Pushdown and Deterministic Transduction

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

AMBIGUITY OF THE SOURCE GRAMMAR

The transduction grammars or schemes of practical interest are one-valued.

If the source grammar is ambiguous → two or more syntax trees
different source and destination trees
different images

source gram. G_2	dest. gram. G_1
$E \rightarrow \text{add}\ \ T\ E$	$E \rightarrow T + E$
$E \rightarrow T$	$E \rightarrow T$
$T \rightarrow \text{mult}\ \ F\ T$	$T \rightarrow F \times T$
$T \rightarrow F$	$T \rightarrow F$
$F \rightarrow E$	$F \rightarrow (E)$
$F \rightarrow i'$	$F \rightarrow i$

G_2 is ambiguous as it admits the following circular derivation

$$E \Rightarrow T \Rightarrow F \Rightarrow E$$

$$E \Rightarrow T \Rightarrow F \Rightarrow i',$$
$$E \Rightarrow T \Rightarrow F \Rightarrow E \Rightarrow T \Rightarrow F \Rightarrow i',\ldots$$

different transductions:

$$E \Rightarrow T \Rightarrow F \Rightarrow i',$$
$$E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow (T) \Rightarrow (F) \Rightarrow (i'),\ldots$$

Even if the source grammar is not ambiguous, the transduction may be multi-valued.

EXAMPLE

$$S \to \frac{a}{b} S \mid \frac{a}{c} S \mid \frac{a}{d}$$

$$G_1 = \{S \to aS \mid a\}$$ is not ambiguous

$$\tau(aa) = \{bd, cd\}$$

PROPERTY

Let T be a transduction grammar (or scheme) such that:

1) the underlying source grammar G_1 is not ambiguous
2) every rule of G_1 corresponds to one rule of T

Then the transduction defined by T is one-valued.

If the transduction grammar T is ambiguous, then also the underlying source grammar G_1 is so. The opposite implication does not hold, in general.

<u>EXAMPLE: unambiguos transduction grammar, ambiguous underlying source grammar.</u>

$$S \rightarrow \frac{\text{if}}{\text{if}} \frac{c}{c} \frac{\text{then}}{\text{then}} S \frac{\varepsilon}{\text{end\_if}} \mid \frac{\text{if}}{\text{if}} \frac{c}{c} \frac{\text{then}}{\text{then}} S \frac{\text{else}}{\text{else}} S \frac{\varepsilon}{\text{end\_if}} \mid a$$

To design a compiler, multi-valued transduction grammars should be avoided, as they may cause multiple transductions.

TRANSDUCTION GRAMMAR AND PUSHDOWN TRANSDUCER AUTOMATON

To compute a transduction defined by means of a transduction grammar, the transducer automaton need have a unbounded pushdown stack memory.

A pushdown transducer (or pushdown IO automaton) is a recognizer pushdown automaton empowered with a one-way write head and an output tape. It can write a character at each transition.

DEFINITION – pushdown transducer

Eight entities:

Q	state set
Σ	source (inpout) alphabet
Γ	stack (memory) alphabet
Δ	destination (output) alphabet
δ	transition and output function
q_0	initial state
Z_0	initial stack symbol
F	subset of final states

Domain of δ: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$

Image of δ: $Q \times \Gamma^* \times \Delta^*$

Meaning: if $(q'', \gamma, y) = \delta(q', a, Z)$ the transducer, from the current state q', when reads a from the input tape and pops Z from the stack, moves to state q'', pushes γ onto tha stack and writes y onto the output tape.

The end condition may be defined by final state or by empty stack.

FROM THE TRANSDUCTION GRAMMAR TO THE AUTOMATON

Transduction schemes and automata are equivalent representations of the same transduction relation. The former is generative, the latter procedural. It is always possible to convert each into the other one.

PROPERTY: a transduction relation is defined by a transduction grammar (or scheme) if and only if it is defined by a pushdown transduction automaton.

CONSTRUCTION OF THE (INDETERMINISTIC) PREDICTIVE PUSHDOWN TRANSDUCER STARTING FROM THE TRANSDUCTION GRAMMAR

Formalize the correspondence between grammar rule and automaton move.

C is the set of the pairs aside (input char, output string):

$$\frac{\varepsilon}{v}, v \in \Delta^{+} \qquad \frac{b}{w}, b \in \Sigma, w \in \Delta^{*}$$

(the automaton is not deterministic, in general; by adding finite states it may be sometimes turned into deterministic form, though not always)

	Rule	Move	Comment
1	$A \rightarrow \frac{\varepsilon}{v} B A_1 \ldots A_n$ $n \geq 0, v \in \Delta^+, B \in V,$ $A_i \in (C \bigcup V)$	**if** (top = A) **then** **write** (*v*) **pop** **push** ($A_n \ldots A_1 B$)	write output string *v* push symbols B $A_1 \ldots A_n$
2	$A \rightarrow \frac{b}{w} A_1 \ldots A_n$ $n \geq 0, b \in \Sigma, w \in \Delta^*,$ $A_i \in (C \bigcup V)$	**if** (cur_char = *b* Λ top = A) **then** **write** (*v*) **pop** **push** ($A_n \ldots A_1$) **shift input head**	read input char *b* write output string *w* push symbols $A_1 \ldots A_n$
3	$A \rightarrow B A_1 \ldots A_n$ $n \geq 0, B \in V,$ $A_i \in (C \bigcup V)$	**if** (top = A) **then** **pop** **push** ($A_n \ldots A_1$)	push symbols $A_1 \ldots A_n$
4	$A \rightarrow \frac{\varepsilon}{v} \quad v \in \Delta^+$	**if** (top = A) **then** **write** (*v*) **pop**	write output string *v*

	Rule	Move	Comment
5	$A \rightarrow \varepsilon$	**if** (top = A) **then** **pop**	
6	for every pair $\frac{\varepsilon}{v} \in C$	**if** (top = ε / *v*) **then** **write** (*v*) **pop**	write output string *v*
7	for every pair $\frac{b}{w} \in C$	**if** (cur_char = *b* Λ top = *b* / *w*) **then** **write** (*w*) **pop** **shift input head**	read input char *b* write output string *v*
8		**if** (cur_char = -| Λ stack empty) **then** **accept** **halt**	accept and end

EXAMPLE – extend to a transducer the recognizer of the language $L = \{a^* a^m b^m \mid m > 0\}$ - $\tau(a^k a^m b^m) = d^m c^k$

	Rule	Move
1	$S \rightarrow \frac{a}{\varepsilon} S \frac{\varepsilon}{c}$	**if** (cur_char = *a* Λ top = S) **then pop**; **push** (ε / *c*S); **shift input head**
2	$S \rightarrow A$	**if** (top = S) then **pop**; **push** (A)
3	$A \rightarrow \frac{a}{d} A \frac{b}{\varepsilon}$	**if** (cur_char = *a* Λ top = A) then **pop**; **write** (*d*); **push** (*b* / εA); **shift input head**
4	$A \rightarrow \frac{a}{d} \frac{b}{\varepsilon}$	**if** (cur_char = *a* Λ top = A) then **pop**; **write** (*d*); **push** (*b* / ε); **shift input head**
5	– – –	**if** (top = ε / *c*) then **pop**; **write** (*c*);
6	– – –	**if** (cur_char = *b* Λ top = *b* / ε) then **pop**; **shift input head**
7	– – –	**if** (cur_char = -| Λ empty stack) then **accept**; **halt**

Not every transduction defined by a transduction free grammar can be computed in a deterministic way.

EXAMPLE – a inherently non-deterministic transduction
(typical example of something not to do)

No det. pushdown transducer automaton can compute the transduction shown aside.

$$\tau(u\text{-|}) = u^R u, \quad u \in \{a, b\}^*$$

A det. transducer should push the string *u* and, soon after reading the terminator, it should pop *u* to output the mirror image. But in this way it looses any information about *u*, and therefore it will not be able to output the direct image of *u* itself.

For practical purposes only the deterministic cases are of importance. Suitable analysis algorithms follow.

SYNTAX ANALYSIS AND ON-LINE TRANSDUCTION

The transduction can be constructed directly and efficiently. A syntactic analyzer is transformed into the corresponding syntactic transducer.

Given a transduction grammar, suppose the underlying source grammar allows the construction of a deterministic syntax analyzer. To compute the transduction, execute the syntax analysis and, as the syntax tree is built, output the corresponding transduction.

TOP-DOWN ANALYZER (LL): can always be transformed into a transducer

BOTTOM-UP ANALYZER (LR): to be transformed into a transducer, grammar rules need satisfy a special restrictive condition - the write action can only occur at the end of the production (when the analyzer performs reduction)

TOP-DOWN DETERMINISTIC TRANSDUCTION

If the source grammar is LL(k), complete the parser with write actions and obtain the corresponding transducer.

This construction is very simple. The transducer can also be designed as a set of recursive syntactic procedures.

EXAMPLE: translate a string into the mirror image

The source grammar is LL(1), and the lookahead sets are {*a*} {*b*} and {-|}.

$$S \to \frac{a}{\varepsilon} S \frac{\varepsilon}{a} \mid \frac{b}{\varepsilon} S \frac{\varepsilon}{b} \mid \varepsilon$$

Automaton moves are:

stack	cc = *a*	cc = *b*	cc = -|	ε
S	pop; push (ε / *a*S)	pop; push(ε / *b*S)	pop	
ε / *a*				write (*a*)
ε / *b*				write (*b*)

EXAMPLE – translate infix expressions to postfix – show recursive version as well

Source language: arithmetic expressions with pharenteses.

The transduction converts the infix form to the postfix form.

Example: v x (v + v) v v v add mult

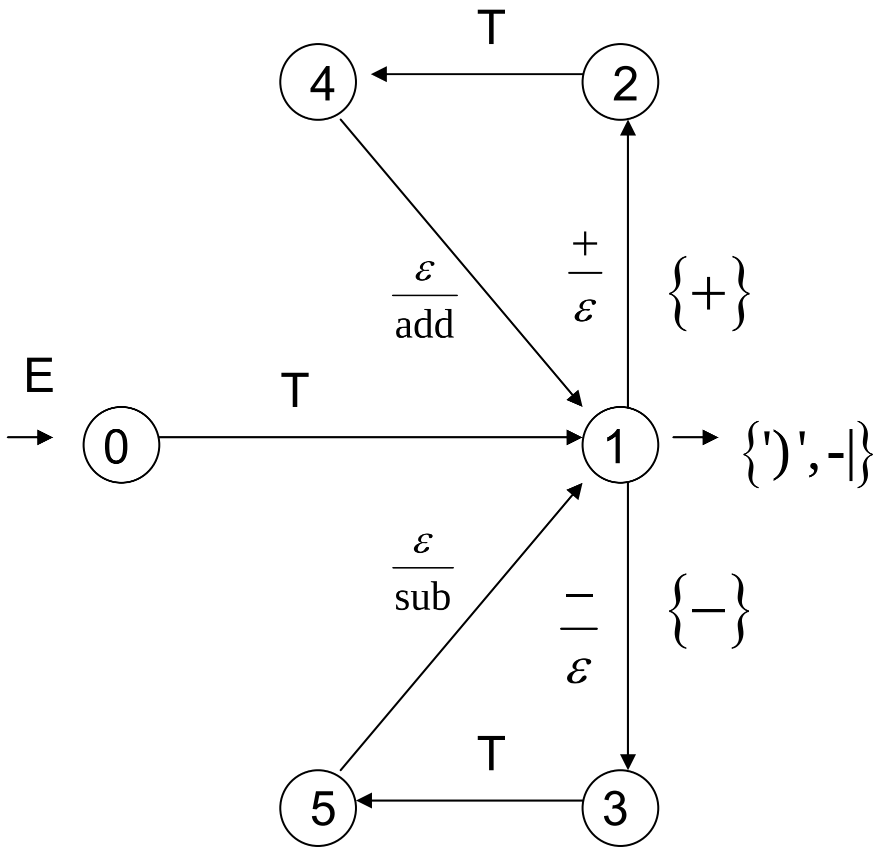
EBNF transduction grammar to postfix form:

$$E \to T\left(\frac{+}{\varepsilon}T\,\frac{\varepsilon}{\text{add}}\mid\frac{-}{\varepsilon}T\,\frac{\varepsilon}{\text{sub}}\right)^*$$

$$T \to F\left(\frac{\times}{\varepsilon}F\,\frac{\varepsilon}{\text{mult}}\mid\frac{\div}{\varepsilon}F\,\frac{\varepsilon}{\text{div}}\right)^*$$

$$F \to \frac{v}{v}\mid\frac{'('}{\varepsilon}E\,\frac{')'}{\varepsilon}$$

$$\Sigma=\{+,\times,-,\div,(,),v\},\quad \Delta=\{\text{add},\text{sub},\text{mult},\text{div},\text{v}\}$$



automaton version

```
procedure E
        call T
        while (cc ∈ {+, -}) do
                case (cc = '+') begin
                        cc := next
                        call T
                        write ('add')
                end
                case (cc = '-') begin
                        cc := next
                        call T
                        write ('sub')
                end
                otherwise error
                end case
        end do
end E
```

recursive version
(with syntatctic procedures)

Bibliography

– S. Crespi Reghizzi, *Linguaggi Formali e Compilazione*, Pitagora Editrice Bologna, 2006

– Hopcroft, Ullman, *Formal Languages and their Relation to Automata*, Addison Wesley, 1969

– A. Salomaa – *Formal Languages*, Academic Press, 1973

– D. Mandrioli, C. Ghezzi – *Theoretical Foundations of Computer Science*, John Wiley & Sons, 1987

– L. Breveglieri, S. Crespi Reghizzi, *Linguaggi Formali e Compilatori: Temi d'Esame Risolti,* web site (eng + ita)