Transduction Regular and Syntactic

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

SYNTAX TRANSDUCTION

1. Abstract definition of the correspondence between two formal languages.
2. Translation based on local transformations of the surce text, by replacing each character with another one (or with an entire string), depending on a table that specifies how to transiltterate the source alphabet into the destination one.
3. Finite transductions defined by means of regular expressions or finite transducers (finite state automata enhanced with the possibility of outputting a string).
4. Syntactic schemes or transduction grammars that use a free grammar to define the source and destination languages (instead of the regular language of point 3). The abstract transducer is a pushdown automaton, and uses the same parsing algorithms as seen in previous lectures.

SYNTAX DIRECTED SEMANTIC TRANSDUCTION

5. Attribute grammars and semantic transductions directed by syntax: this approach is semiformal and exploits the previous concepts to design compilers. Attribute grammars specifiy, rule by rule, the semantic functions to apply and compute the entire transduction in a compositional way.

The addition of attributes and semantic functions to the finite transducers is well documented for instance in lexical analysis, in type checking, in the transduction of conditional instructions and in the semantic directed syntax analysis.

ANALOGIES OF FORMAL LANGUAGE THEORY AND TRANSDUCTION THEORY

As for set-theoretic definitions: the set of the language phrases corresponds to the set of the pairs (source string, destination string) that model the transduction relation.
As for the generative definition: the language grammar becomes a transduction grammar, which generates pairs of phrases (source and destination).
As for the operative definition: the finite state automaton or the pushdown automaton become a transducer automaton or a syntax analyzer that compute the transduction.

1. TRANSDUCTION RELATION AND TRANSDUCTION FUNCTION

Given two alphabets Σ (source) and Δ (destination), a TRANSDUCTION is a correspondence between source and destination strings, which can be modeled as a mathematical binary relation between the two universal languages Σ^* and Δ^*.
A transduction is a subset of the cartesian product $\Sigma^* \times \Delta^*$.

THE TRANSDUCTION RELATION ρ
is a set of pairs (x, y)
with $x \in \Sigma^*$ and $y \in \Delta^*$
L_1 is the projection of ρ over the 1st comp.
L_2 is the projection of ρ over the 2nd comp.

$$\rho = \{(x, y), ...\} \subseteq \Sigma^* \times \Delta^*$$

$$L_1 = \{x \in \Sigma^* \mid \text{ for some } \quad y : (x, y) \in \rho\}$$

$$L_2 = \{y \in \Delta^* \mid \text{ for some } \quad x : (x, y) \in \rho\}$$

or one can proceed as follows:

A TRASNSDUCTION τ
IS A FUNCTION
(in general multi-valued)

$$\tau : \Sigma^* \rightarrow \quad \text{for some} \quad \Delta^*;$$

$$\tau(x) = \{y \in \Delta^* \mid (x, y) \in \rho\}$$

$$L_2 = \tau(L_1) = \{y \in \Delta^* \mid \exists x \in \Sigma^* : y \in \tau(x)\}$$

$$\tau^{-1}(y) = \{x \in \Sigma^* \mid y \in \tau(x)\}$$

The transduction function τ
can always be assumed
to be total.

Depending on the mathematical properties of the function, one has the following transduction cases:

a) total: every string over the source alphabet has an image, of one or more destination strings (and not excluding the empty string)
b) partial: some source strings may not have any image, that is may not be translated into any destination string (and excluding also the empty string)
c) one-valued: no source string has two ore more destination strings as image; it is possible to go back from a destination string to a unique source string, but not always
d) multi-valued: some source strings have two or more (possibly also infinitely many) destination strings as image

The remaining cases make sense only if the transduction function is total and one-valued:

e) injective: different source strings have different image strings; this also means that every string over the destination alphabet is the image of one source string at most, or of none
f) surjective: every string over the destination alphabet is the image of at least one source string, or of more
g) one-to-one or bijective: there is a one-to-one correspondence between source and destination strings (of course a transduction is one-to-one if and only if it is both injective and surjective); clearly a one-to-one transduction is invertible and its inverse transduction is one-to-one as well

2. TRANSLITTERATION

Define a transduction as the repetition, character by character, of a punctual transformation.

The simplest case is TRANSLITTERATION (also called HOMOMORPHISM). Every source character is transduced into a corresponding destination character (or more generally into a string, not excluding the empty string).

Transiltteration is certainly a one-valued transduction, but as said before its inverse transduction is partial, in general, and may be multi-valued. If the translitteration can cancel some source character (replacing it by the empty string), its inverse is certainly multi-valued (not only, infinitely-valued !).

The essence of translitteration is that each character is translated independently of how the sourrounding characters are translated. This notion of transduction is clearly weak and obviously insufficient for compiling all real technical languages.

3. REGULAR (or RATIONAL) TRANSDUCTION

In order to define a transduction relation one can exploit regexps: in this case the regular operators union, concatenation and star are applied in parallel to pairs of strings (source, destination) rather than to individual strings.

EXAMPLE – coherent translitteration of an operator

The source text is a list of unary integers separated by the operator /. The transduction translitterates the operator / into the alternative symbols : or division, but always in the same way. The integers are in unary notation.

$$\Sigma = \{1, /\} \quad \Delta = \{1, :, \div\}$$

$$(3/5/2\ ,\ 3{:}5{:}2)\ ,\ (3/5/2\ ,\ 3 \div 5 \div 2)$$

$$(1,1)^{+}((/,:)(1,1)^{+})^{*} \cup (1,1)^{+}((/,\div)(1,1)^{+})^{*}$$

$$\left(\frac{1}{1}\right)^{+}\left(\frac{/}{:}\left(\frac{1}{1}\right)^{+}\right)^{*} \cup \left(\frac{1}{1}\right)^{+}\left(\frac{/}{\div}\left(\frac{1}{1}\right)^{+}\right)^{*}$$

$$\frac{1}{1}\frac{/}{\div}\frac{1}{1}\frac{1}{1} \quad (1/11,\ 1 \div 11)$$

DEFINITION – regular or rational transduction expression (regexprt)

A regular or rational transduction expression (regexprt) *r* is a regexp containing the union, concatenation and star (or cross) operators, the terms of which are pairs (*u*, *v*) of strings, possibly empty, over the source and destination alphabets, respectively.

$$\tau = \{(x, y), \ldots\} \subseteq \quad \text{finite subsets of} \quad \left(\Sigma^* \times \Delta^*\right)$$

where:

$Z \subset \Sigma^* \times \Delta^*$ is the set of the pairs (*u*, *v*) contained in the regexprt *r*.

The pairs (*x*, *y*) of corresponding source and destination strings are such that:

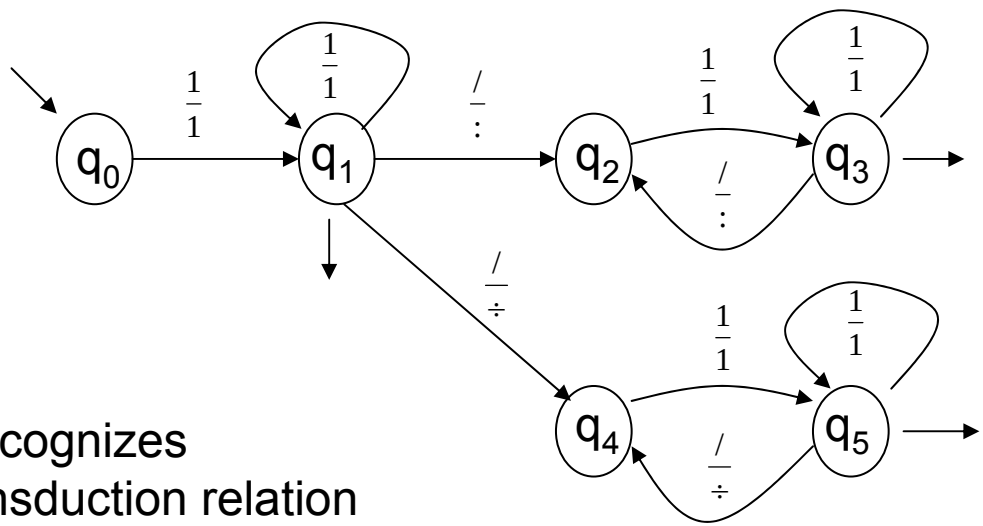
- there exists a string $z \in Z^*$ belonging to the regular set *r*
- *x* and *y* are respectively the projection of *z* on the 1st and the 2nd component

The set of the source strings defined by a regexprt is a regular language. The same holds for the set of the destination strings.

It is not granted, instead, that a relation from source language to destination language, both regular languages, may be definable by means of a regexprt.

TWO-TAPE RECOGNIZER AUTOMATON

Think of the set Z of the pairs contained in the regexprt as terminal symbols; then it is simple to associate to it a recognizer automaton with two input tapes.



Previous EXAMPLE (continued):
The finite state automaton aside recognizes that (11/1, 11:1) belongs to the transduction relation and that (11/1,1:1) does not.

The concept of regexprt and of two-tape recognizer is a method for defining the transduction, and also for building transduction functions in a rigorous way. It also allows to deal with both lexical and syntax analyzers in a uniform way.

FORMS OF THE TWO-TAPE AUTOMATON

When the labels of the arcs of a TWO-TAPE automaton are projected over the first component (source alphabet), one obtains a pure recognizer (with only one input tape) over the source alphabet Σ; this is called **underlying input automaton**, and it is the recognizer of the pure source language.

In describing the two-tape recognizer one can always suppose, without any loss of generaliry, that every move may read one, and only one, character in the source tape.

DEFINITION: TWO-TAPE AUTOMATON

An automaton with two input tapes (two-tape automaton) has, similarly to an ordinary automaton with only one input tape, a state set Q, an initial state q_0, and a subset F of final states. The transition function is:

$$\delta : (Q \times \Sigma \times \Delta^*) \rightarrow \text{finite subsets of } Q$$

If $q' \in \delta(q, a, u)$, when the automatons is in state q and reads a and u in the first and second tape, respectively, it may go to the next state q'. The acceptance condition is the same as that holding in the case of ordinary one-input recognizers.

Normal form: a move may read only one character in either tape, not in both ones. The labels of the arcs can only be of either types:

$$\frac{a}{\varepsilon}, \frac{\varepsilon}{a} \qquad a \in \Sigma \bigcup \Delta$$

More coincisely: $\frac{(a)^* b}{d} \mid \frac{(a)^* c}{e}$ equivalent to: $\frac{(a)^*}{\varepsilon}\frac{b}{d} \mid \frac{(a)^*}{\varepsilon}\frac{c}{e}$

PROPERTY – the familes of regular transductions defined by regexprts and two-tape automata (in general non-deterministic) coincide.

A regexprt defines a regular language R having as alphabet the cartesian product of the source and destination alphabets

$$(u,v) \equiv \frac{u}{v} \text{ where } u \in \Sigma^*, v \in \Delta^*$$

By separating the source and destination components in the regexprt, one obtains an interesting formulation of the regular transduction relation.

NIVAT THEOREM - the following conditions are equivalent:

1. The transduction relation ρ is regular with $\rho \subseteq \Sigma^* \times \Delta^*$

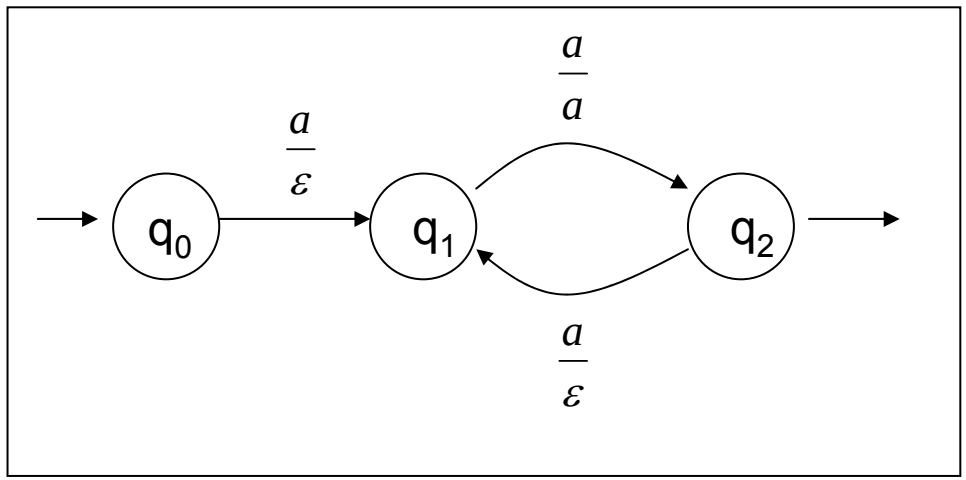
2. There exists a regular language R over the alphabet C and two alphabetic homomorphims (or translitterations) h_1: C → Σ and h_2: C → Δ such that: $\rho = \{(h_1(z), h_2(z)) \mid z \in R\}$

3. If the two alphabets Σ and Δ are disjoint, there exists a language R over the union Σ U Δ such that: $\rho = \{(h_\Sigma(z), h_\Delta(z)) \mid z \in R\}$
where h_Σ and h_Δ are the projections of Σ U Δ over the component source and destination alphabets Σ and Δ, respectively.

EXAMPLE: HALVING – the image string is the halved source string

Transduction relation: $\{(a^{2n}, a^n) \mid n \geq 1\}$ regexprt $\left(\frac{aa}{a}\right)^+$

An equivalent two-tape automaton:



Nivat theorem
(2nd statement)

The two alphabetic hom.s generate the required transduction. For instance, if $z = cdcd \in R$ one has $h_1(z) = aaaa$ and $h_2(z) = aa$

$\left(\frac{a}{\varepsilon}\frac{a}{a}\right)^+ \quad \frac{a}{\varepsilon} = c \quad \frac{a}{a} = d \quad C = \{c, d\}$

$R = (cd)^+$ homomorphisms h_1 h_2

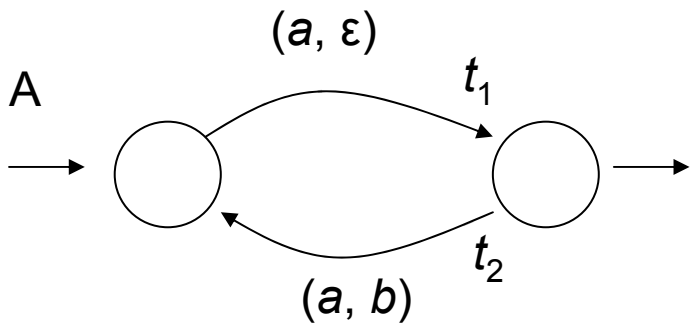
c	a	ε
d	a	a

Nivat theorem
(3rd statement)

Projecting over the source and destination alphabets defines the strings that correspond to each other in the transduction.

$$\Sigma = \{a\}\ \Delta = \{b\} \quad \{(a^{2n}, b^n) \mid n \geq 1\} \quad \left(\frac{a}{\varepsilon}\frac{a}{b}\right)^+$$

$$R \subseteq (\Sigma \cup \Delta)^* \quad R = (a\varepsilon ab)^+ = (aab)^+$$

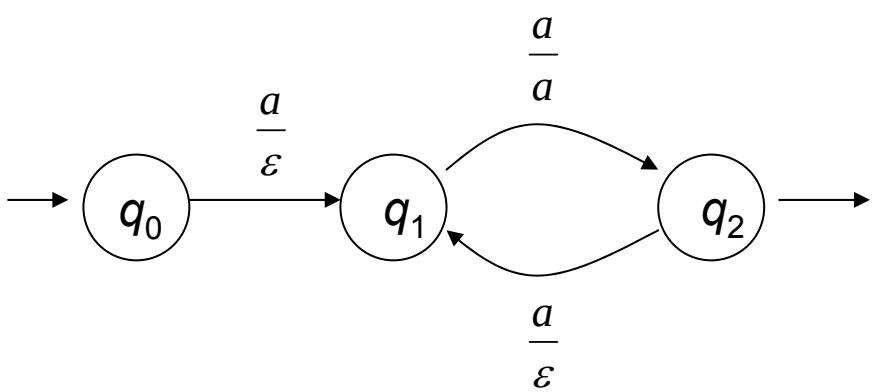


$$T = \{t_1, t_2\} \quad L(A) = (t_1 t_2)^+$$

$$\Pi_\Sigma(t_1) = a \quad \Pi_\Sigma(t_2) = a$$

$$\Pi_\Delta(t_1) = \varepsilon \quad \Pi_\Delta(t_2) = b$$

The correspondence between the finite state automaton model and the right linear grammar model allows to design a **transduction grammar**, along with the two-tape automaton and the regexprt.



$$S \to \frac{a}{\varepsilon} Q_1$$

$$Q_1 \to \frac{a}{a} Q_2$$

$$Q_2 \to \frac{a}{\varepsilon} Q_1 \mid \frac{\varepsilon}{\varepsilon}$$

A grammar-based definition may be preferable in the context of syntax transductions, where the syntactic support is a free grammar.

4. TRANSDUCTION FUNCTION AND TRANSDUCER AUTOMATON

An automaton can be used as an algorithm to compute a transduction function. Define a new model: transducer automaton or input-output automaton (IO automaton). It reads a source string in the input tape and writes a destination string on the output tape.

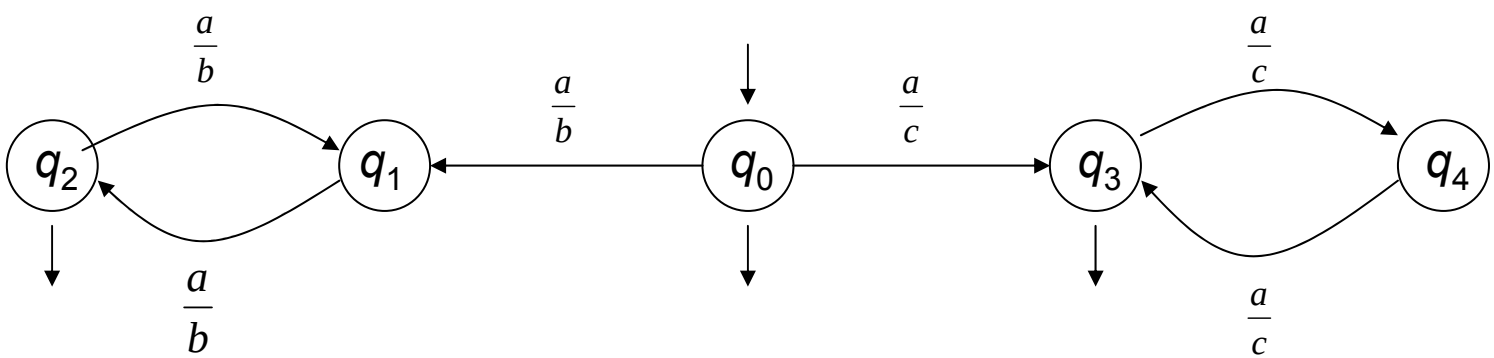
The cases deserving most interest are those of a one-valued transduction function and of a transduction function computable in a deterministic way.

EXAMPLE – a non-deterministic transduction

Is a one-valued transduction, which can not be computed in a deterministic way by means of a deterministic finite state IO-automaton.

$$\rho = \left\{(a^{2n}, b^{2n}) \mid n \geq 0\right\} \cup \left\{(a^{2n+1}, c^{2n+1}) \mid n \geq 0\right\}$$

$$\tau(a^n) = \begin{cases} b^n, n \text{ even} \\ c^n, n \text{ odd} \end{cases} \quad \text{regexprt} \quad \left(\frac{a^2}{b^2}\right)^* \cup \frac{a}{c}\left(\frac{a^2}{c^2}\right)^*$$



The two-tape automaton is det., but the IO automaton is not det.

$$\tau(aa) = bb$$

Given the input string *aa*, two computations are possible: $q_0 \rightarrow q_1 \rightarrow q_2$ or $q_0 \rightarrow q_3 \rightarrow q_4$, but only the former one is valid and reaches a final state.

To compute the transduction function, the input tape is read and the output tape is written. This way of doing is efficient. As soon as the input string is fully scanned, the transducer should be in a final state and can write the last suffix on the output tape, depending on the specific final state it has reached.

DEFINITION – deterministic model of a sequential transducer. A finite state sequential det. transducer T, or IO automaton, is a deterministic machine defined as follows.
It has a finite set Q of states, a source alphabet Σ, an initial state q_0, a subset F of final states and three one-valued functions:

1. the transition function δ, which computes the next state
2. the output function η, which computes the string to write in each move
3. the final function φ, which computes the last suffix to concatenate to the output string when the transducer reaches a final state

The domains and images are: $\delta : Q \times \Sigma \to Q, \quad \eta : Q \times \Sigma \to \Delta^*, \quad \varphi : Q \times \{\text{-}|\} \to \Delta^*$

$$\delta(q,a) = q'$$
$$\eta(q,a) = u$$

$$q \xrightarrow{\frac{a}{u}} q'$$

The transduction $\tau(x)$ computed by T is the concatenation of two strings: the string computed by the output function and that computed by the final function.

$$\left\{ yz \in \Delta^* \mid \exists \text{ a computation labeled by } \frac{x}{y} \quad \text{ending in} \quad r \in F \wedge z = \varphi(r, \text{-}) \right\}$$

The IO automaton T is deterministic if the underlying input automaton (Q, Σ, δ, q_0, F) is deterministic and both the output and the final functions are one-valued.

The IO automaton T is instead non-deterministic, even if the underlying input automaton is deterministic, if between two states of T there is an arc of the type:

$$\frac{a}{\{b\} \cup \{c\}}$$

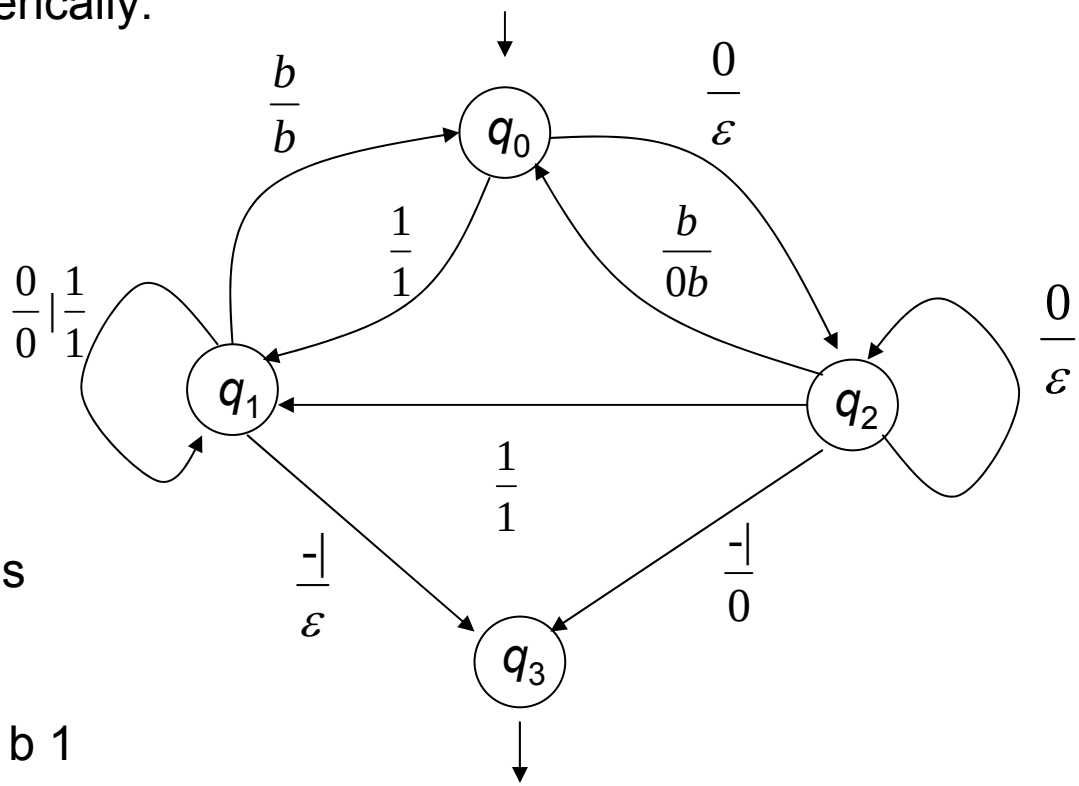
multi-valued output function

A function computable by means of a sequential finite state transducer (IO automaton) is said to be a **sequential function**.

EXAMPLE – elimination of leading zeroes

Text is a list of binary integer numbers, separated by a blank (b). Transduction (one-valued) removes the leading zeroes, which do not matter numerically.

$$\left(\frac{0^{+}b}{0b} \mid \left(\frac{0}{\varepsilon}\right)^{*}\frac{1}{1}\left(\frac{0}{0} \mid \frac{1}{1}\right)^{*}\frac{b}{b}\right)^{*}\left(\frac{0^{+}\text{-}\mid}{0} \mid \left(\frac{0}{\varepsilon}\right)^{*}\frac{1}{1}\left(\frac{0}{0} \mid \frac{1}{1}\right)^{*}\frac{\text{-}\mid}{\varepsilon}\right)$$



Transducing the text

0 0 b 0 1 -|

passes through the states

$q_0\, q_2\, q_0\, q_2\, q_1\, q_3$

and writes

ε ε 0 b ε 1 ε = 0 b 1

EXAMPLE

$$\tau(a^n) = \begin{cases} p, & n \text{ even} \\ d, & n \text{ odd} \end{cases}$$

The example exploits the concept of final function, which is included in the sequential transducer model.

The sequential transducer has two states, both final, corresponding to the parity class of the source string. The transducer does not write anything in the output tape while it is executing the moves, until it reaches a final state; then, depending on the reached final state, it outputs the symbol *p* rather than *d*.

The composition of two sequential functions is still a sequential function.

TWO-PASS SEQUENTIAL TRANSDUCTION WITH MIRRORING – given a generic transduction, specified by means of a regexprt or a two-tape automaton, there does not always exist a deterministic sequential transducer (a det. IO automaton) that can compute it directly. However, it is always possible to compute such a transduction in two passes: the former det. seq. transducer scans the input string from left to right and outputs an intermediate string, which is soon after given as input to the latter det. seq. transducer (different from the former one), which scans it from right to left (opposite direction) and outputs the final string. In this way it is possible, for instance, to compute deterministically the transduction of a^n into b^n if n is even, or in c^n if n is odd.

So far only regular (or rational) transductions have been considered, computed by a deterministic algorithm with a finite memory that examines the source string in one or two passes. But memory finiteness is a limit for many situations in computer science.

EXAMPLE - The transduction relation below is not regular, as it is necessary to store the entire source string before outputting its mirrored image.

$$\left\{ (x, x^R) \mid x \in (a \mid b)^* \right\}$$

Nivat: the transduction can be obtained by Nivat theorem, but it is necessary to construct the language R by merging the source string *x* and the image string $y = (x')^R$ (which should also be translitterated over the alphabet { *a*', *b*' }); however the language R is not regular any longer.

PURE SYNTAX TRANSDUCTON

When the source language is defined by means of a grammar, it is natural to consider transductions where syntactic structures are translated individually. The image of the entire phrase can then be computed by composing the individual transductions.

To this purpose it is necessary to match source and destination grammar rules.

DEFINITION – CONTEXT-FREE (or ALGEBRAIC) TRANSDUCTION

A *transduction grammar* G is a free grammar over the terminal alphabet C consisting of pairs (*u*, *v*) of strings over the source and destination alphabets, respectively.

$$G = (V, \Sigma, \Delta, P, S)$$

$$C \subseteq \Sigma^* \times \Delta^* \qquad \frac{u}{v}$$

The *syntactic transduction scheme* associated with grammar G is a set of pairs of source and destination rules, obtained eliminating from the grammar G the characters of the source and destination alphabets, respectively.

The *transduction relation* defined by grammar G is the following:

$$\tau(G) = \{(x, y) \mid \exists z \in L(G) \wedge x = h_\Sigma(z) \wedge y = h_\Delta(z)\}$$

$$h_\Sigma : C \to \Sigma \quad h_\Delta : C \to \Delta$$

The transduction grammar and the syntactic transduction scheme are simply two notational variants defining the same transduction relation.

EXAMPLE – algebraic transduction that computes mirroring

Transduction grammar:

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

Transduction scheme:

source gramm. G_1	dest. gramm. G_2
$S \to aS$	$S \to Sa$
$S \to bS$	$S \to Sb$
$S \to \varepsilon$	$S \to \varepsilon$

To obtain a pair of strings of the transduction relation:

a) construct a derivation for string *z*

$$S \Rightarrow \frac{a}{\varepsilon} S \frac{\varepsilon}{a} \Rightarrow \frac{a}{\varepsilon}\frac{a}{\varepsilon} S \frac{\varepsilon}{a}\frac{\varepsilon}{a} \Rightarrow$$

$$\Rightarrow \frac{a}{\varepsilon}\frac{a}{\varepsilon}\frac{b}{\varepsilon} S \frac{\varepsilon}{b}\frac{\varepsilon}{a}\frac{\varepsilon}{a} \Rightarrow \frac{a}{\varepsilon}\frac{a}{\varepsilon}\frac{b}{\varepsilon}\frac{\varepsilon}{\varepsilon}\frac{\varepsilon}{b}\frac{\varepsilon}{a}\frac{\varepsilon}{a} = z$$

b) project *z* over the two alphabets

$$h_\Sigma(z) = aab \qquad h_\Delta(z) = baa$$

EXAMPLE – transduction of palindromes – changes the terminal characters of the left and right halves of the string

$$G_p = \{S \to aSa' \mid bSb' \mid \varepsilon\}$$

This transduction grammar is actually a translitterated version of that of the previous example.

$$\frac{a}{\varepsilon} \text{ in } a, \quad \frac{b}{\varepsilon} \text{ in } b, \quad \frac{\varepsilon}{a} \text{ in } a', \quad \frac{\varepsilon}{b} \text{ in } b'$$

The above observation can be generalised, obtaining a version of the Nivat theorem suited for algebraic transductions.

PROPERTY – free languages and syntactic transductions

The following statements are equivalent:

1. The transduction relation is defined by a transduction grammar G
2. There exists a free language L over C and two alphabetic hom.s h_1 and h_2 such that:
3. If Σ and Δ are disjoint alphabets, there exists a language L over Σ U Δ such that:

$$\tau \subseteq \Sigma^* \times \Delta^*$$

$$\tau = \{(h_1(z), h_2(z)) \mid z \in L\}$$

$$\tau = \{(h_\Sigma(z), h_\Delta(z)) \mid z \in L\}$$

EXAMPLE – transducing string x $\in$ (*a* | *b*)* into the mirror image *y* = *x*R

Such a transduction can be expressed by means of the palindrome language.

$$L = \left\{ uu'^{R} \mid u \in (a \mid b)^{*} \wedge u' \in (a' \mid b')^{*} \right\} = \left\{ \varepsilon, aa', ..., abbb'b'a', ... \right\}$$

Here are the two alphabetic homomorphisms (or translitterations):

	h_1	h_2
a	a	ε
b	b	ε
a'	ε	a
b'	ε	b

The string *abb'a'* $\in$ L is projected into the string pair (*ab*, *ba*).

INFIX (or POLISH) NOTATION

An application of syntax transduction: conversion of expressions (arithmetic, logical, etc) into notations differing as for the position of the operators and the possible presence of delimiters, like pharenteses.

Degree of an operator: is the number of arguments (unary, binary, …, operators)
Variadic operator: an operator with variable degree.
Prefix / Postfix / Infix / Mixfix operator.

$$o_o\ arg_1\ o_2\ arg_2\ ...o_{n-1}\ arg_n\ o_n$$

$$\text{if}\ \ arg_1\ \ \text{then}\ \ arg_2\ \ [\text{else}\ arg_3]$$

$$\text{jump\_if\_false}\ \ arg_1\ arg_2$$

Polish notation: does not use pharenteses and the operators are all prefix or all postfix.

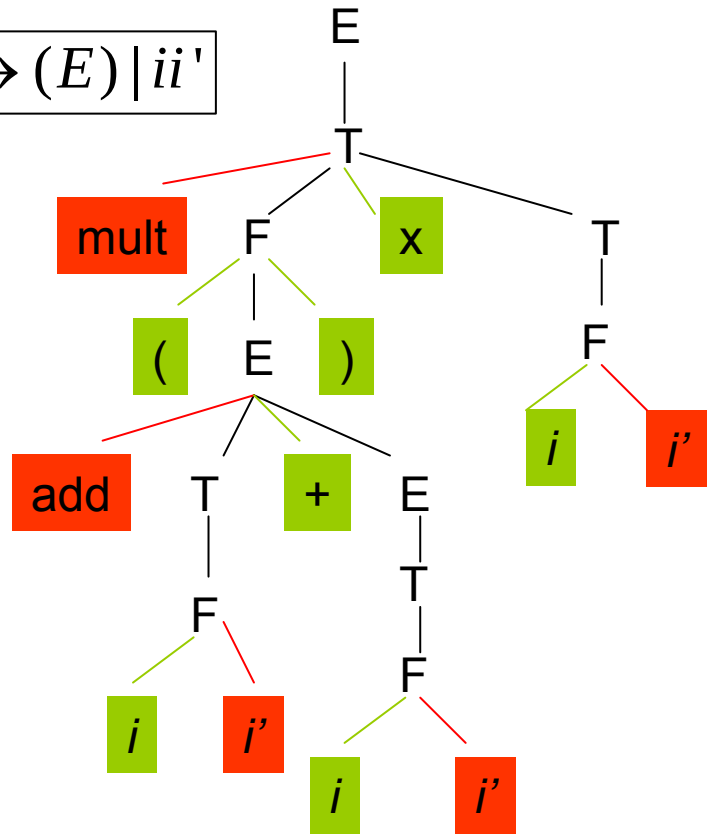
EXAMPLE – Converting an arithmetic expression from the standard notation (infix) into the prefix notation (this is frequently used in compilers to eliminate pharenteses).
Transduction grammar:

$$E \to \text{add } T + E \mid T \quad T \to \text{mult } F \times T \mid F \quad F \to (E) \mid ii'$$

Note: if the source and destination alphabets are disjoint, the transduction grammar can be denoted more simply, as it is not necessary to separate the source and dest. symbols by the fraction line.

$$E \to \text{add } T + E \quad \ldots \quad E \to \frac{\varepsilon}{\text{add}} \; T \frac{+}{\varepsilon} E$$

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



(i + i) x i

mult add i' i' i'

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)