Validity Checking of Bidirectional Transformations

Zhenjiang Hu
National Institute of Informatics (NII), Japan

Joint work with H. Pacheco, S. Fischer, and BiG Members
Validity Checking of Putback Transformations in Bidirectional (Functional) Programming

Zhenjiang Hu
National Institute of Informatics (NII), Japan

Joint work with H. Pacheco, S. Fischer, and BiG Members
Validity Checking of BX

F, B \models roundtrip_property
Bidirectional Transformation
Bidirectional Transformation (BX)

[Nate Foster, et al: POPL 2005]
Roundtrip Properties

Get-Put:
\[ \text{put } s \ (\text{get } s) = s \]

Put-Get:
\[ \text{get } (\text{put } s \ t) = t \]
Bidirectional Computation

BiX: A Bidirectional Tree Transformation Language

Bidirectional Model Transformation: A Compositional Approach

Definition

Basic Properties

Direct Applications

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]

Model Code Coevolution

(Foster et al.: POPL 2005)

ICSE 2012

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]

Zhenjiang Hu

There and Back Again

FM 2014, Singapore, May 15, 2014
Dagstuhl Seminar on BX, 2010

Zhenjiang Hu, Andy Schurr, Perdita Stevens, James Terwilliger,
Bidirectional Transformation Languages
Bidirectional Transformation Languages

- We need languages to support development of software with bidirectional computation

  - **Lens** (for data sync) Univ. of Pennsylvania, ...
    [POPL’05, PODS’06, ICFP’08, ICFP’10, POPL’11,’12]

  - **X/Inv/BiX/UnQL**+ Univ. of Tokyo / NII
    [PEPM’04, ICFP’07, ASE’07, FSE’09, POPL’09, ESOP’10, ICFP’10, MODELS’10, PPDP’11, ICSE’12, PPDP’13, ICFP’13]
Existing Approaches

- Domain Specific Bidirectional Languages
- Automatic Bidirectionization of ATL, XQuery, UnQL

Correctness by Construction
Ambiguity of “Put”

Since get is generally non-injective, many suitable puts correspond to one get, each being useful in different context.
getHeight \((w,h) = h\)

putHeight1 \((w,h) h' = (w,h')\)

PutHeight2 \((w,h) h' = (w*h'/h, h')\)

putHeight3 \(h' \mid h==h' = (w,h) \mid \text{otherwise} = (3,h')\)
An Ad-hoc Solution: Enriching “get”

Enrich “get” with more and more control over “put”

Get-based BX Combinator Library Lenses
(Foster et al.: POPL 2005)

Quotient Lenses
(Foster et al.: POPL'08)

Matching Lenses
(Foster et al.: ICFP'10)

Edit Lenses
(Hoffman et al.: POPL’12)

⇒ Too many versions of “get” ...
What is the essence of bidirectional transformation?
Put is the essence of BX!

• An important but little-known fact:

\[ \text{put uniquely determines get} \]
Derived “get”: Uniqueness

**Lemma**: Given a put function, there exists at most one get function such that GetPut and PutGet hold.

**Proof**: Suppose we have two get functions, say get and get’.

\[
\text{get s} \rightarrow \{\text{GetPut}\} \rightarrow \text{get’ s}
\]
Lemma: Given a put function, there exists at most one get function such that GetPut and PutGet hold.

Proof:
Suppose we have two get functions, say get and get'.

\[
\text{get } s \\
\Rightarrow \{ \text{GetPut} \} \\
\text{get (put } s (\text{get' } s)) \\
\Rightarrow \{ \text{PutGet} \} \\
\text{get' } s
\]
Derived “get”: Existence

Lemma: Given a surjective put function (for any s, there exist s’, v, such that s = put s’ v), the get function defined by

\[\text{get } s = v \text{ such that } \text{put } s \ v = s\]

satisfies GetPut and PutGet.

Proof:

\[
\begin{align*}
\text{get (put } s \ v) & \quad \text{put } s \ (\text{get } s) \\
v & \quad s
\end{align*}
\]
Derived “get”: Existence

**Lemma**: Given a surjective put function (for any s, there exist s’, v, such that s = put s’ v), the get function defined by

\[
\text{get } s = v \text{ such that } \text{put } s \ v = s
\]

satisfies GetPut and PutGet.

**Proof**:

\[
\begin{align*}
\text{get } (\text{put } s \ v) & \quad \text{\{condition for put\}} \\
\text{get } s & \quad \text{\{definition of get\}} \\
= & \quad \text{\{definition of get\}} \\
& \quad \text{\{condition for put\}} \\
v & \quad \text{s}
\end{align*}
\]
Remark

Lemma: if there exists a \( v \) satisfying \( \text{put } s' \ v = s \), then so does \( \text{put } s \ v = s \).

Proof:

\[
\begin{align*}
\text{put } s' \ v &= s \\
\Rightarrow & \quad \{ \text{put} \} \\
\text{put } (\text{put } s' \ v) \ v &= \text{put } s \ v \\
\Rightarrow & \quad \{ \text{PutTwice} \} \\
\text{put } s' \ v &= \text{put } s \ v \\
\Rightarrow & \quad \{ \text{Assumption: put } s' \ v = s \} \\
s &= \text{put } s \ v
\end{align*}
\]
Putback-Style Bidirectional Programming!

“get” (forward transformation)

“put” (backward transformation)

“get” (forward transformation)

“put” (backward transformation)
Validity Checking of Putback Transformations
Well-behaved “put”

Definition: A “put” function is said to be well-behaved, if there exists a (unique) “get” function such that GetPut and PutGet hold.

Question: Are the following put functions well behaved?

- put1 s v = s
- put2 s v = v : s
- put3 [] v = v
  put3 (a : s) v = a : v

Difficult to check because we do not have “get” yet …
Well-behaved “put”

Lemma:
Put is well-behaved, iff
1. View-deterministic
   \[ \text{put } s_1 \ v_1 = \text{put } s_2 \ v_2 \Rightarrow v_1 = v_2 \]
2. View-stable
   for any \( s \), there exists a \( v \), such that \( \text{put } s \ v = s \)

Reference:
A Language for Putback Programming

A treeless language for define primitive well-behaved puts.

+ 

A set of combinators to compose smaller well-behaved puts to form bigger ones
A Treeless Language \textit{PDL}

A Treeless Language for Put-based Bidirectional Programming

Rule

\[
f \; p_1 \; p_2 \; p_3 = t
\]

Treeless Term

\[
t ::= v \quad \{ \text{variable} \} \\
    \mid C \; t_1 \; \ldots \; t_n \quad \{ \text{constructor application} \} \\
    \mid f \; x_s \; x_v \quad \{ \text{put application} \}
\]

Pattern

\[
p ::= x \quad \{ \text{variable} \} \\
    \mid x \; @ \; p \quad \{ \text{look-ahead variable} \} \\
    \mid C \; p_1 \; \ldots \; p_n \quad \{ \text{constructor pattern} \}
Syntactic Assumptions

- **Affine**: each variable appears at most once in rhs
  
  \[
  \text{put (s:ss) vs = s : vs} \quad \text{GOOD}
  \]
  
  \[
  \text{put (s:ss) vs = s : (vs++vs)} \quad \text{BAD}
  \]

- **Structured**: recursive calls are on smaller sub-patterns
  
  \[
  \text{put (s:ss) (v:vs) = v : put ss (v:vs)} \quad \text{GOOD}
  \]
  
  \[
  \text{put (s:ss) (v:vs) = 1 : put ss (1:vs)} \quad \text{BAD}
  \]

- **Total**: patterns are exhausted
Example

\[
\text{putAs} \ [A, 1, A, 2, B, 3, A, 4] \ [10, 11, 12] \rightarrow \ [A, 10, A, 11, B, 3, A, 12] \\
\text{putAs} \ [A, 1, A, 2, B, 3, A, 4] \ [10, 11] \rightarrow \ [A, 10, A, 11, B, 3] \\
\text{putAs} \ [A, 1, A, 2, B, 3, A, 4] \ [10, 11, 12, 13] \rightarrow \ [A, 10, A, 11, B, 3, A, 12, A, 13]
\]

\[
\begin{align*}
\text{putAs} \ [\ ] & \ [\ ] = [\ ] \\
\text{putAs} \ (\text{ss}@[\]) \ (v:\text{vs}) & = A \ v : \text{putAs} \ \text{ss} \ \text{vs} \\
\text{putAs} \ (A \ a : \text{ss}) \ (\text{vs}@[\]) & = \text{putAs} \ \text{ss} \ \text{vs} \\
\text{putAs} \ (A \ a : \text{ss}) \ (v : \text{vs}) & = A \ v : \text{putAs} \ \text{ss} \ \text{vs} \\
\text{putAs} \ (B \ b : \text{ss}) \ \text{vs} & = B \ b : \text{putAs} \ \text{ss} \ \text{vs}
\end{align*}
\]
Example

putAs [A 1, A 2, B 3, A 4] [10, 11, 12] \rightarrow [A 10, A 11, B 3, A 12]
putAs [A 1, A 2, B 3, A 4] [10, 11] \rightarrow [A 10, A 11, B 3, B 4]
putAs [A 1, A 2, B 3, A 4] [10, 11, 12, 13] \rightarrow [A 10, A 11, B 3, A 12, B 0, A 13]

putAs [] [] = []
putAs (ss@[ ]) (v : vs) = A v : B O : putAs ss vs
putAs (A a : ss) (vs@[ ]) = B a : putAs ss vs
putAs (A a : ss) (v : vs) = A v : putAs ss vs
putAs (B b : ss) vs = B b : putAs ss vs
Main Results

Theorem:
Well-behavedness of a put defined in $PDL$ is decidable.

Validation Algorithm:
(Soundness): A validated put is well-behaved.
(Completeness): Any well-behaved put can be validated.
View-Determination Validation

Lemma:
Put is well-behaved, iff

1. View-deterministic
   \[ \text{put } s_1 v_1 = \text{put } s_2 v_2 \Rightarrow v_1 = v_2 \]
2. View-stable
   for any \( s \), there exists a \( v \), such that put \( s \ v = s \)

The relation from updated sources to views forms a function.
The relation from updated sources to views forms a function.

(1) The relation $R$ can be automatically derived from the put defined in $PDL$, which is a finite tree transducer. (In $f \text{ ps pv } = t$, view variables must all appear in $t$)

(2) FACT: Single-valuedness of finite tree transducers is decidable (Seidl:TCS92)
Example: Derivation of $R$

\[
\begin{align*}
\text{putAs } [] & = [] \\
\text{putAs } (ss@[]) & = A \, \text{} \\
\text{putAs } (A \, a : \, ss) & = \text{putAs } ss \, vs \\
\text{putAs } (A \, a : \, ss) & = \text{putAs } ss \, vs \\
\text{putAs } (B \, b : \, ss) & = B \, b : \text{putAs } ss \, vs
\end{align*}
\]

\[
\begin{align*}
R \, [] & = [] \\
R \, (A \, v : \, ps) & = v : R \, ps \\
R \, ps & = R \, ps \\
R \, (A \, v : \, ps) & = v : R \, vs \\
R \, (B \, b : \, ps) & = R \, ps
\end{align*}
\]

$R$ is a function
Lemma:
Put is well-behaved, iff
1. View-deterministic
   \[ \text{put } s_1 \; v_1 = \text{put } s_2 \; v_2 \Rightarrow v_1 = v_2 \]
2. View-stable
   for any \( s \), there exists a \( v \), such that \( \text{put } s \; v = s \)

[v can only be R(s) from view-determination]
Let \( h \; x \; y = \text{put } x \; (R \; y) \). The validation of \( h \; s \; s = s \) is decidable.
View-Stability Validation

(1) $h$ is of treeless form (structural recursion) in $PDL$.

(2) The inductive validity of $h \uparrow_1 \uparrow_2 = p$ is decidable (so does $h \ s \ s = s$) [Giesl&Kapur: IJCAR01]

Let $h \ x \ y = \text{put} \ x \ (R \ y)$. The validation of $h \ s \ s = s$ is decidable.

[v can only be $R(s)$ from View-determination]
Example: Derivation of $h$

$h \, xs \, ys = \text{putAs} \, xs \, (R \, ys)$

$h \, [] \, [] = []$
$h \, [] \, (A \, y : \, ys) = A \, y : \, h \, [] \, ys$
$h \, [] \, (B \, y : \, ys) = h \, [] \, ys$
$h \, (A \, x : \, xs) \, [] = h \, xs \, []$
$h \, (A \, x : \, xs) \, (A \, y : \, ys) = A \, y : \, h \, xs \, ys$
$h \, (A \, x : \, xs) \, (B \, y : \, ys) = h \, (A \, x : \, xs) \, ys$
$h \, (B \, x : \, xs) \, ys = B \, x : \, h \, xs \, ys$

Validation of $h \, xs \, xs = xs$ is decidable.
Conclusion

- **Putback** is the *essence* of BX
- *Validity check of treeless putback* is *decidable*
- **Practical** languages for writing putback
  - Monadic Putback Combinators (PEPM 2014)
  - Update-based Putback Language (ICSE 2014 (NIER))
BiG Team at NII (2008-)

Aim at Language Foundation of Bidirectional Graph Transformation

Zhenjiang Hu  
Prof. (NII), Project Leader  
BX Programming

Hiroyuki Kato  
Assist. Prof. (NII)  
BX Optimization

Soichiro Hidaka  
Assist. Prof. (NII)  
Bidirectionalization

Hugo Pacheco  
Post-doc (NII, 2013)  
BX Languages

Kazuyuki Asada  
Post-doc (NII, 2011-2013)  
BX Semantics

PhD Students:  
- Tao Zan  
- Huu-Phuc Vo

FM 2014, Singapore, May 15, 2014