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...

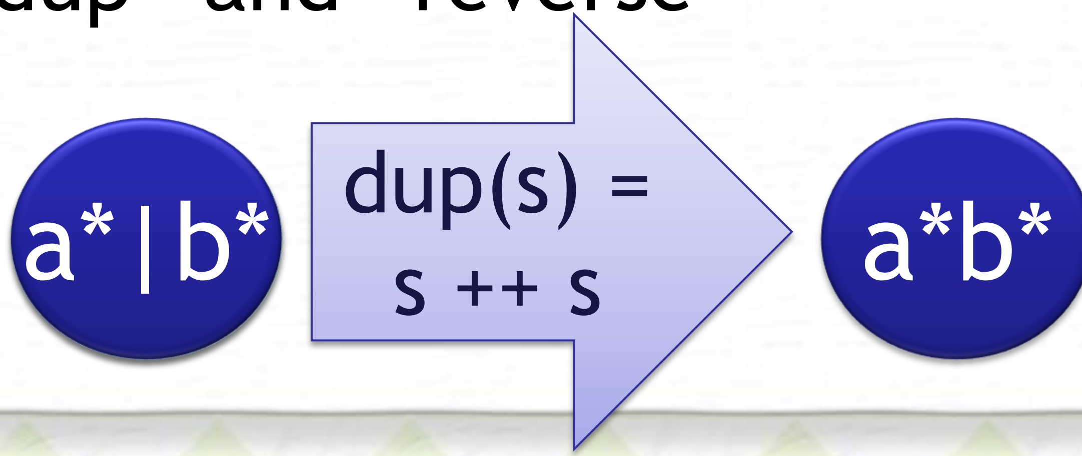
- String/Tree Transformations
  - e.g., dup(s) = s ++ s

    reverse([]) = []
    reverse(x:xs)= reverse(xs)++[x]

  and
- Regular Languages
  - e.g., a*|b* a((a|b)*|c)*b

The Property IRP

- **I**nverse **R**egularity **P**reserving
  - 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:



Agenda

- Why you should be interested in IRP?
  - IRP-based typechecking
  - Always-IRP computation models
- Q: “Do the models cover all IRP?”
- A: “No, IRP$\cap$LSI is not RE.”
  - Proof Tech. 1: Clever diagonalization
  - Proof Tech. 2: Slenderness of languages

Why IRP?

- Typechecking $f :: L_{IN} \to L_{OUT}$?
  - Verify that a transformation always generates valid outputs from valid inputs.

f	L_{IN}	L_{OUT}
XSLT Template for formating bookmarks	XBEL Schema	XHTML Schema
PHP Script	Arbitrary String	String not containing "<script>"

Why IRP?

- Typechecking $f :: L_{IN} \to L_{OUT}$?
  - If f is IRP, we can check this by …

f is type-correct

$\Leftrightarrow f(L_{IN}) \subseteq L_{OUT}$

$\Leftrightarrow L_{IN} \subseteq f^{-1}(L_{OUT})$

(for experts: f is assumed to be deterministic)

FAQ: Why IRP?

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

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

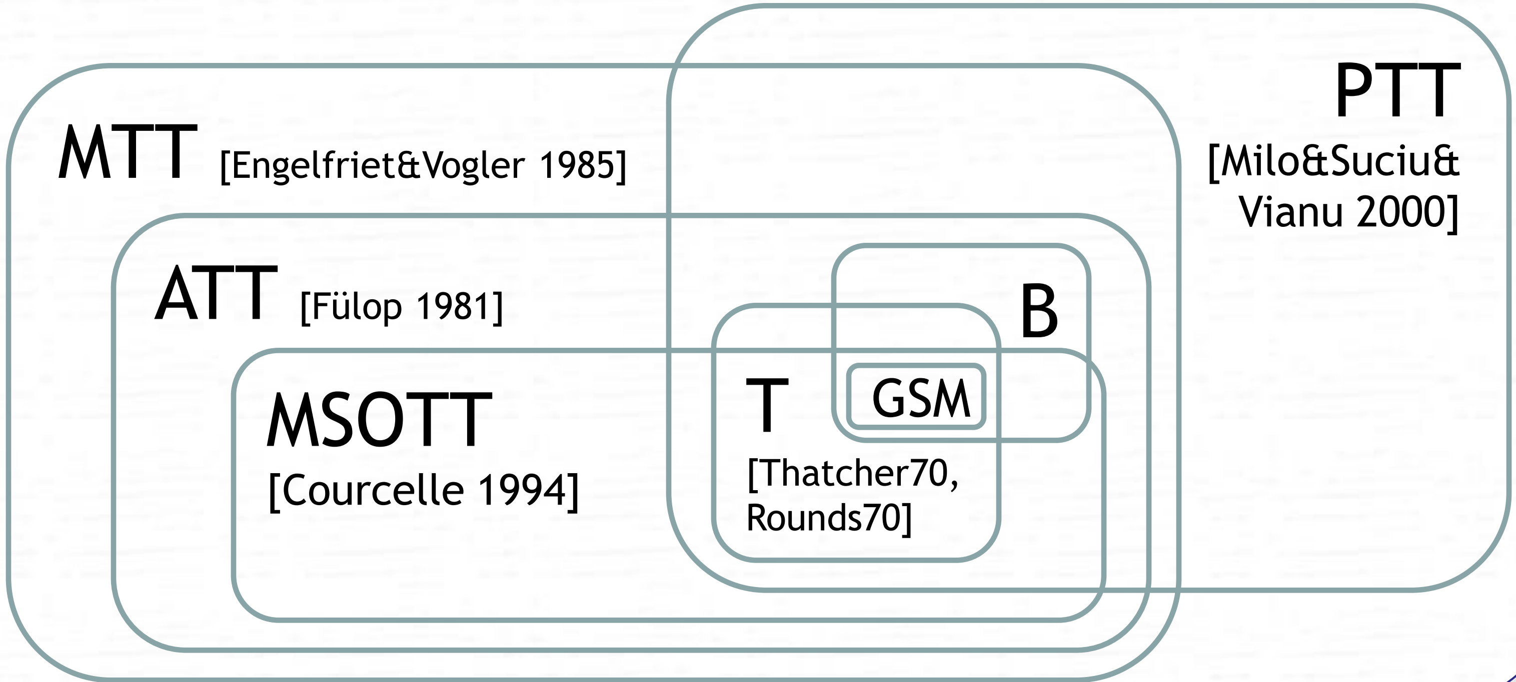
IRP-Based Typechecking

- 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

(note: $X^* := \{f1 \cdot f2 \cdot \ldots \cdot fn \mid n \in Nat,\ fi \in X\}$)

$$MTT^* = PTT^* = ATT^* = \ldots$$



One Example: MTT

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

Syntax

```
MTT ::= FUN … FUN
FUN ::= f(A(x1,…,xn), y1,…,yk) → RHS
RHS ::= C( RHS, … , RHS )
      | f(xi, RHS, …, RHS) | yi
```

Example

```
start( A(x1) )       → double( x1, double(x1, E) )
double( A(x1), y1)   → double( x1, double(x1, y1) )
double( B, y1 )      → C( y1, y1 )
```

Question

- Do they cover all IRP transformations?

- MTT* = PTT* = ATT* = .. **= IRP ?**
  - $\subseteq$ is known
  - $\supseteq$?

(Attribution: I've first heard this question from Sebastian Maneth, who heard it from Keisuke Nakano)

Answer: "No"

- K. Inaba, PPL 2010 Short Presentation

$$\mathrm{tower}(\text{"}\overbrace{a..a}^{n}\text{"}) = \text{"}\overbrace{aa...aa}^{2\hat{}\hat{}n}\text{"}$$

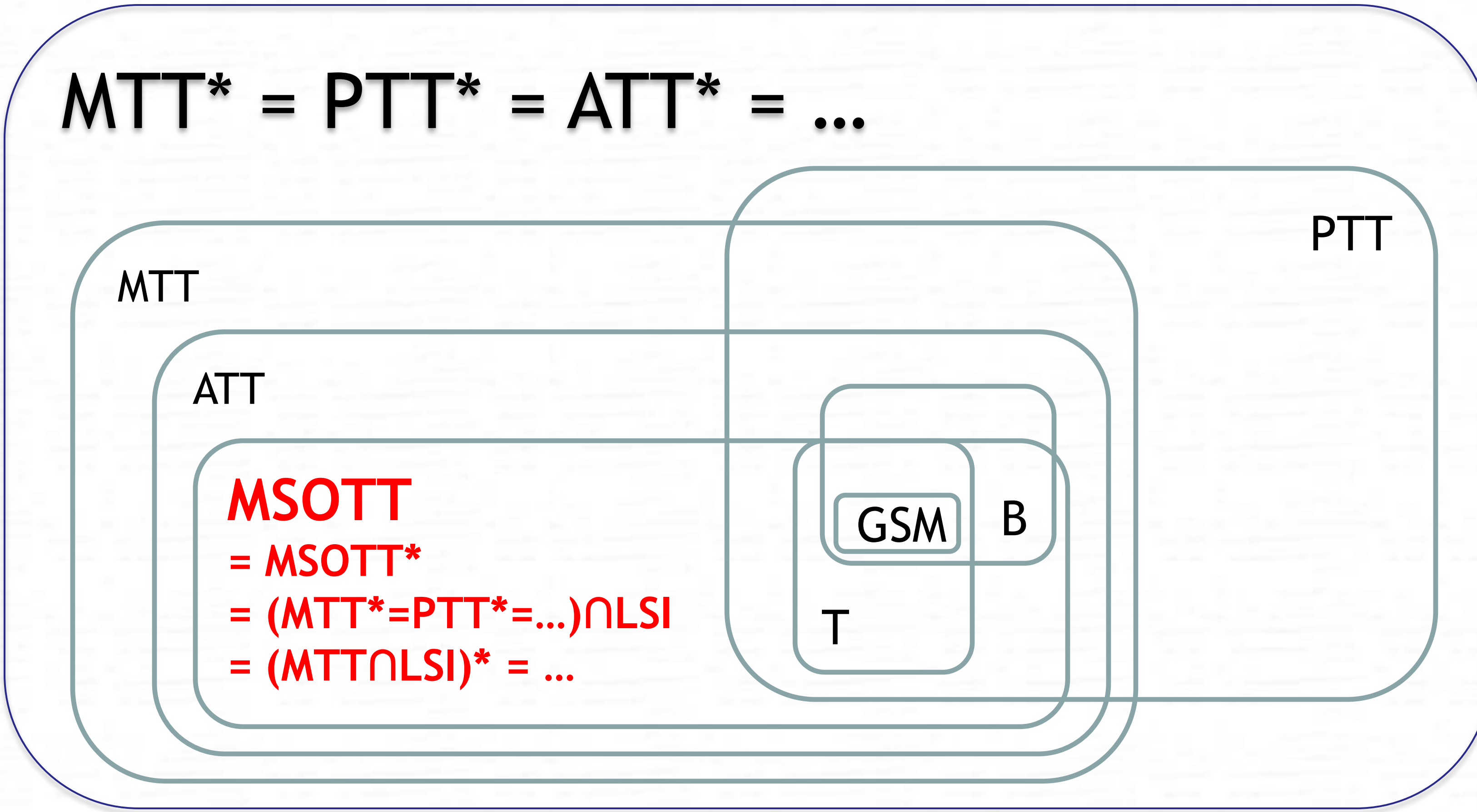
where $2\hat{}\hat{}0 = 1$, $2\hat{}\hat{}(n+1) = 2^{2\hat{}\hat{}n}$

is IRP but not in MTT*

But its growth is tooooooooooo fast!

- Aren't there any "milder" counterexample?

LSI: Linear Size Increase = $\exists c.\ \forall t.\ \mathrm{len}(f(t)) < c \cdot \mathrm{len}(t)$



New Question

- MSOTT
  $= (MTT^*=PTT^*=ATT^*=\ldots)\cap LSI$
  $= IRP \cap LSI$?

  - $\subseteq$ is known
  - $\supseteq$?

Answer: “No“

- There exists a IRP∩LSI transformation that cannot be written in MSOTT

- Main Theorem of This Talk

> ***The class of* IRP∩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 {g1, g2, g3, …} 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: {t1, t2, t3, …}
- Given enumeration {g1, g2, g3, …} of the class
- We construct f as:
  - f(ti) := arbitrary tree except gi(ti)
- Caution!
  - f may not be IRP nor LSI

	t1	t2	t3	t4	…
g1	×				
g2		×			
g3			×		
g4				×	
…					…

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 {g1, g2, g3, …} of the IRP ∩ LSI functions. Then I will show you a function f *not* in the enumeration *but* in IRP∩LSI."**
    - which derives contradiction.

Diagonalization

- We can enumerate all regular languages: {R1, R2, R3, …}
- Given enumeration {g1, g2, g3, …} of the class
- We construct f so that:
  - **f(t) ≠ gi(t) for some**
    $t \leftsquigarrow$ {R1,R2,…,Ri}
  - **f^{-1}(Ri) = *almost* Ri**

	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 R1, R2 ∈ REG then
      - R1 ∩ R2 ∈ REG
      - R1 ∪ R2 ∈ REG
      - ∼R1 ∈ REG
  - “*Slenderness*” is decidable

    [Paun&Salomaa 1993] “Language-Theoretic Problems Arising from Richelieu Cryptosystems”, TCS(116), pp.339-357

Preparation: Slenderness

- A set L of string is slender iff
  - $\exists c.\ \forall n.\ \ \#\{s \mid s \in L,\ len(s)=n\} \ \leqq\ c$
    - {1, 11, 111, 1111, ...} is slender
    - {0, 1, 10, 11, 100, 101, ...} is not slender
    - L1,L2 is slender → L1∪L2 is slender
    - L1,L2 is co-slender → L1∩L2 is co-slender
- Co-slender ⇔ complement is slender
- Not co-slender ⇔ 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.

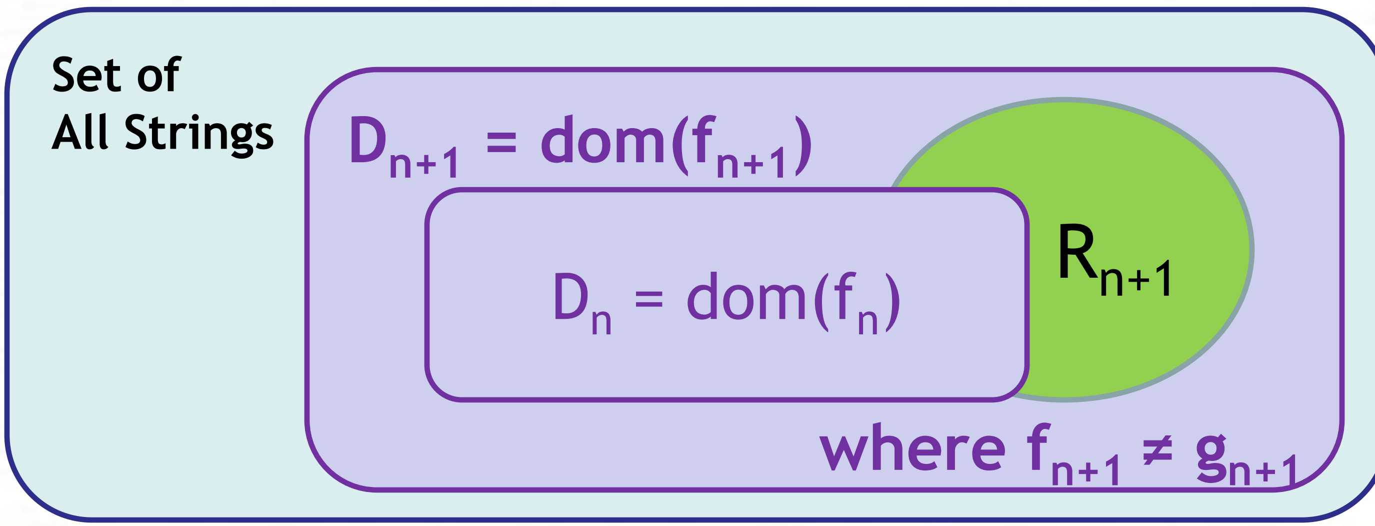
Then we can construct $\{(f_0,D_0), (f_1,D_1), \ldots\}$ such that

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

Proof of the Main

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}

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



Proof of the Main Lemma

D_n is not co-slender. ➔

Take $x, y \in \sim D_n$ s.t. $len(x)=len(y)$ 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

↑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 elems 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 $s \in D_n$
  - if $g_{n+1}(x) = x$
    - $f_{n+1}(x) = y$
    - $f_{n+1}(y) = x$
  - otherwise
    - $f_{n+1}(x) = x$
    - $f_{n+1}(y) = y$
  - $f_{n+1}(s) = s$ 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.

The class of IRP∩LSI *transformations is not recursively enumerable.*

- Suppose it is. By previous lemma,
- let $f = \bigcup_{i \in Nat} f_i$
  - 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 $D_i = dom(f_i)$ eventually
  - f is LSI (in fact, length-preserving)
  - f is IRP
    - next page

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

Contradiction.
Q.E.D.

Notes

- If {g1, g2, … } 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 :: string→string is computable & total
- f^{-1} :: REG→REG is computable & total
- f is length-preserving, IRP, and FRP
- f is not in MSOTT = MTT∩LSI*