Finite and Pushdown Transducers

Stanislav Elbl
Modern Theoretical Computer Science
Technical University of Brno

ABSTRACT: Translation is a relation between source and destination language. It can be defined by finite transducers and pushdown transducers (for regular languages and context free languages, respectively). Transducers are very similar to automatons - they are extended, so they can translate languages in addition to their accepting. They are very closely related to syntax-directed translation schemes. This article defines finite and pushdown transducer and shows examples of translations.

Translation

Formally, translation is a relation between source and destination language, L_I and L_O respectively. If this relation is finite it can be defined by enumeration of pairs (x_i, x_o), where $x_i \in L_I$ is input sentence and $x_o \in L_O$ is output sentence. Infinite translations can be defined by syntax directed translation scheme or by finite or pushdown transducers.

Syntax directed translation scheme (SDTS)

Any recursively enumerable language can be defined by grammar. This article considers only context free grammars. Grammar can be very easily extended, so it generates output language simultaneously with input one. So extended grammar is called syntax directed translation scheme (SDTS).

DEFINITION

A **SDTS** is 5-tuple $\mathbf{G = (N, \Sigma, \Delta, R, S)}$

N - set of nonterminals
Σ - input alphabet
Δ - output alphabet
$S \in N$ - start symbol
R - finite set of rules of the form $A \rightarrow x, y$
$A \in N, x \in (N \cup \Sigma)^*, y \in (N \cup \Delta)^*$

Nonterminals in the string y are permutation of nonterminals in y. Every nonterminal in x has its associated nonterminal in y. We define a relation of direct translation $\Rightarrow$ as follows:

For $u, w \in (N \cup \Sigma)^*, v, z \in (N \cup \Delta)^*, r = A \rightarrow x, y \in R$

$$\mathbf{(u, v) \Rightarrow (w, z) \quad [r]}$$

if $u = \alpha A \beta$, $w = \alpha x \beta$, $v = \gamma A \delta$, $z = \gamma y \delta$ and nonterminals A in u and v are associated. $\Rightarrow^+$, $\Rightarrow^*$ are transitive and reflexive-transitive closures of $\Rightarrow$.

The translation defined by G:

$$\tau(G) = \{ (x, y) \mid x \in \Sigma^*, y \in \Delta^* \text{ and } (S,S) \Rightarrow^* (x,y) \}$$

EXAMPLE 1

Consider SDTS G = ({E, T, F}, {x, +, -, *, /, (,)}, {x, +, -, *, /}, R, E) where R contains rules:

$E \rightarrow E + T,$	E T +
$E \rightarrow E - T,$	E T -
$E \rightarrow T,$	T
$T \rightarrow T * F,$	T F *

$T \rightarrow T / F,$	$T\ F /$
$T \rightarrow F,$	F
$F \rightarrow (E)$	E
$F \rightarrow x ,$	x

G performs a translation of infix expression to postfix expressions. Consider a sentence $x_1 + x_2 * (x_3 - x_4)$ and following derivation:

$$\begin{aligned}
&\mathbf{(E , E)} \Rightarrow (E + T , E\ T +) \Rightarrow (T_1 + T , T_1\ T +) \Rightarrow (F + T , F\ T +) \Rightarrow \\
&\quad \Rightarrow (x_1 + T , x_1\ T +) \Rightarrow (x_1 + T * F , x_1\ T\ F * +) \Rightarrow \\
&\quad \Rightarrow (x_1 + F_1 * F , x_1\ F_1\ F * +) \Rightarrow (x_1 + x_2 * F , x_1\ x_2\ F * +) \Rightarrow \\
&\quad \Rightarrow (x_1 + x_2 * F , x_1\ x_2\ F * +) \Rightarrow (x_1 + x_2 * (E) , x_1\ x_2\ E * +) \Rightarrow \\
&\quad \Rightarrow (x_1 + x_2 * (E - T) , x_1\ x_2\ E\ T - * +) \Rightarrow \\
&\quad \Rightarrow (x_1 + x_2 * (T_1 - T) , x_1\ x_2\ T_1\ T - * +) \Rightarrow \\
&\quad \Rightarrow (x_1 + x_2 * (F - T) , x_1\ x_2\ F\ T - * +) \Rightarrow \\
&\quad \Rightarrow (x_1 + x_2 * (x_3 - T) , x_1\ x_2\ x_3\ T - * +) \Rightarrow \\
&\quad \Rightarrow (x_1 + x_2 * (x_3 - F) , x_1\ x_2\ x_3\ F - * +) \Rightarrow \\
&\quad \Rightarrow \mathbf{(x_1 + x_2 * (x_3 - x_4) , x_1\ x_2\ x_3\ x_4 - * +)}
\end{aligned}$$

A special case of SDTS is **simple SDTS**, in which nonterminals in each rule of the form $A \rightarrow x, y$ occur in same order in x and y.

For every SDTS $G = (N, \Sigma, \Delta, R, S)$ we define **input (underlying) grammar** $G_I = (N, \Sigma, R_I, S)$ where $R_I = \{ A \rightarrow x \mid A \rightarrow x, y \in R \text{ for some } y \in \Delta^* \}$. Similarly we define **output grammar** $G_O = (N, \Sigma, R_O, S)$ where $R_O = \{ A \rightarrow y \mid A \rightarrow x, y \in R \text{ for some } x \in \Sigma^* \}$. This grammars define input and output language of G.

Finite transducer

Finite automatons define family of regular languages. We can extend finite automata, so it defines translation. For such extension we need to generate output string.

DEFINITION

A **finite transducer** is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, q_0, F)$, where

- Q is a finite set of states
- Σ is input alphabet
- Δ is output alphabet
- $q_0 \in Q$ is initial state
- $F \subseteq Q$ is a set of final states
- $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\})) \times (Q \times \Delta^*)$ is a finite set of rules

A **configuration** of M is a triple (q, x, y) where

- $q \in Q$ is current state
- $x \in \Sigma^*$ is remaining part (suffix) of input string
- $y \in \Delta^*$ is generated prefix of output string

A **relation of move** of M $\vdash$

(q, ax, y) $\vdash$ (p, x, yv)

p, q $\in$ Q; a $\in$ ($\Sigma \cup \{\varepsilon\}$); x $\in \Sigma^*$; y, v $\in \Delta^*$ and ((q, a), (p, v)) $\in \delta$. $\vdash^+$ and $\vdash^*$ are defined as transitive and reflexive-transitive closure of $\vdash$.

A translation defined by M:

$$\tau(M) = \{ (x,y) \mid x \in \Sigma^*, y \in \Delta^* \text{ and } (q_0, x, \varepsilon) \vdash^* (q_F, \varepsilon, y), q_F \in F \}$$

Similarly to input grammar for every SDTS there exists **input finite automaton** M_I for every finite transducer M = (Q, Σ, Δ, δ, q_0, F). It is defined as M_I = (Q, Σ, δ_I, q_0, F), where

δ_I = { ((q, a), p) | ((q, a), (p, v)) $\in \delta$ for some v $\in \Delta^*$ }.

Such a finite automaton accepts the input language of M (the domain of τ(M)).

LEMMA 1

A family of translations defined by finite transducers is equal to family of translations defined by SDTS with rules of the form A $\rightarrow$ aB, αB, where A $\in$ N, B $\in$ N $\cup$ { ε }, a $\in$ $\Sigma \cup$ { ε }, $\alpha \in \Sigma^*$.

PROOF:

Consider SDTS G = (N, Σ, Δ, R, S) with rules satisfying condition from lemma 1. Without loss of generality assume, that N $\cap$ { f } = $\varnothing$. Construct finite transducer M = (N $\cup$ { f }, Σ, Δ, δ, S, { f }). If A $\rightarrow$ aB, αB is a rule from R, where A, B $\in$ N, a $\in \Sigma \cup \{\varepsilon\}$, $\alpha \in \Delta^*$ then add ((A, a), (B, α)) to δ. If A $\rightarrow$ a, α is a rule from R, where A $\in$ N, a $\in \Sigma \cup \{\varepsilon\}$, $\alpha \in \Delta^*$ then add ((A, a), (f, α)) to δ.

To complete this part of the proof, we must show, that every translation generated by G is also accepted by M. This is left to the reader.

Now consider a finite transducer M = (N, Σ, Δ, δ, S, F). Construct SDTS G = (N, Σ, Δ, R, S). If ((A, x), (B, y)) $\in \delta$, add A $\rightarrow$ xB, yB to R. In addition, if B $\in$ F, add A $\rightarrow$ x, y to R.

To complete the proof, we must show, that every translation accepted by M is also generated by G. This is left to the reader.

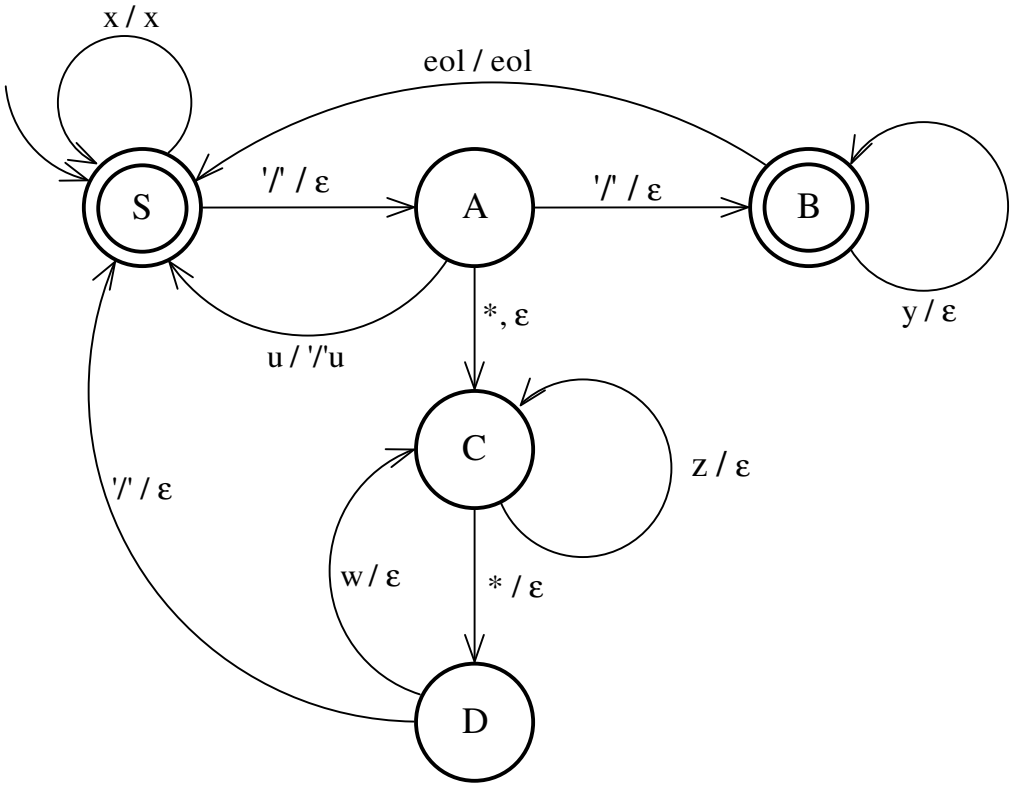
EXAMPLE 2

Consider SDTS G = (N, Σ, P, S) with rules:

S $\rightarrow$ x S,	x S	$\forall$ x $\in \Sigma$ - { '/' }
S $\rightarrow$ / A,	A	possible start of comment
A $\rightarrow$ / B,	B	start of line comment //
A $\rightarrow$ * C,	C	start of block comment /*
A $\rightarrow$ u S,	/ u S	$\forall$ u $\in \Sigma$ - { '*', '/' } - not a comment
B $\rightarrow$ y B,	B	$\forall$ y $\in \Sigma$ - { eol }
B $\rightarrow$ eol S,	eol S	end of line comment

$C \rightarrow * D,$	D	possible end of block comment
$C \rightarrow z C,$	C	$\forall z \in \Sigma - \{ \text{'*'} \}$
$D \rightarrow / S,$	S	end of block comment */
$D \rightarrow w C,$	C	$\forall w \in \Sigma - \{ \text{'/'} \}$

This SDTS defines translation removing line comments and block comments (as defined in C++) from input string. The state diagram of transducer accepting this translation is showed on the picture 1.



Picture 1 - finite transducer

DEFINITION

Finite transducer $M = (Q, \Sigma, \Delta, \delta, q_0, F)$ is **deterministic** if all its states $q \in Q$ satisfy one of following conditions:

a) $|\delta(q, a)| \leq 1$ for all $a \in \Sigma$ and $\delta(q, \varepsilon) = \varnothing$

b) $|\delta(q, \varepsilon)| \leq 1$ and $\delta(q, a) = \varnothing$ for all $a \in \Sigma$

Note, that deterministic finite transducer can define more translations for one input string.

EXAMPLE 3

Consider deterministic transducer $M = (\{q\}, \varnothing, \{a\}, \delta, \{ ((q, \varepsilon), (q, a)) \}, \{q\})$. This one state transducer contains one ε-rule. It defines translation $\{ \varepsilon \} \times \{ a^n \}, n \geq 0$.

For this reason, definition of deterministic transducer can be extended. For example it can forbid ε-rules in final state.
Clearly, there is not equivalent deterministic finite transducer for every finite transducer. As an example consider translation $(a^n, b^n) \cup (a^n, c^n)$, or mapping $(a^{2n}, b^{2n}) \cup (a^{2n+1}, c^{2n+1})$.

Pushdown transducer

Pushdown automatons define family of context free languages. We can extend pushdown automata, so it defines translation. For such extension we need to generate output string.

DEFINITION

A **pushdown transducer** is a 7-tuple
$M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$

- Q - finite set of states
- Σ - input alphabet
- Γ - pushdown alphabet
- Δ - output alphabet
- $q_0 \in Q$ - initial state
- $Z_0 \in \Gamma$ - initial symbol on stack
- $F \subseteq Q$ - set of final states
- $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma^* \times \Delta^*)$ - finite set of rules

A **configuration** of M is a 4-tuple (q, x, α, y)

- $q \in Q$ is current state
- $x \in \Sigma^*$ is remaining part (suffix) of input string
- $\alpha \in \Gamma^*$ is content of the stack
- $y \in \Delta^*$ is generated prefix of output string

A **relation of move** of M $\vdash$

$$\mathbf{(q, ax, Z\gamma, y) \vdash (p, x, \alpha\gamma, yv)}$$

$p, q \in Q;\ a \in (\Sigma \cup \{\varepsilon\});\ x \in \Sigma^*;\ y, v \in \Delta^*; Z \in (\Gamma \cup \{\varepsilon\});\ \alpha, \gamma \in \Gamma^*$
$((q, a, Z), (p, \alpha, v)) \in \delta$. $\vdash^+$ and $\vdash^*$ are defined as transitive and reflexive-transitive closure of $\vdash$.

A translation defined by M

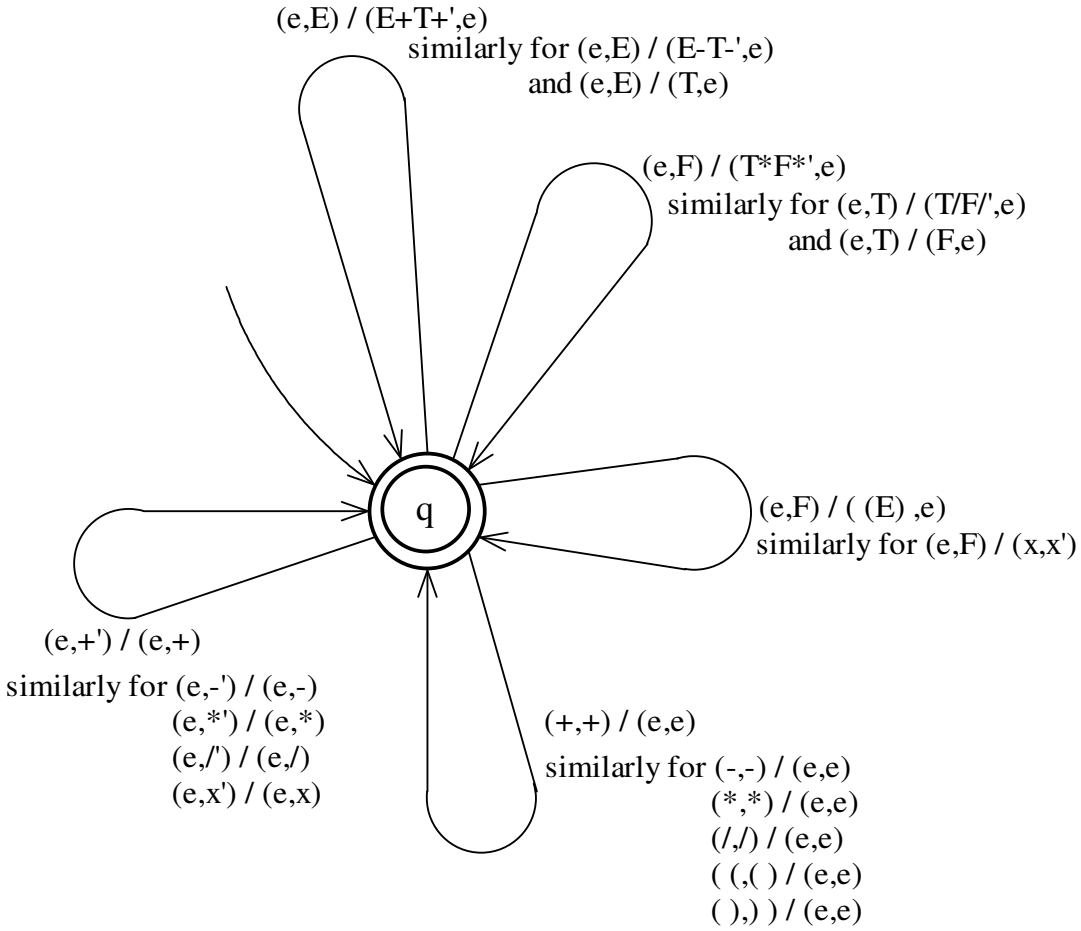
$$\tau(M) = \{ (x,y) \mid x \in \Sigma^*, y \in \Delta^* \text{ and } (q_0, x, Z_0, \varepsilon) \vdash^* (q_F, \varepsilon, \alpha, y), q_F \in F, \alpha \in \Gamma^* \}$$

LEMMA 2

A family of translations defined by pushdown transducers is equal to family of translations defined by simple SDTS. For the proof and construction of pushdown transducer equivalent to simple SDTS see [ULLMAN].

EXAMPLE 4

Consider simple SDTS from the example 1. We can construct pushdown transducer $M = (\{q\}, \{x, +, -, *, /, (,)\}, \{x, +, -, *, /, x', +', -', *', /', E, T, F\}, \{x, +, -, *, /\}, \delta_M, q, E, \{q\})$. The state diagram of transducer is showed on the picture 2.



Picture 2 - Pushdown transducer

DEFINITION

Pushdown transducer $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$ is **deterministic** if all its states $q \in Q$ satisfy following conditions:

a) $|\delta(q, a, z)| \leq 1$ for all $a \in \Sigma \cup \{ \varepsilon \}$ and all $z \in \Gamma$

b) If $\delta(q, \varepsilon, z) \neq \varnothing$, $\delta(q, a, z) = \varnothing$ for all $a \in \Sigma$

Summary

This article discusses about translation. It defines syntax directed scheme and transducers as a algebraic tool to define translation. It also shows relationship between both of them and presents some practical examples.

An object of actual research is the power of pushdown transducers. It shows, that we can constrain output of transducer by linear language and then any recursively enumerable language can be accepted by it.

Open Problems

- Is it decidable whether two deterministic finite transducers are equivalent?
- Is it decidable whether two deterministic pushdown transducers are equivalent?

References

[MEDUNA] Meduna, A.: *Automata and Languages: Theory and Applications*, Springer, London, 2000.

[ULLMAN] Aho, A., V., Ullman, J., D.: *The Theory of Parsing, Translation, and Compiling, Volume I: Parsing*, Prentice - Hall, 1972