Program Transformation

A Presentation for PIMS students.

Robin Cockett and Brett Giles

{robin,gilesb}@cpsc.ucalgary.ca

University of Calgary

Presentation Overview

- An introduction to program transformation.
- Transforming the *collect* function on trees.
- Transforming the *Fibonnacci* function on infinite lists.

Why program transformation?

Given a program, we may use *program transformation* to change it into another program. This can be used to:

- Prove the equality of programs and therefore determine if a program satisfies a specification.
- Optimise a program - For example, change a $\mathcal{O}(n^2)$ program to $\mathcal{O}(n)$.
- Deforest a program - remove intermediate data structures.

Program Transformation and Programming Languages.

- Requirements:
  - Good representation of datatypes.
  - Clear formal specification of language.
  - Simple rules for program equality.
- NOT C, C++, Pascal, Java, ...
- Functional programming languages
  - SML, CAML, Haskell,... (Some issues with termination.)
- Categorical Programming Languages
  - Charity

Data Types.

- Inductive types such as $\mathbb{N}$, lists and trees.
  - $\mathbb{N} = \mu\, x.\{\textsc{Zero} : (), \textsc{Succ} : x\}$
  - $\textsc{List}(A) = \mu x.\{\textsc{Nil} : (), \textsc{Cons} : (f : A, s : x)\}$
  - $\textsc{Tree}(A) = \mu x.\{\textsc{Lf} : A, \textsc{Nd} : (l : x, r : x)\}$
- Co-Inductive types such as infinite lists.
  - $\textsc{InfList}(A)(= \mathbb{IL}(A)) = \nu x.\{\textsc{Hd} : A, \textsc{Tl} : x\}$

The collect *function*

The function $collect$ :: $\textsc{Tree}(A) \to \mathbb{L}(A)$ as

$$\mathsf{collect}(t) = \mu \mathsf{g}.\left[\ \#t' \mapsto \left\{ \begin{array}{l} \textsc{Lf}(a) \mapsto \textsc{Cons}(a, \textsc{Nil}) \\ \textsc{Nd}(l{:}x, r{:}y) \mapsto \mathsf{app}(\mathsf{g}\,x,\ \mathsf{g}\,y) \end{array} \right\} t' \ \right](t)$$

The algorithm.

Here we show a simple collection of a tree.

$$
\begin{aligned}
&= app\left(g\left(\begin{array}{ccc} a & & \\ & b & c \end{array}\right),\ g\left(\begin{array}{cc} d & e \end{array}\right)\right) \\
&= app\left(app\left(g\left(\begin{array}{cc} a & b \end{array}\right), [c]\right),\ app([d], [e])\right) \\
&= app\left(app\left(app([a], [b]), [c]\right),\ [d, e]\right) \\
&= app\left(app([a, b], [c]),\ [d, e]\right) \\
&= app\left([a, b, c], [d, e]\right) \\
&= [a, b, c, d, e]
\end{aligned}
$$

What is the complexity of collect?

Considering the degenerate case where a tree of size n has all of it's elements down the left hand side.

- At the top node, we append $n-1$ elements to 1 element, complexity of $n-1$.
- At the second node, we append $n-2$ elements to 1 element, complexity of $n-2$.
- At the third node, we append $n-3$ elements to 1 element, complexity of $n-3$.
- etc.

Therefore the complexity is
$(n-1)+(n-2)+(n-3)+\cdots = \mathcal{O}(n^2)$.

Transformation - Unwind

$$\mathsf{g}\ t = \mu\_.\left[\ \#x' \mapsto \left\{ \begin{array}{l} \textsc{Lf}(a) \mapsto \textsc{Cons}(a, \textsc{Nil}) \\ \\ \textsc{Nd}(x, y) \mapsto \mathsf{app}(\mathsf{g}\ x,\ \mathsf{g}\ y) \end{array} \right\} t' \ \right] t$$

Abstract over the second argument.

$$\mathsf{app}(\mathsf{g}\ x, z)$$

$$= \mathsf{app}\left(\mu\_.\left[\ \#t' \mapsto \left\{\begin{array}{l} \textsc{Lf}(a) \mapsto \textsc{Cons}(a, \textsc{Nil}) \\ \\ \textsc{Nd}(p, q) \mapsto \mathsf{app}(\mathsf{g}\ p,\ \mathsf{g}\ q) \end{array}\right\} t'\ \right]\ x,\ z\right)$$

Bring the app$(_, z)$ inside.

$$= \mu\_.\left[\ \#t' \mapsto \left\{ \begin{array}{l} \textsc{Lf}(a) \mapsto \mathsf{app}(\textsc{Cons}(a, \textsc{Nil}), z) \\ \\ \textsc{Nd}(p, q) \mapsto \mathsf{app}\,(\mathsf{app}(\mathsf{g}\ p, \mathsf{g}q),\ z) \end{array} \right\} t' \ \right] x$$

Use the associativity of append and its definiton over CONS.

$$= \mu\_.\left[\ \#t' \mapsto \left\{ \begin{array}{l} \textsc{Lf}(a) \mapsto \textsc{Cons}(a, z) \\ \\ \textsc{Nd}(p, q) \mapsto \mathsf{app}\,(\mathsf{g}\ p, \mathsf{app}(\mathsf{g}q,\ z)) \end{array} \right\} t' \ \right] x$$

Tying the knot.

The interior of the equation on slide **??** matches the left hand size of the equation on slide **??**. Note that at the same time, the second argument to the appalso matches the equation. We therefore tie the knot replacing both occurrences with a new inductive function.

$$= \mu \mathsf{f} \left[\#t', y' \mapsto \left\{ \begin{array}{l} \textsc{Lf}(a) \mapsto \textsc{Cons}(a, z') \\ \\ \textsc{Nd}(p, q) \mapsto \mathsf{f}(p, \mathsf{f}(q, z')) \end{array} \right\} t' \right] x$$

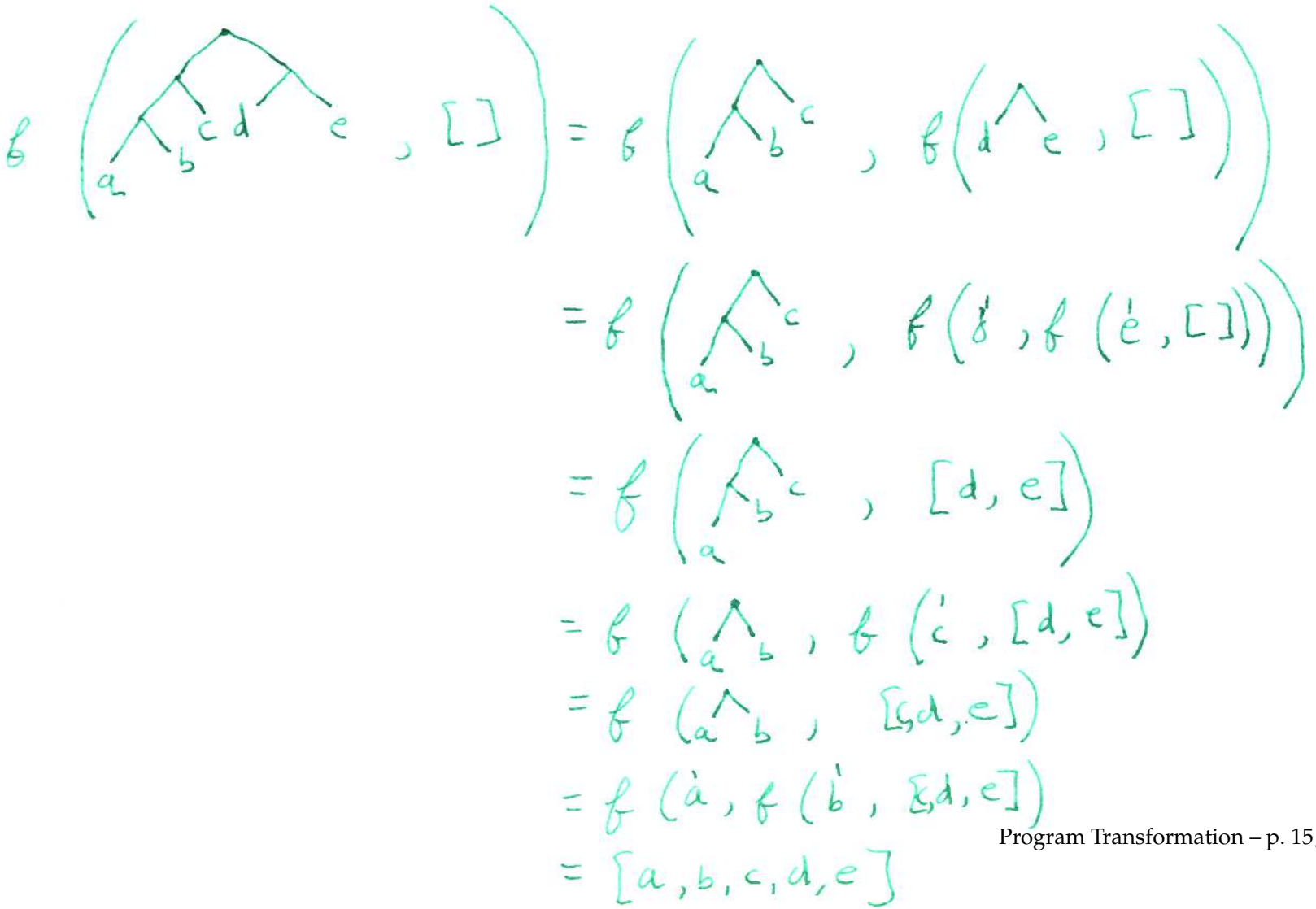
Plug back to our original equation.

- Transformed into a function taking $\mathcal{O}(n)$ time.

$$\mathsf{collect}(t) = \mu\mathsf{fclt}\left[\, \#t', z' \mapsto \left\{ \begin{array}{l} \textsc{Lf}(a) \mapsto \textsc{Cons}(a, z') \\ \\ \textsc{Nd}(x, y) \mapsto \mathsf{fclt}(x, \mathsf{fclt}(y, z')) \end{array} \right\} t' \,\right] (t, \textsc{Nil})$$

The transformed algorithm.

On the same tree as before:



Working with infinite functions

We define the infinite list fib as

$$fib = \nu \mathsf{fb} \left[\#(n,m) \mapsto \left(\begin{array}{l} \textsc{Hd}{:}n \\ \textsc{Tl}{:}\mathsf{fb}(m, n+m) \end{array} \right) \right] (\textsc{Succ}\ \textsc{Zero}, \textsc{Succ}\ \textsc{Zero})$$

Define the function get :: $\mathbb{N} \times \mathbb{IL}(\mathbb{N}) \to \mathbb{N}$ as

$$get(n, L) = \textsc{Hd}\ \mu\, \mathsf{gtt} \left[\#n' \mapsto \left\{ \begin{array}{l} \textsc{Zero} \mapsto L \\ \textsc{Succ}\,(m) \mapsto \textsc{Tl}\ .\mathsf{gtt}(m) \end{array} \right\} n' \right] n$$

Transforming Fibonacci

From this, we can define the Fibonacci function to be:

$$\mathsf{fc}(n) = \mathsf{get}(n, fib)$$

Rewriting fc,

$$\mathsf{fc}(n) = [(v, w) \mapsto \mathsf{get}(n, \mathsf{fb}(v, w))](\textsc{Succ}\ \textsc{Zero}, \textsc{Succ}\ \textsc{Zero})$$

The *fc* Algorithm.

To compute the 6^{th} Fibonnacci number:

$$\begin{aligned}
\mathsf{fc}(6) &= \mathsf{get}(6, [1,1,2,3,5,8,13,\ldots]) = \mathsf{HD}\ \mathsf{TL}\ \mathsf{gtt}(5) \\
&= \mathsf{HD}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{gtt}(4) \\
&= \ldots = \mathsf{HD}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{gtt}(0) \\
&= \mathsf{HD}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ [1,1,2,3,5,8,13,21,...] \\
&= \mathsf{HD}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ \mathsf{TL}\ [1,2,3,5,8,13,21,...] \\
&= \ldots = \mathsf{HD}\ [13,21,...] \\
&= 13
\end{aligned}$$

Unwind the interior.

$$\mathsf{get}(n,\mathsf{fb}(v,w)) =$$

$$\mu\_.\left[\ \#n', L' \mapsto \left\{ \begin{array}{l} \textsc{Zero} \mapsto \textsc{Hd}\ L' \\ \textsc{Succ}\ (m) \mapsto \mathsf{get}(m, \textsc{Tl}\ (L')) \end{array} \right\} n'\ \right]$$
$$(n, \mathsf{fb}(v,w))$$

Substituting **fb**(v, w) *for* L'

$$= \mu\_. \left[\#n' \mapsto \left\{ \begin{array}{l} \textsc{Zero} \mapsto \textsc{Hd}\,\mathsf{fb}(v, w) \\ \\ \textsc{Succ}\,(m) \mapsto \mathsf{get}(m, \textsc{Tl}\,(\mathsf{fb}(v, w))) \end{array} \right\} n' \right] (n)$$

Definition of TL $(\mathit{fb}(v, w))$

$$= \mu\_. \left[\#n' \mapsto \left\{ \begin{array}{l} \textsc{Zero} \mapsto v \\ \\ \textsc{Succ}\,(m) \mapsto \mathsf{get}(m, \mathsf{fb}(w, v + w)) \end{array} \right\} n' \right] (n)$$

Tying the infinite Knot.

The equation on slide **??** is in the same form as the equation on slide **??**. Knot tie and get:

$$\mathsf{get}(n, \mathsf{fb}(v, w))$$

$$= \mu\mathsf{f}.\left[\, \#n', v', w' \mapsto \left\{ \begin{array}{l} \textsc{Zero} \mapsto v' \\ \\ \textsc{Succ}\,(m) \mapsto \mathsf{f}(m, w', v' + w') \end{array} \right\} n' \,\right]_{(n,v,w)}$$

Substitute back.

- A new function that does not use Infinite Lists, that is, intermediate data structures have been removed.

$$\mathsf{fc}(n)$$

$$= \mu\mathsf{f}.\left[\ \#n', v', w' \mapsto \left\{ \begin{array}{l} \textsc{Zero} \mapsto v' \\ \\ \textsc{Succ}\,(m) \mapsto \mathsf{f}(m, w', v' + w') \end{array} \right\} n' \ \right]$$

$$(n, \textsc{Succ}\,\textsc{Zero}, \textsc{Succ}\,\textsc{Zero}))$$

The new fcAlgorithm.

To compute the 6^{th} Fibonnacci number:

$$\begin{aligned}
\mathsf{fc}(6) &= \mathsf{f}(6,1,1) \\
&= \mathsf{f}(5,1,2) \\
&= \mathsf{f}(4,2,3) \\
&= \mathsf{f}(3,3,5) \\
&= \mathsf{f}(2,5,8) \\
&= \mathsf{f}(1,8,13) \\
&= \mathsf{f}(0,13,21) \\
&= 13
\end{aligned}$$