A Functional Reformulation of UnCAL Graph-Transformations

Or, Graph Transformation as Graph Reduction

Kazutaka Matsuda
Tohoku University

Kazuyuki Asada
University of Tokyo

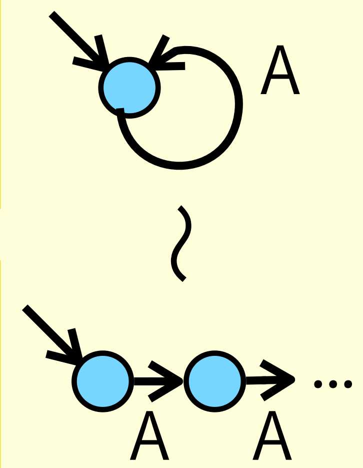
Jan 17 @ PEPM 2017

This Talk is about ...

- FUnCAL:
  a functional graph-transformation language
  - reformulation of UnCAL [Buneman 00]
    - graph transformation language from DB comm.
  - describes ***infinite-tree*** transformations
    - no special operators for graph transformations
      → no special treatment in program manipulation
  - runs as ***terminating finite-graph*** transformations
    - by lazy evaluation (with black holes)

Background: UnCAL

- A functional graph-transformation language [Buneman+ 00]
  - terminating (in polynomial time)
  - graph equality by ***bisimulation***
    - ***graphs = (possibly infinite) regular trees***
      - regular: #{subtrees} < ∞
    - polynomial-time equivalence check
  - refocused recently in bidirectional graph transformations [Hidaka+10~, Sasano+11, Yu+12, ...]



Syntax of (Positive) UnCAL

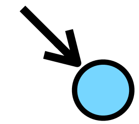
$$
\begin{aligned}
g ::=\ & \{\} \mid \{a{:}g\} \mid g_1 \cup g_2 \\
\mid\ & \&x \mid \&x \triangleright g \mid g_1 \mathbin{@} g_2 \mid \mathbf{cycle}(g) \\
\mid\ & () \mid g_1 \oplus g_2 \\
\mid\ & x \mid \mathbf{srec}(\lambda(z,x).g_1)(g_2)
\end{aligned}
$$

$$
a ::= z \mid L
$$

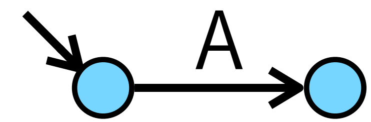
(we removed “label equality test” for simplicity.)

UnCAL Graphs (1/3)

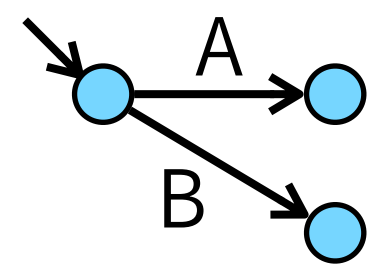
`{}`



`{A:{}}`

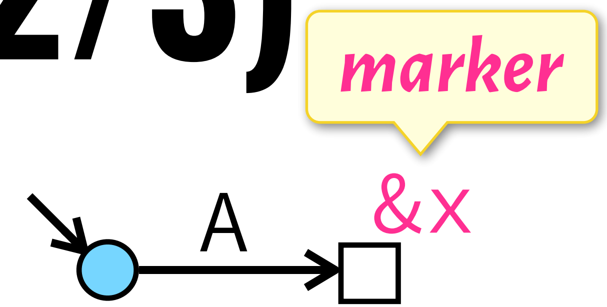


`{A:{}}`
`∪ {B:{}}`

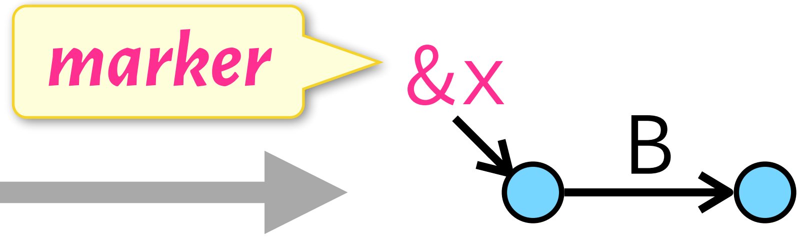


UnCAL Graphs (2/3)

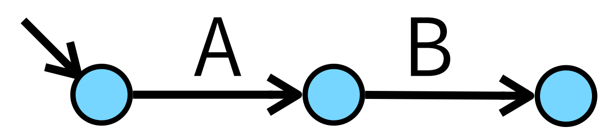
`{A:&x}`



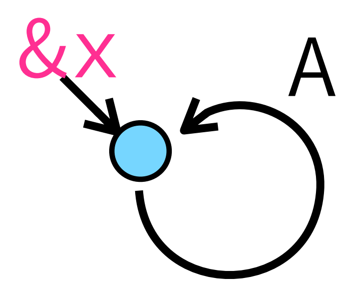
`&x▷{B:{}}`




```
{A:&x} @
 (&x▷{B:{}})
```




```
cycle
 (&x▷{A:&x})
```

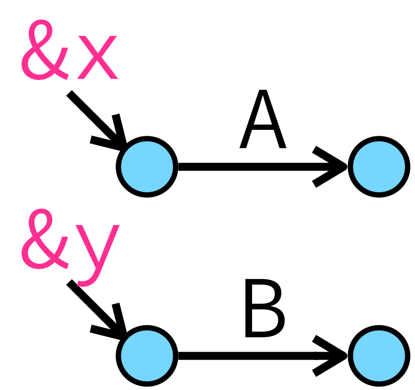


UnCAL Graphs (3/3)

()

Graphs with no root

(&x ▹ {A:{}})
⊕ (&y ▹ {B:{}})

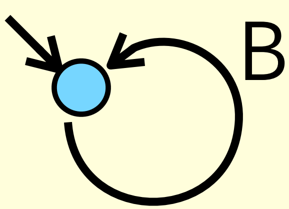


multi-rooted graphs

Structural Recursion: srec

$$\mathbf{srec}(\lambda(z.t).e)(\{z_1:t_1\}\cup\cdots\cup\{z_n:t_n\}) =$$
$$(e[z_1/z,t_1/t])@\mathbf{srec}(\cdots)(t_1)$$
$$\cup\ \cdots\ \cup$$
$$(e[z_n/z,t_n/t])@\mathbf{srec}(\cdots)(t_n)$$

$$\mathbf{srec}(\lambda(z,t).\&y \triangleright \{A:\{z:\&y\}\})$$
$$(\&x@\mathbf{cycle}(\&x \triangleright \{B:\&x\}))$$

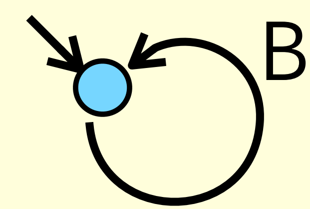


$$\sim\ \{B:\{B:\cdots\}\}$$

Structural Recursion: srec

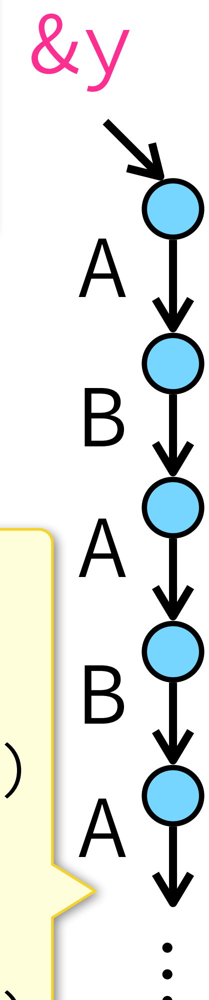
$\mathbf{srec}(\lambda(z.t).e)(\{z_1{:}t_1\}\cup\cdots\cup\{z_n{:}t_n\})\ =$
$\quad(e[z_1/z,t_1/t])\,@\,\mathbf{srec}(\cdots)(t_1)$
$\quad\cup\ \cdots\ \cup$
$\quad(e[z_n/z,t_n/t])\,@\,\mathbf{srec}(\cdots)(t_n)$

$\mathbf{srec}(\lambda(z,t).\&y \triangleright \{A{:}\{z{:}\&y\}\})$
$(\&x\,@\,\mathbf{cycle}(\&x \triangleright \{B{:}\&x\}))$



$\sim\ \{B{:}\{B{:}\cdots\}\}$

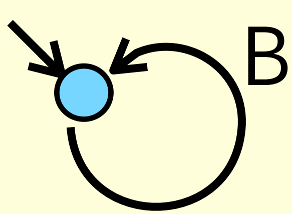
$\mathbf{srec}(\cdots)(\{B{:}\{B{:}\cdots\}\})$
$=\ (\&y \triangleright \{A{:}\{B{:}\&y\}\})$
$\quad@\,\mathbf{srec}(\cdots)(\{B{:}\{B{:}\cdots\}\})$
$=\ (\&y \triangleright \{A{:}\{B{:}\&y\}\})$
$\quad@(\&y \triangleright \{A{:}\{B{:}\&y\}\})$
$\quad@\,\mathbf{srec}(\cdots)(\{B{:}\{B{:}\cdots\}\})$



Structural Recursion: srec

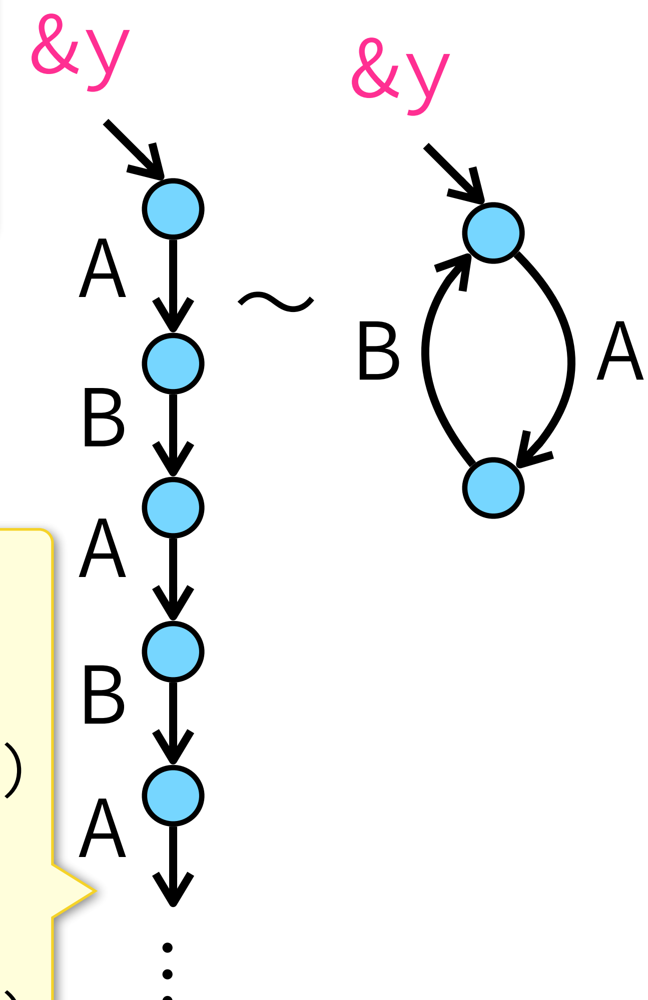
$$\mathbf{srec}(\lambda(z.t).e)(\{z_1{:}t_1\}\cup\cdots\cup\{z_n{:}t_n\}) =$$
$$(e[z_1/z,t_1/t])@\mathbf{srec}(\cdots)(t_1)$$
$$\cup\ \cdots\ \cup$$
$$(e[z_n/z,t_n/t])@\mathbf{srec}(\cdots)(t_n)$$

$$\mathbf{srec}(\lambda(z,t).\&y\triangleright\{A{:}\{z{:}\&y\}\})$$
$$(\&x@\mathbf{cycle}(\&x\triangleright\{B{:}\&x\}))$$



$\sim\ \{B{:}\{B{:}\cdots\}\}$

$$\mathbf{srec}(\cdots)(\{B{:}\{B{:}\cdots\}\})$$
$$=\ (\&y\triangleright\{A{:}\{B{:}\&y\}\})$$
$$@\mathbf{srec}(\cdots)(\{B{:}\{B{:}\cdots\}\})$$
$$=\ (\&y\triangleright\{A{:}\{B{:}\&y\}\})$$
$$@(\&y\triangleright\{A{:}\{B{:}\&y\}\})$$
$$@\mathbf{srec}(\cdots)(\{B{:}\{B{:}\cdots\}\})$$



Problem

- ***Special*** operators for ***graph*** transformation
  - especially ***marker-related operations***
    - `&x`, `&x`$\triangleright$`g`, `@`, **`cycle`**, `()`, $\oplus$, **`srec`**
  - prevent us from applying FP-based program manipulation techniques directory to UnCAL
    - optimization, verification, implementation, …

Approach

- Express UnCAL graph transformations as ***usual*** functional programs on ***infinite trees***
  - leveraging the fact "***graphs = (possibly-infinite) regular trees***"
  - cf. ***special*** operators for ***graph*** transformation

Contributions

- FUnCAL: functional reformulation of UnCAL
  - describes ***infinite-tree*** transformations
    - ***no special operators*** for graph manipulation
      → more ***FP-based program-manipulation*** friendly
    - ***expressive as UnCAL***
  - runs as ***terminating finite-graph*** transformation
    - by lazy evaluation with black holes [Ariola&Klop 96]
      - a variant of [Nakata&Hasegawa 09]
    - ensured by our ***type system***

Outline

- Introduction
- FUnCAL
- FUnCAL Programs as Graph Transformations
- Conclusion

FUnCAL

- $\lambda^{\to,\times}$ + constructors + restricted recursions
  - call-by-name (for now)

```
e ::= x | λx.e | e₁ e₂
    | πᵢ | (e₁,⋯,e₂)
    | ● | e₁ : e₂ | e₁ ∪ e₂ | L
    | fix e | foldₙ e
```

$$\begin{aligned} e ::= {} & x \mid \lambda x.e \mid e_1\ e_2 \\ \mid {} & \pi_i \mid (e_1, \cdots, e_2) \\ \mid {} & \bullet \mid e_1 : e_2 \mid e_1 \cup e_2 \mid L \\ \mid {} & \mathbf{fix}\ e \mid \mathbf{fold}_n\ e \end{aligned}$$

FUnCAL

- $\lambda^{\rightarrow,\times}$ + constructors + restricted recursions
  - call-by-name (for now)

```
e ::= x | λx.e | e₁ e₂
    | πᵢ | (e₁,⋯,e₂)
    | ● | e₁ : e₂ | e₁ ∪ e₂ | L
    | fix e | foldₙ e
```

```
data G = ●
       | Label : G
       | G ∪ G
```

FUnCAL

- $\lambda^{\to,\times}$ + constructors + restricted recursions
  - call-by-name (for now)

```
data G = ●
       | Label : G
       | G ∪ G
```

$$
\begin{aligned}
e ::= {} & x \mid \lambda x.e \mid e_1\ e_2 \\
\mid {} & \pi_i \mid (e_1, \cdots, e_2) \\
\mid {} & \bullet \mid e_1 : e_2 \mid e_1 \cup e_2 \mid L \\
\mid {} & \mathbf{fix}\ e \mid \mathbf{fold}_n\ e
\end{aligned}
$$

$\mathbf{fix} : (G^n \to G^n) \to G^n$

FUnCAL

- $\lambda^{\to,\times}$ + constructors + restricted recursions
  - call-by-name (for now)

```
e ::= x | λx.e | e₁ e₂
    | πᵢ | (e₁,⋯,e₂)
    | ● | e₁ : e₂ | e₁ ∪ e₂ | L
    | fix e | foldₙ e
```

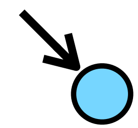
```
data G = ●
       | Label : G
       | G ∪ G
```

```
fix : (Gⁿ → Gⁿ) → Gⁿ
```

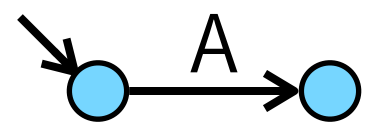
```
foldₙ f ●       = (●,...,●)
foldₙ f (a:x)   = f a (foldₙ f x)
foldₙ f (x ∪ y) = (foldₙ f x) ∪ₙ (foldₙ f y)
```

Examples

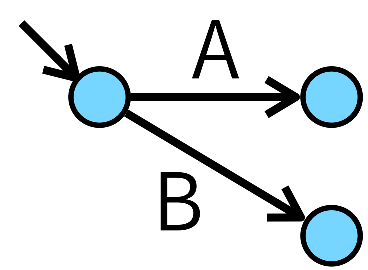
$\bullet \qquad \sim$



$A : \bullet \qquad \sim$



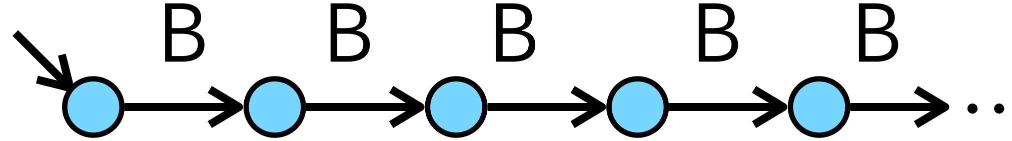
$\{A : \bullet\} \cup \{B : \bullet\} \qquad \sim$

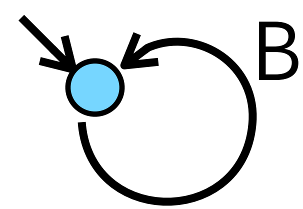


Examples

`fix (λx.B:x)` $\sim$ `B:fix(λx.B:x)`

$\sim$ `B:B:B:B:`$\cdots$

$\sim$ 

($\sim$ 

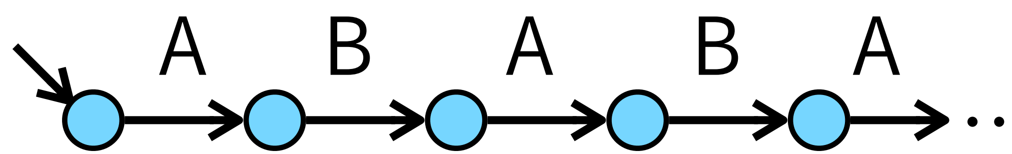
)

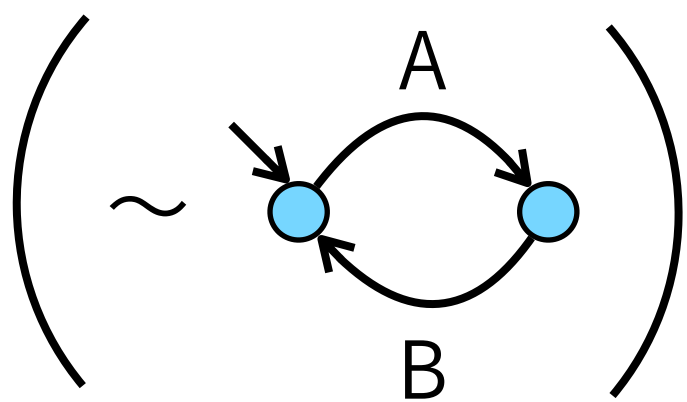
Examples

fold$_1$ (λz.λt.A:z:t) (**fix** (λx.B:x))

∼ **fold**$_1$ (λz.λt.A:z:t) (B:B:B:B:B:⋯)

∼ A:B:A:B:A:B:A:B:⋯

∼ 



Relationship to UnCAL

- Typed UnCAL programs [Buneman+ 96] can be converted to typed FUnCAL programs
  - see our paper for details

```
cycle(&x▹{A:&x})  ⟶  λz.fix (λx.A:x)

{A:&x}@(&x▹{B:{}})  ⟶  λz.(λx.A:x)
                            ((λy.B:●) z)
```

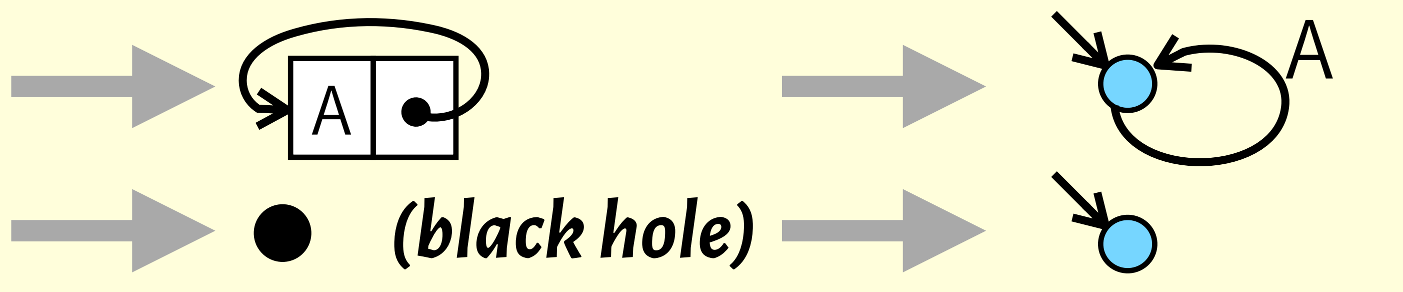
Outline

- Introduction
- FUnCAL
- FUnCAL Programs as Graph Transformations
- Conclusion

Goal

- ❖ Run FUnCAL programs as ***graph*** transformation
  - semantics
    - a variant of lazy evaluation with black holes
      - [Nakata&Hasegawa 09] + memoized fold

```
fix (λx.A:x)
fix (λx.x)
```



  - finer-type system
    - simple types are not enough (see the next page)

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

```
fix (λx.B:insA x)
```

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

```
fix (λx.B:insA x)
→* B:insA (fix ...)
```

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

```
fix (λx.B:insA x)
→* B:insA (fix ...)
→* B:insA (B:insA (fix ...))
```

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

$\mathtt{insA} = \mathbf{fold_1}\ (\lambda z.\lambda t.\mathtt{A{:}z{:}t})$

$\mathbf{fix}\ (\lambda x.\mathtt{B{:}insA}\ x)$

$\to^*\ \mathtt{B{:}insA}\ (\mathbf{fix}\ \ldots)$

$\to^*\ \mathtt{B{:}insA}\ (\mathtt{B{:}insA}\ (\mathbf{fix}\ \ldots))$

$\to^*\ \mathtt{B{:}A{:}B{:}insA}^2\ (\mathbf{fix}\ \ldots)$

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

```
fix (λx.B:insA x)
→* B:insA (fix ...)
→* B:insA (B:insA (fix ...))
→* B:A:B:insA² (fix ...)
→* B:A:B:insA² (B:insA (fix ...))
```

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

$$\mathtt{insA} = \mathbf{fold_1}\ (\lambda z.\lambda t.\mathtt{A{:}z{:}t})$$

fix (λx.B:insA x)

$\rightarrow^*$ B:insA (**fix** ...)

$\rightarrow^*$ B:insA (B:insA (**fix** ...))

$\rightarrow^*$ B:A:B:insA^2 (**fix** ...)

$\rightarrow^*$ B:A:B:insA^2 (B:insA (**fix** ...))

$\rightarrow^*$ B:A:B:A:A:A:B:insA^3 (**fix** ...)

Problematic Example

```
insA = fold₁ (λz.λt.A:z:t)
```

$\textbf{fix}\ (\lambda x.B{:}\mathrm{insA}\ x)$

$\to^*\ B{:}\mathrm{insA}\ (\textbf{fix}\ \ldots)$

$\to^*\ B{:}\mathrm{insA}\ (B{:}\mathrm{insA}\ (\textbf{fix}\ \ldots))$

$\to^*\ B{:}A{:}B{:}\mathrm{insA}^2\ (\textbf{fix}\ \ldots)$

$\to^*\ B{:}A{:}B{:}\mathrm{insA}^2\ (B{:}\mathrm{insA}\ (\textbf{fix}\ \ldots))$

$\to^*\ B{:}A{:}B{:}A{:}A{:}A{:}B{:}\mathrm{insA}^3\ (\textbf{fix}\ \ldots)$

$\to^*\ BABA^3BA^7BA^{15}BA^{31}BA^{63}B\ldots$

Non-regular, can't be a finite graph!

Observation & Idea

- Without "**`fold`**", everything is regular
- "**`fold`** `f t`" is regular if "`t`" is a regular tree constructed ***beforehand***
  - **Ok:** `insA (`**`fix`**` (λx.B:x))`
  - **Bad:** **`fix`**` (λx.B:insA x)`

Stratify trees

to avoid traversing trees that we are constructing

Our Type System

(ignoring products)

$$\tau ::= \mathtt{Label} \mid \mathtt{G}<n> \mid \tau_1 \rightarrow \tau_2$$

trees (graphs) at generation n

$$\frac{\Gamma \vdash e : \mathtt{G}<n> \rightarrow \mathtt{G}<n>}{\Gamma \vdash \mathbf{fix}\ e : \mathtt{G}<n>}$$

$$\frac{\Gamma \vdash e : \mathtt{Label} \mathbin{\texttt{->}} \mathtt{G}<m> \rightarrow \mathtt{G}<m> \qquad n < m}{\Gamma \vdash \mathbf{fold_1}\ e : \mathtt{G}<n> \rightarrow \mathtt{G}<m>}$$

Theorems

- FUnCAL programs converted from UnCAL are well-typed also in the finer type system

- If $\vdash$ `e : G<n>`, then `e` yields a regular tree
  - its bisimilar graph is obtained by lazy evaluation with black holes
    - a variant of [Nakata&Hasegawa 09] + memoized fold

Related Work

- Graph transformation frameworks in FP based on the graph isomorphism
  - [Erwig 92, 2001, Fegaras&Sheard 96, Hamana 10, Oliveira&Cook 12, ...]
  - Different "graphs"
    - Ours are based on the graph bisimulation

Related Work

- **srec**-like computation
  - [Nishimura&Ohori 99]
    - Framework for parallel programming
      - Basis for in OODB query [Nishimura+96]
    - **foreach**: similar but a bit weaker than **srec**
  - CoCaml [Jeannin+13]
    - Various ways to compute fixed-points
      - including memoized recursion for cyclic data
    - No formal discussions on correctness

Conclusion

- FUnCAL: functional reformulation of UnCAL
  - describes ***infinite-tree*** transformations
    - ***no special operators*** for graph manipulation
      → more ***FP-based program-manipulation*** friendly
    - ***expressive as UnCAL***
  - runs as ***terminating finite-graph*** transformation
    - by lazy evaluation with black holes [Ariola&Klop 96]
      - a variant of [Nakata&Hasegawa 09]
    - ensured by our ***type system***

Future Work

- Fusion
  - short-cut fusion [Gill+93]?
- Bidirectionalization [M&Wang 13, 15]
- Generalization of the idea
  - for cyclic data structures in general
    - like [Hamana 2016]?
  - for ordered trees
- More finer & general type system