Two-Way Coupled Finite Automata and Its Usage in Translators

Tomáš Hruška, Dušan Kolář, Roman Lukáš, Eva Zámečníková
Faculty of Information Technology
Department of Information Systems
Božetěchova 2, Brno 61266
Czech Republic

Abstract: This article defines two-way coupled finite automata. Two-way coupled finite automaton enable to make a translation from input language to output language and from output language to input language too. There is discussed deterministic parsing by using coupled finite automaton in this article. For example, this deterministic model can be used for translation between assembly language and a binary code.

Key–Words: Two-way coupled finite automaton, lazy finite automaton, lazy finite transducer, deterministic finite transducer, translator, assembly language, binary code, HW/SW co-design.

1 Introduction

The goal of our project is the creation and especially the implementation of a language - the language name is ISAC - Instruction Set Architecture C, which is used for description of microprocessor architecture. For good applicability of the language, it is necessary to create a development environment, which provides development of both software tools and simultaneously of microprocessor hardware. Due to the concurrent work on hardware and software (hardware/software co-design), the total time of the development will be reduced and the developmental cycle will be shortened. Our project consists of several parts. A basic research in this area is running now, and the project comes to more intensive phase, which should lead to practicable results in short time. The work on the project is concentrated on the design of typical constructions of the description language and on the implementation of the software tools (compiler, universal assembler, disassembler, linker, and simulator). In this article we present, how we can use two-way coupled finite automata as a model for assembler and disassembler and how we can make deterministic parsing by using this model.

2 Preliminaries

This paper assumes that the reader is familiar with the formal language theory. For a set, Q, $card(Q)$ denotes the cardinality of Q. For an alphabet, V, V^* represents the free monoid generated by V under the operation of concatenation. The unit of V^* is denoted by ε. Set $V^+ = V^* - \{\varepsilon\}$; algebraically, V^+ is thus the free semigroup generated by V under the operation of concatenation.

- For every $w \in V^*$, $|w|$ denotes the length of w.
- For every $w \in V^*$, $alph(w)$ denotes the set of symbols ocuring in w.
- For every $w \in V^*$, $perm(w)$ denotes the set of all permutations of string w. Formally, $perm(\varepsilon) = \{\varepsilon\}$ and for all $a \in V, w \in V^*, perm(aw) = \{xay : xy = z, z \in perm(w)\}$.

Example 1 $perm(abc) = \{$*abc, acb, bac, bca, cab, cba*$\}$.

3 Definitions

Definition 2 *A* <u>*lazy finite automaton*</u> *is a quintuple,* $M = (Q, \Sigma, R, s, F)$, *where*

- *Q is a finite set of states,*
- *Σ is an input alphabet,*
- *R is a finite set of rules of the form $px \rightarrow q$, where $p, q \in Q$ and $x \in \Sigma^*$,*
- *$s \in Q$ is the start state of M,*
- *$F \subseteq Q$ is a set of final states.*

A <u>*configuration of M*</u> *is a string px, where $p \in Q$ and $x \in \Sigma^*$.*

Let pxy and qy be two configurations of M, where $p, q \in Q$, and $x, y \in \Sigma^*$. Let $r = px \rightarrow q \in R$

be a rule. Then M makes a move from pxy to qy according to r, written as $pxy \vdash qy[r]$ or, simply, $pxy \vdash qy$.

M makes zero moves from χ to χ; in symbols, $\chi \vdash \chi[\varepsilon]$ or, simply $\chi \vdash \chi$.

Let $\chi_0, \chi_1, ..., \chi_n$ be a sequence of configurations, where $n \geq 1$, and $\chi_{i-1} \vdash \chi_i[r_i]$, where $r_i \in R$, for all $i = 1, ..., n$. Then M makes n moves from χ_0 to χ_n, written as $\chi_0 \vdash^n \chi_n[r_1...r_n]$ or, simply $\chi_0 \vdash^n \chi_n$.

If $\chi_0 \vdash^n \chi_n[\varrho]$ for some $n \geq 1$, then we write $\chi_0 \vdash^+ \chi_n[\varrho]$.

If $\chi_0 \vdash^n \chi_n[\varrho]$ for some $n \geq 0$, then we write $\chi_0 \vdash^* \chi_n[\varrho]$.

The language accepted by M, $L(M)$, is defined as $L(M) = \{w : w \in \Sigma^*, sw \vdash^* f, f \in F\}$.

Definition 3 *Let M be a lazy finite automaton. M is unambiguous lazy finite automaton, if for each $x \in L(M)$ there exists exactly one sequence of rules ϱ such that $sx \vdash^* f[\varrho], f \in F$.*

Definition 4 *A lazy finite transducer is a sextuple, $M = (Q, \Sigma, \Omega, R, s, F)$, where*

- *Q is a finite set of states,*
- *Σ is an input alphabet,*
- *Ω is an output alphabet,*
- *R is a finite set of rules of the form $px \rightarrow yq$, where $p, q \in Q$, $x \in \Sigma^*$ and $y \in \Omega^*$,*
- *$s \in Q$ is the start state of M,*
- *$F \subseteq Q$ is a set of final states.*

A configuration of M is a string vpu, where $p \in Q$, $u \in \Sigma^*$ and $v \in \Omega^*$.

Let $vpxu$ and $vyqu$ be two configurations of M, where $p, q \in Q$, $x, u \in \Sigma^*$ and $v, y \in \Omega^*$. Let $r = px \rightarrow yq \in R$ be a rule. Then M makes a move from $vpxu$ to $vyqu$ according to r, written as $vpxu \vdash vyqu[r]$ or, simply, $vpxu \vdash vyqu$.

M makes zero moves from χ to χ; in symbols, $\chi \vdash \chi[\varepsilon]$ or, simply $\chi \vdash \chi$.

Let $\chi_0, \chi_1, ..., \chi_n$ be a sequence of configurations, where $n \geq 1$, and $\chi_{i-1} \vdash \chi_i[r_i]$, where $r_i \in R$, for all $i = 1, ..., n$. Then M makes n moves from χ_0 to χ_n, written as $\chi_0 \vdash^n \chi_n[r_1...r_n]$ or, simply $\chi_0 \vdash^n \chi_n$.

If $\chi_0 \vdash^n \chi_n[\varrho]$ for some $n \geq 1$, then we write $\chi_0 \vdash^+ \chi_n[\varrho]$.

If $\chi_0 \vdash^n \chi_n[\varrho]$ for some $n \geq 0$, then we write $\chi_0 \vdash^* \chi_n[\varrho]$.

Translation of M, $T(M)$, is defined as $T(M) = \{(x, y) : sx \vdash^* yf, x \in \Sigma^*, y \in \Omega^*, f \in F\}$

Definition 5 *Let M be a lazy finite transducer. M is unambiguous lazy finite transducer, if for each $(x, y) \in T(M)$ there exists exactly one sequence of rules ϱ such that $sx \vdash^* yf[\varrho], f \in F$ and there exists no z and $\overline{\varrho}$ such that $sx \vdash^* z\overline{f}[\overline{\varrho}], \overline{f} \in F$ and $\varrho \neq \overline{\varrho}$.*

Definition 6 *Let $M = (Q, \Sigma, \Omega, R, s, F)$ be a lazy finite transducer. M is finite transducer, if for each $px \rightarrow yq \in R$ holds $x \in \Sigma \cup \{\varepsilon\}$.*

Definition 7 *Let $M = (Q, \Sigma, \Omega, R, s, F)$ be a finite transducer. M si ε-free finite transducer, if card(F) = 1 and each rule from R is of the form $pa \rightarrow yq$, where $p, q \in Q - F$, $a \in \Sigma$, $y \in \Omega^*$ or $p \rightarrow yf$, where $p \in Q - F$, $y \in \Omega^*$, $f \in F$.*

Definition 8 *Let $M = (Q, \Sigma, \Omega, R, s, F)$ be an ε-free finite transducer. M si strict deterministic finite transducer, if for each $a \in \Sigma$ and $p \in Q$ there exist no more than one rule of the form $pa \rightarrow yq$, where $p, q \in Q - F$, $y \in \Omega^*$ and for each $p \in Q$ there exist no more than one rule of the form $p \rightarrow yf$, where $f \in F$ and $y \in \Omega^*$.*

Definition 9 *A two-way coupled finite automaton is a triple, $\Gamma = (M_1, M_2, h)$, where*

- *$M_i = (Q_i, \Sigma_i, R_i, s_i, F_i)$ is a lazy finite automaton for each $i \in \{1, 2\}$*
- *h is a bijective mapping from R_1 to R_2.*

Let h^* be a mapping from R_1^* to R_2^* defined as: $h^*(\varepsilon) = \{\varepsilon\}$ and for $r_1, r_2, ..., r_n \in R_1$, $h^*(r_1 r_2 ... r_n) = h(r_1)(r_2)...(r_n)$, where $n \geq 1$.

Then, the translation of Γ, $T(\Gamma)$, is defined as $T(\Gamma) = \{(w_1, w_2) : w_1 \in \Sigma_1^*, w_2 \in \Sigma_2^*, s_1 w_1 \vdash^* f_1[\varrho_1]$ in M_1, $s_2 w_2 \vdash^* f_2[\varrho_2]$ in M_2, $f_1 \in F_1, f_2 \in F_2, \varrho \in perm(\varrho_1), \varrho_2 = h^*(\varrho)\}$.

Example 10 *$\Gamma = (M_1, M_2, h)$, where*

- *$M_1 = (\{s_1, f_1\}, \{a, b\}, \{1 : s_1 \rightarrow as_1, 2 : s_1 \rightarrow f_1, 3 : f_1 \rightarrow bf_1\}, s_1, \{f_1\})$;*
- *$M_2 = (\{s_2, f_2\}, \{a, b\}, \{1 : s_2 \rightarrow bs_2, 2 : s_2 \rightarrow f_2, 3 : f_2 \rightarrow af_2\}, s_2, \{f_2\})$;*
- *$h = \{(1, 3), (2, 2), (3, 1)\}$*

is two-way coupled finite automaton.

For example, $(a^2b^3, b^3a^2) \in T(\Gamma)$, because $s_1a^2b^3 \vdash^* f_1[112333], s_2b^3a^2 \vdash^* f_2[111233], f_1 \in F_1, f_2 \in F_2, 333211 \in perm(112333)$ and $111233 = h^*(333211)$.

Notice that $T(\Gamma) = \{(a^m b^n, b^n a^m) : m, n \geq 0\}$.

4 Results

Algorithm 11 *Convertion from lazy finite transducer to finite transducer*

- *Input:* *Lazy finite transducer* $M_1 = (Q_1, \Sigma, \Omega, R_1, s, F)$
- *Output:* *Finite transducer* $M_2 = (Q_2, \Sigma, \Omega, R_2, s, F)$, *such that* $T(M_1) = T(M_2)$
- *Method:*

```
$Q_2 := Q_1$;
for each $px \rightarrow yq \in R_1$ do
if $|x| \leq 1$ then
  add $px \rightarrow yq$ to $R_2$
else
  let $x = a_1a_2 \ldots a_n$, where $n \geq 2$:
  add $\langle p, a_1\rangle, \langle p, a_1a_2\rangle, \ldots, \langle p, a_1a_2 \ldots a_{n-1}\rangle$ to $Q$
   add $pa_1 \rightarrow \langle p, a_1\rangle$, $\langle p, a_1\rangle a_2 \rightarrow \langle p, a_1a_2\rangle$, $\ldots \langle p, a_1a_2 \ldots a_{n-2}\rangle a_{n-1} \rightarrow \langle p, a_1a_2 \ldots a_{n-1}\rangle$, $\langle p, a_1a_2 \ldots a_{n-1}\rangle a_n \rightarrow yq$ to $R_2$
```

Theorem 12 *For every lazy finite transducer* M_1 *there exists a finite transducer* M_2 *such that* $T(M_1) = T(M_2)$.

Proof: Use the Algorithm for convertion from lazy finite transducer to finite transducer. □

Definition 13 *Let* $M = (Q, \Sigma, \Omega, R, s, F)$ *be a finite transducer and* $q \in Q$. $\varepsilon - closure(q)$ *is defined as:* $\varepsilon - closure(q) = \{(p, y) : q \vdash^* yp, p \in Q, y \in \Omega^*\}$

Algorithm 14 *Computation of* ε*-closure*

- *Input:* *Lazy finite transducer* $M_1 = (Q_1, \Sigma, \Omega, R_1, s, F)$; $q \in Q$
- *Output:* $\varepsilon - closure(q)$ *or error of ambiguity, if there exist* $p \in Q$ *and different* $u, v \in \Omega^*$ *such that* $q \vdash^* up$ *and* $q \vdash^* vp$
- *Method:*

```
$S_{undone} := \{(q, \varepsilon)\}$;
$S_{done} := \emptyset$;
while $S_{undone} \neq \emptyset$ do begin
  let $(p, v) \in S_{undone}$:
   if exists $w \in \Omega^*, w \neq v$ such that $(p, w) \in S_{done}$
  then error(ambiguity)
  $S_{undone} := S_{undone} - \{(p, v)\}$;
  $S_{done} := S_{done} \cup \{(p, v)\}$;
  $S_{undone} := S_{undone} \cup \{(t, vy) : p \rightarrow yt \in R\}$;
end
$\varepsilon - closure(q) := S_{done}$;
```

Algorithm 15 *Convertion from finite transcuder to* ε*-free finite transducer*

- *Input:* *Finite transducer* $M_1 = (Q_1, \Sigma, \Omega, R_1, s, F_1)$, $\varepsilon - closure(q)$ *for all* $q \in Q$
- *Output:* ε*-free finite transducer* $M_2 = (Q_2, \Sigma, \Omega, R_2, s, F_2)$, *such that* $T(M_1) = T(M_2)$
- *Method:*

```
$Q_2 := Q_1 \cup \{f_2\}$;
$R_2 := \emptyset$;
$F_2 := \{f_2\}$;
for each $q \in Q$ do
  $R_2 := R_2 \cup \{qa \rightarrow xyt : pa \rightarrow yt \in R_1, (p, x) \in \varepsilon - closure(q), a \in \Sigma, t \in Q\} \cup \{q \rightarrow xf_2 : (f, x) \in \varepsilon - closure(q), f \in F\}$;
```

Theorem 16 *For every unambiguous finite transducer* $M_1 = (Q_1, \Sigma, \Omega, R_1, s, F_1)$ *there exists an* ε*-free finite transducer* $M_2 = (Q_2, \Sigma, \Omega, R_2, s, F_2)$ *such that* $T(M_1) = T(M_2)$.

Proof: Use the Algorithm for convertion from finite transcuder to ε-free finite transducer. If M_1 is unambiguous finite transducer, there not exist $p \in Q_1$ and different $u, v \in \Omega^*$ such that $q \vdash^* up$ and $q \vdash^* vp$ and it is possible to compute $\varepsilon - closure(q)$ for each $q \in Q_1$. □

Algorithm 17 *Convertion from* ε*-free finite transducer to strict deterministic finite transducer*

- *Input:* ε*-free finite transducer* $M_1 = (Q_1, \Sigma, \Omega, R_1, s_1, \{f_1\})$,
- *Output:* *deterministic finite transducer* $M_2 = (Q_2, \Sigma, \Omega, R_2, s_2, \{f_2\})$, *such that* $T(M_1) = T(M_2)$ *or error, if* M_1 *is not unambiugous or if* M_1 *is not convetable to* M_2 *using this algorithm.*
- *Method:*

```
$s_2 := \{\langle s_1, \varepsilon\rangle\}$;
$f_2 := \{\langle f_1, \varepsilon\rangle\}$;
$S_{undone} := \{s_2\}$;
```

```
$S_{done} := \emptyset$;
$S_{cycle} := \{(s_2, \varepsilon)\}$;
while $S_{undone} \neq \emptyset$ do begin
  let $X \in S_{undone}$:
  $S_{undone} := S_{undone} - \{X\}$;
  $S_{done} := S_{done} \cup \{X\}$;
  for each $a \in \Sigma \cup \{\varepsilon\}$ do
    $Q_{in} := \{\langle p, z\rangle : \langle p, z\rangle \in X, pa \rightarrow yq \in R_1, y, z \in \Omega^*, q \in Q_1\}$;
    if $|Q_{in}| = 1$ then
      let $\langle p, z\rangle \in Q_{in}$:
      $Q_{out} := \{\langle q, y\rangle : q \in Q_1, y \in \Omega^*, pa \rightarrow yq \in R_1\}$;
      if $|Q_{out}| = 1$ then
        let $\langle q, y\rangle \in Q_{out}$:
        $Q_{out} := \{\langle q, \varepsilon\rangle\}$;
        add $Xa \rightarrow zyQ_{out}$ to $R_2$;
      if $|Q_{out}| > 1$ then
        add $Xa \rightarrow zQ_{out}$ to $R_2$;
    if $|Q_{in}| > 1$ then
      $Q_{out} := \{\langle q, zy\rangle : \langle p, z\rangle \in X, pa \rightarrow yq \in R_1, y, z \in \Omega^*, q \in Q_1\}$;
      add $Xa \rightarrow Q_{out}$ to $R_2$;
    $S_{undone} := S_{undone} \cup \{Q_{out}\}$;
    if exists $q \in Q_1$ such that $|\{y : \langle q, y\rangle \in Q_{out}\}| > 1$ then error(ambiguity)
  let $(X, x) \in S_{cycle}$:
  if exists $(Y, y) \in S_{cycle}$ such that $y$ is a prefix of $x$, $X \neq Y$, $\{p : \langle p, u\rangle \in X\} = \{q : \langle q, v\rangle \in Y\}$ then error(cycle)
    add $(Q_{out}, xa)$ to $S_{cycle}$;
end
$Q_2 := S_{done}$;
```

Theorem 18 *For every unambiguous ε-free finite transducer M_1, where $T(M_1)$ is a finite relation, there exists a strict deterministic finite transducer M_2 such that $T(M_1) = T(M_2)$.*

Proof: Use the Algorithm for convertion from ε-free finite transducer to strict deterministic finite transducer. □

Notice: The basic idea of construction is following. There is described a determinization of ε-free finite automaton to deterministic finite automaton in [6]. Each state in deterministic finite automaton represents a set of states in original finite automaton. In previous algorithm, this detemininization is modified for finite transducers. Each state in deterministic finite transducer represents a set of states in original finite transducer. If a state in deterministic finite transducer contains more than one state from original finite transducer, then it is not known, which of them is simulated in the original automaton. Hence, it is not possible to generate an output. These parts of generated outputs are assigned to their states and nothing is generated. If some state in deterministic finite transducer contains exactly one state from original finite transducer, then it is possible to generate the whole history of outputs, which is known from this state. Notice that this algorithm can convert only unambiguous ε-free finite transducer (ambiguity is detected), which specifies the finite translation. There exist some unambiguous ε-free finite transducers, which specify infinite translation and this algorithm cannot convert them to strict deterministic finite transducers.

Example 19 *Consider ε-free finite transducer $M_1 = (Q_1, \Sigma, \Omega, R_1, s_1, \{f_1\})$, where:*

- $Q_1 = \{s_1, q_1, q_2, q_3, q_4, f_1\}$
- $\Sigma = \{a, b\}$
- $\Omega = \{1, 2, 3, 4, 5, 6, 7\}$
- $R_1 = \{s_1a \rightarrow 1q_1, s_1b \rightarrow 2q_1, s_1a \rightarrow 4q_3, s_1b \rightarrow 6q_4, q_1a \rightarrow 3q_2, q_3b \rightarrow 5q_2, q_4b \rightarrow 7q_2, q_2 \rightarrow f_1\}$

The previous algorithm constructs a strict deterministic finite transducer $M_2 = (Q_2, \Sigma, \Omega, R_2, \langle s_1, \varepsilon\rangle, \{\langle f_1 \varepsilon\rangle\})$, where:

- $Q_2 = \{\{\langle s_1, \varepsilon\rangle\}, \{\langle q_1, 1\rangle, \langle q_3, 4\rangle\}, \{\langle q_1, 2\rangle, \langle q_4, 6\rangle\}, \{\langle q_2, \varepsilon\rangle\}, \{\langle f_1, \varepsilon\rangle\}\}$
- $\Sigma = \{a, b\}$
- $\Omega = \{1, 2, 3, 4, 5, 6, 7\}$
- $R_2 = \{\{\langle s_1, \varepsilon\rangle\}a \rightarrow \{\langle q_1, 1\rangle, \langle q_3, 4\rangle\},$
  $\{\langle s_1, \varepsilon\rangle\}b \rightarrow \{\langle q_1, 2\rangle, \langle q_4, 6\rangle\},$
  $\{\langle q_1, 1\rangle, \langle q_3, 4\rangle\}a \rightarrow 13\{\langle q_2, \varepsilon\rangle\},$
  $\{\langle q_1, 1\rangle, \langle q_3, 4\rangle\}b \rightarrow 45\{\langle q_2, \varepsilon\rangle\},$
  $\{\langle q_1, 2\rangle, \langle q_4, 6\rangle\}a \rightarrow 23\{\langle q_2, \varepsilon\rangle\},$
  $\{\langle q_1, 2\rangle, \langle q_4, 6\rangle\}b \rightarrow 67\{\langle q_2, \varepsilon\rangle\},$
  $\{\langle q_2, \varepsilon\rangle\} \rightarrow \{\langle f_1, \varepsilon\rangle\}\}$.

Notice that M_1 is unambiguous and specifies the finite translation.

Example 20 *Consider ε-free finite transducer $M_1 = (Q_1, \Sigma, \Omega, R_1, s_1, \{f_1\})$, where:*

- $Q_1 = \{s_1, q_1, f_1\}$
- $\Sigma = \{a\}$
- $\Omega = \{1, 2\}$
- $R_1 = \{s_1a \rightarrow 1s_1, s_1a \rightarrow 2q_1, q_1 \rightarrow f_1\}$

The previous algorithm constructs a strict deterministic finite transducer $M_2 = (Q_2, \Sigma, \Omega, R_2, \langle s_1, \varepsilon\rangle, \{\langle f_1, \varepsilon\rangle\})$*, where:*

- $Q_2 = \{\{\langle s_1, \varepsilon\rangle\}, \{\langle s_1, 1\rangle, \langle q_1, 2\rangle\}, \{\langle f_1, \varepsilon\rangle\}\}$
- $\Sigma = \{a\}$
- $\Omega = \{1, 2\}$
- $R_2 = \{\{\langle s_1, \varepsilon\rangle\}a \to \{\langle s_1, 1\rangle, \langle q_1, 2\rangle\}, \{\langle s_1, 1\rangle, \langle q_1, 2\rangle\}a \to 1\{\langle s_1, 1\rangle, \langle q_1, 2\rangle\}, \{\langle s_1, 1\rangle, \langle q_1, 2\rangle\} \to 2\{\langle f_1, \varepsilon\rangle\}\}$.

Notice that M_1 *is unambiguous and specifies the infinite translation.*

5 Applications

Two-way coupled finite automata can be used as effective models for description of translation, especially for the finite translation. For example, we use this model as an inner representation of relations between assembly instructions and their binary codes in our project. Then, there is generated an assembler (translator, which translates assembly language to binary code) and a disassembler (translator, which translates binary code to assembly language) in automatic way. The automatic construction of assembler and disassembler can be described in the following steps:

- Relations between assembly and binary codes of instructions are described in language ISAC 0.0.
- Description in language ISAC 0.0 is converted to equivalent two-way coupled finite automaton $\Gamma = (M_1, M_2, h)$, where M_1 is a lazy finite automaton for assembly format's description of instructions, M_2 is a lazy finite automaton for binary format's description of instructions, and h describes translating relations between rules of M_1 and M_2.
- **Deterministic translation from assembly language to binary code and automatic construction of this translator:** Lazy finite automaton M_1 is converted to lazy finite transducer $\overline{M_1}$, which generates the sequence of accepting moves to the output. It means that the label of each rule from M_1 is added to this rule itself as a generated output. Lazy finite transducer $\overline{M_1}$ is converted to equivalent strict deterministic finite transducer $\overline{M_d}$ by using algorithms described in previous section. It is possible because this translation is a finite relation (the length of each instruction is fixed) and M_1 is unambiguous because relation between assembly and binary code of instructions is a bijection. This model is a basic structure of assembler-translator. An assembler-translator reads assembly code of instruction and simulates the activity of the strict deterministic finite transducer $\overline{M_d}$. The output sequence of symbols is not generated to the output, but this sequence activates corresponding edges in M_2. After accepting this instruction, M_2 generates corresponding output by using activated edges. Analogically, **deterministic translation from binary code to assembly language and automatic construction of this translator** is realized.

6 Conclusion

This article showed that we can use effectively two-way coupled finite automaton as a model for translations. Especially, two-way coupled finite automaton can be used as a model for inner structure's description of language ISAC 0.0. Assembler and dissassembler can be created in an automatic way by using this model.

References:

[1] Češka M., Hruška T., Beneš, M., Překladače, VUT Brno, 1993.

[2] Hruška T., Instruction Set Architecture C., FIT VUT Brno, 2004.

[3] Lukáš R., Hruška T., Kolář D., Masařík K., it Two-Way Deterministic Translantion and Its Usage in Practice, Proceedings of 8th Spring International Conference - ISIM'05, pp. 101-107, 2005.

[4] Masařík K., Hruška T., Kolář D., Lukáš R., System for design and simulation of microprocessors, Proceedings of 8th Spring International Conference - ISIM'05, pp. 269-276, 2005.

[5] Masařík K., Hruška T., Kolář D., Language and Development Environment For Microprocessor Design Of Embedded Systems, Proceedings of IFAC Workshop on PROGRAMMABLE DEVICES and EMBEDDED SYSTEMS PDeS, pp. 120-125, 2006.

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