A Compositional Approach to Bidirectional Model Transformation

Bidirectional Computation for Software Engineering

Zhenjiang Hu

GRACE Center
National Institute of Informatics

March 6, 2009

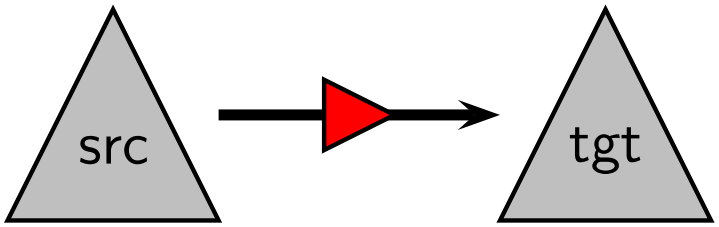
Joint Work with Soichiro Hidaka, Hiroyuki Kato and Keisuke Nakano

How do you synchronize your calenders?

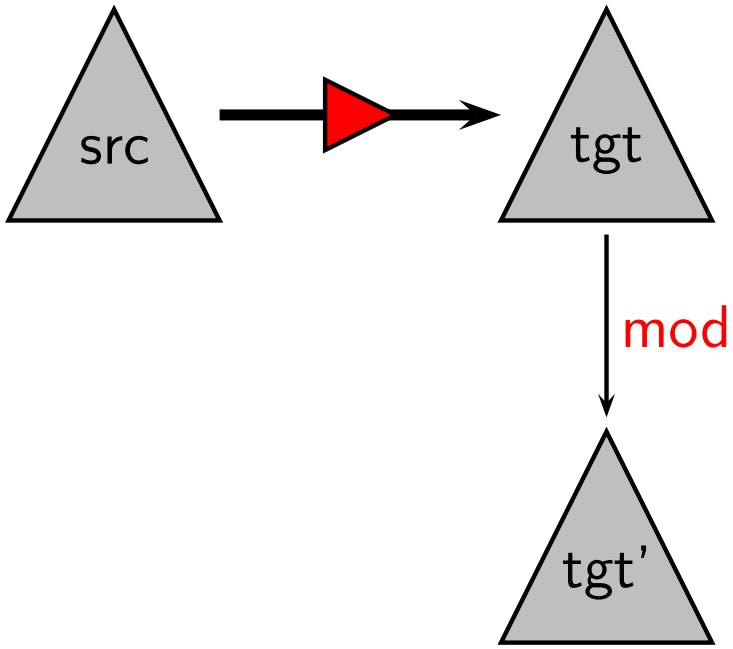
Bidirectional Computation (Bidirectional Transformation)

Bidirectional Computation (Bidirectional Transformation)

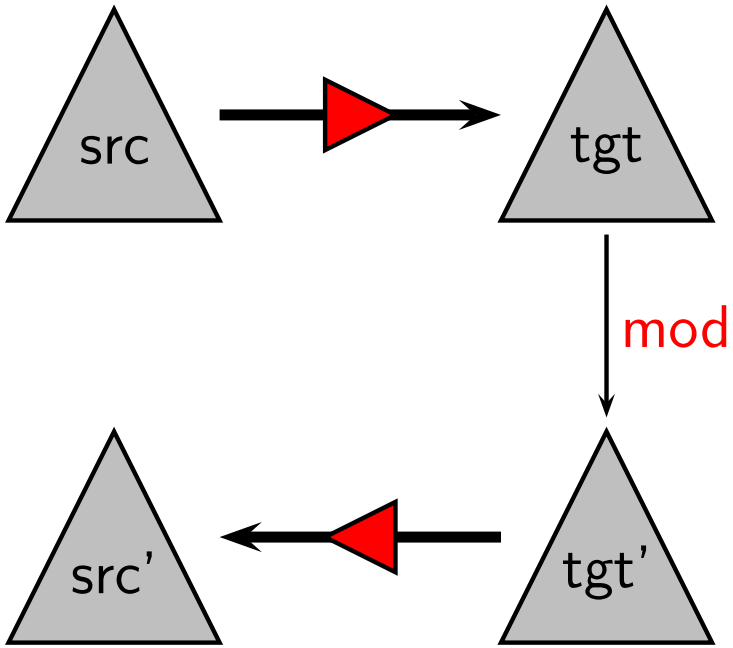
Bidirectional Computation (Bidirectional Transformation)



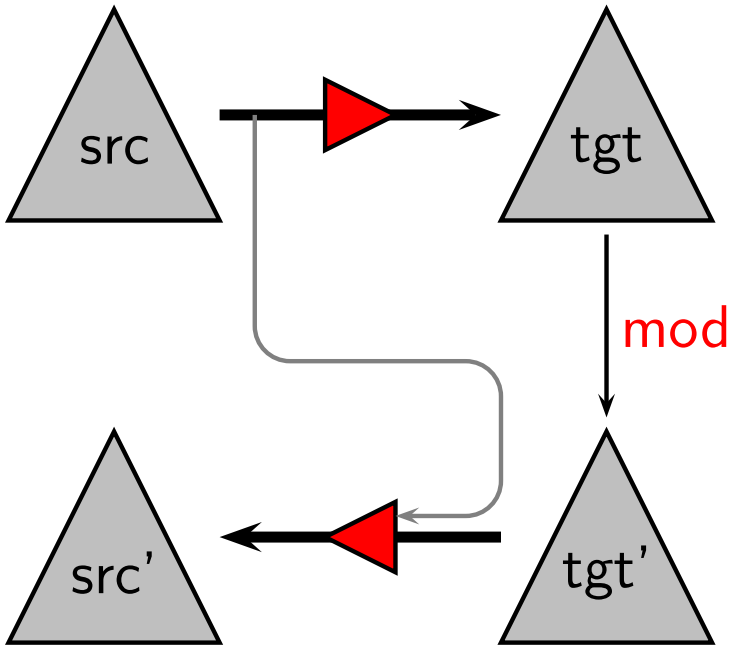
Bidirectional Computation (Bidirectional Transformation)



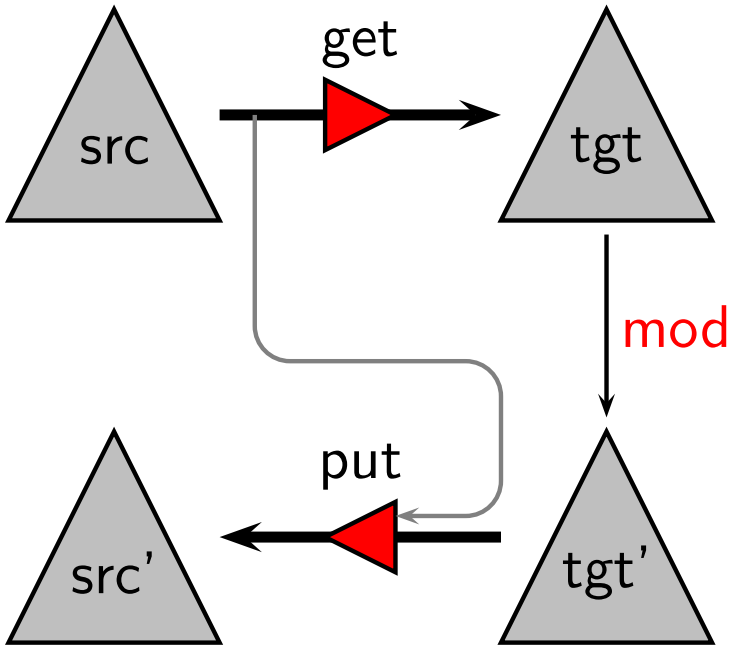
Bidirectional Computation (Bidirectional Transformation)



Bidirectional Computation (Bidirectional Transformation)



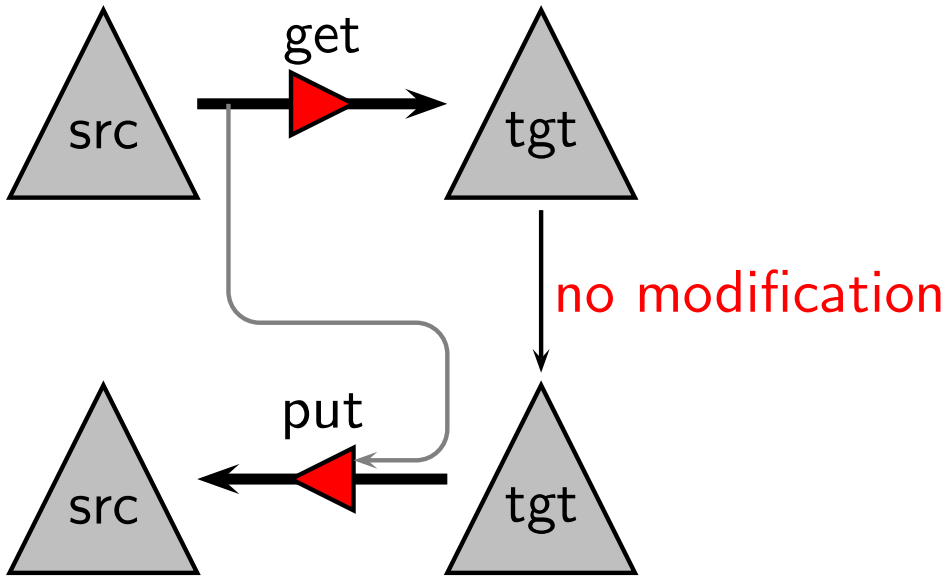
Bidirectional Computation (Bidirectional Transformation)



It consists of a pair of computation forward and backward.

Stability

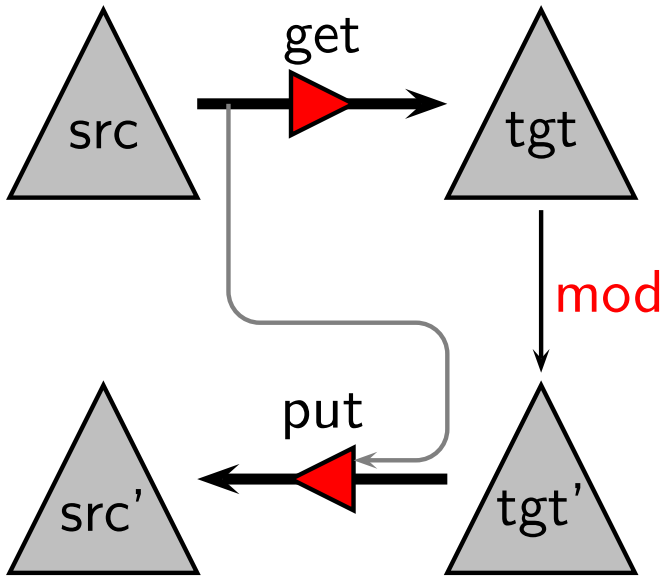
No change on the target implies no change on the source.



$$\mathrm{put}(\mathrm{get}(s), s) = s$$

Reflectivity

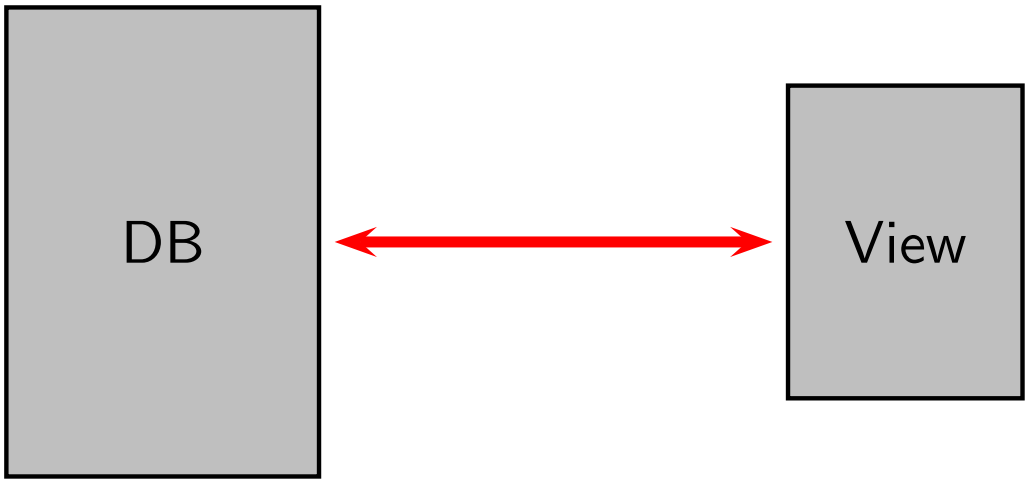
Permitted changes on the target should be reflected to the source.



$$\mathrm{get}(\mathrm{put}(t', s)) = t'$$

View Updating

Reflect changes on the view to the original relational database.



Ref: many studies in the database community

[Bancilhon&Spyratos:81, Dayal&Bernstein, Gottlob et. al.:88]

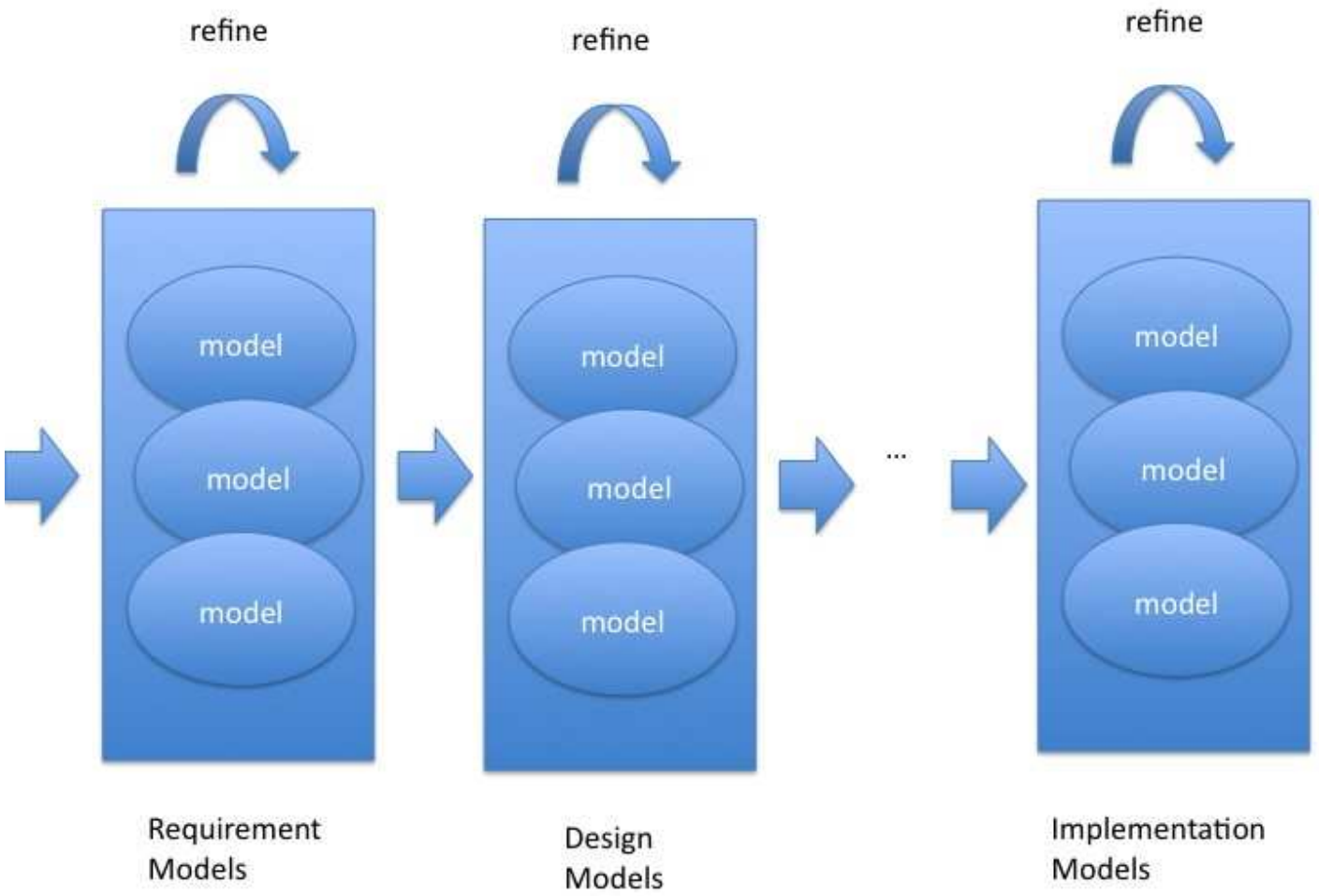
Replicated Data Synchronization

Synchronization of data in different formats.

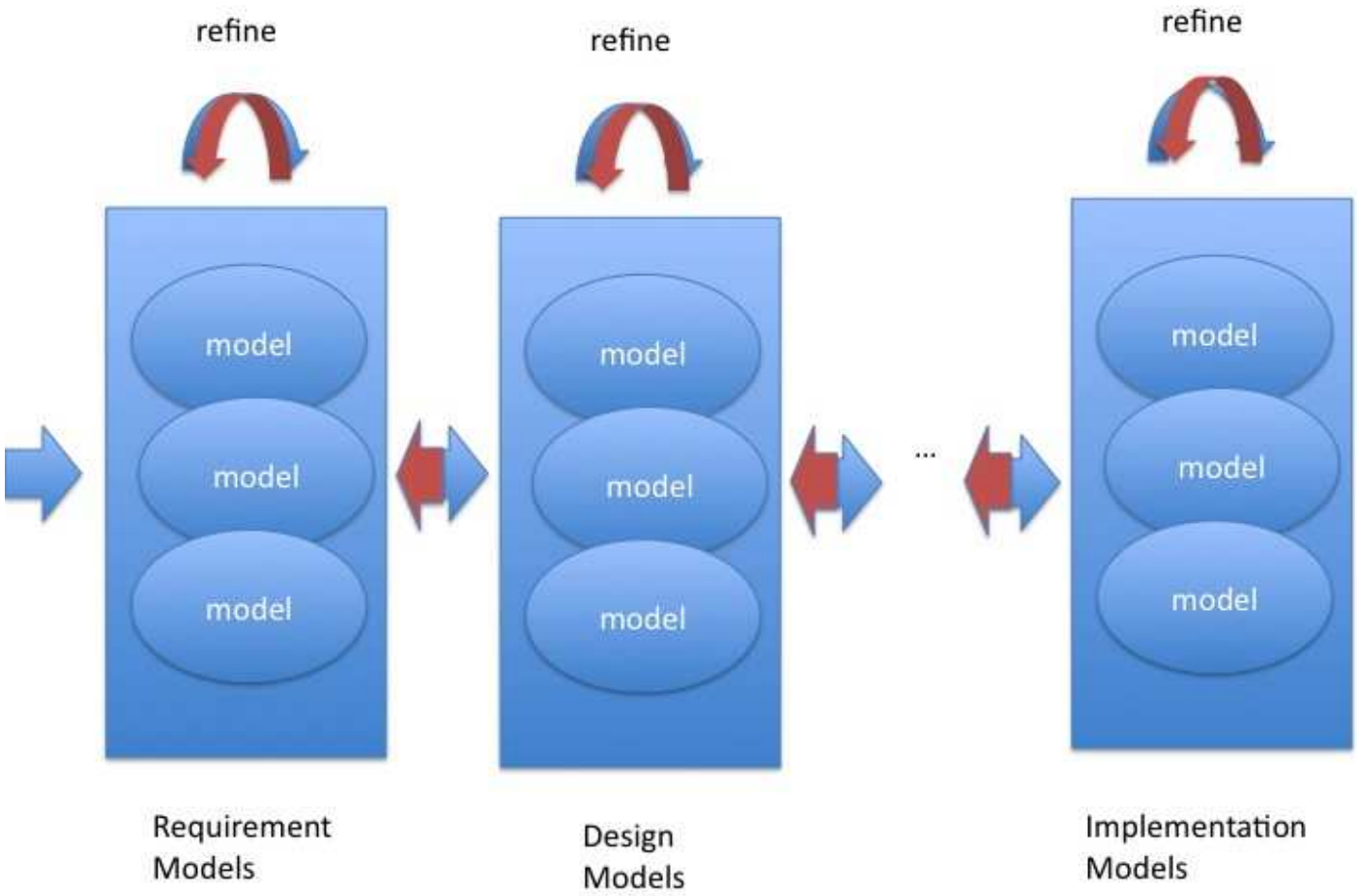
Ref: the Harmony project in Univ. of Pennsylvania

[Pierce et. al: POPL'05, PODS'06, POPL'08].

Bidirectional Computation is WANTED in SE



Bidirectional Computation is WANTED in SE



Challenges of BC for SE

- Software Artifacts: Graphs
- Software Development: Graph Transformations

Challenges of BC for SE

- Software Artifacts: Graphs
- Software Development: Graph Transformations

Perdita Stevens: **Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions**, Best Paper in MoDELS 2007.

> *It is a big challenge to design a powerful language for specifying graph computation with clear bidirectional semantics that can support practical software development.*

BiX: A Bidirectional XML Transformation Language

BiX: A Bidirectional XML Transformation Language

- Inv: an injective language for revertible computation

  [MPC'04, APLAS'04]

- X: a domain-specific bidirectional language for interactive document construction (with the dup primitive)

  [ACM PEPM'04, HOCS:08]

- BiX: a bidirectional transformation language for general XML processing (with function definitions and local bindings)

  [ACM PEPM'07]

- ... a general bidirectional functional language [ACM ICFP'07]

BiX: A Bidirectional XML Transformation Language

- Inv: an injective language for revertible computation

  [MPC'04, APLAS'04]

- X: a domain-specific bidirectional language for interactive document construction (with the dup primitive)

  [ACM PEPM'04, HOCS:08]

- BiX: a bidirectional transformation language for general XML processing (with function definitions and local bindings)

  [ACM PEPM'07]

- ... a general bidirectional functional language [ACM ICFP'07]

Write forward transformation and get backward transformation for free!

BiX

- A Combinator-based language

BiX

- A Combinator-based language
  - Primitive bidirectional transformations for tree manipulation, which is extensible.

BiX

- A Combinator-based language
  - Primitive bidirectional transformations for tree manipulation, which is extensible.
  - Combinators for composing smaller bidirectional transformations

BiX

- A Combinator-based language
  - Primitive bidirectional transformations for tree manipulation, which is extensible.
  - Combinators for composing smaller bidirectional transformations

We only need to prepare a pair of transformations for each primitive bidirectional transformation, other backward transformations are obtained for free.

Primitive Bi-Transformations

Primitive bidirectional transformations for tree manipulation:

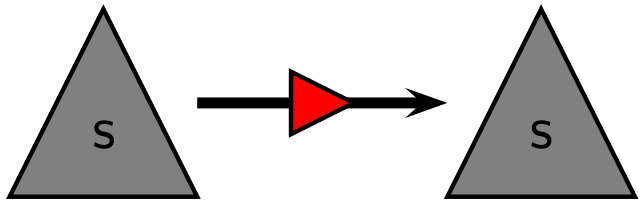
- for tree destruction, e.g.,
  - *xid*
  - *xleftchild*
- for tree construction, e.g.,
  - *xconst t*
  - *xdup*
  - *xnewroot n*

xid

- Identity Transformation

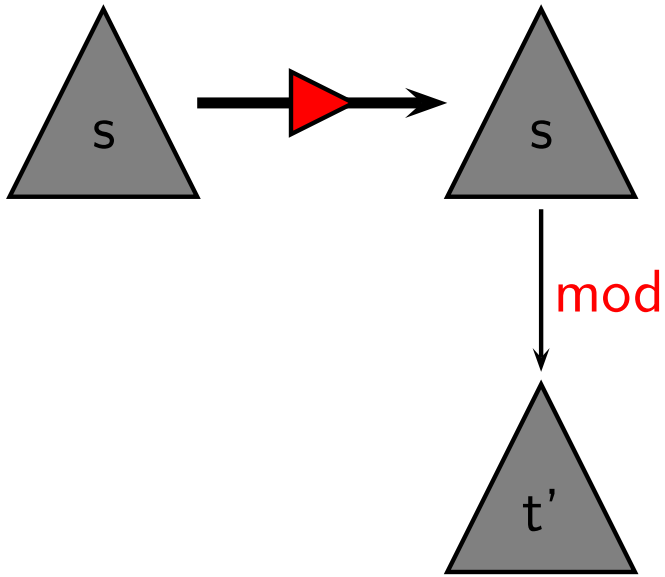
xid

- Identity Transformation



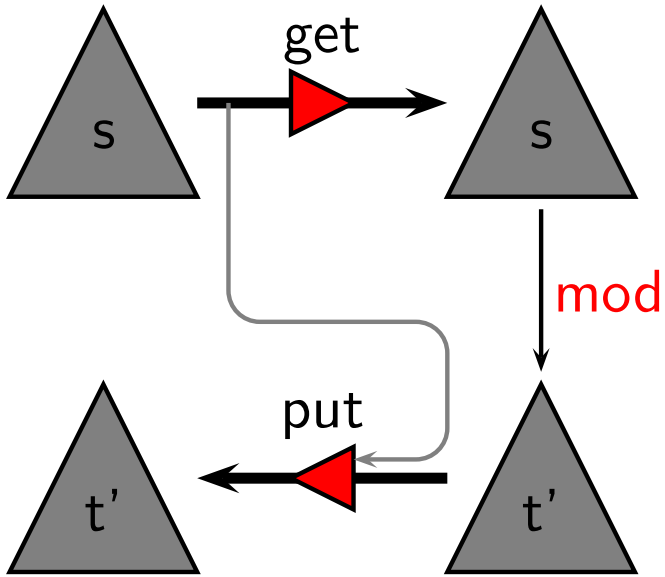
xid

- Identity Transformation



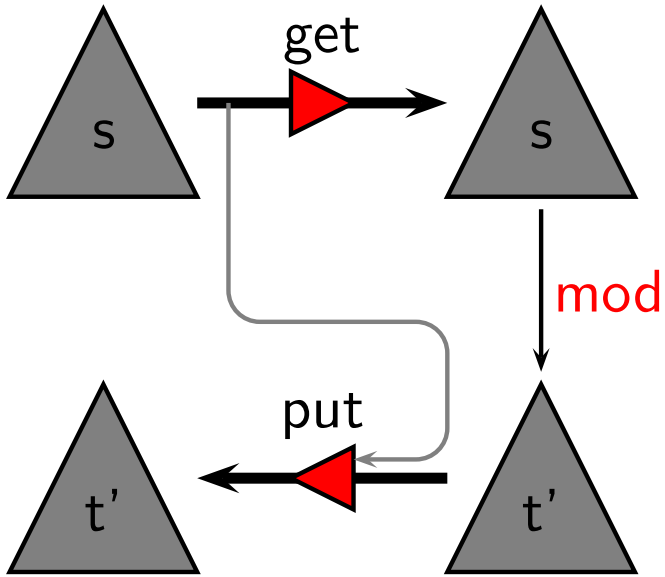
xid

- Identity Transformation



xid

- Identity Transformation



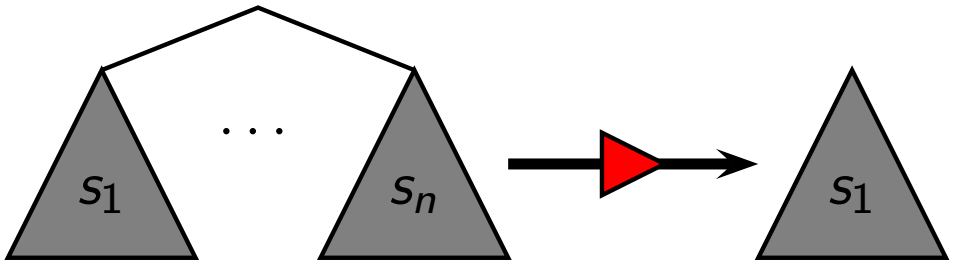
$$\begin{aligned} \text{get } s &= s \\ \text{put } t'\ s &= t' \end{aligned}$$

xleftchild

- Select the leftmost child of the root

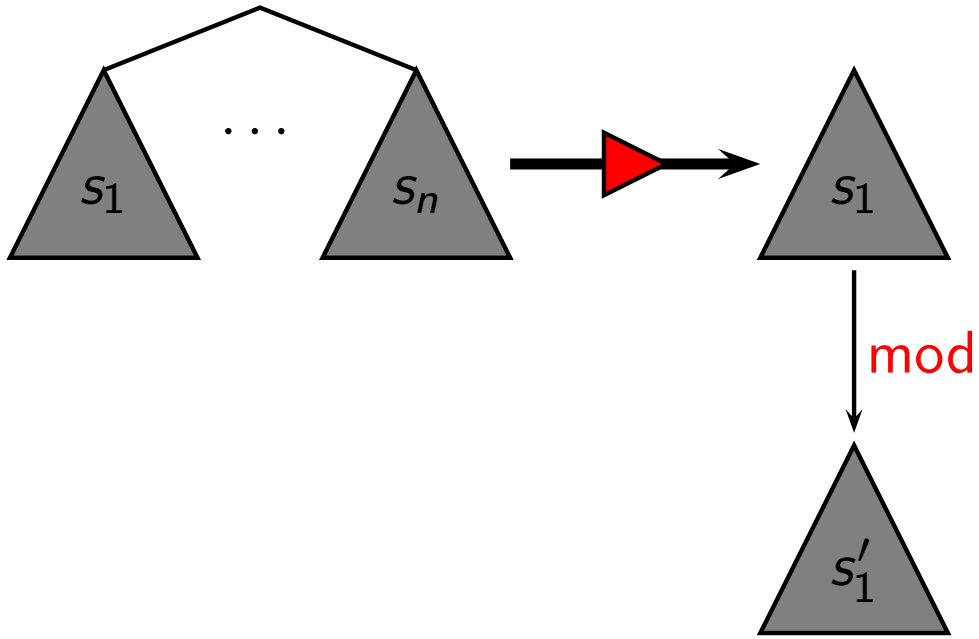
xleftchild

- Select the leftmost child of the root



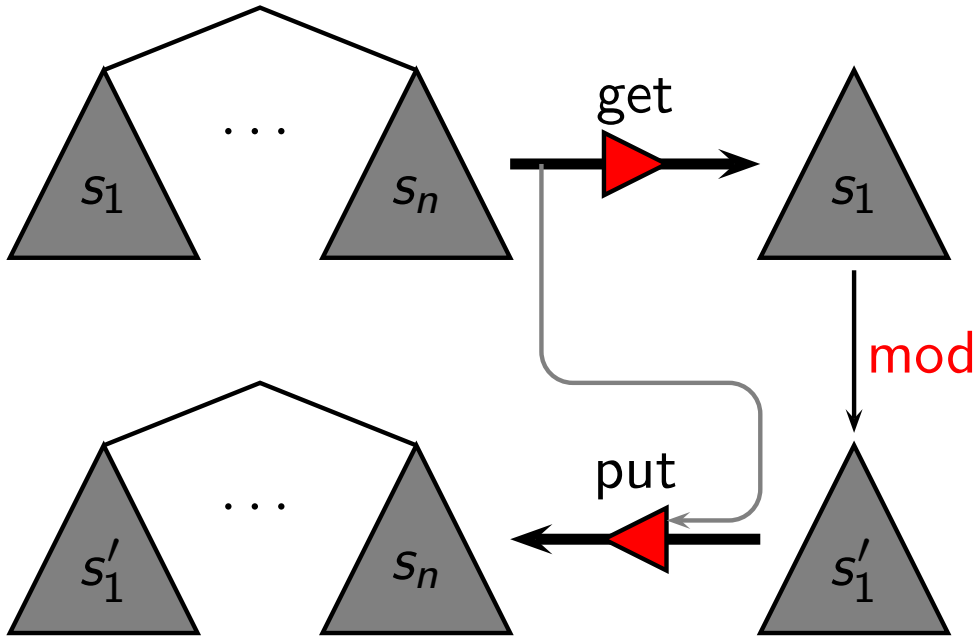
xleftchild

- Select the leftmost child of the root



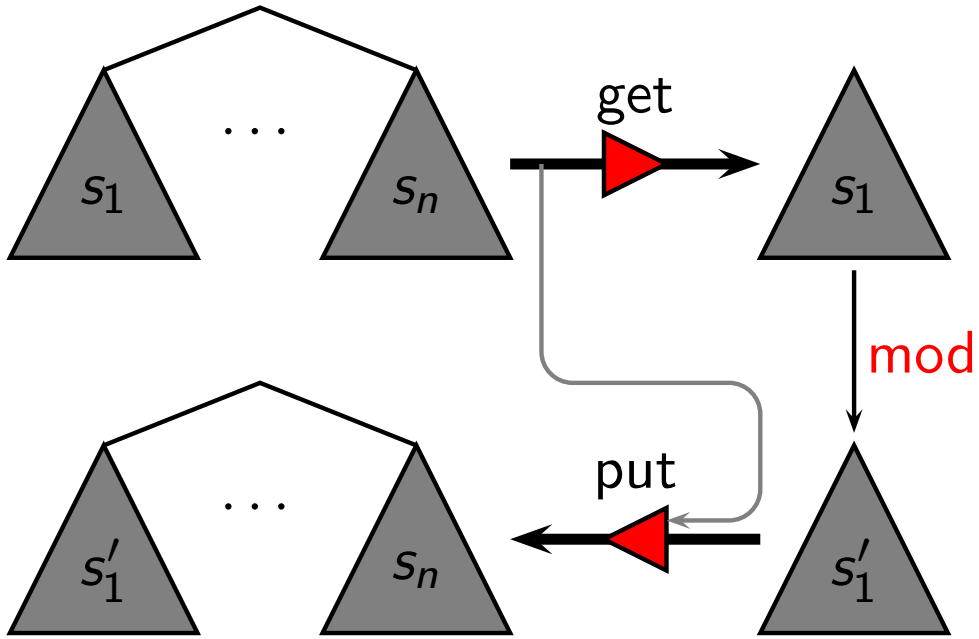
xleftchild

- Select the leftmost child of the root



xleftchild

- Select the leftmost child of the root



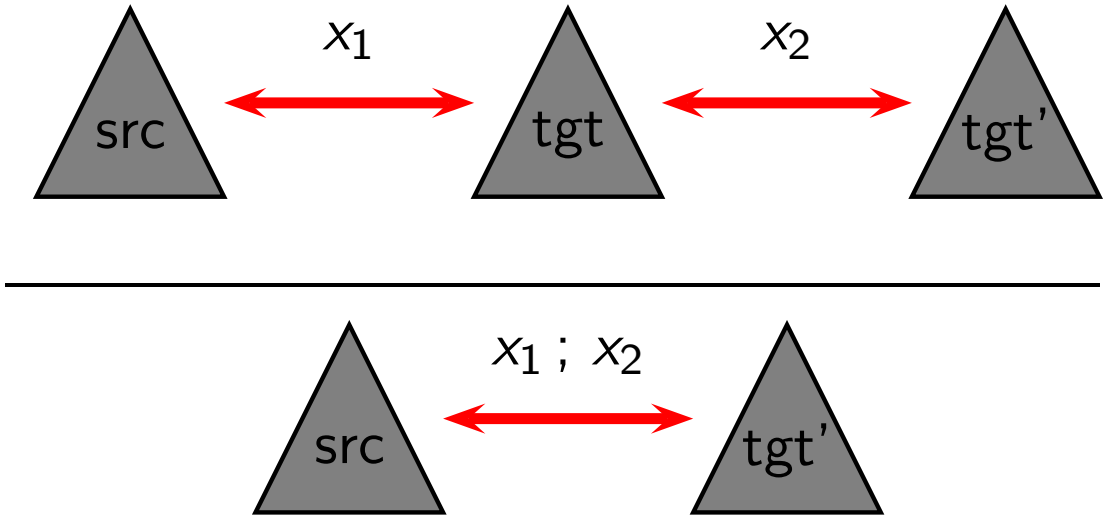
$$\text{get}\ (\mathit{Node}\ n\ [s_1, \ldots, s_n]) = s_1$$
$$\text{put}\ s'_1\ (\mathit{Node}\ n\ [s_1, \ldots, s_n]) = \mathit{Node}\ n\ [s'_1, \ldots, s_n]$$

Combinators

Used to construct bigger bidirectional transformations by composing smaller bidirectional transformations.

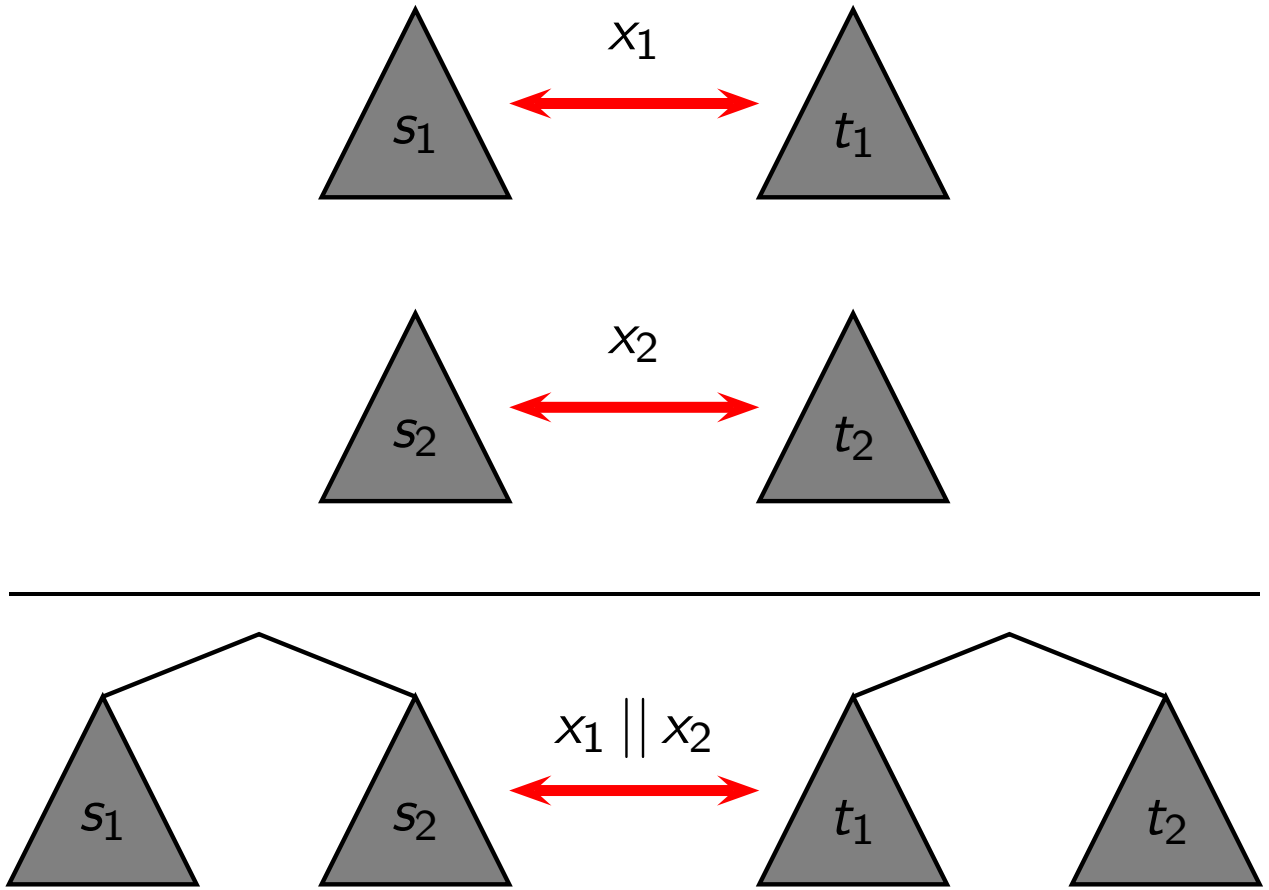
- Sequential composition: $x_1 \; ; \; x_2$
- Parallel composition: $x_1 \; || \; x_2$
- Map to each: xmap x
- Conditional: xif $p \; x_1 \; x_2$

Sequential Composition

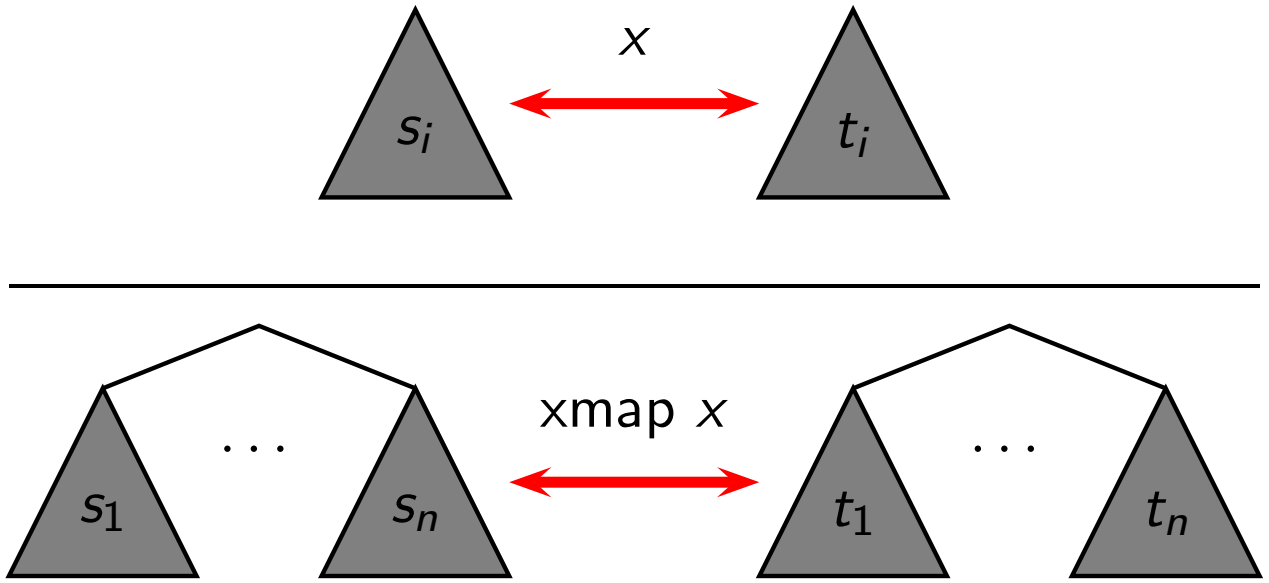


$$\begin{aligned} \text{get}_{(x_1\,;\,x_2)}\ s &= \text{get}_{x_2}(\text{get}_{x_1}\ s) \\ \text{put}_{(x_1\,;\,x_2)}\ t'\ s &= \text{put}_{x_1}\ (\text{put}_{x_2}\ t'\ (\text{get}_{x_1}\ s))\ s \end{aligned}$$

Parallel Composition

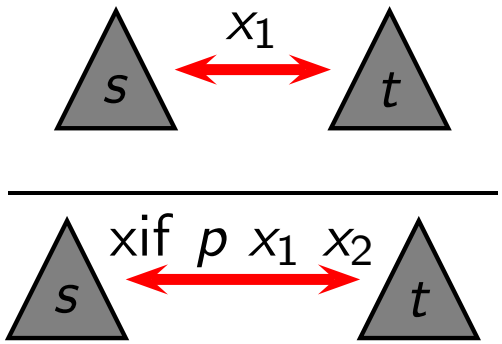


Map-to-each

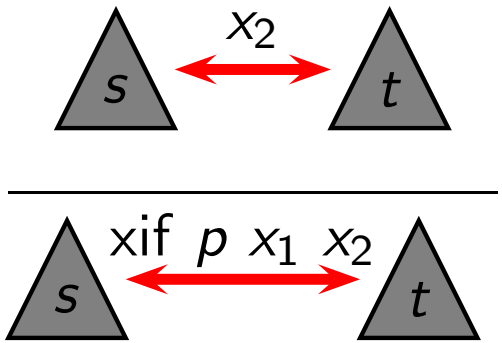


Conditional

If $p\ s$ holds, then



otherwise,



How Powerful is BiX?

- The core XSLT can be bidirectionalized with BiX [JSSST CS'07]
- The core XQuery can be bidirectionalized with BiX [ACM PEPM'07]

http://www.ipl.t.u-tokyo.ac.jp/~liu/BiXQuery.html

Application: Web Site Maintenance

Vu-X: update web pages on browsers in the WYSIWYG manner.

http://www.psdlab.org/vux/

Can we define a combinator language for bidirectional model (Graph) transformation?

Can we define a combinator language for bidirectional model (Graph) transformation?

A subset of ATL can be bidirectionalized (ASE 2007)!

Can we define a combinator language for bidirectional model (Graph) transformation?

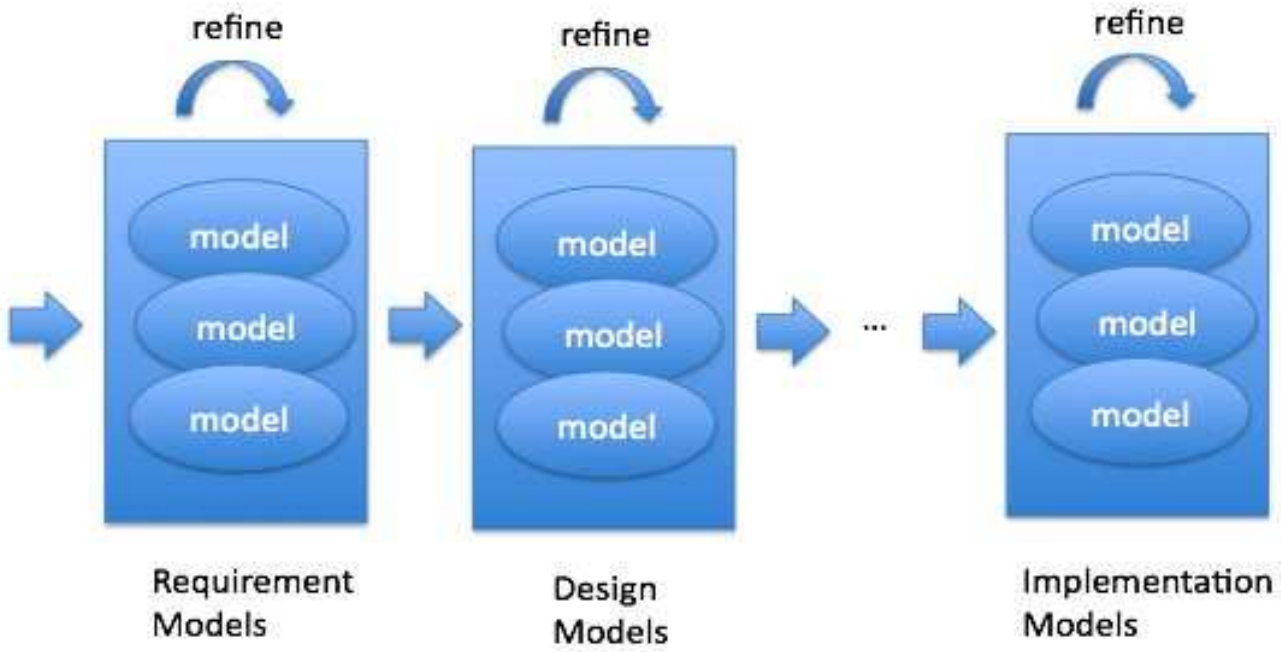
$\Downarrow$

A subset of ATL can be bidirectionalized (ASE 2007)!

A General Compositional Framework for
Bidirectional Graph Transformations (SAC'09, NIER Track of ICSE'09)

Model-Driven Software Development

- Software Artifacts: Models
- Software Development: Model Transformations

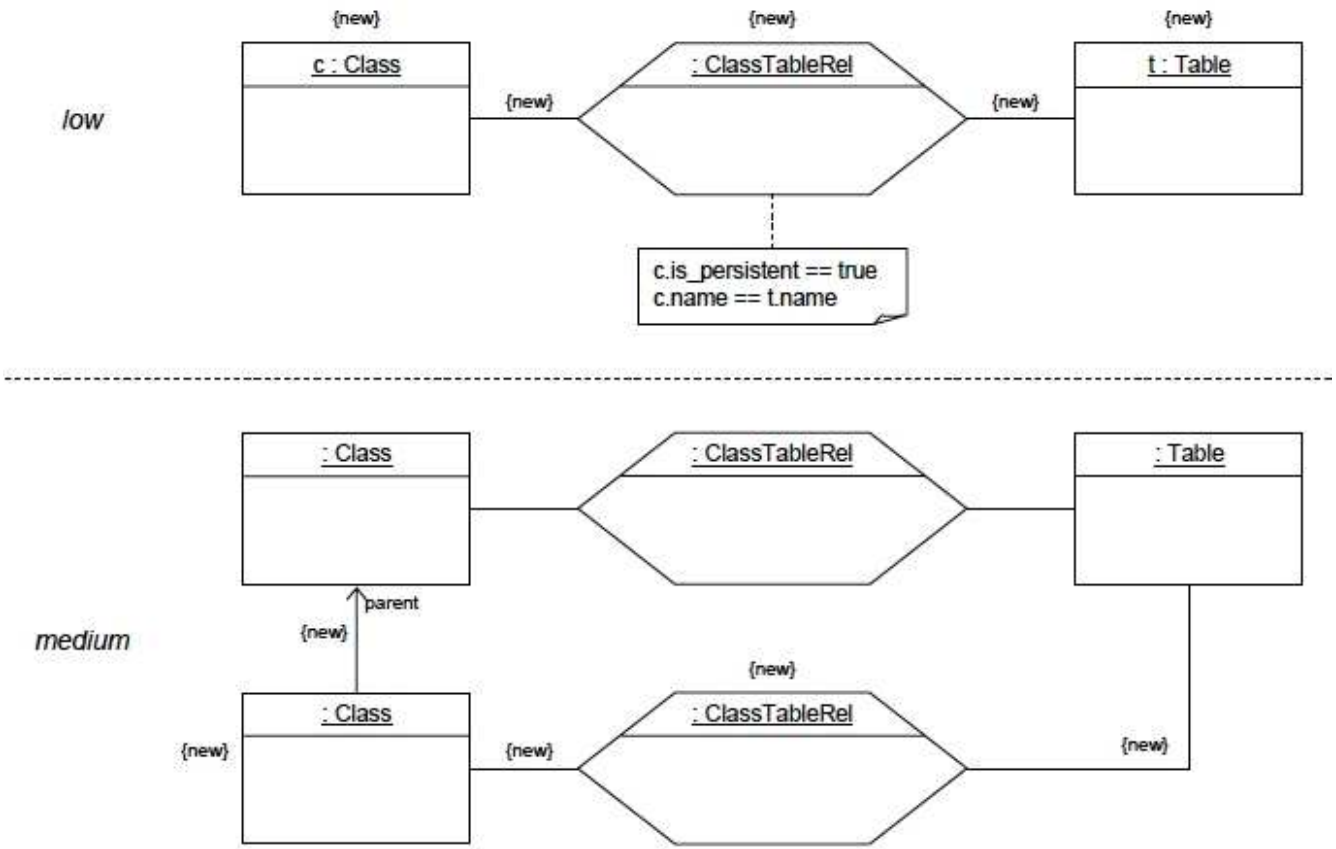


Model Transformations

- Graph-based Approach:
  - Models: Graphs
  - Model Transformations: Graph Transformations
- Existing Frameworks:
  - AGG: attributed graph grammars
  - TGG: triple graph grammars
  - QVT/ATL: Query/View/Transformation

The TGG/QVT Approach

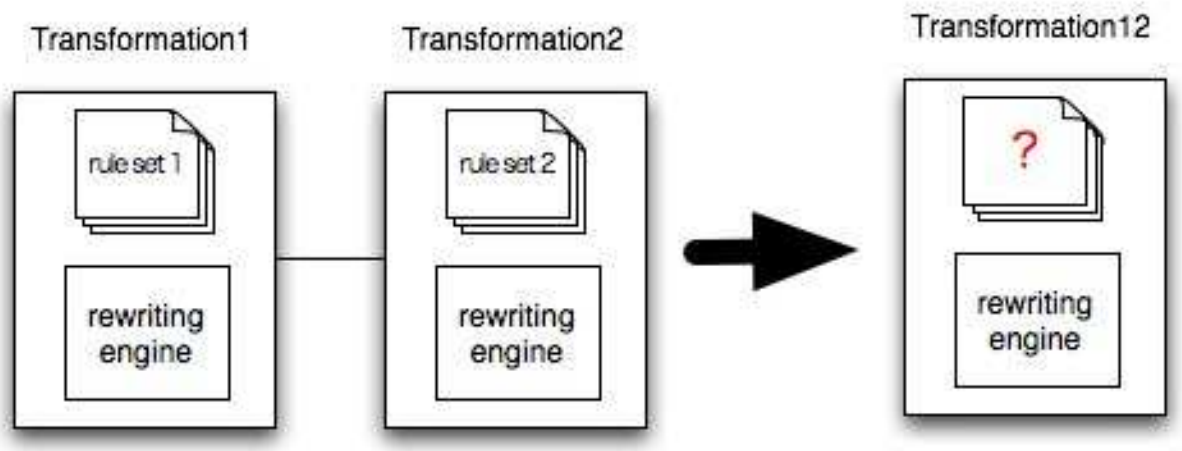
Triple Graphs: source graph + link graph + target graph



Graph Transformation Composition

The survey paper [Ehrig et al. 2005] says:

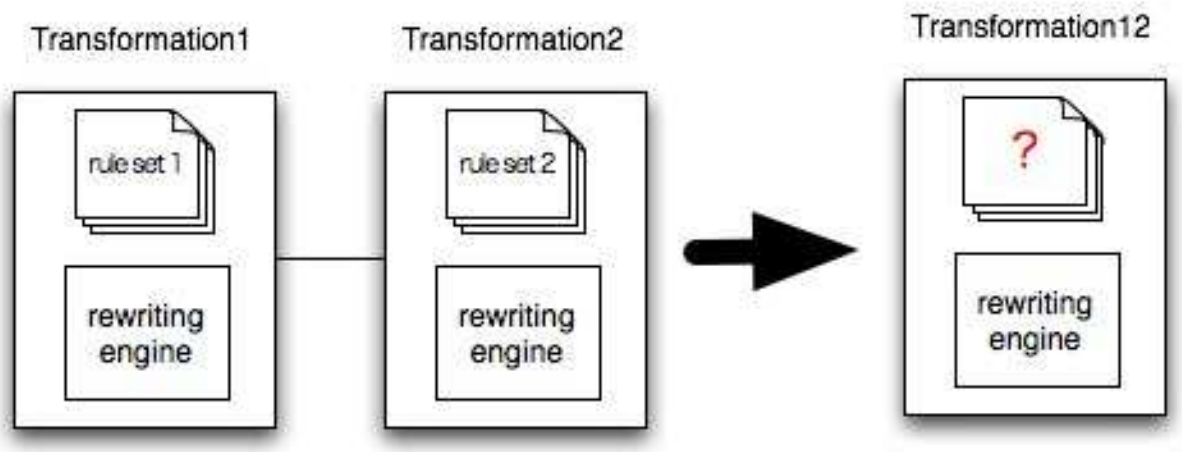
> *Open issues for all graph transformation approaches are elaborated concepts to compose transformations ...*



Graph Transformation Composition

The survey paper [Ehrig et al. 2005] says:

> *Open issues for all graph transformation approaches are elaborated concepts to compose transformations ...*



Composition is WANTED for systematic development of model transformation in the large [Klar et al: FSE07] besides bidirectionalization.

UnQL: Graph Querying

UnQL [Buneman et al. 2000] is a

compositional framework for graph querying

- Easy to use: select ... where ...
- Has a core graph algebras for arbitrary graph construction
- Use structural recursion for manipulating graphs

UnQL: Graph Querying

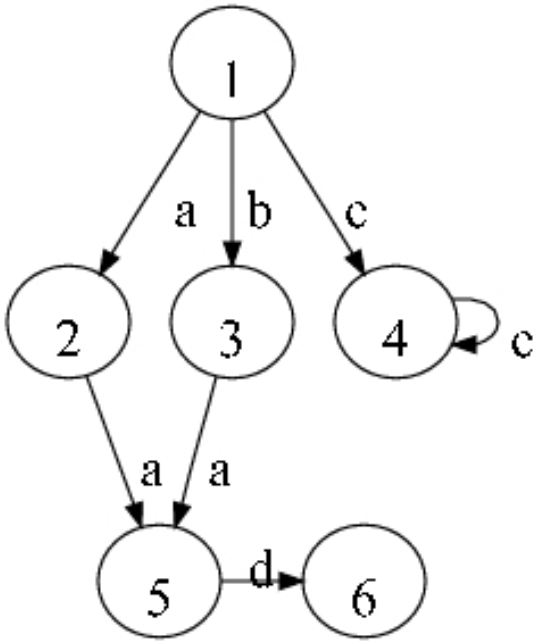
UnQL [Buneman et al. 2000] is a

compositional framework for graph querying

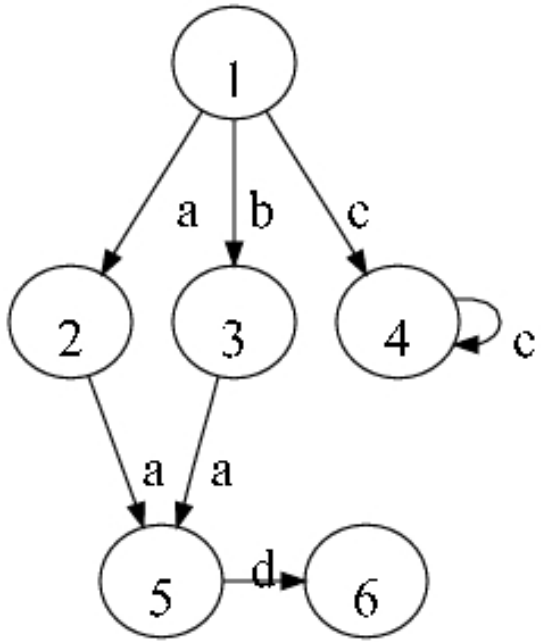
- Easy to use: select ... where ...
- Has a core graph algebras for arbitrary graph construction
- Use structural recursion for manipulating graphs

Can we extend UnQL
from graph querying to graph transformation
that can be bidirectionalized?

Edge-labelled Graphs



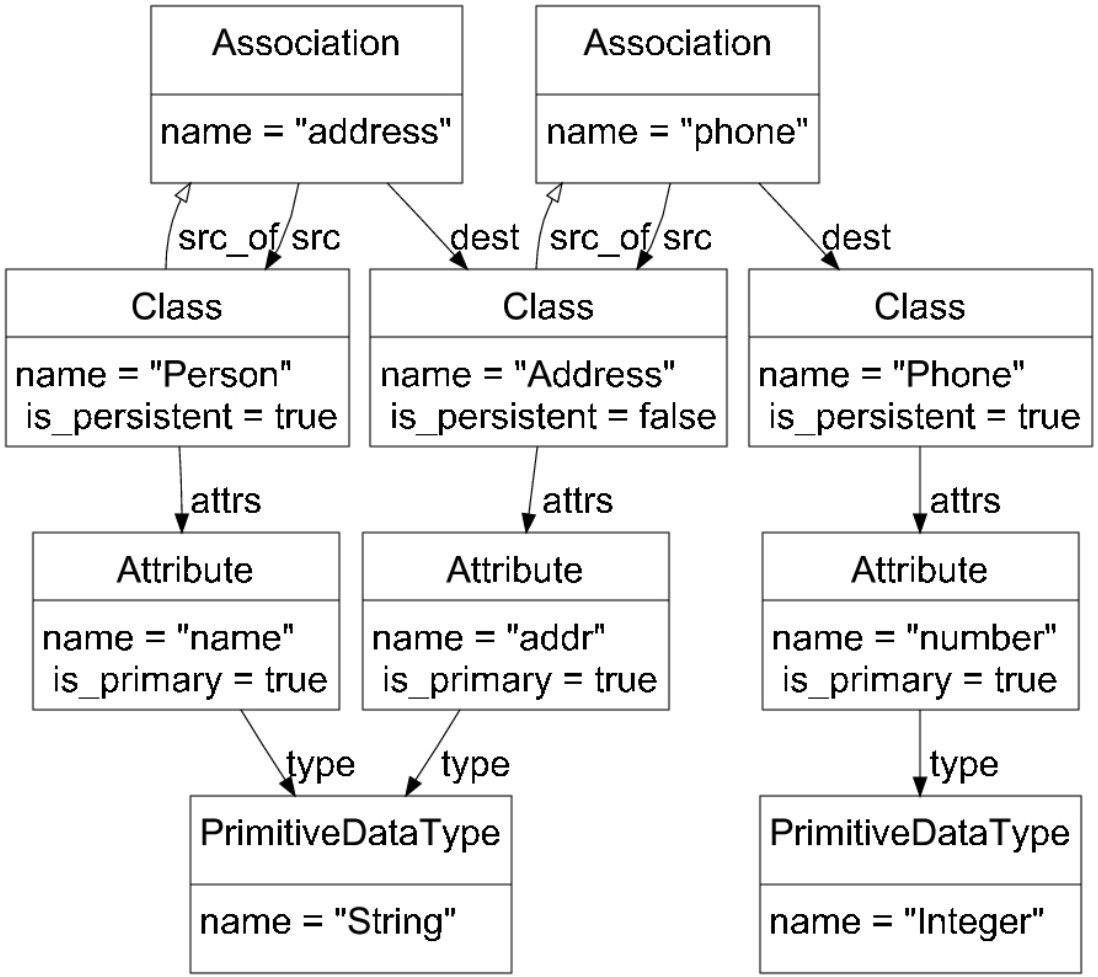
Edge-labelled Graphs



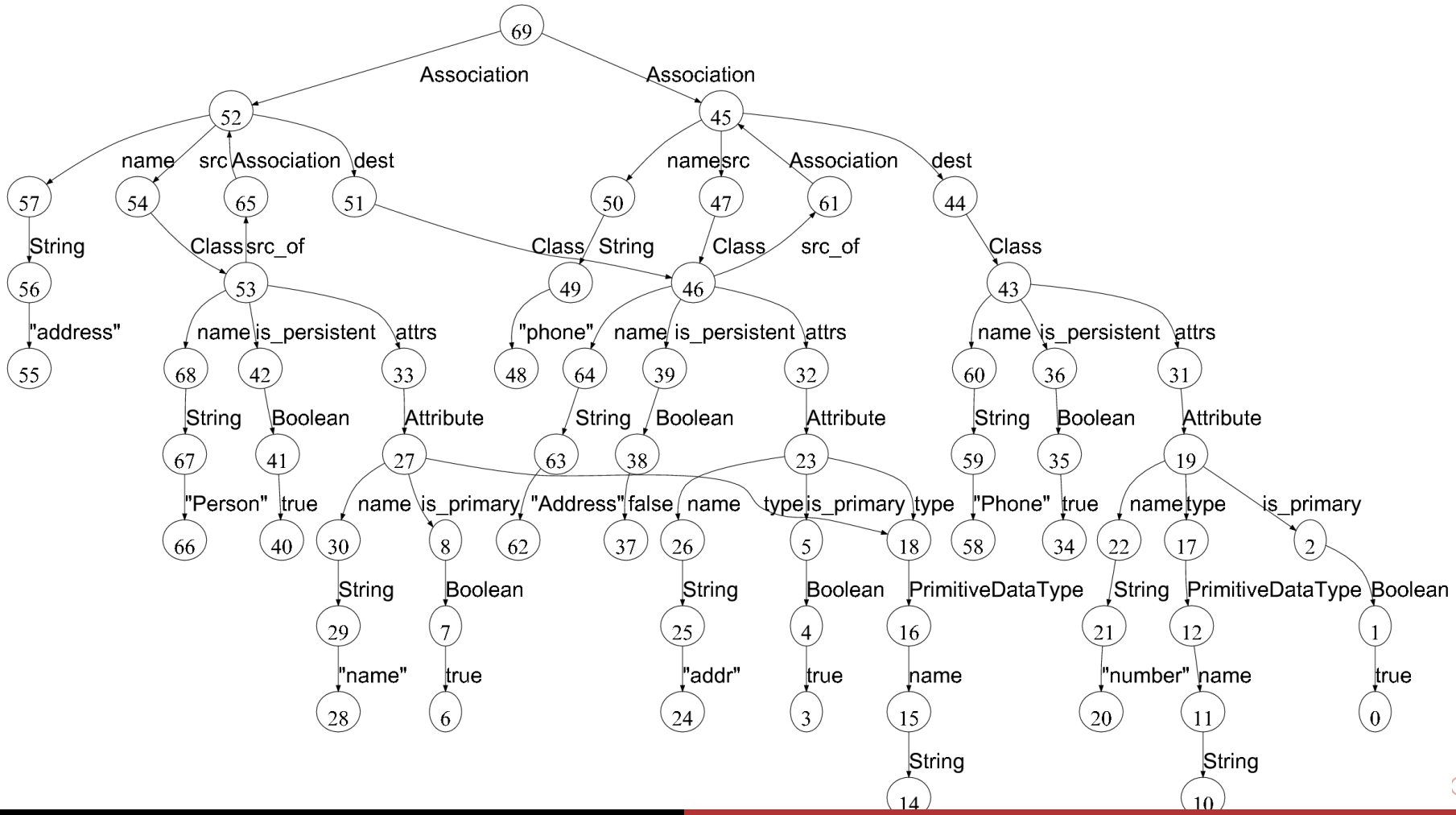
$$
\begin{aligned}
g &= \{a : \{a : g_1\}, b : \{a : g_1\}, c : g_2\} \\
g_1 &= \{d : \{\}\} \\
g_2 &= \{c : g_2\}
\end{aligned}
$$

Model Representation: An Example

A Class Diagram:



Model Representation: An Example



Structural Recursion: Manipulating Graphs

Structural Recursion:

$$\begin{aligned} f\ (\{\}) &= \{\} \\ f\ (\{l : g\}) &= l \odot f(g) \\ f\ (g_1 \cup g_2) &= f(g_1) \cup f(g_2) \end{aligned}$$

Structural Recursion: Manipulating Graphs

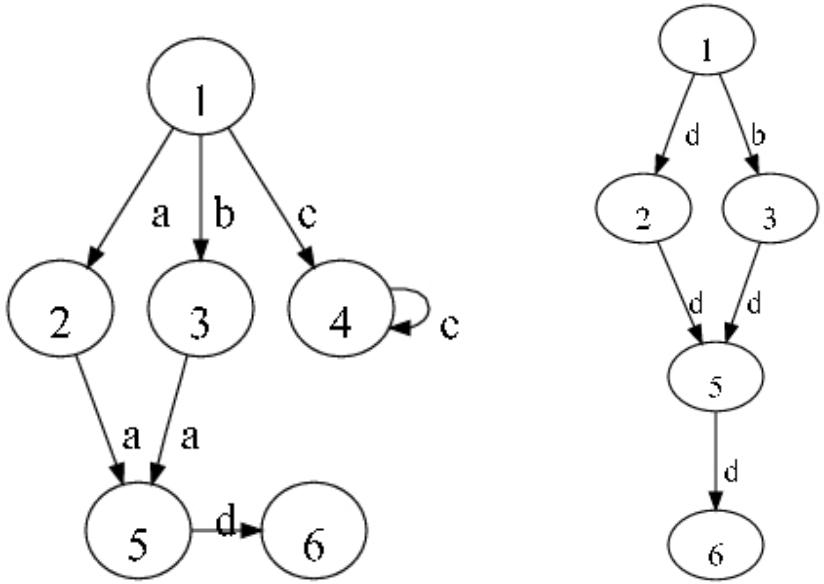
Structural Recursion:

$$\begin{array}{lcl} f\ (\{\}) & = & \{\} \\ f\ (\{l : g\}) & = & l \odot f(g) \\ f\ (g_1 \cup g_2) & = & f(g_1) \cup f(g_2) \end{array}$$

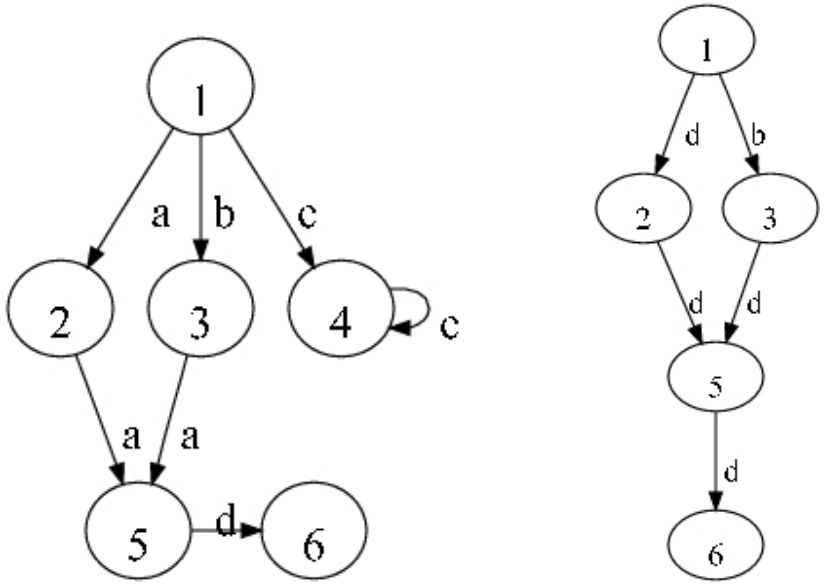
Or written as:

$$\text{sfun}\ \ f\ (\{l : g\}) = l \odot f(g)$$

An Example

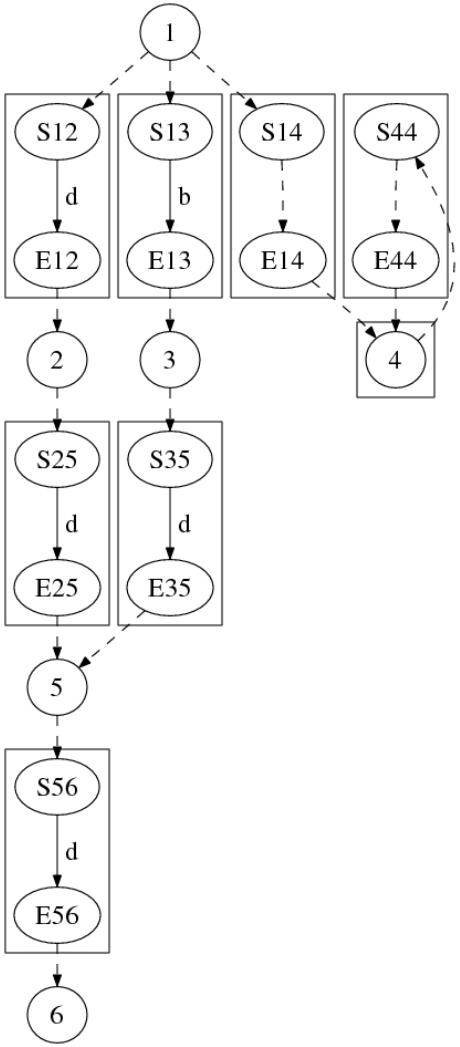


An Example

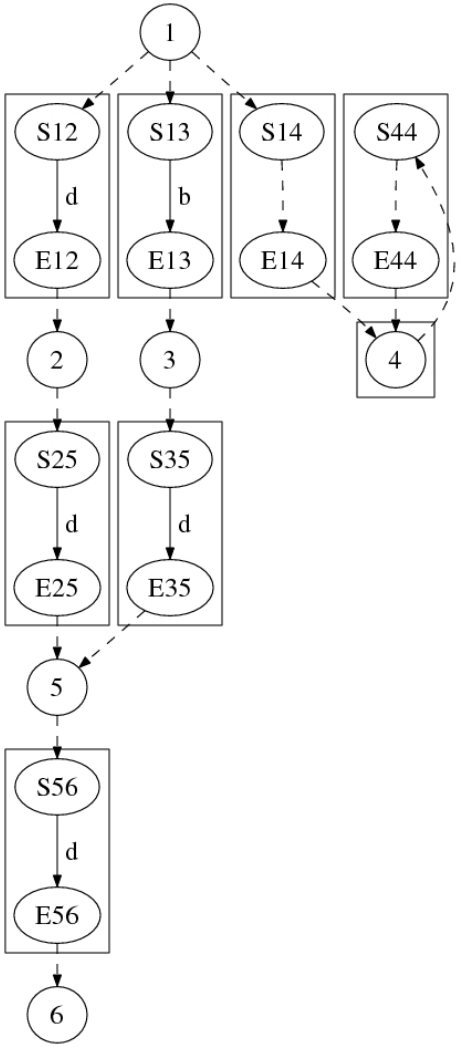


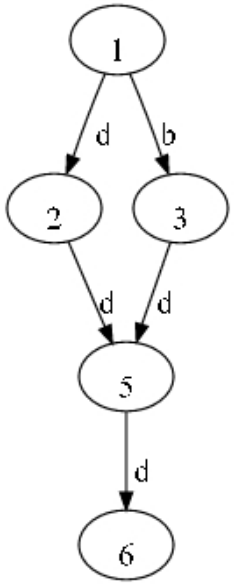
$$\text{sfun } a2d\_xc\ (\{l : g\}) = \begin{array}{l} \text{if } l = a \text{ then } \{d : a2d\_xc(g)\} \\ \text{else if } l = c \text{ then } a2d\_xc(g) \\ \text{else } \{l : a2d\_xc(g)\} \end{array}$$

The Bulk Semantics of Structural Recursion

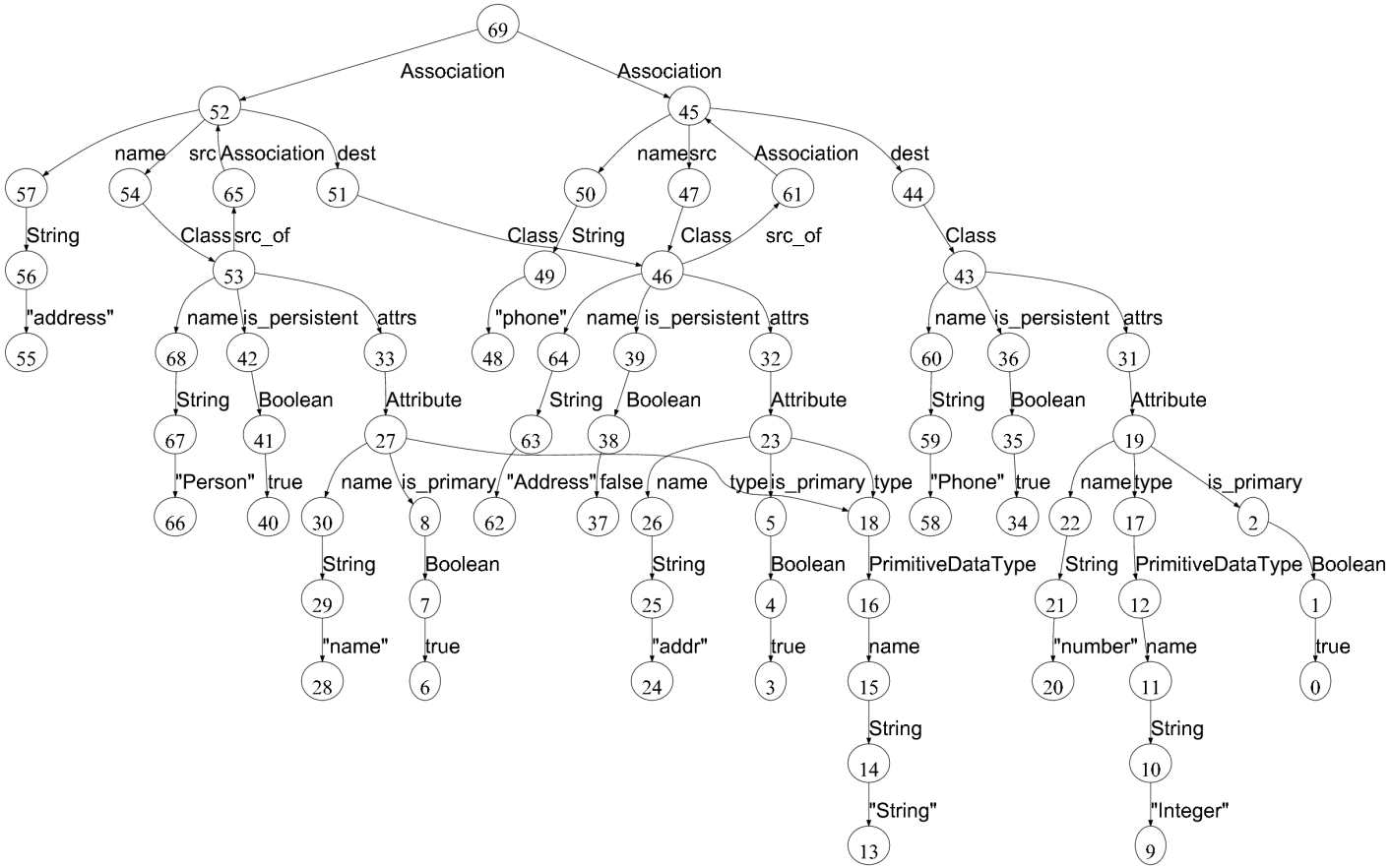


The Bulk Semantics of Structural Recursion

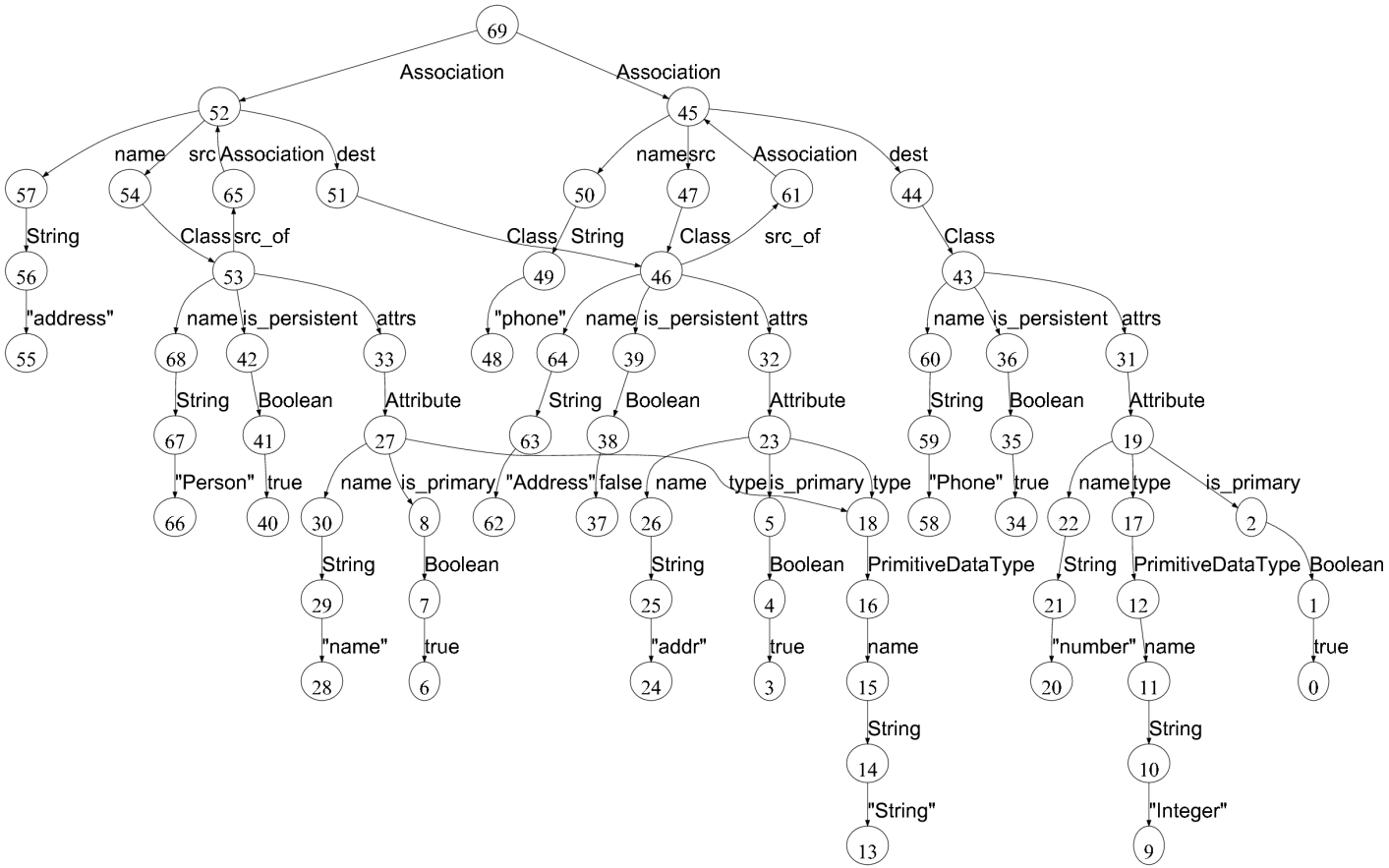




Graph Querying in UnQL



Graph Querying in UnQL



How to extract all persistent classes?

Graph Querying in UnQL

select $\$class$ where
$\{Association.(src|dest).Class : \$class\}$ in $\$classDB$,
$\{is\_presistent : \{Boolean : true\}\}$ in $\$class$

Graph Querying in UnQL

select $\$class$ where
$\{Association.(src|dest).Class : \$class\}$ in $\$classDB$,
$\{is\_presistent : \{Boolean : true\}\}$ in $\$class$

Remarks:

- We do not need to use structural recursion explicitly!
- Any UnQL expression can be described as a composition of structural recursions.

UnQL$^+$: An Extension of UnQL

$$
\begin{array}{l}
\textcolor{red}{\text{replace}}\ \{\$name : \{\}\} \\
\textcolor{red}{\text{by}}\quad\quad \{(\text{"}class\_\text{"} + \$name) : \{\}\} \\
\textcolor{red}{\text{where}} \\
\quad \{\_ * .Class.name.String : \{\$name : \{\}\}\}\ \text{in}\ \$classDB
\end{array}
$$

UnQL$^+$: An Extension of UnQL

$$
\begin{array}{l}
\textcolor{red}{\text{replace}}\ \{\$name : \{\}\} \\
\textcolor{red}{\text{by}}\quad\quad \{(\text{"}class\_\text{"} + \$name) : \{\}\} \\
\textcolor{red}{\text{where}} \\
\quad \{\_ * .Class.name.String : \{\$name : \{\}\}\}\ \text{in}\ \$classDB
\end{array}
$$

Property 1

Any expression in UnQL$^+$ can be described as a composition of structural recursions.

UnQL$^+$: An Extension of UnQL

$$
\begin{array}{l}
\textcolor{red}{\text{replace}}\ \{\$name : \{\}\} \\
\textcolor{red}{\text{by}}\quad\quad \{(\text{"}class\_\text{"} + \$name) : \{\}\} \\
\textcolor{red}{\text{where}} \\
\quad \{\_ * .Class.name.String : \{\$name : \{\}\}\}\ \text{in}\ \$classDB
\end{array}
$$

Property 1

Any expression in UnQL$^+$ can be described as a composition of structural recursions.

Property 2

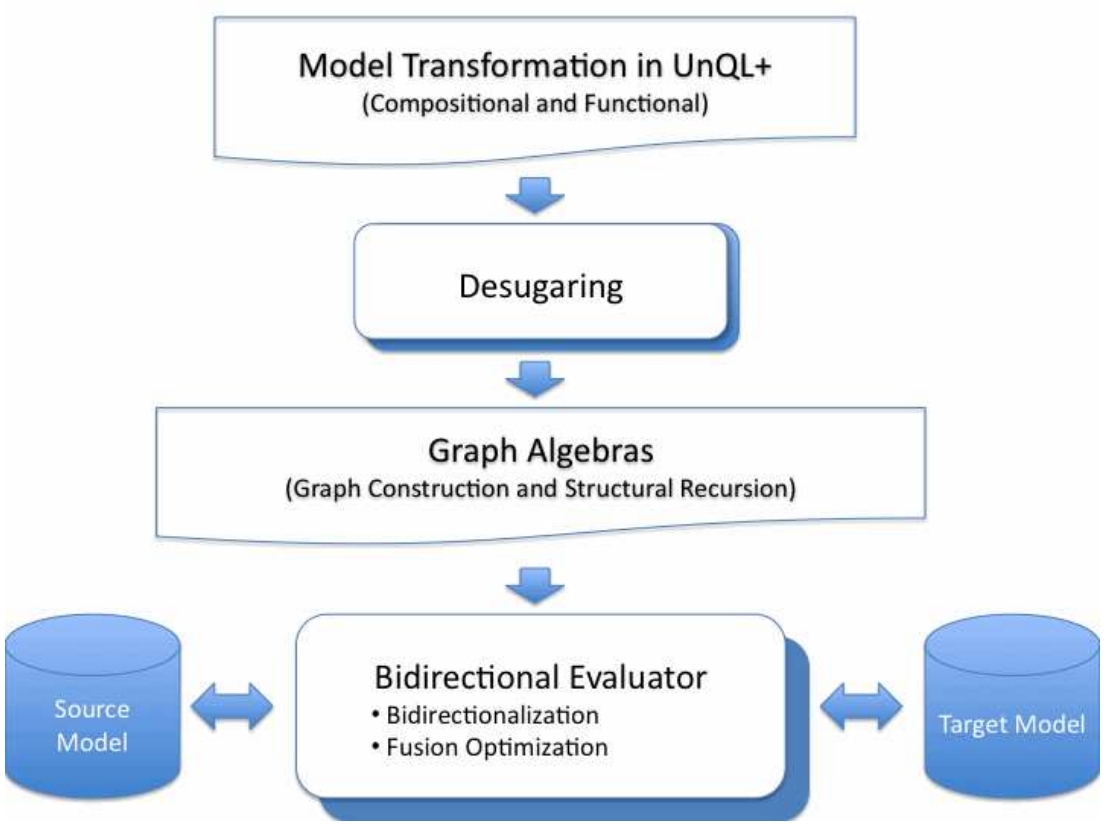
Unnecessary compositions can be automatically removed.

Transformation Compositions

Extract all persistent classes, and transform them to tables by replacing *attrs* by *cols* and *Attribute* by *Column*.

```
(* replace Attribute *)
replace{$l_A : $g} by{Column : $g} where
  $db in
    (* replace attrs *)
    (replace{$l_a : $A} by{cols : $A} where
        $class in
          (* select classes *)
          (select $class where
          {Association.(src|dest).Class : $class}
                in $classDB,
          {is_presistent : {Boolean : true}} in $class),
       {$l_a : $A} in $class, $l_a = attrs),
    {cols : {$l_A : $g}} in $db, $l_A = Attribute
```

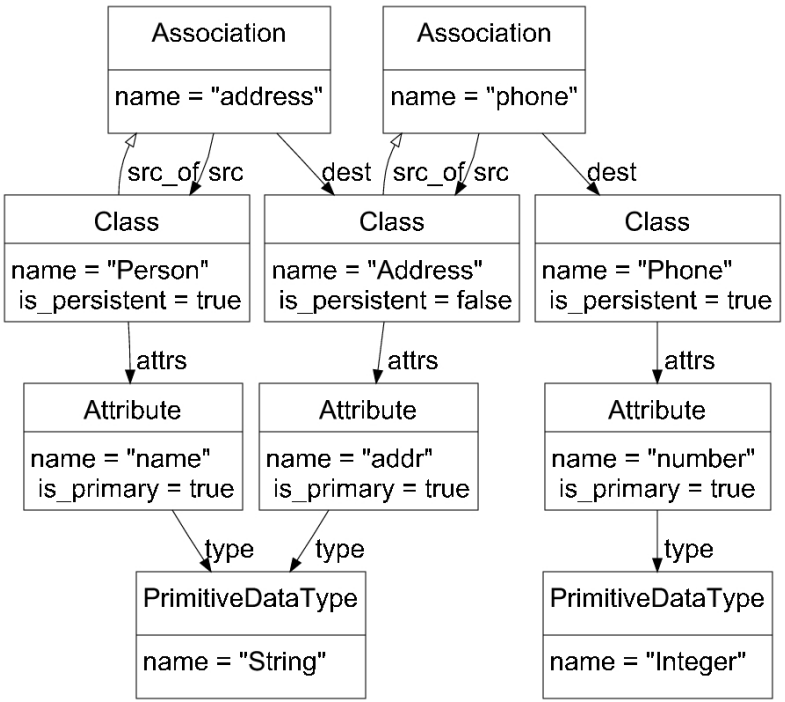
System Overview

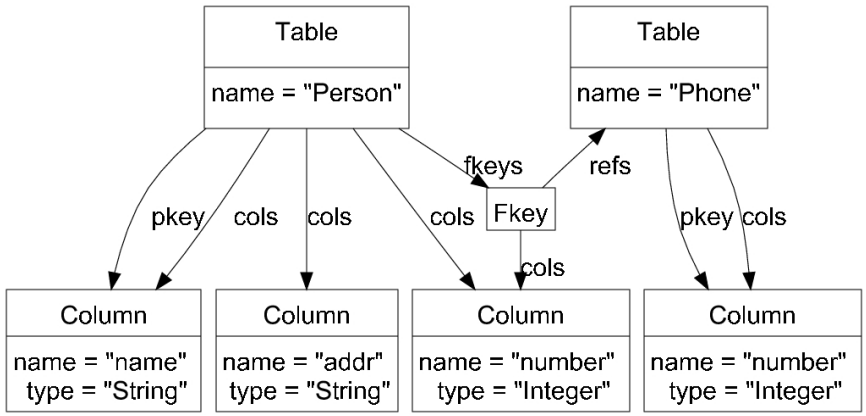


(8,500 lines of code in OCaml)

Application: Class2RDB

Transformation from "Class to RDB", a nontrivial example proposed at MTiP'05.





Class2RDB in UnQL$^+$

```
select $tables where
  $tables in
    {select $tables where
       {Class:$class} in (select $assoc where {Association.{src|dest}:$assoc} in $db),
       {is_persistent.Boolean:true} in $class,
       $dests in {select {Class:$dest} where {{src_of.Association.dest.Class}+:$dest} in $class},
       $related in {{Class:$class} U $dests},
       $cols in {select {cols:{Column:{name:$n,type:$t}}} where {Class.attrs.Attribute:{name:$n,type:$t}} in $related
       $tables in {select {Table:{name:$cname} U $cols} where {name:$cname} in $class},
       $tables in {extend $table with $pkeys U $fkeys where
                     {Table:$table} in $tables,
                     {cols:$cols} in $table,
                     {Column.name.String:{$cname:{}}} in $cols,
                     $pkeys in {select {pkey:$cols} where
                                  {attrs.Attribute: {is_primary.Boolean:true, name.String:{$pname:{}}}} in $class,
                                  $cname = $pname},
                     $fkeys in {select {fkeys:{Fkey:{cols:$cols, ref:$ref}}} where
                                  {Class:{is_persistent.Boolean:true,
                                          attrs.Attribute.name.String:{$aname:{}}, name:$ref}} in $dests,
                                  $cname = $aname}},
  $tables in {replace $ref by {Table:$table} where
                {Table.fkeys.Fkey.ref:$ref, Table:$table} in $tables,
                {String:{$rname:{}}} in $ref,
                {name.String:{$tname:{}}} in $table,
                $tname = $rname}
```

Conclusion

- Bidirectional transformation is fun!
  - Static $\Rightarrow$ dynamic
- Bidirectional transformation is useful!
  - Synchronization of data in different formats
  - Document development
  - Software Engineering
- Bidirectional transformation is challenging!
  - Semantic issues, compositional construction, development environment, ...

Conclusion

- Bidirectional transformation is fun!
  - Static $\Rightarrow$ dynamic
- Bidirectional transformation is useful!
  - Synchronization of data in different formats
  - Document development
  - Software Engineering
- Bidirectional transformation is challenging!
  - Semantic issues, compositional construction, development environment, ...

$\Downarrow$

Linguistic Foundation for Bidirectional Graph Transformation

http://www.biglab.org