IRP ∩ LSI is not RE

( Introduction of the manuscript: Tetsuya Ishiu, “IRP is Strictly Larger Than MTT”
http://twitdoc.com/c/xrwhnm )

presentation by Kazuhiro Inaba (NII, kinaba@nii.ac.jp)

IPL-Group Seminar
May 25, 2010
The Talk is about a Property on...

- **String/Tree Transformations**
  - e.g., `dup(s) = s ++ s`
  - `reverse([]) = []`
  - `reverse(x:xs) = reverse(xs) ++ [x]`

- **Regular Languages**
  - e.g., `a* | b*`
  - `a((a|b)* | c)*b`
The Property IRP

Inverse Regularity Preserving

A function $f$ is IRP iff

For any regular language $L$, the inv. img. $f^{-1}(L) = \{s \mid f(s) \in L\}$ is regular

Example: “dup” and “reverse”

Example:

$\text{dup}(s) = s++s$

$a^* | b^*$
Agenda

Why you should be interested in IRP?
- IRP-based typechecking
- Always-IRP computation models

Q: “Do the models cover all IRP?”
A: “No, $\text{IRP} \cap \text{LSI}$ is not RE.”

Proof Tech. 1: Clever diagonalization
Proof Tech. 2: Slenderness of languages
Why IRP?

- Typechecking

Verify that a transformation always generates valid outputs from valid inputs.

<table>
<thead>
<tr>
<th></th>
<th>$L_{IN}$</th>
<th>$L_{OUT}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>XSLT Template for formatting bookmarks</td>
<td>XBEL Schema</td>
</tr>
<tr>
<td></td>
<td>Arbitrary String</td>
<td>String not containing &quot;&lt;script&gt;&quot;</td>
</tr>
</tbody>
</table>
Why IRP?

- Typechecking \( f :: L_{IN} \rightarrow L_{OUT} \)?

If \( f \) is IRP, we can check this by ...

\[
f \text{ is type-correct} \iff f(L_{IN}) \subseteq L_{OUT} \iff L_{IN} \subseteq f^{-1}(L_{OUT})
\]

(for experts: \( f \) is assumed to be deterministic)
FAQ: Why IRP?

- ForwardRP also enables typechecking
- **FRP**-based checking: $\text{f}(L_{\text{IN}}) \subseteq L_{\text{OUT}}$
- **IRP**-based checking: $L_{\text{IN}} \subseteq \text{f}^{-1}(L_{\text{OUT}})$

Reasons

- IRP provides more useful counter examples.
- Many functions in practice tend to be IRP, but not so for FRP. E.g., “dup”.
IRP-Based Typechecking

- Not all transformations are IRP

- The trend is to define a restricted language whose programs are always IRP and present sound & complete typechecking for them

- or use them as clearly defined targets for approximate checking
Famous Computation Models of IRP Tree Transformations

\[ X^* := \{ f_1 \cdot f_2 \cdot \ldots \cdot f_n \mid n \in \text{Nat}, f_i \in X \} \]

\[ \text{MTT}^* = \text{PTT}^* = \text{ATT}^* = \ldots \]

- MTT [Engelfriet & Vogler 1985]
- ATT [Fülop 1981]
- MSOTT [Courcelle 1994]
- \( T \) [Thatcher 70, Rounds 70]
- B
- GSM

\[ \text{PTT} \] [Milo & Suciu & Vianu 2000]
One Example: MTT

- MTT = The class of functions on trees defined by (mutual) structural recursion + accumulating parameters
- MTT* = Finite composition of MTTs

**Syntax Example**

\[
\begin{align*}
\text{MTT} &::= \text{FUN} \ldots \text{FUN} \\
\text{FUN} &::= f(A(x_1, \ldots, x_n), y_1, \ldots, y_k) \rightarrow \text{RHS} \\
\text{RHS} &::= C( \text{RHS}, \ldots, \text{RHS} ) \\
&\quad \mid f(x_i, \text{RHS}, \ldots, \text{RHS}) \mid y_i
\end{align*}
\]

- \( \text{start}( A(x_1) ) \rightarrow \text{double}( x_1, \text{double}(x_1, E) ) \)
- \( \text{double}( A(x_1), y_1 ) \rightarrow \text{double}( x_1, \text{double}(x_1, y_1) ) \)
- \( \text{double}( B, y_1 ) \rightarrow C(y_1, y_1) \)
Do they cover all IRP transformations? 

$\text{MTT}^* = \text{PTT}^* = \text{ATT}^* = .. = \text{IRP}$? 

$\subseteq$ is known

$\supseteq$?

(Attribution: I’ve first heard this question from Sebastian Maneth, who heard it from Keisuke Nakano)
Answer: “No”

\[
\text{tower( “a..a” )} = “aa...aa”
\]

where \( 2^{^0} = 1, \quad 2^{^{(n+1)}} = 2^{^2^{^n}} \)

is IRP but not in MTT*

But its growth is tooooooooo fast!

Aren’t there any “milder” counterexample?
LSI: Linear Size Increase = \( \exists c. \forall t. \text{len}(f(t)) < c \cdot \text{len}(t) \)

\[
\text{MTT}^* = \text{PTT}^* = \text{ATT}^* = \ldots
\]

\[
\text{MSOTT} = \text{MSOTT}^* = (\text{MTT}^* = \text{PTT}^* = \ldots) \cap \text{LSI} = (\text{MTT} \cap \text{LSI})^* = \ldots
\]
New Question

MSOTT
= (MTT*=PTT*=ATT*=...) ∩ LSI
= IRP ∩ LSI ?

⊆ is known
⊇ ?
There exists a \( \text{IRP} \cap \text{LSI} \) transformation that cannot be written in MSOTT

Answer: “No”

Main Theorem of This Talk

*The class of \( \text{IRP} \cap \text{LSI} \) transformations is not recursively enumerable.*

(There’s no Turing machine that enumerates all of them)
THE PROOF
Overview

The class of IRP ∩ LSI transformations is not recursively enumerable. (There’s no Turing machine that enumerates all of them)

Basic idea is the Diagonalization (対角線論法)

“Give me a enumeration \{g_1, g_2, g_3, \ldots\} of the class of functions. Then I will show you a function \( f \) not in the enumeration.”
Diagonalization

(assuming a fixed alphabet,) we can enumerate all string/trees: \{t_1, t_2, t_3, \ldots\}

Given enumeration \{g_1, g_2, g_3, \ldots\} of the class

We construct \( f \) as:

\[ f( t_i ) := \text{arbitrary tree except } g_i(t_i) \]

Caution!

\( f \) may not be IRP nor LSI

\[
\begin{array}{cccc}
  & t_1 & t_2 & t_3 & t_4 & \ldots \\
 g_1 & \times &   &   &   &   \\
 g_2 &   & \times &   &   &   \\
 g_3 &   &   & \times &   &   \\
 g_4 &   &   &   & \times &   \\
 \cdots &   &   &   &   & \cdots
\end{array}
\]
Diagonalization

The class of IRP ∩ LSI transformations is not recursively enumerable. (There’s no Turing machine that enumerates all of them)

What we really want is this:

“Give me a enumeration \{g_1, g_2, g_3, \ldots\} of the IRP \cap LSI functions. Then I will show you a function \( f \) not in the enumeration but in IRP \cap LSI.”

which derives contradiction.
We can enumerate all regular languages: \{R1, R2, R3, \ldots\}

Given enumeration \{g1, g2, g3, \ldots\} of the class of regular languages, we construct f so that:

- \( f(t) \neq g_i(t) \) for some \( t \in \{R1,R2,\ldots,R_i\} \)
- \( f^{-1}(R_i) = \text{almost } R_i \)

|     | R1 | R2 | R3 | R4 | ...
|-----|----|----|----|----|----
| g1  |    |    | ×  |    | ...
| g2  |    | ×  |    |    | ...
| g3  | ×  |    |    |    | ...
| g4  |    |    | ×  |    | ...
| ... |    |    |    |    | ...
Preparation

Known facts on Regular Languages

- All finite sets are regular
- They are closed under boolean ops.
  - If $R_1, R_2 \in \text{REG}$ then $R_1 \cap R_2 \in \text{REG}$
  - $R_1 \cup R_2 \in \text{REG}$
  - $\sim R_1 \in \text{REG}$

“Slenderness” is decidable

Preparation: Slenderness

A set $L$ of string is slender iff

$$\exists c. \forall n. \#\{s \mid s \in L, \text{len}(s)=n\} \leq c$$

- $\{1, 11, 111, 1111, \ldots\}$ is slender
- $\{0, 1, 10, 11, 100, 101, \ldots\}$ is not slender
- $L_1, L_2$ is slender $\Rightarrow$ $L_1 \cup L_2$ is slender
- $L_1, L_2$ is co-slender $\Rightarrow$ $L_1 \cap L_2$ is co-slender

Co-slender $\iff$ complement is slender

Not co-slender $\iff$ a plenty of supply of non-members
Main Lemma

Let \( \{g_1, g_2, \ldots \} \) be an enumeration of total functions.
Let \( \{R_1, R_2, \ldots \} \) be an enumeration of all regular langs.

Then we can construct \( \{(f_0,D_0), (f_1,D_1), \ldots \} \) such that

- \( \Phi = f_0 \subseteq f_1 \subseteq f_2 \subseteq \ldots \) \hspace{1cm} \# increasing list of partial functions
- \( \Phi = D_0 \subseteq D_1 \subseteq D_2 \subseteq \ldots \) \hspace{1cm} \# eventually covers all regular languages
- Either \( R_i \subseteq D_i \) or \( \sim R_i \subseteq D_i \)
- \( \exists x \in D_i. \ f_i(x) \neq g_i(x) \) \hspace{1cm} \# different from every \( g_i \)
- \( D_i \) is not co-slender \hspace{1cm} \# technical detail
- \( f_i \) is bijective on \( D_i \)
- For all but finitely many \( x \in D_i \), \( f_i(x) = x \) \hspace{1cm} \# almost identity (hence IRP)
- \( \forall x \in D_i. \ \text{len}(f_i(x)) = \text{len}(x) \) \hspace{1cm} \# linear size increase
Proof of the Main Lemma

By Induction

- \( f_0 = D_0 = \Phi \)
- Suppose we already have \( f_n \) construct \( f_{n+1} \) and \( D_{n+1} \).

Set of All Strings

\[ D_{n+1} = \text{dom}(f_{n+1}) \]

\[ D_n = \text{dom}(f_n) \]

where \( f_{n+1} \neq g_{n+1} \)

Requirements
- \( D_{n+1} \) must cover either \( R_{n+1} \) or \( \sim R_{n+1} \)
- \( D_{n+1} \) must not be co-slender
- \( D_{n+1} \) must have elems to distinguish \( g_{n+1} \) and \( f_{n+1} \)
Proof of the Main Lemma

D_n is not co-slender. \( \Rightarrow \)

Take \( x, y \in \sim D_n \) s.t. \( \text{len}(x) = \text{len}(y) \) but \( x \neq y \)

Then Take

- \( D_{n+1} := D_n \cup \{x, y\} \cup R_n \)
- if it is not co-slender

- \( D_{n+1} := D_n \cup \{x, y\} \cup \sim R_n \)
- otherwise \( \uparrow \text{this becomes not-co-slender!} \)

Requirements

- \( D_{n+1} \) must cover either \( R_{n+1} \) or \( \sim R_{n+1} \)
- \( D_{n+1} \) must not be co-slender
- \( D_{n+1} \) must have elements to distinguish \( g_{n+1} \) and \( f_{n+1} \)
Proof of the Main Lemma

We then construct $f_{n+1}$

- $f_{n+1}(s) = f_n(s)$ if $s \in D_n$
- $f_{n+1}(x) = x$ if $g_{n+1}(x) = x$
- $f_{n+1}(x) = y$ if $f_{n+1}(y) = x$
- Otherwise, $f_{n+1}(s) = s$ for all other $s \in D_{n+1}$

Requirements
- $f_n \subseteq f_{n+1}$
- $f_{n+1}$ is bijection on $D_{n+1}$
- $f_{n+1}$ is length preserving
- $f_{n+1}$ differs from $g_{n+1}$
- $f_{n+1}$ is almost identity

Q.E.D.
Suppose it is. By previous lemma, let \( f = \bigcup_{i \in \mathbb{Nat}} f_i \)

- \( f \) is equal to none of \( \{g_1, g_2, \ldots\} \)
- \( f \) is a total function

Because each singleton \( \{s_i\} \) regular set must be covered by \( D_i = \text{dom}(f_i) \) eventually

- \( f \) is LSI (in fact, length-preserving)
- \( f \) is IRP
Main Theorem

- **f is IRP** (In fact, f is FRP by almost the same proof, too.)

- Take any regular set $R_i$.
  - If $R_i \subseteq D_i$
    - Since $f_i$ is bijection & identity except fin. points,
      $f^{-1}(R_i) = f_i^{-1}(R_i)$ differs only finitely from $R_i$
      $\Rightarrow$ regular

- If $\sim R_i \subseteq D_i$
  - Similarly, $f^{-1}(\sim R_i)$ is regular
  - f is also a bijection, so $f^{-1}(R_i) = \sim f^{-1}(\sim R_i)$
    $\Rightarrow$ regular

Contradiction. Q.E.D.
If \{g_1, g_2, \ldots\} is an enumeration of computable total functions, then the \( f \) is a computable function. \( f^{-1} \) (as a mapping on regular languages) is computable.

<Summary> There exists \( f \) such that
- \( f : \text{string} \rightarrow \text{string} \) is computable & total
- \( f^{-1} : \text{REG} \rightarrow \text{REG} \) is computable & total
- \( f \) is length-preserving, IRP, and FRP
- \( f \) is not in MSOTT = MTT^* \cap LSI