EXISTENTIAL LABEL FLOW INFERENCE VIA CFL REACHABILITY

Polyvios Pratikakis
Jeffrey S. Foster
Michael Hicks

University of Maryland, College Park

Flow Analysis — Applications

- Points-to Analysis
- Information Flow
- Type Qualifier Inference
- Code Optimizations
- “Guarded-by” analysis (race detection)
  - Used in LOCKSMITH race detection tool, PLDI 2006

Precise Flow Analysis

- Analyze function calls context sensitively
  - As if every function was inlined at every call site
- Problem with data structures
  - Most analyses conflate all elements of a data structure
  - Usually, important flow relations occur among members of each individual element
  - Such flow is sound, even when the element cannot be precisely identified

Analyzing Data Structures

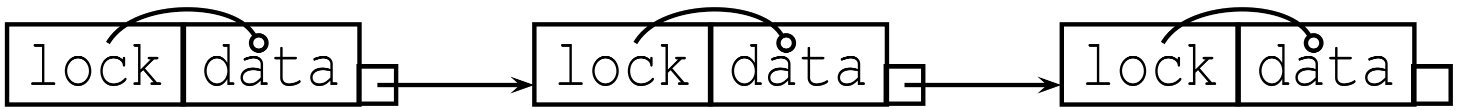
- Motivation: inference of "guarded by" relation between elements of a struct:

```
struct list {
  lock_t lock;
  int* data;
  struct list *next;
}
```

- Within *each* element, `lock` protects `*data`

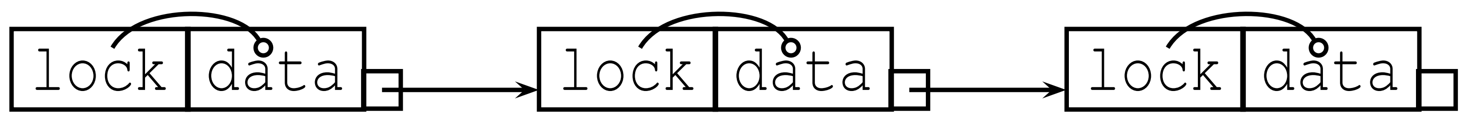
The Problem

- Actual data structure

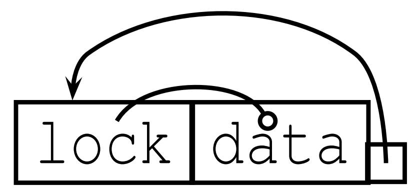


The Problem

- Actual data structure

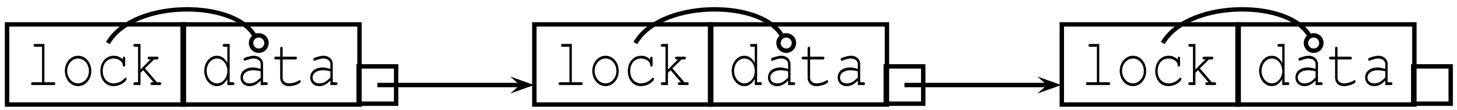


- Summarized by the analysis

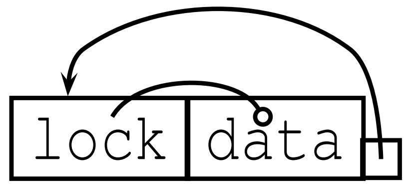


The Problem

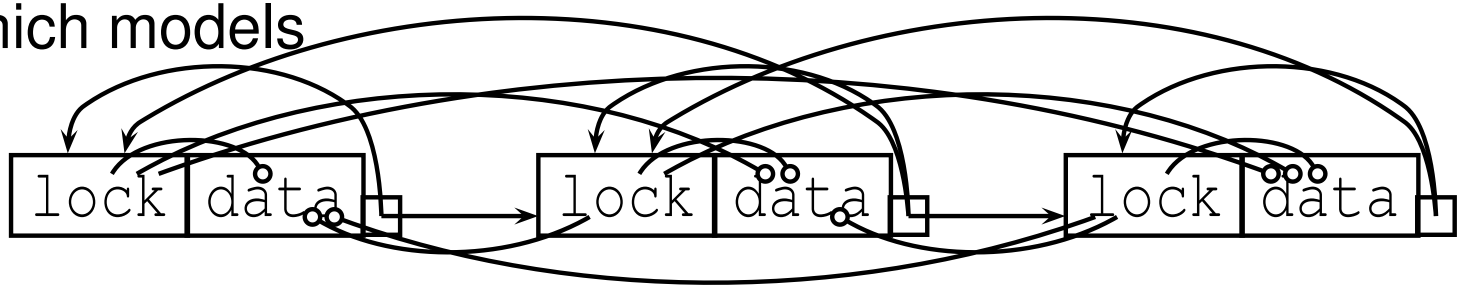
- Actual data structure



- Summarized by the analysis

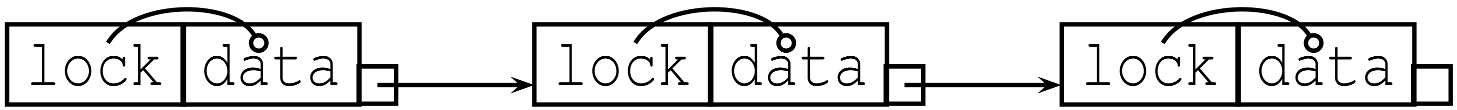


- Which models

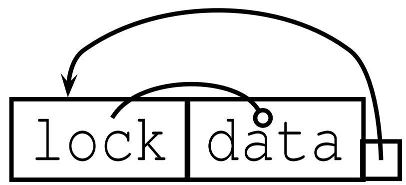


The Problem

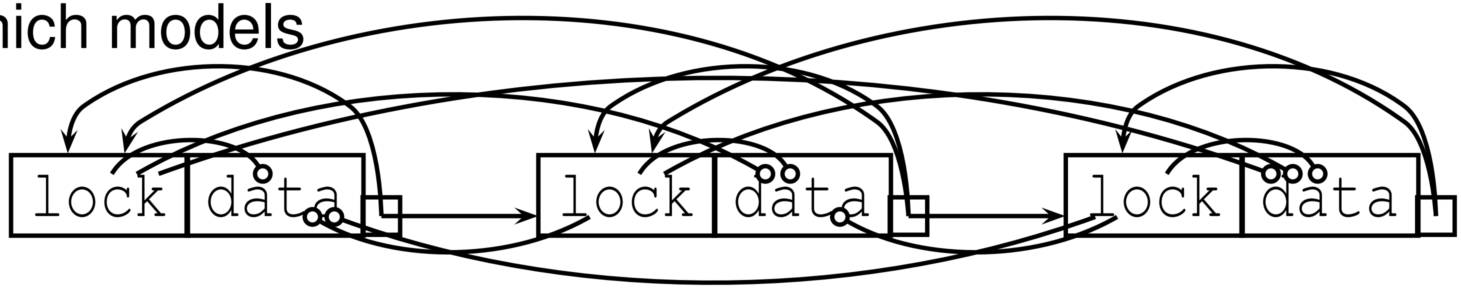
- Actual data structure



- Summarized by the analysis



- Which models



...a “blob”

Contributions

- Label flow analysis with support for flow within data structure elements
  - Formalized as type-based label flow analysis
  - Flow within data structure elements is existentially quantified
- Proof of soundness
  - Type-based formulation allows us to use type-system proof techniques
  - Type-soundness implies sound analysis
- Encoded as a CFL reachability problem
  - Solvable in $O(n^3)$

Previous Work

- Context insensitive type-based label flow analysis
- Add context sensitivity
  - Abstract over the context where a function is defined
  - Instantiate to the calling context when it is called
  - Encoded as (bounded) universal polymorphism [Mossin]: $\forall\vec{\ell}[C].\tau$
  - Can be implemented without copying using CFL reachability [Fähndrich et al]

Main idea

- Use existential polymorphism to model data structures
- Universal and existential polymorphism are dual
  - $\forall$: Abstract context at the definition and instantiate (inline) on every use
  - $\exists$: Abstract the context of every use and instantiate at the definition
- Idea: use $\forall/\exists$ duality to encode existential polymorphism
  - Allows reuse of the same techniques used for normal context sensitivity
- Unfortunately, it's not trivial — complications:
  - Existential types are first-class
  - Possible ambiguity when we can quantify both existentially and universally

Type-Based Flow Analysis

"Does the value of expression e_1 flow to expression e_2?"

- Annotate all types with labels ℓ (e.g. $int^{\langle\ell\rangle}$)
- Typecheck the program, creating *flow constraints*: "ℓ_1 flows to ℓ_2" ($\ell_1 \leq \ell_2$) forming a *flow graph*
- Answer flow question:
  - Type expressions e_1 and e_2 with annotated types $e_1 : \tau_1^{\langle\ell_1\rangle}$, $e_2 : \tau_2^{\langle\ell_2\rangle}$
  - Check for flow from ℓ_1 to ℓ_2 in the graph

Context Insensitive Analysis

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}$

y

ℓ_a	ℓ'_a
L_1	ℓ_x
L_2	ℓ_y

- `id` has type $int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$ where ℓ_a flows to ℓ'_a
- `x`, `y` have types $int^{\langle \ell_x \rangle}, int^{\langle \ell_y \rangle}$
- `1`, `2` have types $int^{\langle L_1 \rangle}, int^{\langle L_2 \rangle}$

Context Insensitive Analysis

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}$

y

$$\begin{tikzcd} \ell_a \arrow[r] & \ell'_a \\ L_1 & \ell_x \\ L_2 & \ell_y \end{tikzcd}$$

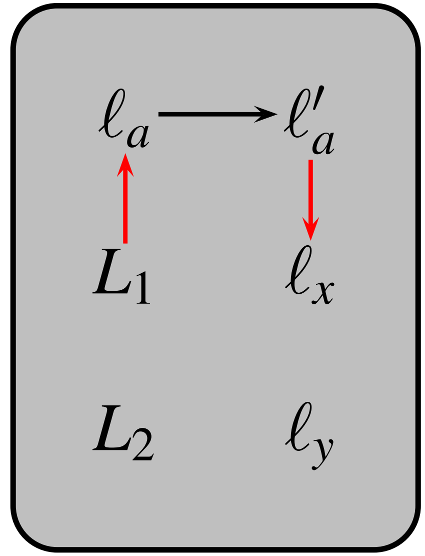
Context Insensitive Analysis

let $id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}$ in

let $x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}$ in

let $y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}$ in

y



Context Insensitive Analysis

$$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$$

$$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$$

$$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}$$

$$y$$

$$\begin{tikzcd}
\ell_a \arrow[r] & \ell'_a \arrow[d] \arrow[dd, bend left] \\
L_1 \arrow[u] & \ell_x \\
L_2 \arrow[uu, bend left] & \ell_y
\end{tikzcd}$$

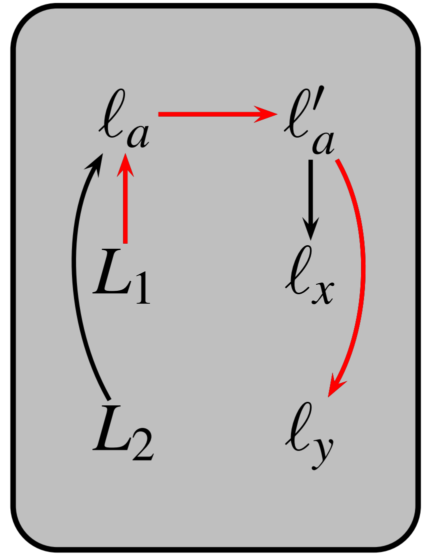
Context Insensitive Analysis

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}$

y



Imprecise! 1 flows to y!

Let's Add Context Sensitivity

Analyze a function f:

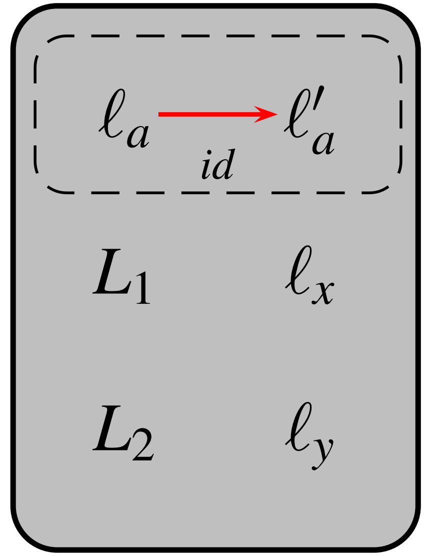
- Generate the flow graph C_f from the function body
- Assign the type $\forall \ell_1, \dots, \ell_n[C_f].\tau \longrightarrow \tau'$ to the function
- At every call site, *instantiate* C_f:
  - Insert a fresh copy of C_f
- Amounts to inlining the function body on all calls

Example — Context-Copying

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}\ y$



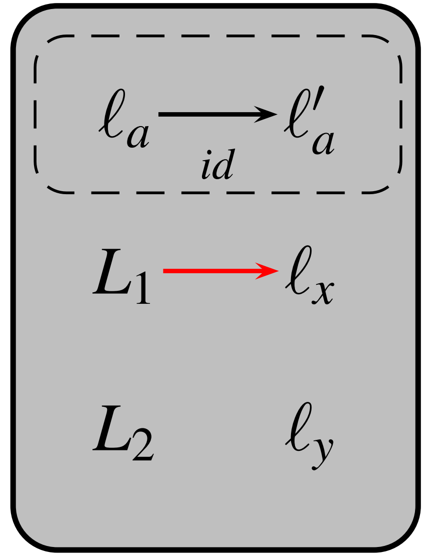
- `id` has type $int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$ or $\forall \ell_a, \ell'_a[\ell_a \leq \ell'_a].int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$

Example — Context-Copying

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}\ y$



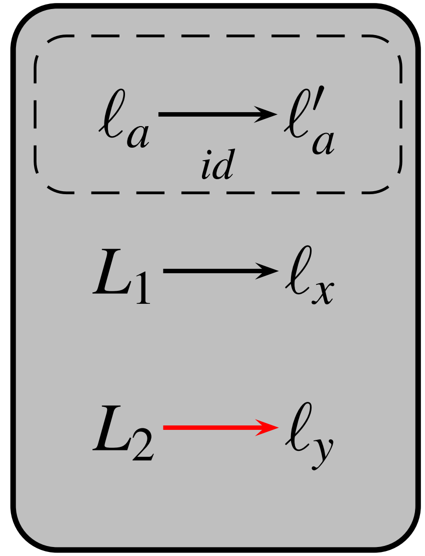
- `id` has type $int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$ or $\forall \ell_a, \ell'_a [\ell_a \leq \ell'_a].int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$
- In the first call, we *instantiate* `id` to $int^{\langle L_1 \rangle} \longrightarrow int^{\langle \ell_x \rangle}$

Example — Context-Copying

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}\ y$



- `id` has type $int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$ or $\forall \ell_a, \ell'_a[\ell_a \leq \ell'_a].int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$
- In the first call, we *instantiate* `id` to $int^{\langle L_1 \rangle} \longrightarrow int^{\langle \ell_x \rangle}$
- In the second call, we *instantiate* `id` to $int^{\langle L_2 \rangle} \longrightarrow int^{\langle \ell_y \rangle}$

Example — Context-Copying

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id\ 2^{\langle L_2 \rangle}\ \texttt{in}\ y$

$$\ell_a \xrightarrow[id]{} \ell'_a$$

$$L_1 \longrightarrow \ell_x$$

$$L_2 \longrightarrow \ell_y$$

- `id` has type $int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$ or $\forall \ell_a, \ell'_a[\ell_a \leq \ell'_a].\, int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$
- In the first call, we *instantiate* `id` to $int^{\langle L_1 \rangle} \longrightarrow int^{\langle \ell_x \rangle}$
- In the second call, we *instantiate* `id` to $int^{\langle L_2 \rangle} \longrightarrow int^{\langle \ell_y \rangle}$
- The *function body* subgraph ($\longrightarrow$) is copied on every call

Can We Avoid Copying Subgraphs?

- More efficient encoding:
  - Reuse the function body subgraph, without copying it
  - Still differentiate between call sites
- Use a unique name i per call site
- Link the function body subgraph to the call site context as in the context insensitive case
- Name all the “link” edges with the name of the call site
- Only consider flow along paths that correspond to valid call-return pairs

Encoding as CFL Reachability

- Reps et al first proposed using CFL reachability for program analysis
- Fähndrich et al encoded polymorphic label flow as parenthesis-matching
  - When flow enters a function's subgraph at call site i, label edges with $(_i$
  - When flow exits a function's subgraph at call site i, label edges with $)_i$
  - Valid flow only on paths without mismatched parentheses
  - Parenthesis matching (CFL-reachability) is solvable in $O(n^3)$
  - Proof by reduction to Context-Copying system

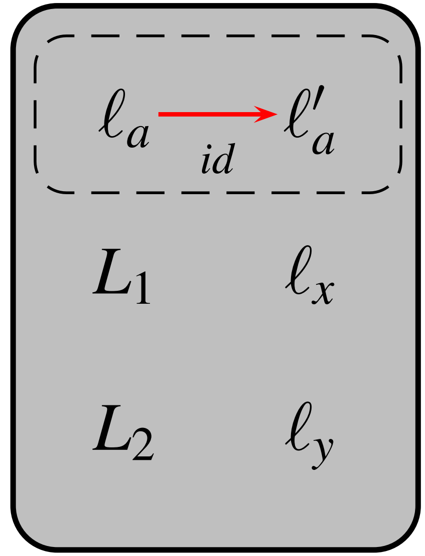
Example — CFL

$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$

$\texttt{let}\ x^{\langle \ell_x \rangle} = id^{j}\ 1^{\langle L_1 \rangle}\ \texttt{in}$

$\texttt{let}\ y^{\langle \ell_y \rangle} = id^{k}\ 2^{\langle L_2 \rangle}\ \texttt{in}$

y



- `id` has type $\forall \ell_a, \ell'_a . int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$

Example — CFL

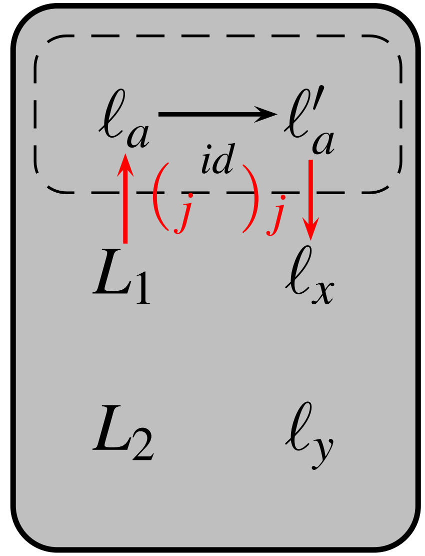
$$\texttt{let}\ id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}\ \texttt{in}$$

$$)_j \qquad (_j$$

$$\texttt{let}\ x^{\langle \ell_x \rangle} = id^{j}\ 1^{\langle L_1 \rangle}\ \texttt{in}$$

$$\texttt{let}\ y^{\langle \ell_y \rangle} = id^{k}\ 2^{\langle L_2 \rangle}\ \texttt{in}$$

$$y$$



- `id` has type $\forall \ell_a, \ell'_a.\, int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$
- In context i, we *instantiate* `id` to $int^{\langle \ell_1 \rangle} \longrightarrow int^{\langle \ell_x \rangle}$

Example — CFL

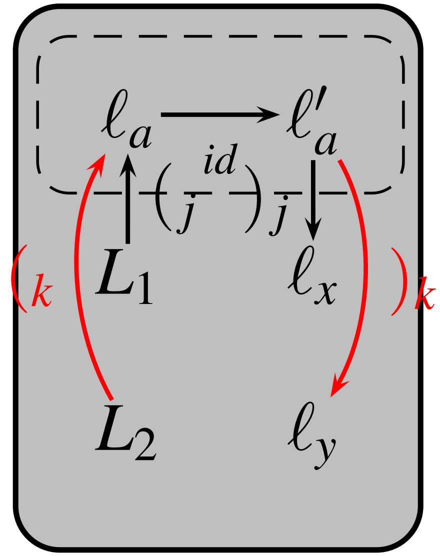
let $id = \lambda a^{\langle \ell_a \rangle} . a^{\langle \ell'_a \rangle}$ in

let $x^{\langle \ell_x \rangle} = id^j \; 1^{\langle L_1 \rangle}$ in

$)_k$ $(_k$

let $y^{\langle \ell_y \rangle} = id^k \; 2^{\langle L_2 \rangle}$ in

y



- id has type $\forall \ell_a, \ell'_a . int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$
- In context i, we *instantiate* id to $int^{\langle \ell_1 \rangle} \longrightarrow int^{\langle \ell_x \rangle}$
- In context j, we *instantiate* id to $int^{\langle \ell_2 \rangle} \longrightarrow int^{\langle \ell_y \rangle}$

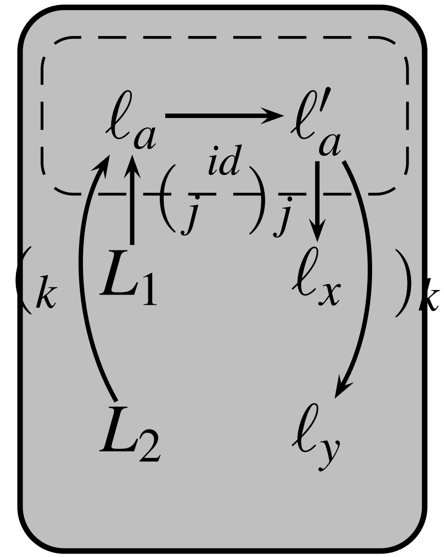
Example — CFL

let $id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle}$ in

let $x^{\langle \ell_x \rangle} = id^{j}\ 1^{\langle L_1 \rangle}$ in

let $y^{\langle \ell_y \rangle} = id^{k}\ 2^{\langle L_2 \rangle}$ in

y



- id has type $\forall \ell_a, \ell'_a . int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$
- In context i, we *instantiate* id to $int^{\langle \ell_1 \rangle} \longrightarrow int^{\langle \ell_x \rangle}$
- In context j, we *instantiate* id to $int^{\langle \ell_2 \rangle} \longrightarrow int^{\langle \ell_y \rangle}$
- There is *no* explicit constraint copying

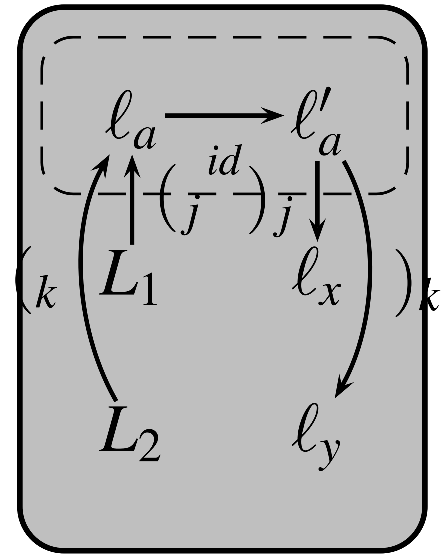
Example — CFL

$\texttt{let } id = \lambda a^{\langle \ell_a \rangle}.a^{\langle \ell'_a \rangle} \texttt{ in}$

$\texttt{let } x^{\langle \ell_x \rangle} = id^j\ 1^{\langle L_1 \rangle} \texttt{ in}$

$\texttt{let } y^{\langle \ell_y \rangle} = id^k\ 2^{\langle L_2 \rangle} \texttt{ in}$

y



- `id` has type $\forall \ell_a, \ell'_a.\, int^{\langle \ell_a \rangle} \longrightarrow int^{\langle \ell'_a \rangle}$
- In context i, we *instantiate* `id` to $int^{\langle \ell_1 \rangle} \longrightarrow int^{\langle \ell_x \rangle}$
- In context j, we *instantiate* `id` to $int^{\langle \ell_2 \rangle} \longrightarrow int^{\langle \ell_y \rangle}$
- There is *no* explicit constraint copying
  - Solution in $O(n^3)$

The Problem with Data Structures

$\texttt{let}\ f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}\ \texttt{in}$

$\texttt{let}\ g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}\ \texttt{in}$

$\texttt{let}\ p = \texttt{if}\ \ldots\ \texttt{then}$

$\quad (f^{j}, 1)^{\langle L_1 \rangle}$

$\quad \texttt{else}$

$\quad (g^{k}, 2)^{\langle L_2 \rangle}$

$\texttt{in}$

$\texttt{let}\ (p1, p2) = p\ \texttt{in}$

$\quad p1\ p2$

$$\begin{array}{ccc}
\ell_f \rightarrow & L_3 & \ell'_f \\
\ell_g & L_4 & \ell'_g \\
 & L_1 & \\
 & L_2 & \\
\ell_1 & \ell_3 & \ell_2
\end{array}$$

p: $(int^{\langle \ell_1 \rangle} \rightarrow int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$

- f is only applied to 1, g is only applied to 2
- *Constructor* L_1 is only consumed by *destructor* L_3, L_2 by L_4

The Problem with Data Structures

$\texttt{let}\ f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}\ \texttt{in}$

$\texttt{let}\ g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}\ \texttt{in}$

$\texttt{let}\ p = \texttt{if ... then}$

$\quad (f^{j}, 1^{\langle L_1 \rangle})$

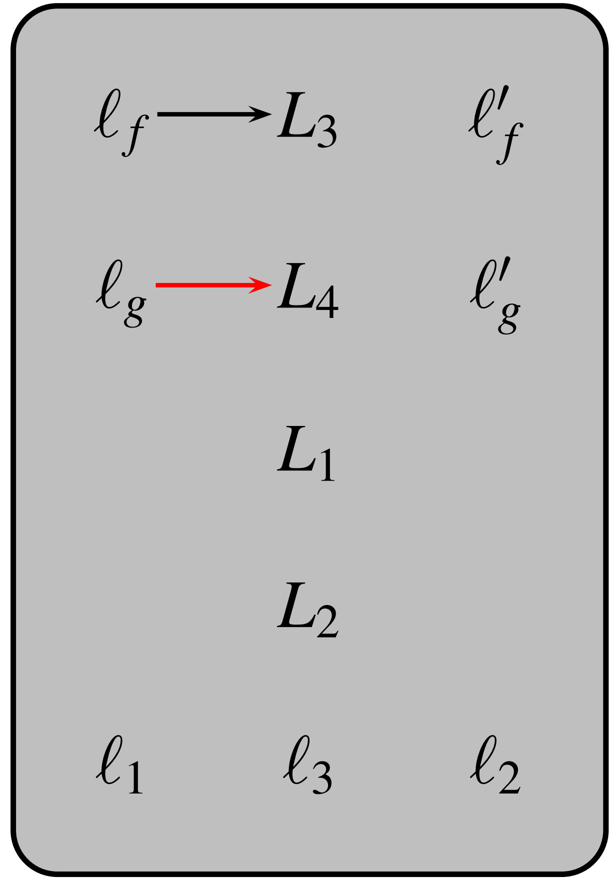
$\quad \texttt{else}$

$\quad (g^{k}, 2^{\langle L_2 \rangle})$

$\texttt{in}$

$\texttt{let}\ (p1, p2) = p\ \texttt{in}$

$\quad p1\ p2$



$\texttt{p}:\ (int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$

The Problem with Data Structures

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in

let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in

let $p =$ if ... then

$\quad (f^{j}, 1^{\langle L_1 \rangle})$

$\quad$ else $(_j$

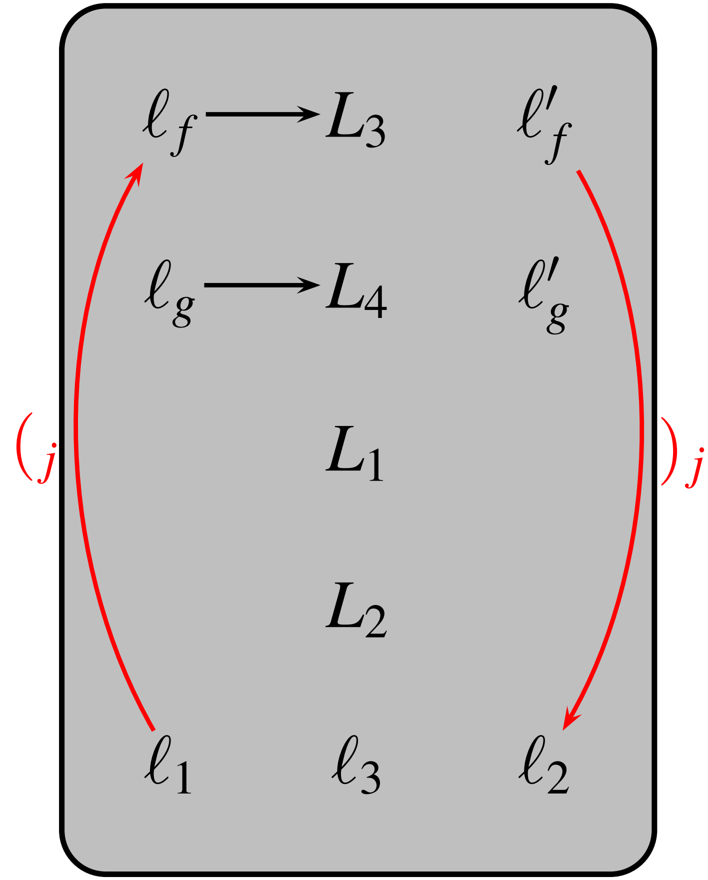
$\quad (g^{k}, 2^{\langle L_2 \rangle})$ $\quad)_j$

in

let $(p1, p2) = p$ in

$\quad p1\ p2$

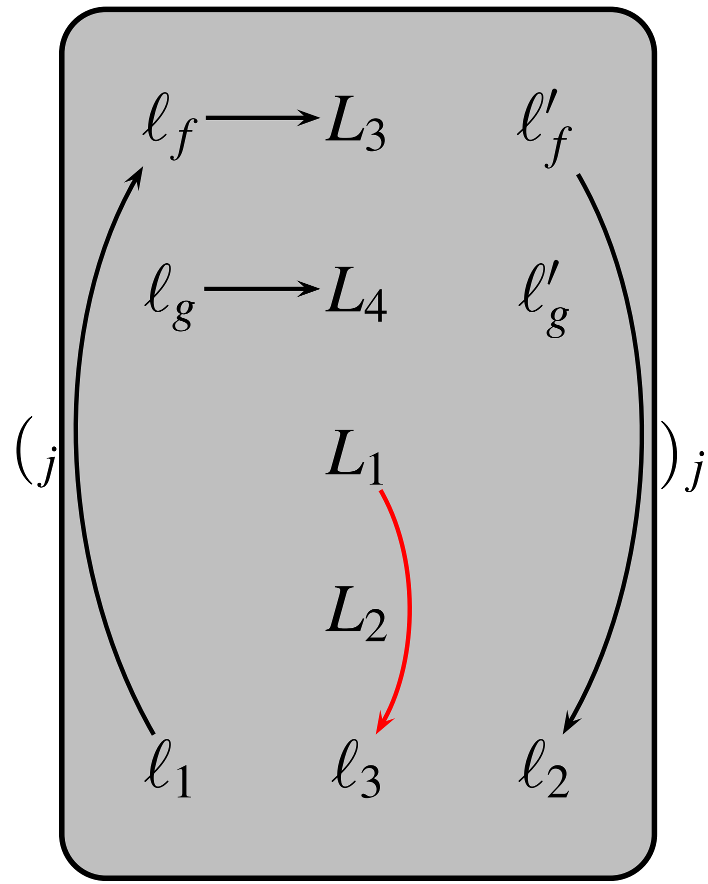
p: $(int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



The Problem with Data Structures

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in
let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in
let $p =$ if ... then
    $(f^{j}, 1^{\langle L_1 \rangle})$
  else
    $(g^{k}, 2^{\langle L_2 \rangle})$
 in
let $(p1, p2) = p$ in
  $p1\ p2$

p: $(int^{\langle \ell_1 \rangle} \rightarrow int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



The Problem with Data Structures

`let` $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ `in`

`let` $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ `in`

`let` $p =$ `if` ... `then`

$\quad (f^{j}, 1^{\langle L_1 \rangle})$

`else`

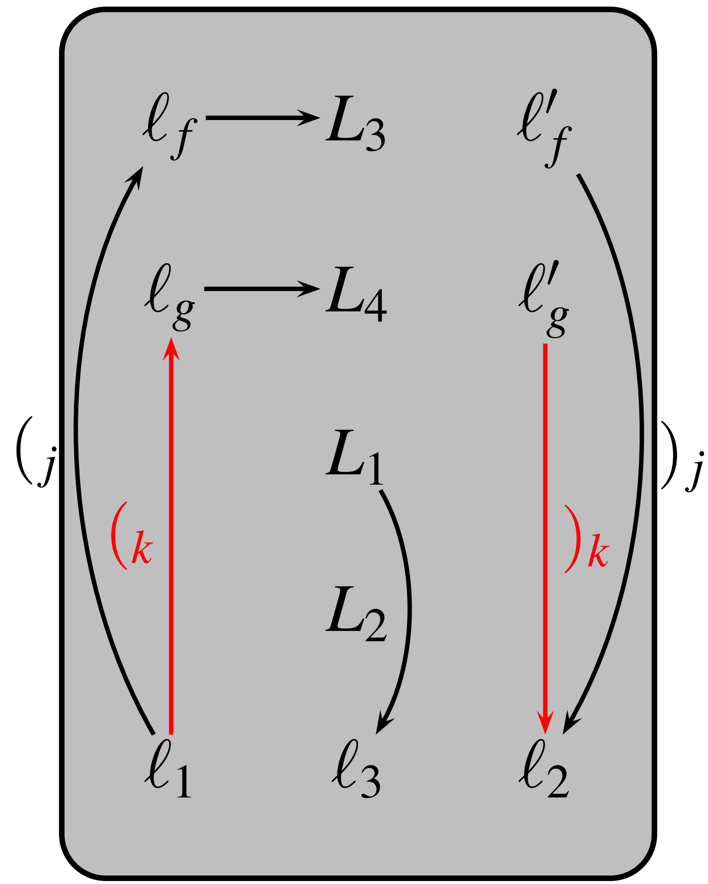
$\quad (g^{k}, 2^{\langle L_2 \rangle})$ $(_k$ $\qquad)_k$

`in`

`let` $(p1, p2) = p$ `in`

$\quad p1\ p2$

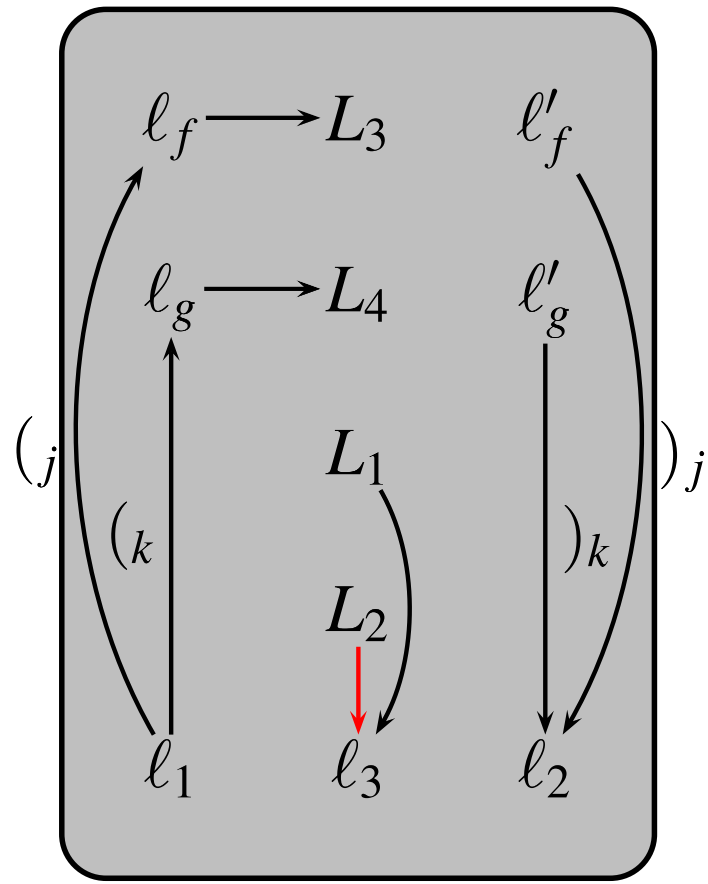
`p`: $(int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



The Problem with Data Structures

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in
let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in
let $p =$ if ... then
  $(f^j, 1^{\langle L_1 \rangle})$
 else
  $(g^k, 2^{\langle L_2 \rangle})$
in
let $(p1, p2) = p$ in
 $p1\ p2$

p: $(int^{\langle \ell_1 \rangle} \rightarrow int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



The Problem with Data Structures

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in

let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in

let $p =$ if ... then

$(f^{j}, 1^{\langle L_1 \rangle})$

else

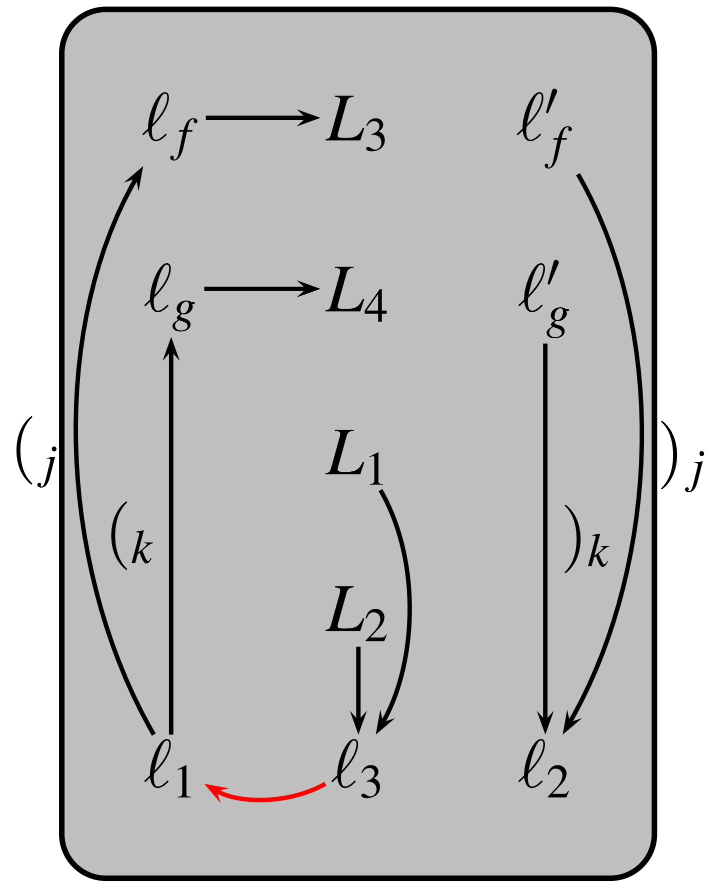
$(g^{k}, 2^{\langle L_2 \rangle})$

in

let $(p1, p2) = p$ in

$p1\ p2$

p: $(int^{\langle \ell_1 \rangle} \rightarrow int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



The Problem with Data Structures

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in

let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in

let $p =$ if … then

  $(f^j, 1^{\langle L_1 \rangle})$

 else

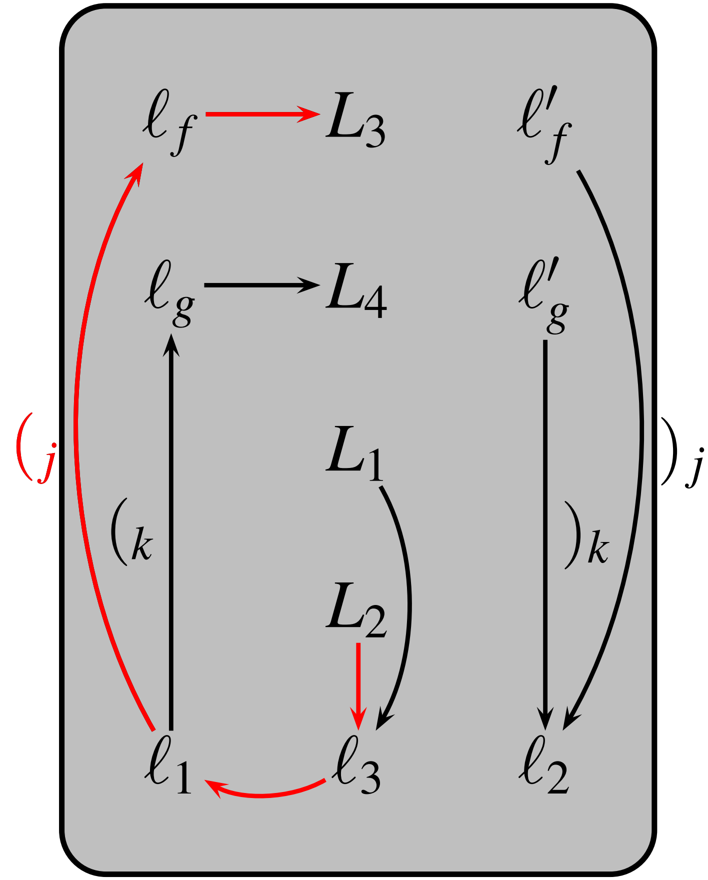
  $(g^k, 2^{\langle L_2 \rangle})$

in

let $(p1, p2) = p$ in

 $p1\ p2$

p: $(int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Duality of $\forall$ and $\exists$

- Existential abstraction is dual to universal abstraction
- For functions:
  - Abstract the body of the function when it is defined
  - Instantiate at every use of the function
  - Amounts to copying the flow graph from the definition to the use
- For data structures it is dual:
  - Abstract (capture) the context on every use
  - Instantiate (inline) at the definition
  - Amounts to copying the flow graph from the use to the definition

Duality: Direction of Inlining Contexts

Functions

$\texttt{let}\ f = \boxed{\ldots}\ \texttt{in}$

$\ldots$

$\boxed{f}$

$\ldots$

$\boxed{f}$

$\ldots$

Data structures

$\texttt{let}\ p = \texttt{pack}(\boxed{\ldots})\ \texttt{in}$

$\ldots$

$\texttt{unpack}\ p\ \texttt{in}$

$\quad\boxed{\ldots}$

$\ldots$

$\texttt{unpack}\ p\ \texttt{in}$

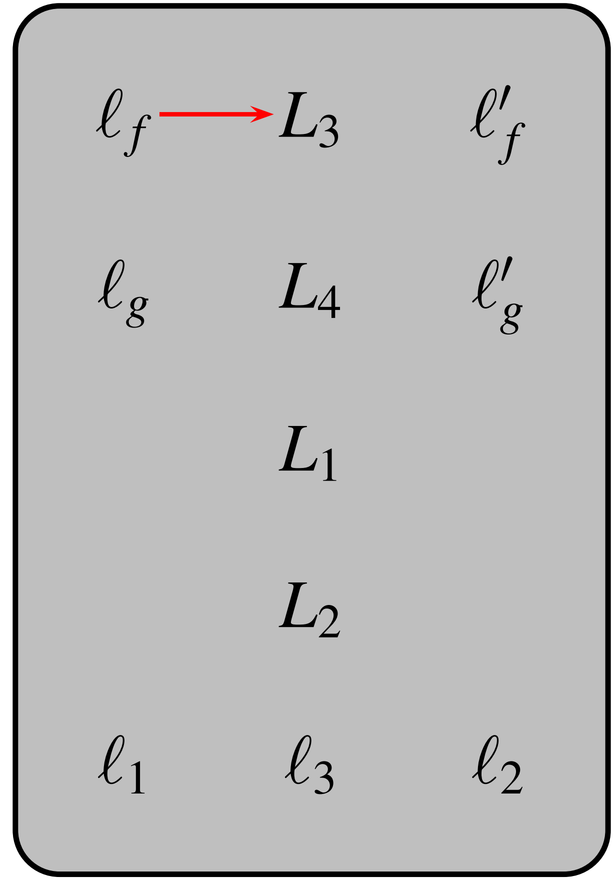
$\quad\boxed{\ldots}$

$\ldots$

Example — Existential Polymorphism

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in
let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in
let $p =$ if ... then
    pack$^m(f^j, 1^{\langle L_1 \rangle})$
  else
    pack$^n(g^k, 2^{\langle L_2 \rangle})$
 in
unpack $(p1, p2) = p$ in
  $p1\ p2$

p: $(int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Example — Existential Polymorphism

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in

let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in

let $p =$ if ... then

  pack$^m(f^j, 1^{\langle L_1 \rangle})$

 else

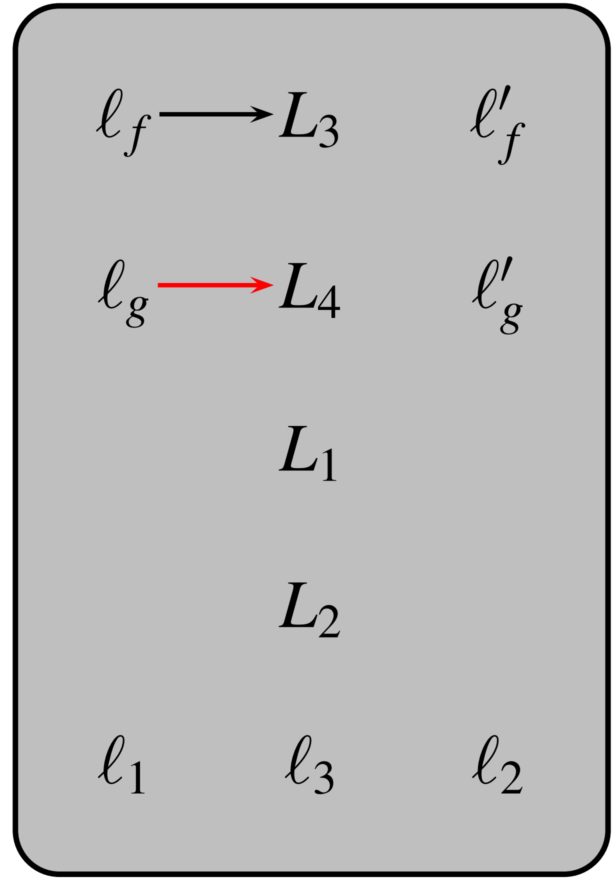
  pack$^n(g^k, 2^{\langle L_2 \rangle})$

in

unpack $(p1, p2) = p$ in

 $p1\ p2$

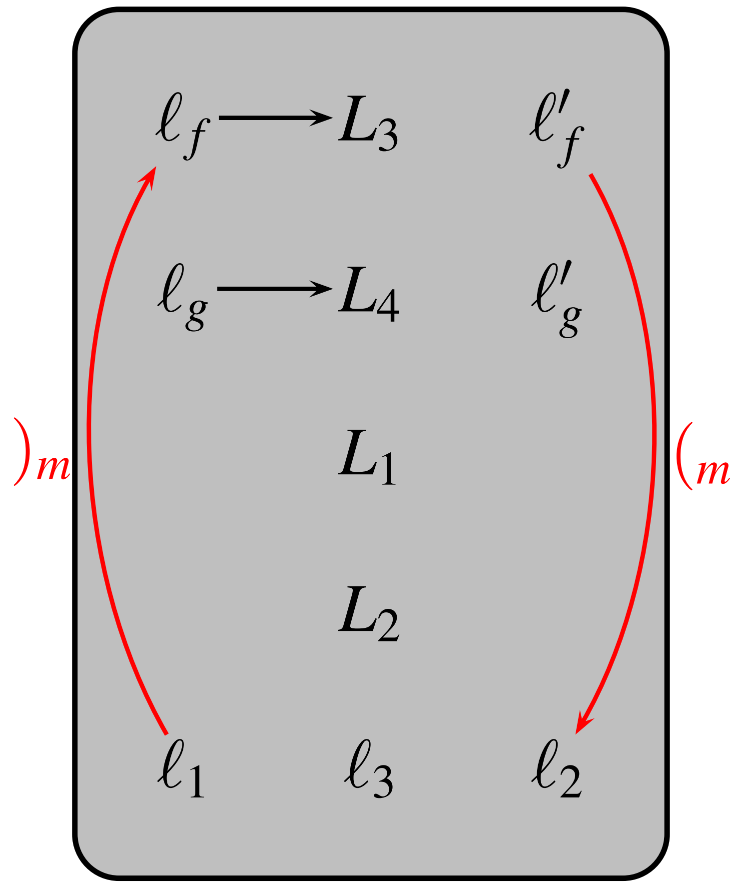
p: $(int^{\langle \ell_1 \rangle} \rightarrow int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Example — Existential Polymorphism

```
let f = λa⟨ℓf⟩.(a +⟨L3⟩ 42)⟨ℓ'f⟩ in
let g = λb⟨ℓg⟩.(b −⟨L4⟩ 42)⟨ℓ'g⟩ in
let p = if ... then
    pack^m(f^j, 1⟨L1⟩)
  else
    pack^n(g^k, 2⟨L2⟩)
 in
unpack (p1, p2) = p in
  p1 p2
```

p: $(int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Example — Existential Polymorphism

$\texttt{let}\ f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}\ \texttt{in}$

$\texttt{let}\ g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}\ \texttt{in}$

$\texttt{let}\ p = \texttt{if}\ \ldots\ \texttt{then}$

$\quad \texttt{pack}^m(f^j, 1^{\langle L_1 \rangle})$

$\quad \texttt{else}$

$\quad \texttt{pack}^n(g^k, 2^{\langle L_2 \rangle})$

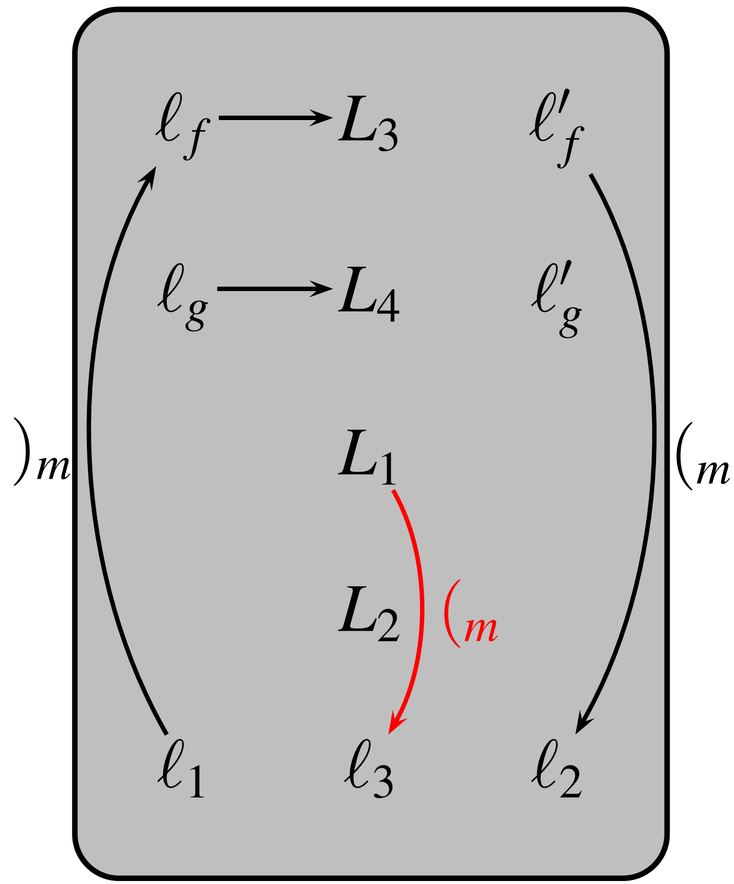
$\texttt{in}$

$\texttt{unpack}\ (p1, p2) = p\ \texttt{in}$

$\quad p1\ p2$

$($$_m$

$\texttt{p:}\ (int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Example — Existential Polymorphism

$\texttt{let}\ f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}\ \texttt{in}$

$\texttt{let}\ g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}\ \texttt{in}$

$\texttt{let}\ p = \texttt{if}\ \ldots\ \texttt{then}$

$\quad \texttt{pack}^m(f^j, 1^{\langle L_1 \rangle})$

$\quad \texttt{else}$

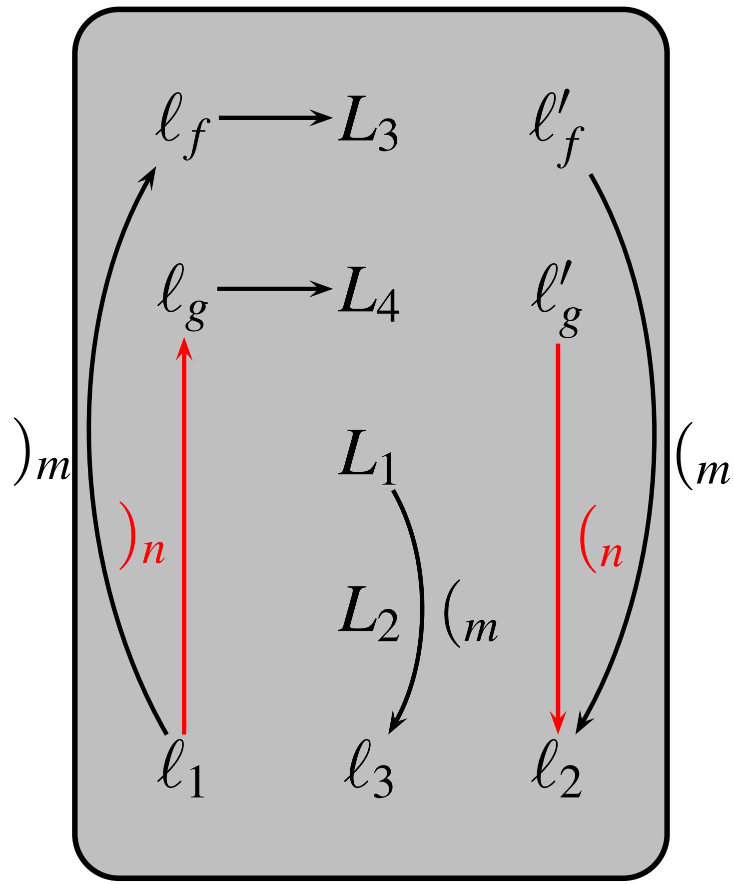
$\quad \texttt{pack}^n(g^k, 2^{\langle L_2 \rangle})$

$\texttt{in}$

$\texttt{unpack}\ (p1, p2) = p\ \texttt{in}$

$\quad p1\ p2$

$\texttt{p:}\ (int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Example — Existential Polymorphism

`let` $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ `in`

`let` $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ `in`

`let` $p =$ `if ... then`

    `pack`$^{m}(f^{j}, 1^{\langle L_1 \rangle})$

  `else`

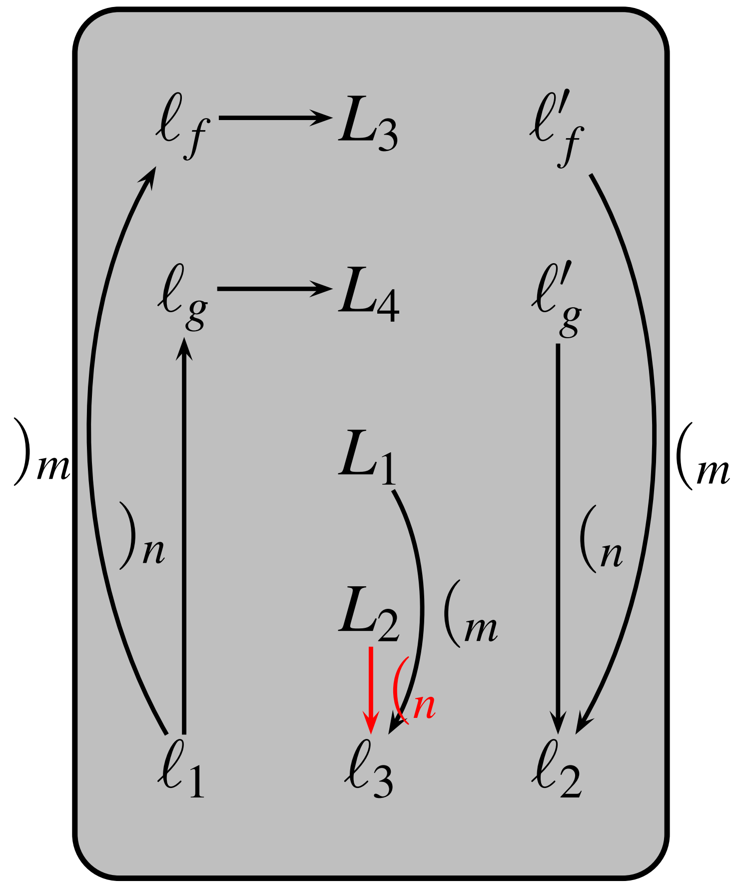
    `pack`$^{n}(g^{k}, 2^{\langle L_2 \rangle})$

`in`

`unpack` $(p1, p2) = p$ `in` $(_n$

  $p1\ p2$

`p`: $(int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Example — Existential Polymorphism

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in

let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in

let $p =$ if ... then

  pack$^m(f^j, 1^{\langle L_1 \rangle})$

  else

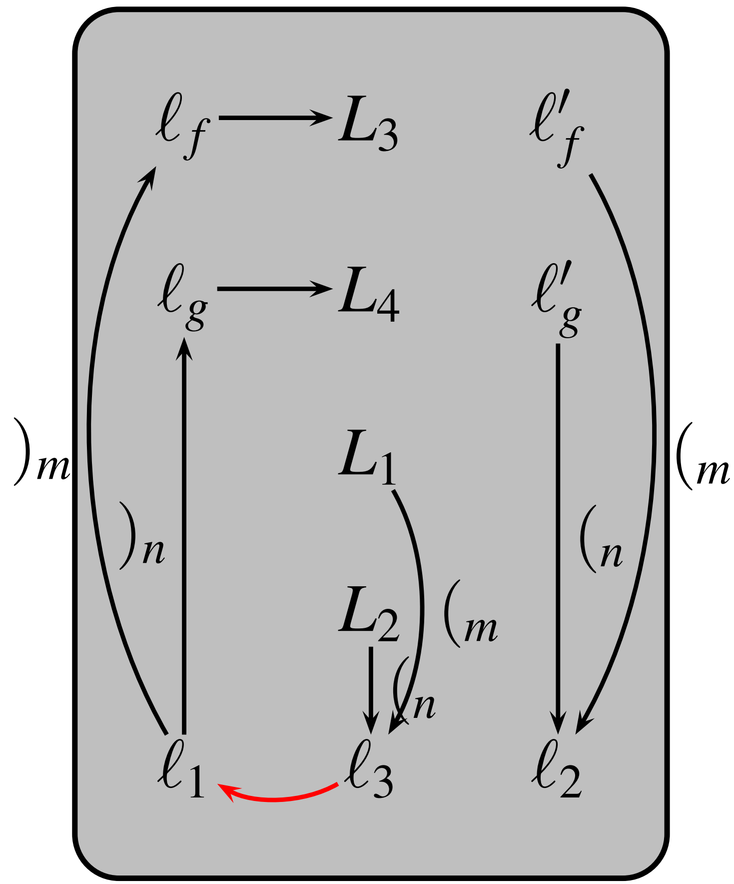
  pack$^n(g^k, 2^{\langle L_2 \rangle})$

in

unpack $(p1, p2) = p$ in

  $p1\ p2$

p: $(int^{\langle \ell_1 \rangle} \rightarrow int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$



Example — Existential Polymorphism

let $f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}$ in

let $g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}$ in

let $p =$ if ... then

  pack$^m (f^j, 1^{\langle L_1 \rangle})$

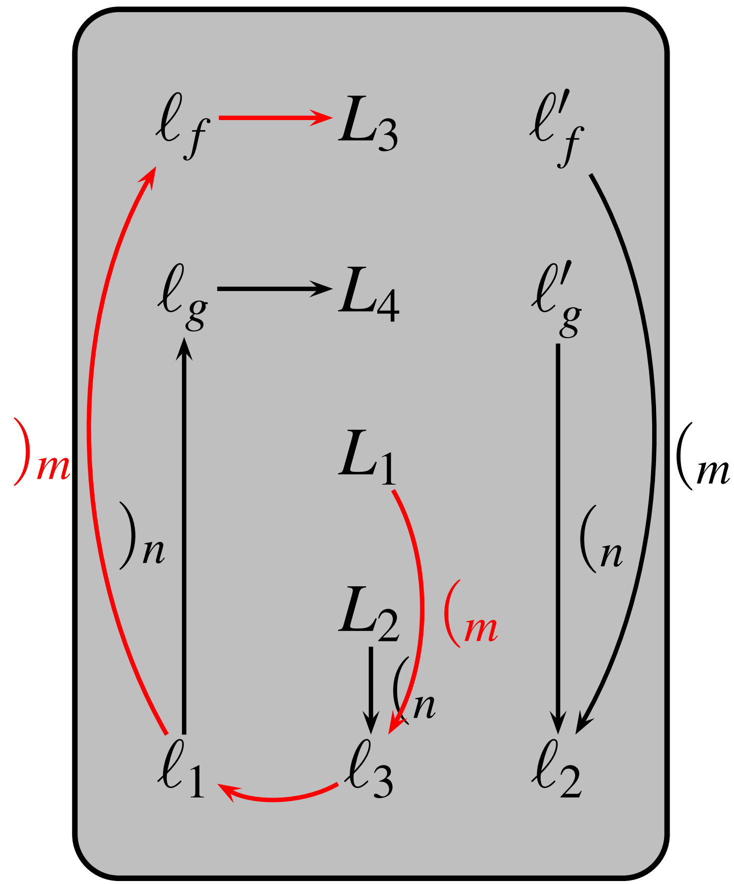
 else

  pack$^n (g^k, 2^{\langle L_2 \rangle})$

in

unpack $(p1, p2) = p$ in

 $p1\ p2$



p: $(int^{\langle \ell_1 \rangle} \to int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$

Example — Existential Polymorphism

$\texttt{let}\ f = \lambda a^{\langle \ell_f \rangle}.(a +^{\langle L_3 \rangle} 42)^{\langle \ell'_f \rangle}\ \texttt{in}$

$\texttt{let}\ g = \lambda b^{\langle \ell_g \rangle}.(b -^{\langle L_4 \rangle} 42)^{\langle \ell'_g \rangle}\ \texttt{in}$

$\texttt{let}\ p = \texttt{if}\ \ldots\ \texttt{then}$

$\quad \texttt{pack}^{m}(f^{j}, 1^{\langle L_1 \rangle})$

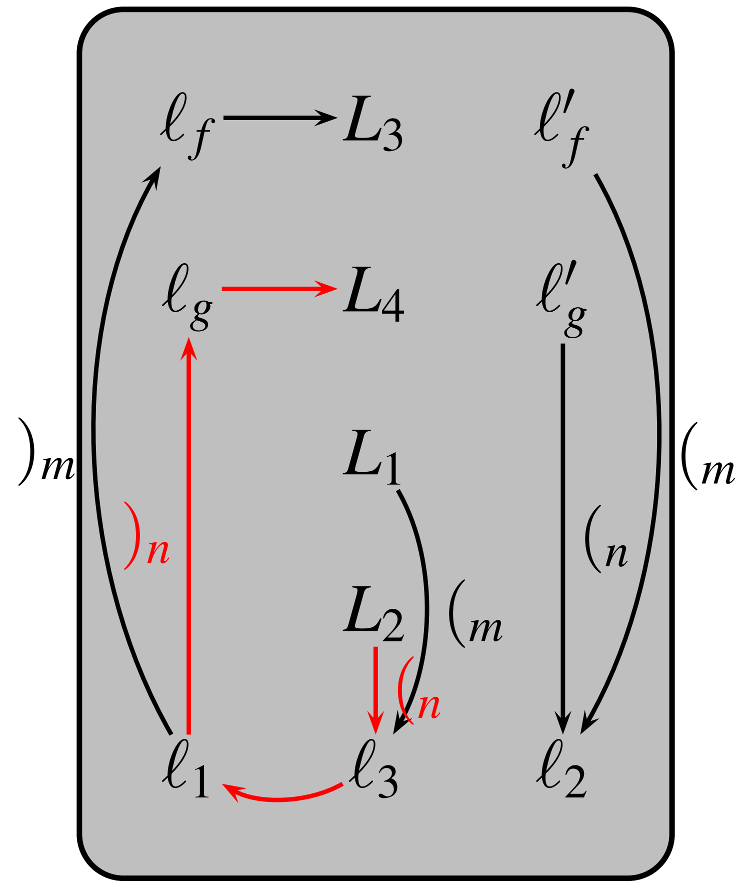
$\quad \texttt{else}$

$\quad \texttt{pack}^{n}(g^{k}, 2^{\langle L_2 \rangle})$

$\texttt{in}$

$\texttt{unpack}\ (p1, p2) = p\ \texttt{in}$

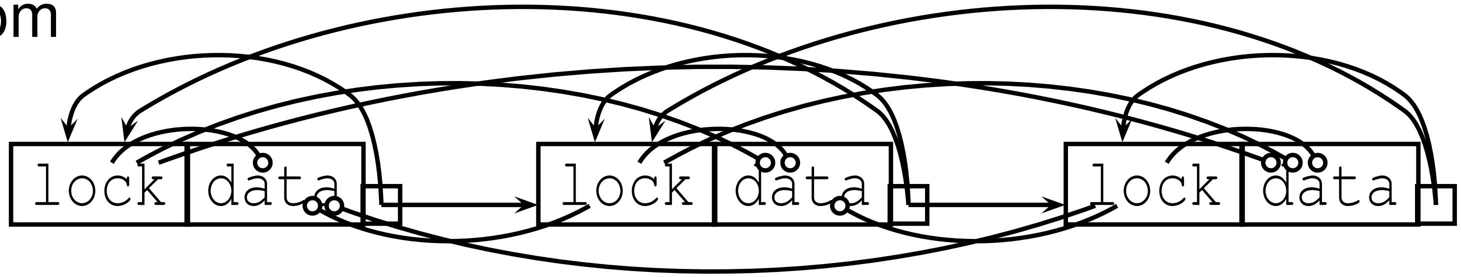
$\quad p1\ p2$



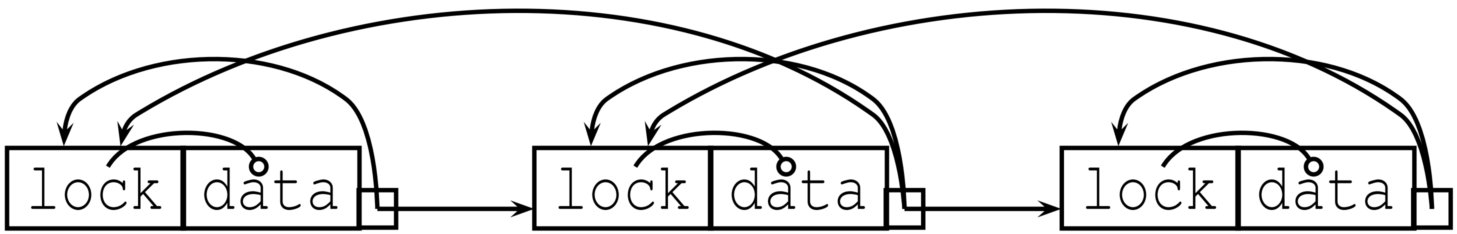
p: $(int^{\langle \ell_1 \rangle} \rightarrow int^{\langle \ell_2 \rangle}) \times int^{\langle \ell_3 \rangle}$

Finally...

- From



- to



- Benefit
  - Internal flow is precisely modeled. . .
  - . . . even when elements are conflated by static analysis

Differences

- Existentials are first-class
  - Can be passed around, across contexts before unpacking
  - Our solution: restrict existentially quantified variables from escaping the unpack
- Quantified types can include other quantified types
  - Sometimes it is possible to quantify label ℓ both existentially and universally: $\forall \ell . \exists \emptyset . int^{\ell}$ or $\forall \emptyset . \exists \ell . int^{\ell}$
  - There is no optimal strategy that always yields smaller (more precise) flow
  - Our solution: existentials explicitly state which labels are quantified

Other Uses of Existentials

- Data structures containing:
  - Closures: a function together with its arguments
  - Objects: a set of functions together with a `this` pointer
  - Array bounds: an array with an int that corresponds to its length
  - ...

Conclusions

- Existential Label Flow Analysis
  - Better handling of data structures
- Dual to universal polymorphism
  - Use the same context-sensitivity techniques
  - "Copy" the constraints backwards, from use to definition
  - Inference of flow graph and solution in $O(n^3)$
- Proof of soundness
  - Formalized context-copying system, proved sound
  - Formalized CFL system, proved by reduction to copying
- Future work
  - Infer what can be existentially quantified
  - Allow flow to escape unpacks similarly to the universal case