VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FIT FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Deterministic translation of LL(1) languages using reduced pushdown automata

Adam Husár,
ihusar@fit.vutbr.cz,
FIT, BUT, 11/12/07.

Contents

Introduction, motivation

- Assembler input language can be divided into 3 grammars describing:
  - o assembler file structure and directives,
  - o expressions and
  - o instruction set.
- Needed to "regularize" context-free grammar – to use a modification of finite automata that will be able to translate expressions.
- Result: algorithm that allows us to transform any CFG into an equivalent reduced pushdown automaton.

Global Index Grammar Inspired Grammars (GIGIG)

- $G = (N, T, I, S, \#, P)$,
- N, T, S are defined as usually,
- I is the set of indices (stack symbols),
- # is the starting stack symbol,
- P is the set of productions of the following forms

a. $A \underset{\varepsilon}{\rightarrow} \alpha$ *(epsilon),*

b. $A \underset{x}{\rightarrow} \alpha$ *(push),*

c. $A \underset{\bar{x}}{\rightarrow} \alpha$ *(pop),*

where $x \in I, y \in I \cup \{\#\}, A \in N, \alpha \in (N \cup T)^*$.

GIGIG – derivation relation, example

- Derivation relation $\Rightarrow$ is defined as follows:

  if rule is of the form:

  a. $A \underset{\varepsilon}{\rightarrow} \alpha$, then $\delta\#\beta A\gamma \underset{\varepsilon}{\Rightarrow} \delta\#\beta\alpha\gamma$,

  b. $A \underset{x}{\rightarrow} \alpha$, then $\delta\#\beta A\gamma \underset{x}{\Rightarrow} x\delta\#\beta\alpha\gamma$,

  c. $A \underset{\bar{x}}{\rightarrow} \alpha$, then $x\delta\#\beta A\gamma \underset{\bar{x}}{\Rightarrow} \delta\#\beta\alpha\gamma$.

- Generated language: $L(G) = \{w \mid \#S \rightarrow^{*} \#w, w \in T^{*}\}$.
- Example: let's have a GIGIG with following rules:

  $S \underset{i}{\rightarrow} aSd$, $S \rightarrow B$, $B \underset{\bar{i}}{\rightarrow} bBc$, $B \rightarrow \varepsilon$,

  then a derivation sequence of sentential forms for string *aabbccdd* would be the following:

$$\#S \underset{i}{\Rightarrow} i\#aSd \underset{i}{\Rightarrow} ii\#aaSdd \Rightarrow ii\#aaBdd \underset{\bar{i}}{\Rightarrow} i\#aabBcdd \underset{\bar{i}}{\Rightarrow} \#aabbBccdd \Rightarrow aabbccdd .$$

Reduced Pushdown Automata (RPDA)

- $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$,
- $Q, \Sigma, \Gamma, q_0, z_0, F$ are defined in the same way as for pushdown automata,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow 2^{Q \times (\Gamma \cup \{\varepsilon\})}$, where for any $q \in Q, a \in (\Sigma \cup \{\varepsilon\})$ holds:

    if $(q', z') \in \delta(q, a, z)$,

    then $(z = \varepsilon \wedge z' = \varepsilon) \vee (z \in \Gamma \wedge z' = \varepsilon) \vee (z = \varepsilon \wedge z' \in \Gamma)$.

- When a transition is made, either we do not manipulate with the pushdown, one symbol is pushed onto the pushdown or one symbol is popped.

Step 1 - Transformation CFG -> GIGIG

- Let's have a CFG *G* = (*N, T, P, S*), an equivalent GIGIG is *G'* = ($N \cup I \cup \{X\}$, *T, I, S, #, P'*), where the production set *P'* and index set *I* is constructed in this way:

  For every rule *p*, $p \in P$, of the form $A \rightarrow w_1 A_1 w_2 A_2 \ldots A_{n-2} w_{n-1} A_{n-1} w_n$ add to P' following rules:

$$A \underset{A_1^p}{\rightarrow} w_1 A_1, \qquad X \underset{\overline{A_1^p}}{\rightarrow} A_1^p,$$

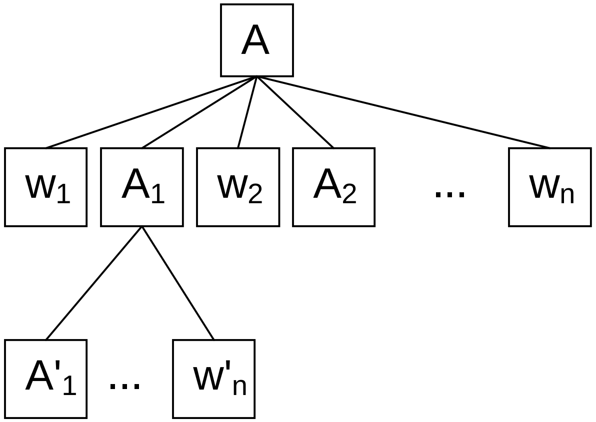
$$A_1^p \underset{A_2^p}{\rightarrow} w_2 A_2, \qquad X \underset{\overline{A_2^p}}{\rightarrow} A_2^p,$$

... ...

$$A_{n-2}^p \underset{A_{n-1}^p}{\rightarrow} w_{n-1} A_{n-1}^p, \qquad X \underset{\overline{A_{n-1}^p}}{\rightarrow} A_{n-1}^p,$$

$$A_{n-1}^p \rightarrow w_n,$$

$$A_{n-1}^p \rightarrow w_n X$$ and



  to *I* add $A_1^p, A_2^p, \ldots, A_{n-1}^p$, where $A_i \in N, n \geq 1, 1 \leq i \leq n-1$, $w_j \in T^*, 1 \leq j \leq n$

- Result of the step 1 is a tuple (*G'*, *X*).

Step 2 - Transformation GIGIG with right-linear rules -> GIGIG with right-regular rules

- Classical algorithm for conversion of right-linear grammar to right-regular grammar with one small modification would be used.
- If a rule manipulates with the pushdown, this action would be associated with the first rule created from it.

Step 3 - Transformation GIGIG -> RPDA

- Let's have tuple (*G*, *X*) obtained from a CFG using steps 1 and 2.

  G = (N, T, I, S, #, P) and *X* is a special nonterminal, $X \in N$.
- An equivalent reduced pushdown automaton is

  $M = (N \cup \{f\}, T, I \cup \{\#\}, \delta, S, \#, \{f\})$, where the transition function

  $\delta : N \times (T \cup \{\varepsilon\}) \times (I \cup \{\varepsilon\}) \rightarrow 2^{(N \cup \{f\}) \times (I \cup \{\varepsilon\})}$ is constructed in this way:

  For every rule *p*, $p \in P$ of the form

  a) $A \rightarrow aB$, let $(B, \varepsilon) \in \delta(A, a, \varepsilon)$,

  b) $A \underset{z}{\rightarrow} aB$, let $(B, z) \in \delta(A, a, \varepsilon)$,

  c) $A \underset{\bar{z}}{\rightarrow} B$, let $(B, \varepsilon) \in \delta(A, \varepsilon, z)$ and

  d) $A \rightarrow a$, let $(X, \varepsilon) \in \delta(A, a, \varepsilon)$.

  Further, let $(f, \varepsilon) \in \delta(X, \varepsilon, \#)$.
- Result of step 3 is a RPDA that translates original CFG.

Example 1

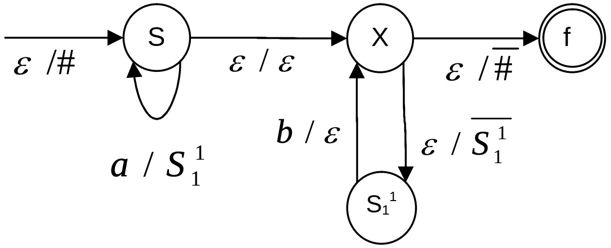
- Let's have a CFG with following rules: 1: $S \rightarrow aSb$, 2: $S \rightarrow \varepsilon$.
- Using step 1 and step 2, we obtain GIGIG with following regular rules:

1: $S \underset{S_1^1}{\rightarrow} aS$, $X \underset{\overline{S_1^1}}{\rightarrow} S_1^1$,

$S_1^1 \rightarrow b$, $S_1^1 \rightarrow bX$,

2: $S \rightarrow \varepsilon$, $S \rightarrow X$.

- Then the step 3 is applied and we get this RPDA:



- For input string *aabb*, this automaton goes throught the following sequence of configurations:

$$(S, aabb, \#) \mapsto (S, abb, S_1^1\#) \mapsto (S, bb, S_1^1 S_1^1\#) \mapsto (X, bb, S_1^1 S_1^1\#) \mapsto (S_1^1, bb, S_1^1\#) \mapsto (X, b, S_1^1\#) \mapsto$$
$$(S_1^1, b, \#) \mapsto (X, \varepsilon, \#) \mapsto (f, \varepsilon, \varepsilon)$$

Deterministic RPDA

- Any RPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, where for the transition function δ holds

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

  can be transformed to a deterministic RPDA.

- A step relation for deterministic RPDA first tries to use a transition that uses a symbol from the input or a symbol from the pushdown. Only in the case that none of these can be used, an epsilon transition ($\delta(q, \varepsilon, \varepsilon)$) can be applied.

Theorem 1

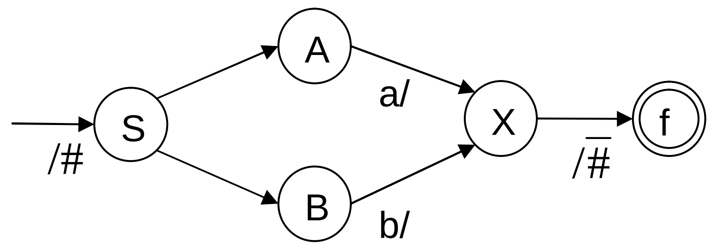
For a reduced pushdown automaton M_1, created from context-free grammar G_1 using steps 1, 2 and 3, holds $L(M_1) = L(G_1)$.

Theorem 2

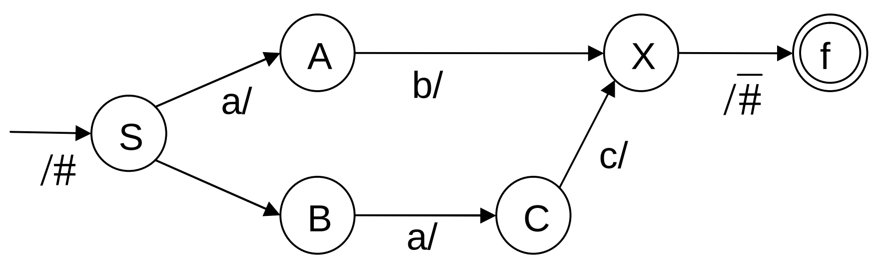
If a reduced pushdown automaton M_2, created from LL(1) grammar G_2 using steps 1, 2 and 3, can be transformed to a deterministic pushdown automata M_{2D}, then $L(M_{2D}) = L(G_2)$ holds.

Theorem 2, explanations

- Let's have a grammar with rules $S \to A \mid B, A \to a, B \to b$,
    - o is LL(1), but the created automaton is not deterministic.



- Another grammar with rules $S \to aA \mid B, A \to b, B \to aC, C \to c$,
    - o is not LL(1) (in fact is LL(2)), the created automaton is deterministic (in the way we have defined determinism for deterministic RPDA), but does not accept the same language.



- Note: The ZAP course project grammar.

Example 2

- Let's have a CFG with following rules:

$$1: E \to FD$$
$$2: D \to +FD$$
$$3: D \to \varepsilon$$
$$4: F \to (E)$$
$$5: F \to i$$

- Then an equivalent GIG will have these rules:
-

1: $E \underset{E_1^1}{\to} F$, $X \underset{\overline{E_1^1}}{\to} E_1^1$,

$E_1^1 \underset{E_2^1}{\to} D$, $X \underset{\overline{E_2^1}}{\to} E_2^1$,

$E_2^1 \to \varepsilon$, $E_2^1 \to X$,

2: $D \underset{D_1^2}{\to} +F$, $X \underset{\overline{D_1^2}}{\to} D_1^2$,

$D_1^2 \underset{D_2^2}{\to} D$, $X \underset{\overline{D_2^2}}{\to} D_2^2$,

$D_2^2 \to \varepsilon$, $D_2^2 \to X$,

3: $D \to \varepsilon$, $D \to X$,

4: $F \underset{F_1^4}{\to} (E$, $X \underset{\overline{F_1^4}}{\to} F_1^4$,

$F_1^4 \to)$, $F_1^4 \to)X$,

5: $F \to i$, $F \to iX$.

Example 2, continued

- We have a GIGIG with these rules:

1: $E \underset{E_1^1}{\to} F$, $\quad X \underset{\overline{E_1^1}}{\to} E_1^1$,

$E_1^1 \underset{E_2^1}{\to} D$, $\quad X \underset{\overline{E_2^1}}{\to} E_2^1$,

$E_2^1 \to \varepsilon$, $\quad E_2^1 \to X$,

2: $D \underset{D_1^2}{\to} + F$, $\quad X \underset{\overline{D_1^2}}{\to} D_1^2$,

$D_1^2 \underset{D_2^2}{\to} D$, $\quad X \underset{\overline{D_2^2}}{\to} D_2^2$,

$D_2^2 \to \varepsilon$, $\quad D_2^2 \to X$,

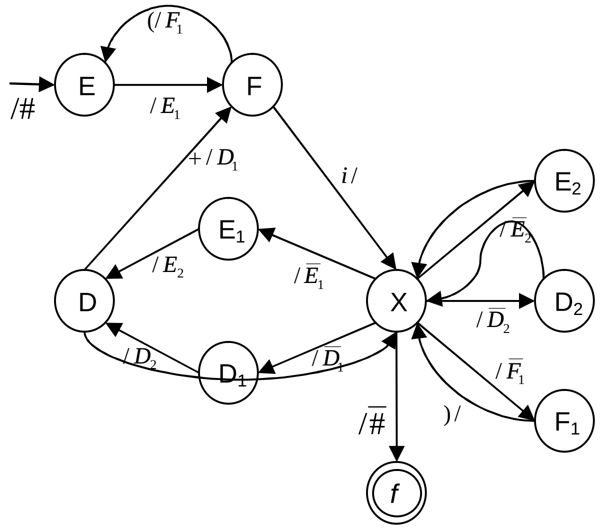
3: $D \to \varepsilon$, $\quad D \to X$,

4: $F \underset{F_1^4}{\to} (E$, $\quad X \underset{\overline{F_1^4}}{\to} F_1^4$,

$F_1^4 \to)$, $\quad F_1^4 \to)X$,

5: $F \to i$, $\quad F \to iX$.

- Using step 3 we get following RPDA:



- This automaton translates the language generated by the original grammar.

(Upper indices representing rule numbers are omitted.)

Attributed Translation

- We need to deal with attributes and semantic actions during translation.

- For this, we add to a RPDA a new stack onto which we will store attribute values.

- Operations on this attribute stack are these:
  - *st*[*n*] –access to the *n*-th value below the top of the stack,
  - *push*(*a*) – pushes value *a* onto the stack and
  - *pop*(*m*) – removes *m* items (values) from the top of the stack.

- Further, there are two types of semantic actions associated with rules:
  - 1. for terminals - push attribute value when a terminal symbol is encountered,
  - 2. for rules – they are executed when a rule expansion is finished, consist of three steps: 1) new value calculation, 2) popping of not anymore needed attribute values from the attribute stack and 3) pushing of a new value.

Example 3 - Attributed Translation

- Let's have a grammar with assigned semantic actions:

$1: E \rightarrow FD$ $\{\ a := st[0];\ pop(2);\ push(a);\ \}$

$2: D \rightarrow +\ \{\ push(\perp);\ \}\ FD$ $\{\ a := st[3] + st[1];\ pop(3);\ push(a);\ \}$

$3: D \rightarrow \varepsilon$ $\{\ a := \perp;\ pop(0);\ push(a);\ \}$

$4: F \rightarrow (\ \{\ push(\perp);\ \}\ E\ )\ \{\ push(\perp);\ \}$ $\{\ a := st[1];\ pop(3);\ push(a);\ \}$

$5: F \rightarrow i\ \{\ push(i.value);\ \}$ $\{\ a := st[0];\ pop(1);\ push(a);\ \}$

- Then we get a GIGIG containing rules with semantic actions, GIGIG rules for original rule 2 are shown:

2: $D \underset{D_1^2}{\rightarrow} +\ \{\ push(\perp);\ \}\ F,$ $X \underset{\overline{D_1^2}}{\rightarrow} D_1^2,$

$D_1^2 \underset{D_2^2}{\rightarrow} D,$ $X \underset{\overline{D_2^2}}{\rightarrow} D_2^2,$

$D_2^2 \rightarrow \varepsilon$ $\{\ a := st[3] + st[1];\ pop(3);\ push(a);\ \},$

$D_2^2 \rightarrow X$ $\{\ a := st[3] + st[1];\ pop(3);\ push(a);\ \}.$

Example 3 - Attributed Translation, continued

- Now we can try to translate string *i+i,* where the first *i* has attribute value 1 and the second *i* has attribute value 2.

Step	State	Input	RPDA pushdown	Attribute pushdown	Executed semantic actions	Original rule
1	*E*	i+i	#			
2	*F*	i+i	E_1 ,#			
3	*X*	+i	E_1 ,#	1 1	*push(i.value);* *a:= st[0]; pop(1); push(a);*	$F \rightarrow i$
4	E_1	+i	#	1		
5	*D*	+i	E_2 ,#	1		
6	*F*	i	D_1, E_2 ,#	$\perp$,1	*push(*$\perp$*);*	
7	*X*		D_1, E_2 ,#	2,$\perp$,1 2,$\perp$,1	*push(i.value);* *a := st[0]; pop(1); push(a);*	$F \rightarrow i$
8	D_1		E_2 ,#	2,$\perp$,1		
9	*D*		D_2, *E* ,$\#_2$	2,$\perp$,1		
10	*X*		D_2, E_2 ,#	$\perp$,2,$\perp$,1	*a :=* $\perp$*; pop(0); push(a);*	$D \rightarrow \varepsilon$
11	D_2		E_2 ,#	$\perp$,2,$\perp$,1		
12	*X*		E_2 ,#	3,1	*a := st[3] + st[1]; pop(3);* *push(a);*	$D \rightarrow +FD$
13	E_2		#	3,1		
14	*X*		#	3	*a := st[0]; pop(2); push(a);*	$E \rightarrow FD$
15	*f*			3		

Conclusions, further work

- Global index grammar inspired grammars, non-deterministic and deterministic reduced pushdown automata.
- Presented algorithm allows us to transform in a straightforward way any context-free grammar to a reduced pushdown automaton. Also, if the original grammar was LL(1), we can obtain a deterministic RPDA.

- Is it possible to transform any LR(k) grammar to a deterministic RZA?
- Find a simple algorithm that transforms any nondeterministic CFG to a deterministic pushdown automaton with multiple stacks?
- Applications in hardware?
- Classes of languages accepted deterministically by presented automata, their hierarchy. (Similarly to LL(1), LL(2), ...)

- Notes: LL(1) translation table creation, left parse generation.

Final remark

- **Finite automata without any stack = Regular Languages**
- **Finite automata with 1 stack = Context-Free Languages**
- **Finite automata with 2 stacks = Recursively Enumerable Languages**

- Where do the context-sensitive/recursive/Turing-decidable languages fit in?
- What impacts do have undecidable problems on finite automata with two stacks?
- If we impose the same restriction as for linear bounded automata on finite automata with 1 stack, what will we receive?

Thank you for your attention

Acronym RPDA by The Free Dictionary

RPDA	Remote Power Distribution Assembly
RPDA	Ruggedized Personal Digital Assistant

Rugged - drsný, nerovný, hrbolatý, kostrbatý, neotesaný, mrzutý, náročný, namáhavý, zbrázděný, rozeklaný, nevlídný (podle slovník.seznam.cz).

References

[Cas04] Castano, J. M.: *Global Index Languages*. PhD. Thesis, The Faculty of the Graduate School of Arts and Sciences, Brandeis University, 2004. Document available on the WWW: <http://www.cs.brandeis.edu/~jcastano/thesis3.pdf>.

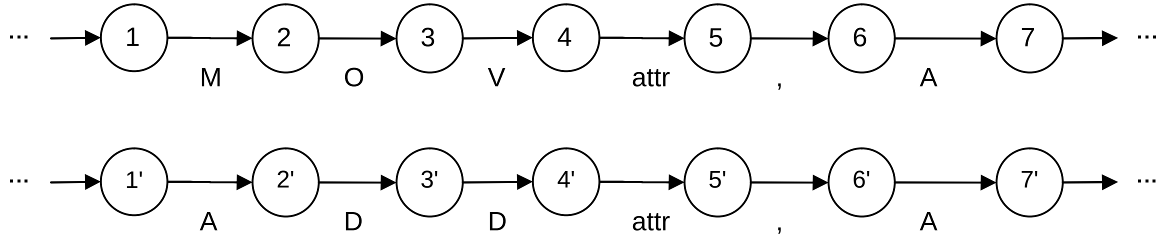
[Ces92] Češka, M.: *Gramatiky a jazyky*. FIT VUT v Brně, 1992. Document available on the WWW: <http://www.fit.vutbr.cz/study/courses/TI1/public/Texty/ti.pdf>.

Substitution of RPDA into a finite automaton

- To use such an expression translating automaton, we need to substitute it into the finite automaton used in assembler to translate input.
- Example: let's have two operations with following assembler sections:

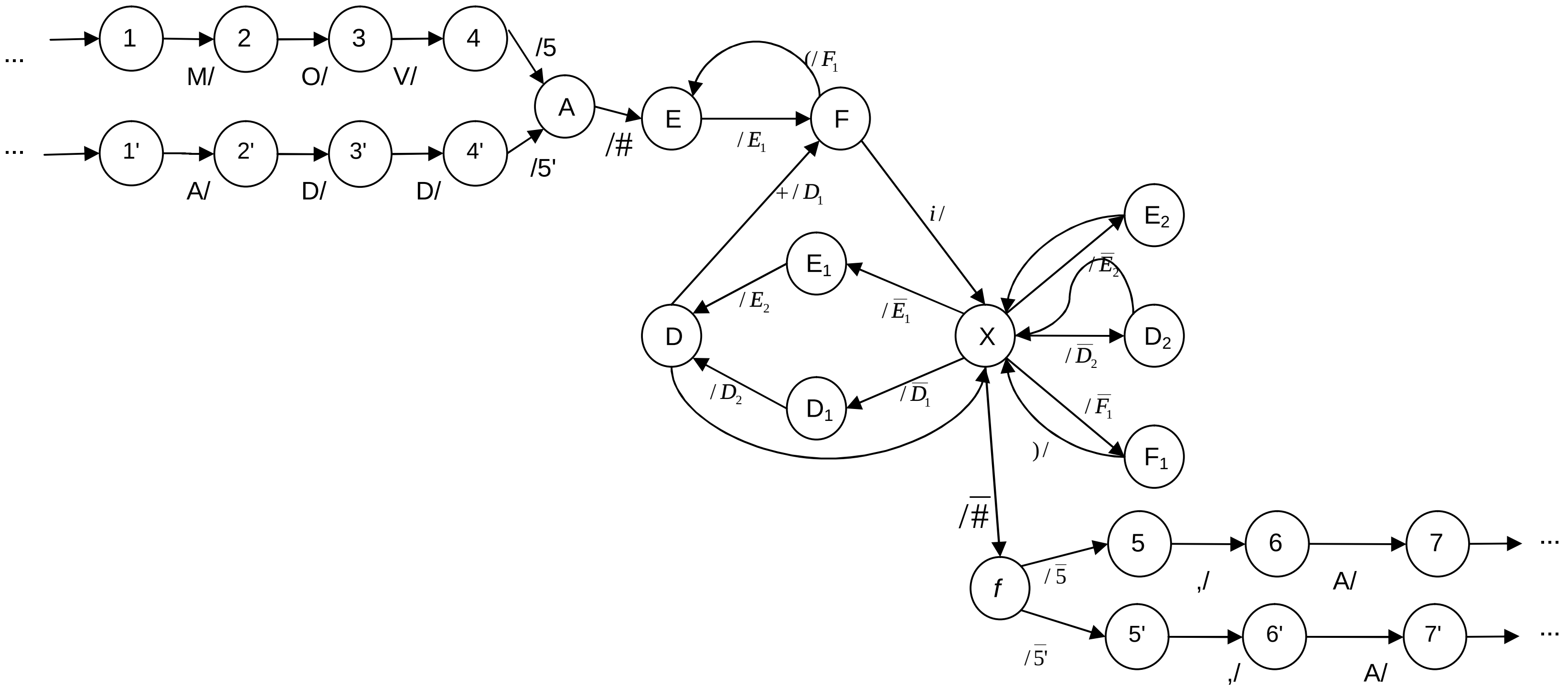
```
ASSEMBLER { "MOV" attr "," "A" },
ASSEMBLER { "ADD" attr "," "A" }.
```

- From the instruction set description we obtain an automaton that translates this instruction set. Only parts relevant for this example are shown here:



Substitution of RPDA into a finite automaton, continued

- Now we will substitute RPDA created from expression generating grammar to the finite automata obtained from instruction set description.



Note: substituted RPDA accepts input without needing to consume it completely.