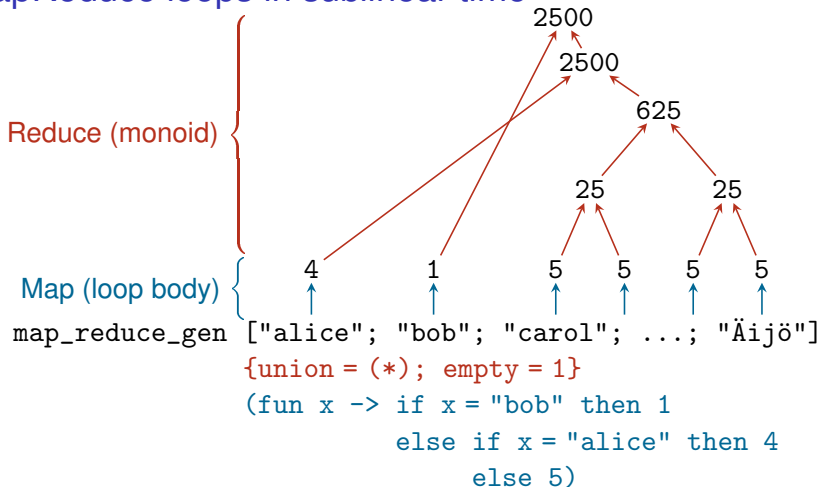
MapReduce loops in sublinear time



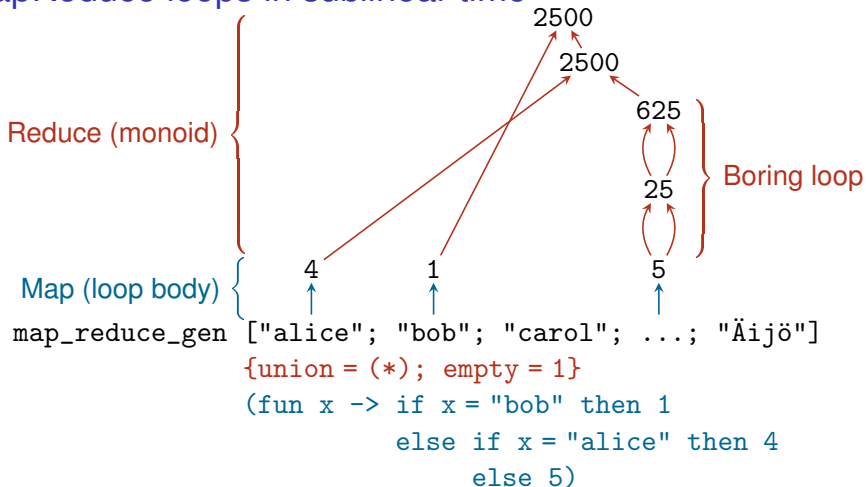
MapReduce loops in sublinear time



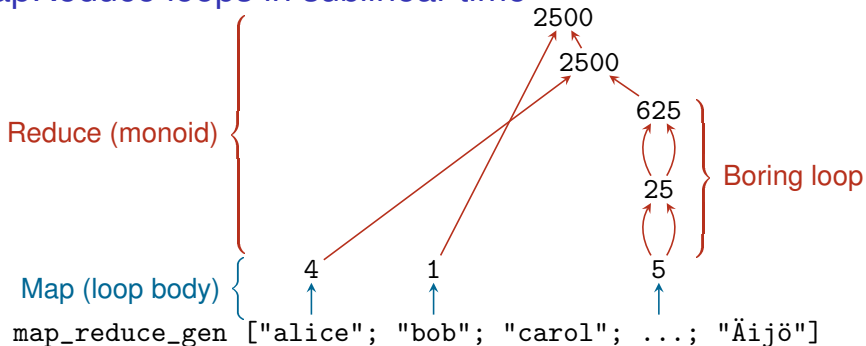
MapReduce loops in sublinear time



MapReduce loops in sublinear time



MapReduce loops in sublinear time



```
{union = (*); empty = 1}
```

```
(fun x -> if x = "bob" then 1
```

```
    else if x = "alice" then 4
```

```
    else 5)
```

```
= map_reduce_sw {union = (*); empty = 1}
```

```
(Case ("bob", 1,
```

```
    Case ("alice", 4,
```

```
        Default 5)))
```

MapReduce loops in sublinear time

\prod_x let $f(S) =$ (if $\neg I \wedge \neg S$ then 9 else 1)
· (if $I \wedge \neg S$ then 4 else 1)
· (if $x = a \wedge S$ then 0 else 1)
in if $x = b$ then $f(S(b))$ else $f(t) + f(f)$

```
(fun x -> if x = "bob" then 1
          else if x = "alice" then 4
          else 5)
```

```
(Case ("bob", 1,
      Case ("alice", 4,
            Default 5)))
```

MapReduce loops in sublinear time

$$\prod_x \text{let } f(S) = (\text{if } \neg I \wedge \neg S \text{ then } 9 \text{ else } 1)$$
$$\quad \cdot (\text{if } I \wedge \neg S \text{ then } 4 \text{ else } 1)$$
$$\quad \cdot (\text{if } x = a \wedge S \text{ then } 0 \text{ else } 1)$$
$$\text{in if } x = b \text{ then } f(S(b)) \text{ else } f(t) + f(f)$$

Reflect

```
(fun x -> if x = "bob" then 1
          else if x = "alice" then 4
          else 5)
```

Reify

```
(Case ("bob", 1,
      Case ("alice", 4,
            Default 5)))
```

Outline

MapReduce loops in sublinear time

► **Normalizing loop bodies by evaluation**

Nested loops

Loops over tuples and subsets

Normalizing loop bodies by evaluation

Object language = OCaml + an abstract individual with equality

```
type var = Val of string | Var      (* abstract *)  
val equ : var -> var -> bool
```

Normalizing loop bodies by evaluation

Object language = OCaml + an abstract individual with equality

```
type var = Val of string | Var          (* abstract *)
val equ : var -> var -> bool

fun x:var ->
  let f s = (if not I && not s then 9 else 1)
            * (if I && not s then 4 else 1)
            * (if equ x (Val "alice") && s then 0 else 1)
  in if equ x (Val "bob")
     then f(S(b))
     else f true + f false
```

Suppose $I = S(b) = \text{true}$ for example.

Normalizing loop bodies by evaluation

Object language = OCaml + an abstract individual with equality

```
type var = Val of string | Var      (* abstract *)
val equ : var -> var -> bool

fun x:var ->
  let f s =
    (if      not s then 4 else 1)
    * (if equ x (Val "alice") && s then 0 else 1)
  in if equ x (Val "bob")
     then f true
     else f true + f false
```

Normalizing loop bodies by evaluation

Object language = OCaml + an abstract individual with equality

```
type var = Val of string | Var      (* abstract *)
val equ : var -> var -> bool

reify (fun x ->
  let f s =
    (if      not s then 4 else 1)
    * (if equ x (Val "alice") && s then 0 else 1)
  in if equ x (Val "bob")
    then f true
    else f true + f false)
```

Apply map to Var under 'debugger' (delimited control).

Set 'breakpoint' at equ:

```
type 'r req = Done of 'r
           | Compare of var * var * (bool -> 'r req)
```



```
loop [] let f s = (if not s then 4 else 1)
           * (if equ Var (Val "alice") && s
              then 0 else 1)
in if equ Var (Val "bob") then f true
   else f true + f false
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b -> let f s = ...
                              in if b then f true
                                  else f true + f false ))
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                          in if b then f true
                              else f true + f false ))
```

```
Case ("bob", loop_known "bob" let f s = ...
                                in f true ,
```

```
loop ["bob"] let f s = ...
              in f true + f false )
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                           in if b then f true
                               else f true + f false))
```

```
Case ("bob", loop_known "bob" let f s = ...
                                in f true ,
```

```
loop ["bob"] let f s = ...
              in f true + f false)
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                          in if b then f true
                              else f true + f false))
```

```
Case ("bob", 1,
```

```
loop ["bob"] let f s = ...
              in f true + f false)
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                          in if b then f true
                              else f true + f false))
```

```
Case ("bob", 1,
```

```
loop ["bob"] let f s = ...
              in f true + f false)
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                          in if b then f true
                              else f true + f false))
```

```
Case ("bob", 1,
```

```
Case ("alice", loop_known "alice" let f s = ...
                                     in 0 + f false,
```

```
loop ["alice"; "bob"] let f s = ...
                       in 1 + f false))
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                           in if b then f true
                               else f true + f false))
```

```
Case ("bob", 1,
```

```
Case ("alice", loop_known "alice" let f s = ...
                                       in 0 + f false,
```

```
loop ["alice"; "bob"] let f s = ...
                       in 1 + f false))
```



```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                           in if b then f true
                               else f true + f false))
```

```
Case ("bob", 1,
```

```
Case ("alice", 4,
```

```
loop ["alice"; "bob"] let f s = ...
                       in 1 + f false))
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                           in if b then f true
                               else f true + f false))
```

```
Case ("bob", 1,
```

```
Case ("alice", 4,
```

```
loop ["alice"; "bob"] let f s = ...
                       in 1 + f false))
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                          in if b then f true
                              else f true + f false))
```

```
Case ("bob", 1,
```

```
Case ("alice", 4,
```

```
Default 5))
```

```
loop [] let f s = (if not s then 4 else 1)
          * (if equ Var (Val "alice") && s
              then 0 else 1)
          in if equ Var (Val "bob") then f true
              else f true + f false
```

```
loop [] (Compare (Var, Val "bob",
                  fun b-> let f s = ...
                          in if b then f true
                              else f true + f false))
```

```
Case ("bob", 1,
```

```
Case ("alice", 4,
```

```
Default 5))
```

Putting it together

Compute $1 \times 4 \times 5^{n-2}$ quickly (in polylog time):

```
map_reduce_sw
```

```
{union = (*); empty = 1}
```

```
(reify (fun x ->
```

```
  let f s = (if not s then 4 else 1)
```

```
    * (if equ x (Val "alice") && s then 0 else 1)
```

```
  in if equ x (Val "bob")
```

```
    then f true
```

```
    else f true + f false))
```

Outline

MapReduce loops in sublinear time

Normalizing loop bodies by evaluation

► **Nested loops**

Loops over tuples and subsets

Nested loops

```
let sum_monoid = {union = (+); empty = 0}
```

```
map_reduce_sw sum_monoid (reify (fun x ->
  map_reduce_sw sum_monoid (reify (fun y ->
    if equ x (Val "alice") || equ y (Val "bob")
    then 0 else 1))))
```

Want $(n - 1)^2$.

Nested loops

```
let sum_monoid = {union = (+); empty = 0}
```

```
map_reduce_sw sum_monoid (reify (fun x ->
  map_reduce_sw sum_monoid (reify (fun y ->
    if equ x (Val "alice") || equ y (Val "bob")
    then 0 else 1))))
```

Want $(n - 1)^2$. Get $n(n - 2)$, because x is confused with y .

Nested loops

```
let sum_monoid = {union = (+); empty = 0}

map_reduce_sw sum_monoid (reify (fun x ->
  map_reduce_sw sum_monoid (reify (fun y ->
    if equ x (Val "alice") || equ y (Val "bob")
    then 0 else 1))))
```

Each level should add a new abstract individual.

```
type var = Val of string | Var
type 'r switch = Default of 'r
                | Case of string * 'r * 'r switch
```

Nested loops

```
let sum_monoid = {union = (+); empty = 0}
top (fun () ->
map_reduce_sw sum_monoid (reify (fun x ->
  map_reduce_sw sum_monoid (reify (fun y ->
    if equ x (Val "alice") || equ y (Val "bob")
    then 0 else 1))))))
```

Each level should add a new abstract individual.

```
type var = Val of string | Var of unit ref
type 'r switch = Default of 'r
               | Case of var * 'r * 'r switch
```

Now reify handles equ by invoking equ metacircularly!
Each level tracks its own variable's equality or disequalities.

Polymorphism and optimization

This nesting is

- ▶ **inexpressive**: All loop variables have the same type `var`.
- ▶ **inefficient**: Each level delegates `equ` to the next outer level.

Fix: use **multi-prompt** delimited control

to associate each loop variable with its `reify` in one fell swoop.

Ongoing work with Yuki Yoshi Kameyama to extract this pattern of *call-by-need delimited control*.

Outline

MapReduce loops in sublinear time

Normalizing loop bodies by evaluation

Nested loops

▶ **Loops over tuples and subsets**

Loops over tuples and subsets

Factoring was easy:

$$\sum_{\substack{S(z) \in \{f, t\} \\ \text{for each } z \neq b}} \left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right) \times (\text{if } S(a) \text{ then } 0 \text{ else } 1)$$

but what if individuals interact?

$$\sum_{\substack{S(z) \in \{f, t\} \\ \text{for each } z \neq b}} \left(\prod_{I \wedge S(x) \wedge S(y)} 2 \right) \times \dots$$

Need counting arguments.

Loops over tuples and subsets

Factoring was easy:

$$\sum_{\substack{S(z) \in \{f, t\} \\ \text{for each } z \neq b}} \left(\prod_{\neg I \wedge \neg S(x)} 9 \right) \times \left(\prod_{I \wedge \neg S(x)} 4 \right) \times (\text{if } S(a) \text{ then } 0 \text{ else } 1)$$

but what if individuals interact?

$$\sum_{\substack{S(z) \in \{f, t\} \\ \text{for each } z \neq b}} \left(\prod_{I \wedge S(x) \wedge S(y)} 2 \right) \times \dots$$

Step 1: from tuples to individuals

Step 2: from individuals to subsets

Step 1: from loops over tuples to loops over individuals

$$\prod_{x,y \in S} f(x,y)$$

Step 1: from loops over tuples to loops over individuals

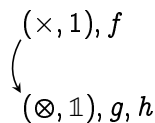
Reify f into a switch of switches.

$$\prod_{x,y \in S} f(x,y) \\ = h\left(\bigotimes_{x \in S} g(x)\right)$$

$$\begin{array}{l} (\times, 1), f \\ \curvearrowright \\ (\otimes, \mathbb{1}), g, h \end{array}$$

Step 1: from loops over tuples to loops over individuals

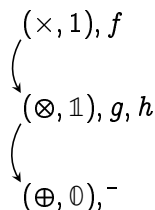
Reify f into a switch of switches.

$$\begin{aligned} & \sum_{S \subseteq D} \prod_{x, y \in S} f(x, y) \\ = & \sum_{S \subseteq D} h\left(\bigotimes_{x \in S} g(x)\right) \end{aligned}$$


$(\times, 1), f$
 $(\otimes, \mathbb{1}), g, h$

Step 2: from loops over individuals to loops over subsets

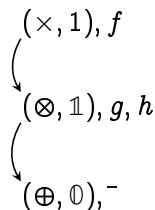
Reify f into a switch of switches.

$$\begin{aligned} & \sum_{S \subseteq D} \prod_{x, y \in S} f(x, y) \\ &= \sum_{S \subseteq D} h\left(\bigotimes_{x \in S} g(x)\right) \\ &= \sum_{S \subseteq D} \bigoplus_{x \in S} \overline{h\left(\bigotimes_{x \in S} g(x)\right)} \\ &= \sum_{S \subseteq D} \bar{h}\left(\bigoplus_{x \in S} \overline{\bigotimes_{x \in S} g(x)}\right) \\ &= \sum_{S \subseteq D} \bar{h}\left(\bar{\bigotimes}_{x \in D} (\bar{\mathbb{1}} \oplus \overline{g(x)})\right) \end{aligned}$$


Factor sum into bags.

Step 2: from loops over individuals to loops over subsets

Reify f into a switch of switches.

$$\begin{aligned} & \sum_{S \subseteq D} \prod_{x, y \in S} f(x, y) \\ = & \sum_{S \subseteq D} h\left(\bigotimes_{x \in S} g(x)\right) \\ = & \sum_{S \subseteq D} \bigoplus_{x \in S} \overline{h\left(\bigotimes_{x \in S} g(x)\right)} \\ = & \sum_{S \subseteq D} \bar{h}\left(\bigoplus_{x \in S} \overline{\bigotimes_{x \in S} g(x)}\right) \\ = & \sum_{S \subseteq D} \bar{h}\left(\bar{\bigotimes}_{x \in D} (\bar{\mathbb{1}} \oplus \overline{g(x)})\right) \end{aligned}$$


Factor sum into bags. Result: a loop over individuals, which computes binomial coefficients by convolution.

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

(256, 2, 1, 1, 4, 1, 16)

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

(256 , 2 , 1 , 1 , 4 , 1 , 16)

(16 , 2 , 0 , 0 , 4 , 1 , 1) (16 , 0 , 1 , 1 , 1 , 1 , 16)

2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
2	2	2	2	1	1	1
1	1	1	1	1	1	1
1	1	1	1	16	1	1
1	1	1	1	1	1	1

(65536 , 4, 0, 0, 16, 1, 1)

(16 , 2, 0, 0, 4, 1, 1) (16 , 2, 0, 0, 4, 1, 1)

Conclusion

Run MapReduce loops in sublinear time,
reifying loop bodies by evaluating them under delimited control.

Further connections:

- ▶ **constraint solving**
Equality and disequalities (chromatic polynomial).
Other constraints: inequalities, function symbols, subsets?
- ▶ **metacircular interpretation**
Multi-prompt delimited control for expressivity and efficiency.
Enforce no-throw safety?
- ▶ **loop combinators**
Tuples and subsets.
Convert from streaming algorithms efficiently?
- ▶ **artificial intelligence**
Metareasoning without interpretive overhead!
Predict resource usage, to optimize and represent queries.