The Complexity of Tree Transducer Output Languages

FSTTCS 2008, Bengaluru

The Univ. of Tokyo  Kazuhiro Inaba
NICTA, and UNSW  Sebastian Maneth
“Complexity of Output Languages”

- Given...
  - A language \( L \subseteq T_\Sigma \) (Trees over \( \Sigma \))
  - A relation (nondeterministic translation)
    \( \tau \subseteq T_\Sigma \times T_\Delta \) (from \( T_\Sigma \) to \( T_\Delta \))

- What is the complexity of the language \( \tau(L) \subseteq T_\Delta \)?

  (i.e., for \( t \in T_\Delta \), how is it computationally hard to determine whether \( t \in \tau(L) \) or not?)
Classic Results

- $\tau$: Program of Turing-Machine
  - Undecidable

- $L$: Regular String Language

- $\tau$: Nondeterministic Finite State Transduction
  - $\tau(L)$ is regular!
    - The membership of $\tau(L)$ is solved in $O(n)$ time, $O(1)$ space

- Corollary: for $\tau \in$ Finitely Many Compositions of Nondeterministic FST, $\tau(L)$ is regular
Trees?

- $L$: Regular Tree Language
- $\tau$: Finitely Compositions of Nondet. Finite-State Tree Transducers
  - Beyond Regular Tree Language
    - (Intuitively…) Due to Copying
      - $\tau(t) \rightarrow x(t, t)$ is an instance of FSTT
  - In $\text{DSPACE}(n)$ [Baker1978]
    - i.e., Deterministic Context-Sensitive
Recent Result [Maneth2002, FSTTCS]

- L : Regular Tree Language
- $\tau$: Finite Compositions of Total Deterministic Macro Tree Transducers
  - $\exists \Rightarrow$ Tree Transducers extended with “accumulating parameters” for each state
  - In $\text{DSPACE}(n)$
    - Still, Deterministic Context-Sensitive
Today’s Target!

- $L$: Regular Tree Language
- $\tau$: Finite Compositions of Nondeterministic Macro Tree Transducer

- Is it still context-sensitive? – Yes. $\text{NSPACE}(n)$
- What about the time complexity? – $\text{NP}$-complete
Outline

- What is/Why Macro Tree Transducers?
- Review of the Proof for Deterministic Case
  - “Garbage-free” Lemma
  - “Translation Membership” Problem
- Summary
Macro Tree Transducer (MTT)

- $Q$: Finite Set of States
- $q_0$: Initial State
- $\Sigma$: Input Alphabet
- $\Delta$: Output Alphabet
- $R$: Set of Rewrite Rules of form:
  
  \[
  <q, \sigma(x_1,\ldots,x_k)>(y_1,\ldots,y_m) \rightarrow r
  \]
  
  where $r ::= \delta(r,\ldots,r) \mid <q,x_i>(r,\ldots,r) \mid y_j$
Example of an MTT

- `<q0, a(x)>()` → f( `<q1, x>( a(e) ), <q2,x>() )
- `<q0, b(x)>()` → f( `<q1, x>( b(e) ), <q2,x>() )
- `<q1, a(x)>(y)` → `<q1, x>( a(y) )`
- `<q1, b(x)>(y)` → `<q1, x>( b(y) )`
- `<q1, e>(y)` → y
- `<q2, a(x)>()` → a( `<q2,x>()` )
- `<q2, b(x)>()` → b( `<q2,x>()` )
- `<q2, e>()` → e

<q0, a(b(b(e)))>()
→ f( `<q1,b(b(e))>(a(e)), <q2,a(e)>() )
→ f( `<q1,b(e)>(a(a(e)), <q2,a(e)>() )
→ f( `<q1,e>(a(a(a(e))), <q2,a(e)>() ) → ...
(Choice of Semantics)

- Functional Programming + Laziness + Nondeterminism 😊

- We take the Runtime-Choice Semantics:
  - \(<\text{coin, a}> \rightarrow 0 \mid 1\>
  - \(<\text{twocoin}, a> (y) \rightarrow c(y, y)\>
  - \(<\text{twocoin}, a> (\langle \text{coin, a> () } \rangle) \rightarrow *\>
    - \{ c(0,0), c(0,1), c(1,0), c(1,1) \}

- Because of its composability: \(\text{MTT} ; \text{LT} \subseteq \text{MTT}\)
MTT*(REGT)
= PTT*(REGT)
= ATT*(REGT)
= ...

DtMTT*(REGT)

IO-Hierarchy
Context Free

OI-Hierarchy

MSOTT*(REGT)

T*(REGT)

Regular
Review:

**DSpace(n) Membership for Det. MTTs**

- Given a (fixed) pair of
  - Input regular language $L$ and
  - Composition sequence $\tau_1; \ldots; \tau_n$ of total deterministic MTTs

- and a tree $t$,

- How can we test $t \in (\tau_1; \ldots; \tau_n)(L)$ in linear space wrt $|t|$?
Review:

DSPACE(n) Membership for Det. MTTs

- Guess the input $s \in L$
- Calculate $(\tau_1 ; \ldots ; \tau_n)(s)$
- If $(\tau_1 ; \ldots ; \tau_n)(s) = t$, then $t$ is in the output language!
- Otherwise, try another input tree $s$

Is this a possible output from $\tau_1 ; \ldots ; \tau_n$?
Review:

**DSPACE(n) Membership for Det. MTTs**

- In order to carry out the algorithm in $\text{DSPACE}(|t|)$ …
  - The sizes $|s|, |s_1|, |s_2|, \ldots, |s_n|$ must be linearly bounded by $|t|$  
    - i.e., there must be a constant $c$ independent from $t$ s.t. $|s| \leq c|t|$
  - Each step $\tau$ of the computation must be done in linear space

The translation must have ‘no garbage’!
Review:

**DSPACE(n) Membership for Det. MTTs**

- **‘Garbage-Free’ Lemma**
  - For any input language $L$ and mtt $\tau_1, \ldots, \tau_n$, there exists $L'$ and $\tau'_1, \ldots, \tau'_n$ such that
    \[(\tau_1;\ldots;\tau_n)(L) = (\tau'_1;\ldots;\tau'_n)(L')\]
    and every $\tau'_i$ is ‘non-deleting’ ($|in| \leq 2|out|$)

- **Linear Time (and Space) Computation**
  - For any total deterministic mtt $\tau$ and a tree $s$, $\tau(s)$ can be computed in time $O(|s| + |\tau(s)|)$
    (already known as a folklore result)
NSPACE(n)/NP Output Membership for Nondeterministic MTTs

- Guess the input $s \in L$ and all the intermediate trees $s_1, \ldots, s_{n-1}$
- Check whether $(s, s_1) \in T_1$, $(s_1, s_2) \in T_2$, \ldots, $(s_{n-1}, t) \in T_n$
- If it is, then $t$ is in the output language!
- Otherwise, try another $s$, $s_2$, \ldots, $s_{n-1}$
Key Lemmas


- NP/NSPACE(n) “Translation Membership” for a single mtt translation
Key Lemma (1):

Basic Idea

- “Factor out” the deletion

\[ T_1 ; T_2 \equiv T_1 ; (D ; T'_{2}) \]
\[ \equiv (T_1 ; D) ; T'_{2} \]
\[ \equiv \rho_1 ; T'_{2} \]

- Decompose \( T_2 \) to ‘deleting part’ \( D \) and ‘nondeleting’ \( T'_{2} \)
- Associativity
- Compose \( T_1 \) with \( D \)
Three Types of Deletion

- **“Erasure”**
  - \( <q,\sigma>(y_1, y_2) \rightarrow y_1 \)
  - No new output node is generated at this \( \sigma \) node. Only returning its parameter.

- **“Input-Deletion”**
  - \( <q, \sigma(x_1, x_2)>() \rightarrow \delta( <q, x_1>() ) \)
  - Discarding the “\( x_2 \)” subtree!

- **“Skipping”**
  - \( <q, \sigma(x_1)>() \rightarrow <q, x_1>() \)
  - Occurs only at monadic node. No new output is generated here. Just going down to its child node.

Lemma:
If no erasing, input-deleting, or skipping rule is used during the computation, then \(|\text{in}| \leq 2|\text{out}|\)
Eliminating
The Three Types of Deletion

- Achieved by heavily manipulating the rules
  - For details, please consult the paper

- One of the difficulties compared to the deterministic case: **Inline-Expansion**
  - \(<q, a> (y) \rightarrow y\)
  - \(<q, b(x_1,x_2)> \rightarrow c( <p,x_1>(<q,x_2>(e)) )\)
    (Assume we know that ‘b’’s child is always ‘a’)
  - \(<q, b(x_1,x_2)> \rightarrow c( <p,x_1>( e ) )\)
With Nondeterminism, Inline-Expansion is Not Easy

- \( \langle q, a() \rangle \rightarrow e \)
- \( \langle q, a() \rangle \rightarrow f \)
- \( \langle q, b(x)() \rangle \rightarrow \langle p, x\rangle(\langle q, x() \rangle) \)
- \( \langle p, a\rangle(y) \rightarrow c(y, y) \)

Different Translation!

- \( \langle q, b(a)() \rangle \rightarrow \langle p, a\rangle(\langle q, a() \rangle) \)
- \( \rightarrow c(\langle q, a() \rangle, \langle q, a() \rangle) \)
- \( \rightarrow c( e, f ) \)
Solution:
“MTT with Choice and Failure”

- We have extended MTTs with “inline” nondeterminism
  - Allows inline-expansion for free!
  - Actually, we prove the output language complexity for mtt-cfs

\[
\begin{align*}
&<q, a>() \rightarrow e \\
&<q, a>() \rightarrow f \\
&q, b(a)() \rightarrow <p,a>(+ (e,f)) \\
&c(+ (e,f), + (e,f)) \rightarrow c(e, f) \\
&q, b(x)() \rightarrow <p,x>(+ (e,f)) \\
&p, a(y) \rightarrow c(y, y)
\end{align*}
\]
Key Lemma (2):
“Translation Membership” of single $\tau_i$

- Given a pair $(s_{i-1}, s_i)$ of trees, we can determine whether $(s_{i-1}, s_i) \in \tau_i$ in linear-space & polynomial time wrt $|s_{i-1}| + |s_i|$ in nondet. Turing machine

- Naively Applying the folklore deterministic computation takes $O(|s_{i-1}| + |\tau(s_{i-1})|)$ time/space → New Idea is Necessary
“Translation Membership” of single $\tau_i$

- Naively Applying the Linear Time Computation for Deterministic MTTs:
  - Fails.
  - It relied on the decomposition of an MTT into Linear MTTs (each input variable $x_i$ occurs at most once in each rule),
  - ...and the fact that deterministic linear MTTs read each input node at most once,
  - ...which allows to compress the output tree as a DAG for both saving space and sharing computations.

- Need More Sophisticated Compression!

- Bad. Nondet. Linear MTTs may read each node multiple times

- OK. Similar decomposition works also for Nondet. MTTs

- Linear
Example: Linear Nondet. MTT
Reading Some Node Twice

- \( <q, b(x_1, x_2)>() \) \rightarrow \( <p, x_1>(<q, x_2>() ) \)
- \( <q, a>() \) \rightarrow \ e
- \( <q, a>() \) \rightarrow \ f
- \( <p, a>(y) \) \rightarrow \ g(y, y) \)
Solution: Compression by Context-Free Tree Grammar

- The set all outputs $\tau_i(s_{i-1})$ of an MTT can be represented by a CFTG of size proportional to $|s_{i-1}|$ [MB04]

- Navigation (up, 1st child, nextsibl) on the compressed representation is efficient for linear mtts
Summary

- Composition sequence $\tau_1 \ldots \tau_n$ of mtts can be converted to an equivalent ‘garbage-free’ composition.
- Translation Membership of any mtt is in NP/NSPACE($n$).

$\rightarrow$ Altogether, the output language complexity of mtt-compositions is NP/NSPACE($n$).
- Corollary: OI-hierarchy, PTT*(REGT), ATT*(REGT), … is in NP/NSPACE($n$).
- Current Status (Unpublished): NSPACE($n$) $\rightarrow$ DSPACE($n$).