An extended computation model

Pushdown Automata

- Finite state automata are a handy model, but are not able to count an arbitrary number of items
- Idea: add a simple memory to the computation model: a stack
- The stack is a Last-In-First-Out memory (LIFO)
- The read operation on the memory erases the value (pop operation)
- The resulting automaton is known as a PushDown Automaton (PDA)
- For this computation model the non-determinism enhances the computation capability
A recognizer PDA is formally defined as a 7-tuple \((Q, I, \Gamma, \delta, q_0, F, Z_0)\), where:

- \(Q\), is the set of states of the automata
- \(I\) is the alphabet of the input string which will be checked
- \(\Gamma\) is the alphabet of the symbols on the stack
- \(\delta : Q \times I \times \Gamma \mapsto Q \times (\Gamma \cup \epsilon)\) the transition function
- \(q_0 \in Q\) the (unique) initial state from where the automaton starts
- \(F \subseteq Q\) the set of final accepting states of the automaton
- \(Z_0\) is the symbol which indicates the bottom of the stack
Structure

Advantages

- The main advantage of a PDA is that it is able to count letters
- The transition function relies on reading both a symbol from the input and a symbol from the stack to perform a transition
- Nondeterminism takes place when two transitions have both the same input and the same stack symbol as a trigger (from the same state)...
- ...or with $\epsilon$-transitions, as always
A first attempt

$L = a^n b^{2n}, n \geq 1$

- To obtain a synthetic notation, the convention is to denote on an arc $<input><stack> | <stack><stack>$

\[
\begin{align*}
&\text{start} \quad q_0 \quad bA|A \\
&aA|AA \\
&\quad q_1 \quad bA|\epsilon \\
&aZ_0|AZ_0 \\
&\quad q_2 \\
&\quad \epsilon Z_0|Z_0 \\
&\quad q_3
\end{align*}
\]
A first attempt

$L = a^n b^{2n}, n \geq 1$

- Eliminating nondeterminism with an extra symbol
A **transducer** PDA is formally defined as a 7-tuple $(Q, I, \Gamma, \delta, q_0, F, Z_0, O, \eta)$, where:

- $Q$, is the set of states of the automata
- $I$ is the alphabet of the input string which will be checked
- $\Gamma$ is the alphabet of the symbols on the stack
- $\delta : Q \times I \times \Gamma \mapsto Q \times (\Gamma \cup \epsilon)$ the transition function
- $q_0 \in Q$ the (unique) initial state from where the automaton starts
- $F \subseteq Q$ the set of final accepting states of the automaton
- $Z_0$ is the symbol which indicates the bottom of the stack
- $O$ the output alphabet (may coincide with $I$)
- $\eta : Q \times I \times \Gamma \mapsto \{O \cup \epsilon\} \times (\Gamma \cup \epsilon)$ the transduction function
A simple transducer

\[ \tau(a^k b^h c^h) = d^{3h} e^k, \quad h, k \geq 1 \]

- Hint: “one b is worth two d” is a nice strategy
- Notation convention

\[
\langle \text{input} \rangle \langle \text{stack} \rangle \mid \langle \text{stack} \rangle \langle \text{stack} \rangle, \langle \text{output} \rangle
\]

```
\begin{align*}
\begin{array}{c}
aZ_0 & | AZ_0, \epsilon \\
\text{start} & \quad \rightarrow \\
q_0 & \quad bA|BA, dd \\
& \quad \rightarrow \\
q_1 & \quad cB|\epsilon, d \\
& \quad \rightarrow \\
q_2 & \quad \epsilon A|\epsilon, e \\
& \quad \rightarrow \\
q_3 & \quad bA|\epsilon, e
\end{array}
\end{align*}
```

```
The ultimate computing model

Definition

- Willing to further extend the capabilities of the PDA we substitute the stack with one (or more) tapes
- The resulting automaton is the Turing Machine, which is able to do anything we define as computable\(^a\)
- The TM is able to choose whether to advance, go back or stand still with the cursor on the tape
- Multiple tapes can be added, but they do not increase the computing power

\(^a\)or, at least, Alonso Church believed so, and we pretty much agree
The ultimate computing model

Definition

- For greater comfort, we use a marker symbol $Z_0$ to indicate the beginning of the tape (in case it’s not there, we can add it)
- We assume that the input is positioned on the input tape
- All the other tapes, if any, are filled with the blank symbol □
- All the actions on the tapes contents are driven by a FSA
- Notation: $<\text{input\_tape}>, <\text{tape}_1>, \ldots, <\text{tape}_n> \mid <\text{tape}_1>, \ldots, <\text{tape}_n>, (R \mid S \mid L)^{n+1}$
A $k$-tapes transducer TM is formally defined as a 7-tuple $(Q, I, \Gamma, \delta, q_0, F, Z_0)$, where:

- $Q$, is the set of states of the automata
- $I \cup \{■\}$ is the alphabet of the input string which will be checked
- $\Gamma \cup \{■\}$ is the alphabet of the symbols on the tapes
- $\delta : Q \times I \times \Gamma^k \mapsto Q \times \Gamma^k \times \{R, S, L\}^{k+1}$ the transition function
- $q_0 \in Q$ the (unique) initial state from where the automaton starts
- $F \subseteq Q$ the set of final accepting states of the automaton
$L = a^n b^n c^n, \ n \geq 1$

- The trick is that we can sweep over the memory tape as many times as we want

![Diagram](image-url)
A bit more complex

$L = a^n b^{n/2} c^{n/2}, \ n \geq 1$

- A slight deviation from the common idea of counting
- More than a single solution possible (it’s the same as saying: “devise a program that writes such strings”)
- One method to perform division: stop on the input tape, continue in memory
- Other methods involve writing twice the symbols on the tape
A bit more complex
FSA determinization

Removing nondeterminism

- Removing nondeterminism from an FSA
  \((Q_{ND-FSA}, I_{ND-FSA}, \delta_{ND-FSA}, q_{0ND-FSA}, F_{ND-FSA})\) is “easy”, although comes at a price

- The state set of the deterministic FSA is (in the worst case) is \(\wp(Q_{ND-FSA})\), thus \(|\wp(Q_{ND-FSA})| = 2^{Q_{ND-FSA}}\)

- Luckily some of them may not be reachable and can be eliminated

- The algorithm for determinization applies step by step the definition

- Starting from \(q_{0ND-FSA}\) add a state as soon as a transition goes there

- A state is final if at least one of the ND-FSA states represented by it was final
FSA determinization
FSA determinization

Expanding $q_0$

Start $\rightarrow q_0$ $\rightarrow q_1$ $\rightarrow q_1q_2$

Transitions:
- $q_0 \xrightarrow{1} q_1$
- $q_0 \xrightarrow{0} q_1q_2$
FSA determinization

Expanding $q_1$
FSA determinization

Expanding $q_3$
FSA determinization

Expanding $q_2$

Diagram showing states and transitions of a finite automaton.
FSA determinization

Expanding $q_{13}$
**FSA determinization**

Expanding $q_{013}$
FSA determinization

Expanding $q_{123}$
Expanding $q_{01}$
FSA determinization

Expanding $q_{23}$