

# Bidirectional Graph Transformation using UnCAL

Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, Keisuke  
Nakano

National Institute of Informatics  
The University of Electro-Communications

Dec. 17 2008

GRACE-bx08

# In this talk

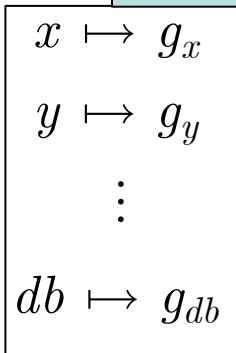
## what's unique in our framework

- Bidirectional transformation in the presence of **cycles**
  - Thanks to unique semantics of original **UnCAL**
  - Compositional
- Forward transformation
  - Preserving (part of) the result of transformation for backward computation
  - Cf. view complement
- Backward transformation
  - **Decompose** the result to proceed transformation of operands
  - Result is obtained as a set of variable binding instead of a graph
- **$\epsilon$ -edges** – artifacts in the original (unidirectional) semantics are conveniently used while backward transformation
  - Mechanism used: Transitive Closure (TC) computation on graphs
- Performance issues and optimization
  - Fusion, pruning
  - Notion of equivalence

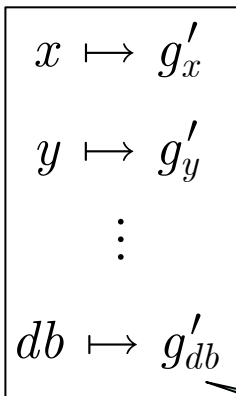
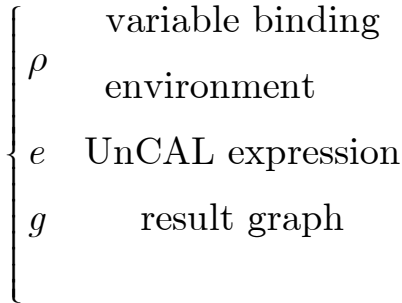
# Semantics

## General form

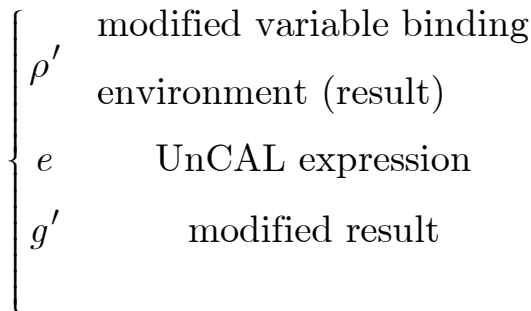
environment is a map from variable to values



$$\rho \xrightarrow{e} g$$



$$\rho' \xleftarrow[\rho]{e} g'$$



modified source

- Forward computation
  - expression  $e$  under environment  $\rho$  yields graph  $g$
- Backward computation
  - new environment  $\rho'$  is obtained by the backward evaluation of  $e$  starting from graph  $g'$
  - any number of inputs are allowed! (as free variables in an expression)

# Semantics

## augmentation of the result graph $g$

- Extend the AST to host the result of evaluation
- The value is used during backward evaluation (especially for decomposing)

$$\rho \xrightarrow{e} e^g$$

$$\rho' \xleftarrow[\rho]{e^g} g'$$

# Empty tree

- Modification is not allowed for empty tree
- Only chance to succeed in backward transformation is that  $g$  is an empty tree

$$\frac{v = \text{newnode}() \quad g = (\{v\}, \emptyset, \emptyset, \emptyset)}{\{\} \quad \rho \longrightarrow g}$$

$$\frac{g = (\{v\}, \emptyset, \emptyset, \emptyset)}{\{\} \quad \rho \longleftarrow_{\rho} g}$$

# GetPut and PutGet

- Same environment  $\rho$  is obtained by unmodified graph  $g$

$$\frac{\rho \xrightarrow{e} g}{\rho \xleftarrow[\rho]{e} g} \quad [\text{GetPut}]$$

- Another forward transformation from the modified environment  $\rho'$  produces  $g'$

$$\frac{\rho' \xleftarrow[\rho]{e} g'}{\rho' \xrightarrow{e} g'} \quad [\text{PutGet}]$$

# Unary and binary operators

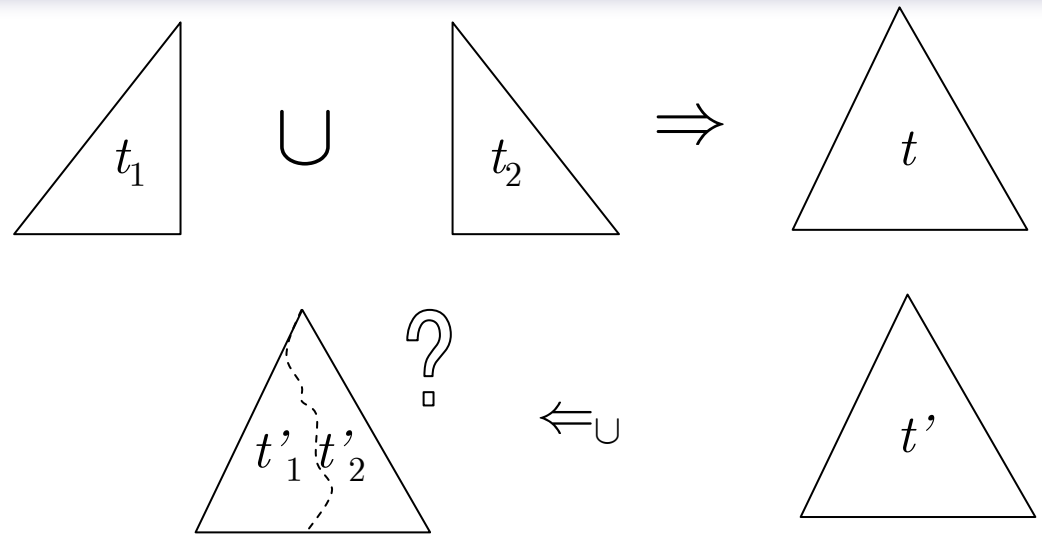
- Decompose the (possibly modified result) to feed the backward transformations of the operands

$$\frac{\rho \xrightarrow{e_1} g_1 \quad \rho \xrightarrow{e_2} g_2 \quad g_1 \odot g_2 \Rightarrow g}{\rho \xrightarrow{e_1 \odot e_2} g}$$

$$\frac{g' \Rightarrow_{\odot} (g'_1, g'_2) \quad \rho'_1 \xleftarrow{\rho}^{e_1} g'_1 \quad \rho'_2 \xleftarrow{\rho}^{e_2} g'_2}{\rho'_1 \uplus_{\odot} \rho'_2 \quad \xleftarrow{\rho}^{e_1 \odot e_2} g'}$$

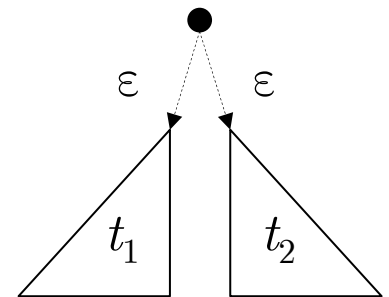
# The role of $\varepsilon$ -edges

- Remember (store) the separation points
- Example:  $\text{union}(\cup)$



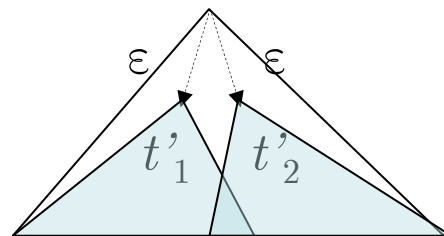
*How do you decide*

*Instead*

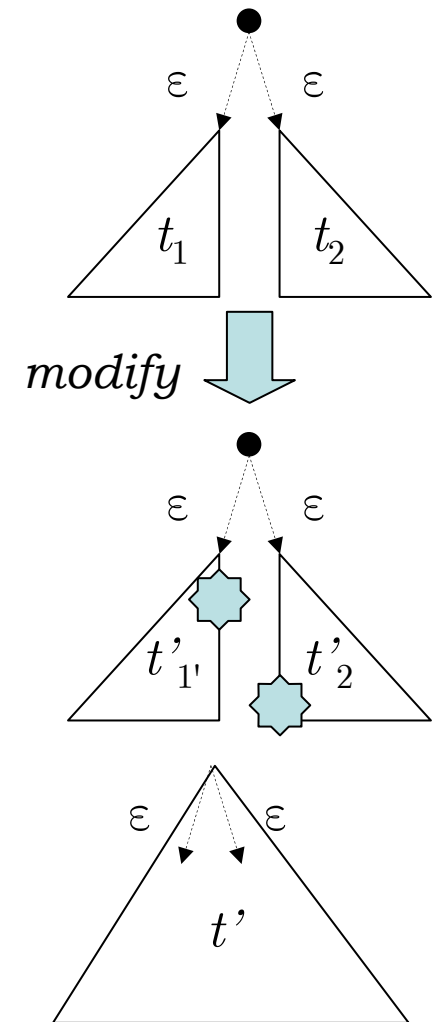


# The role of $\varepsilon$ -edges (cont.)

- Given modified tree  $t'$ 
  - Even in the presence of overlap
- Compute  $\text{reachable}(g, v)$  to grab reachable parts of the graph  $g$  from node
- and apply backward transformation recursively



$\Leftarrow \cup$



# Variable reference

## Propagation of modification (1)

- Forward: whatever  $\rho$  says
- Backward:  $\rho$  with the binding of  $v$  replaced by modified graph  $g'$
- Remark: Any modification is permitted for expression  $\$v$

$$\rho \xrightarrow{\$v} \rho(\$v)$$

$$\rho[\$v \mapsto g'] \xleftarrow[\rho]{\$v} g'$$

# Let expression

## Propagation of modification (2)

- Forward: trivial
- Backward:
  - $\rho$  extended with the result  $g_1$  is used to transform the body backwards
  - pick the new binding of  $v$  and use it and apply backward transformation of  $e_1$

$$\frac{\rho \xrightarrow{e_1} g_1 \quad \rho[v \mapsto g_1] \xrightarrow{e_2} g_2}{\rho \xrightarrow{\text{let } \$v = e_1 \text{ in } e_2} g_2}$$

- Remark: variable bindings works as a medium of modification propagation

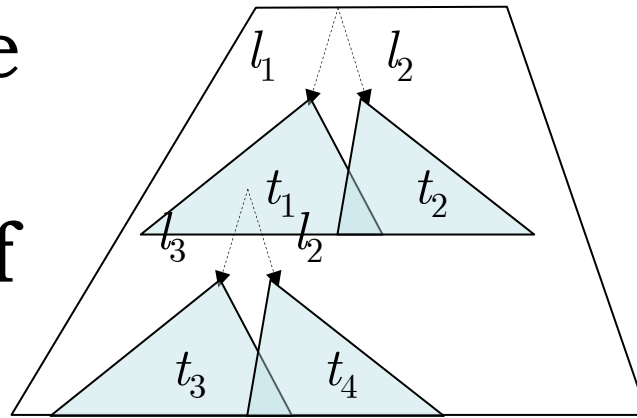
$$\frac{\rho_1 = \rho[v \mapsto g_1] \quad \rho'_1 \xrightarrow{\rho_1} g' \quad \rho' \xrightarrow{\rho} \rho'_1(v)}{\rho' \uplus (\rho_1 \setminus [v \mapsto g'']) \xrightarrow{\rho} (\text{let } \$v = e_1^{g_1} \text{ in } e_2^{g_2})^g g'}$$

# Structural recursion relying on the bulk semantics

$$\begin{array}{c}
 g' \Rightarrow g'_1, \dots, g'_n \quad \rho(v) \Rightarrow g_1, \dots, g_n \\
 \rho'_i \xleftarrow{e} \rho[\{l:g\} \leftarrow g_i] \quad g'_i \\
 \rho' = \text{mergeEnv}_\rho(\rho'_1, \dots, \rho'_n) \\
 g'' = \text{merge}(\{\rho_1(l) : \rho_1(g)\}, \dots, \{\rho_n(l) : \rho_n(g)\}) \\
 \rho'' = \rho'[v \leftarrow g''] \\
 \hline
 \rho'' \xleftarrow{\text{rec}(\lambda(l,g).e)(v)} \rho \quad g'
 \end{array}
 \quad (\text{BWD\_REC})$$

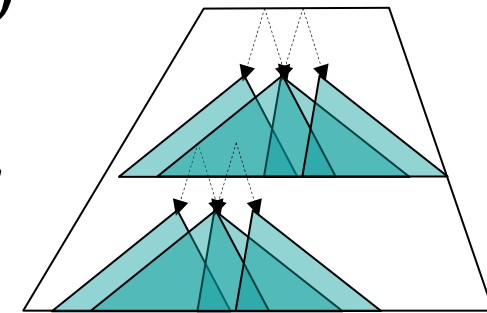
# Structural recursion relying on the bulk semantics

- Preserve every result of  $e(l_i, t_j)$

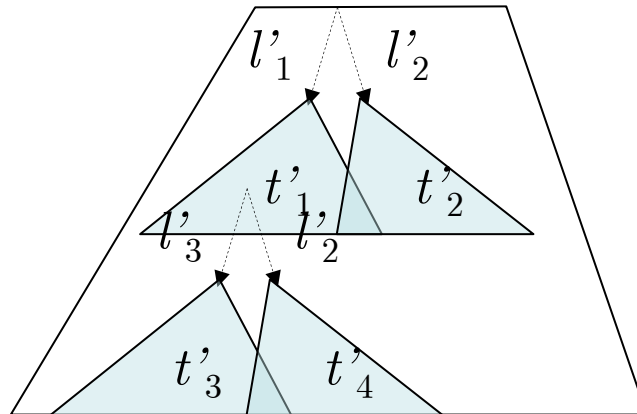


$$e(l_i, t_j)$$

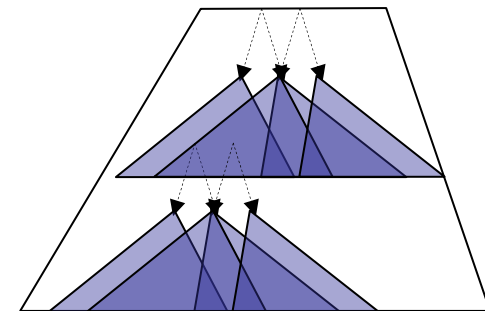
$$\rho \xrightarrow{e} g$$



modify 



$$\rho'_i \xleftarrow[\rho_i]{e_i} g'_i$$



# Fusion rules

$$\text{rec}(e_2) \circ \text{rec}(e_1) = \text{rec}(\text{rec}(e_2)) \circ e_1$$

**if  $e_2(l, t)$  does not depend on  $t$**

$$\begin{aligned} &\text{rec}(e_2) \circ \text{rec}(e_1) \\ &= \text{rec}(\lambda(l, t). \text{rec}(e_2)(e_1(l, t) @ \text{rec}(e_1)(t))) \end{aligned}$$

**for arbitrary  $e_2(l, t)$**

# Conclusion

- Graph algebra UnCAL is bidirectionalized
- Forward transformation: environment to graph
  - Preserves (part of) the result of transformation
- Backward transformation: modified graph to modified environment
  - Decompose the result using  $\epsilon$ -edges and TC computation
- Demonstration available at <http://www.biglab.org>
- Performance issues and optimization
  - Fusion, pruning
  - Notion of equivalence