Penguin grammars

Adam Pilát

1. Mating call (1 point)

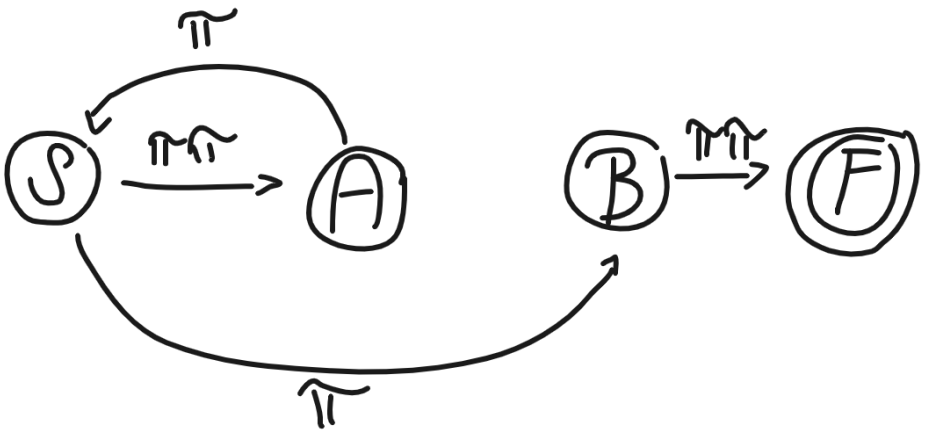
- **G** = (N, Σ, P, S)

→ N = {S, A, B, F}

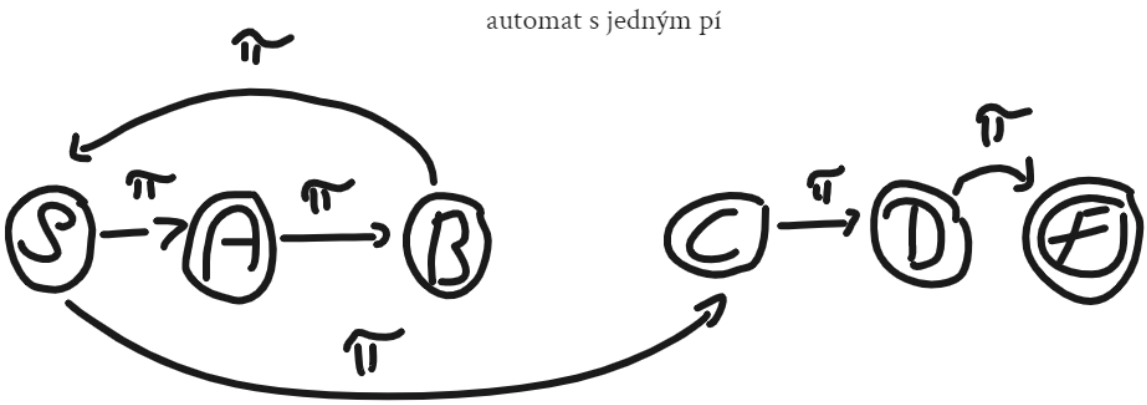
→ Σ = {π, ππ}

→ P:

- **S -> ππA | πB**
- **A -> πS**
- **B -> ππ**



- *Automaton has one cycle, the carousel on "π", which always generates this symbol three times. If he wants to quit, he will have to go through three more "π".*



2. Intellectual debates (2 points)

- **G** = (N, Σ, P, S)

→ N = {S, A, B, C, D, F}

→ Σ = {k, s, i}

→ P:

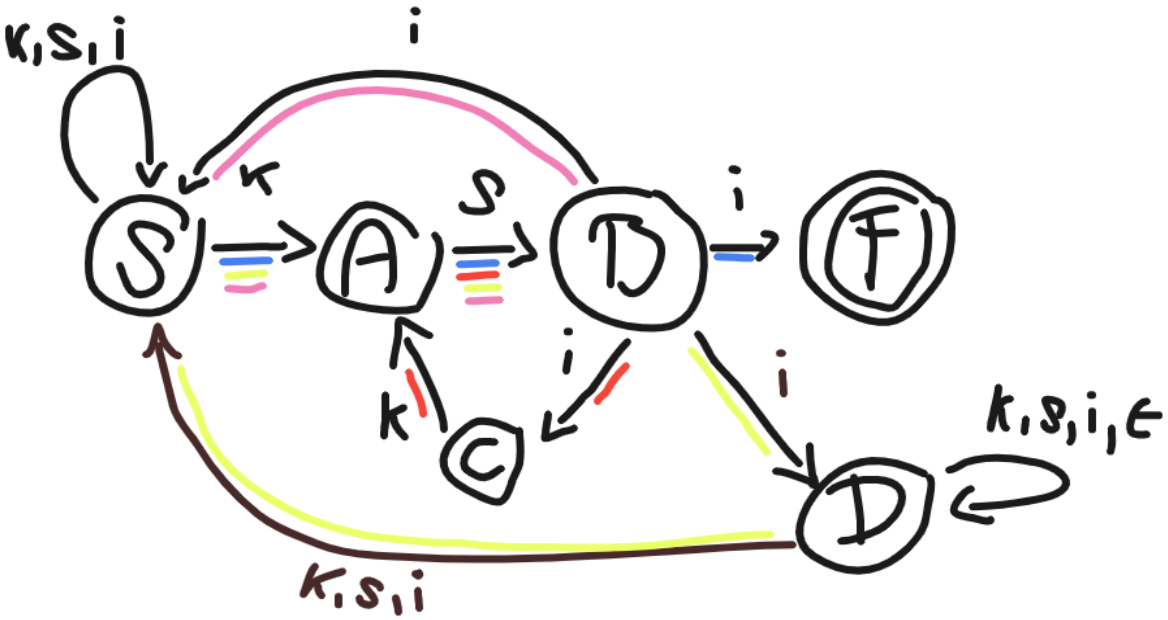
➢ **S -> kS | sS | iS | kA**

➢ **A -> sB**

➢ **B -> iS | iC | iD | i**

➢ **C -> kA**

➢ **D -> kD | sD | iD | kS | sS | iS | ε**



- **Explanation:**
- **Blue path**

→ *Classic case no. 1, "ksi" itself.*

- **Red path**

→ *In this case, I've also included a red path that shows several strings of "ksi" in a row. Although I think it wasn't necessary because of the pink path.*

- **Pink path**

→ *Probably the most likely case that will occur, i.e. a loop of the letters "k,s,i" followed by the string "ksi" and "B -> iS" sends it back to the beginning, where the automaton can choose any path.*

- **Yellow path**

→ *The yellow path is in charge of the loop as our initial state, only with the difference of ending the generation after a successful string "ksi" or continuing further to the initial state "S".*

- *Cases I think my automaton should handle:*
- ***Ksi** (blue path)*
- ***Ksi**ikkskskskskskskskskskiiiiii (yellow path)*
- ***Ksiksiksiksiksiksi** (red path)*
- *Ikikikikikssssssikikikik**ksiksiksiksiksi** (loop S, red path)*
- ***Ksiksiksi**sisisikkkksisi**ksi** (almost all together)*
- ***...***

3. Beer conversation (3 points)

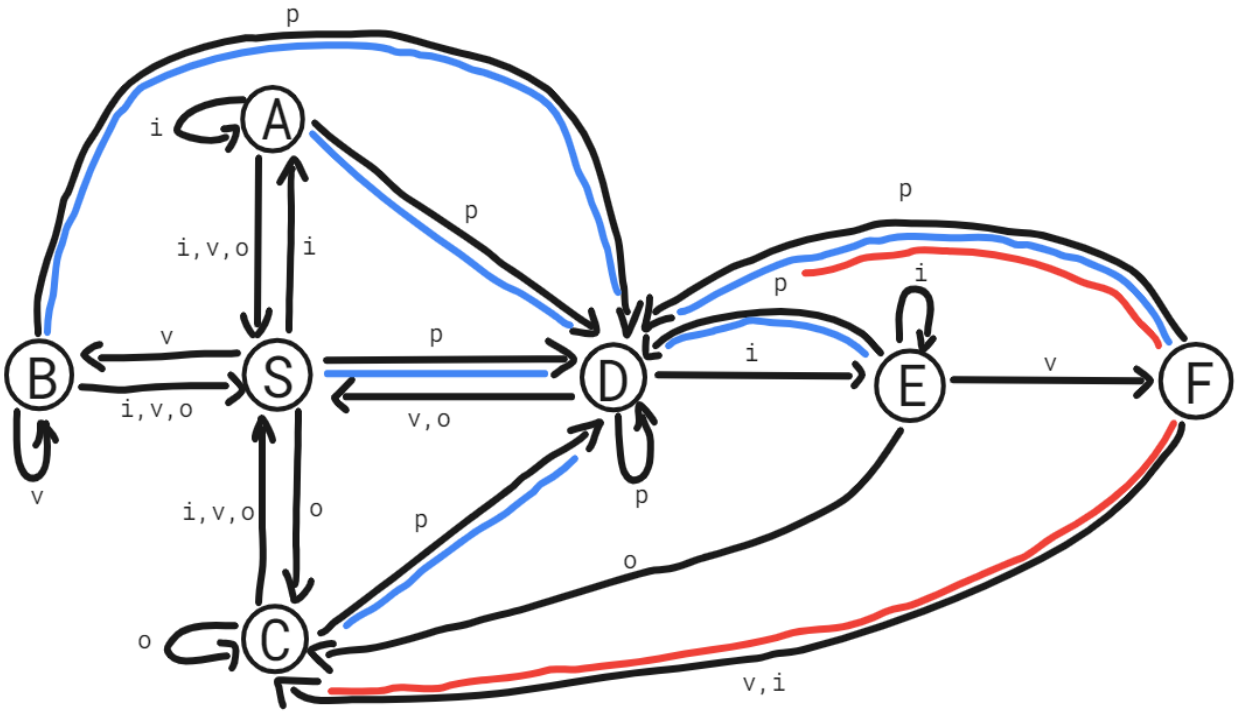
- **G** = (N, Σ, P, S)

→ N = {S, A, B, C, D, E, F}

→ Σ = {p, i, v, o}

→ P:

- **S -> pD | iA | vB | oC | ε**
- **A -> iA |iS | vS | oS | pD | ε**
- **B -> vB | iS | vS | oS | pD | ε**
- **C -> oC | iS | vS | oS | pD | ε**
- **D -> pD | iE | vS | oS | ε**
- **E -> iE | vF | pD | oC | ε**
- **F -> pD | vC | iC | ε**



- **Explanation:**
- Btw. “pivo” translates as beer in Slovakia.
- **Blue path**

→ *Every state that contains the letter "p" leads to the state "D", that is, the state "D" is the beginning of our check if we are trying to sub-word "pivo" in the word.*

- **Červená cesta Red path**

→ *If we get to the state where we have "...pivo", then there cannot be a situation where the letter "o" would be assigned to this word, thus we cannot create any sub-word "pivo". It can be said that the red path prevents the blue one from creating this subword.*

- *I added an empty string to each state, since it can be the word infinite or just something like "vi", even nothing.*

- *Cases I think my automaton should handle:*

- *vipovipip*
- *pivvo*
- ***...***

4. Sign language (1 point)

- **G** = (N, Σ, P, S)

→ N = {S, A, B, C, D, F}

→ Σ = {^, v}

→ P:

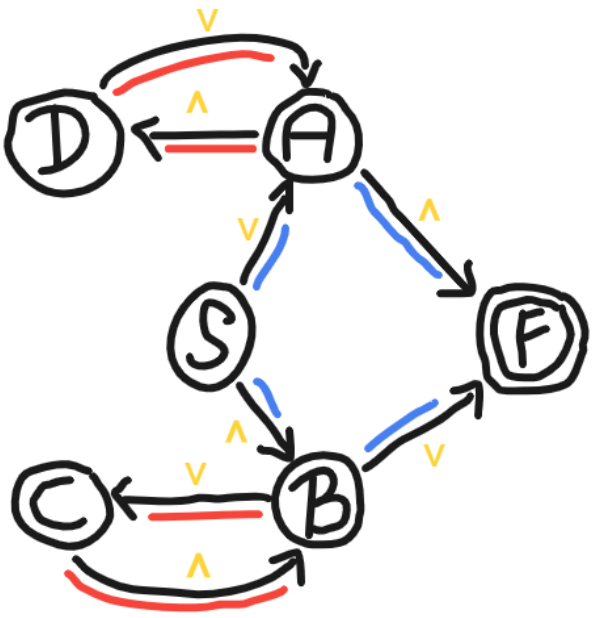
➢ **S -> ^B | vA**

➢ **A -> ^D | ^**

➢ **B -> vC | v**

➢ **C -> ^B**

➢ **D -> vA**



- **Explanation:**
- **Blue path:**

→ *Since the word must contain an even number of symbols and these symbols must not be two in a row, I decided to create two paths, the possibilities that can occur. By that I mean the first symbol of a word (either "^" or "v"), which cannot also be the last symbol of that word. The blue path also handles the shortest possible acceptable words ("^v" or "v^").*

- **Red path**

→ *A simple loop that ensures that there is never a situation where two symbols are repeated.*

5. Math (3 points)

- **G** = (N, Σ, P, S)

→ N = {S, A, B, C, F}

→ Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

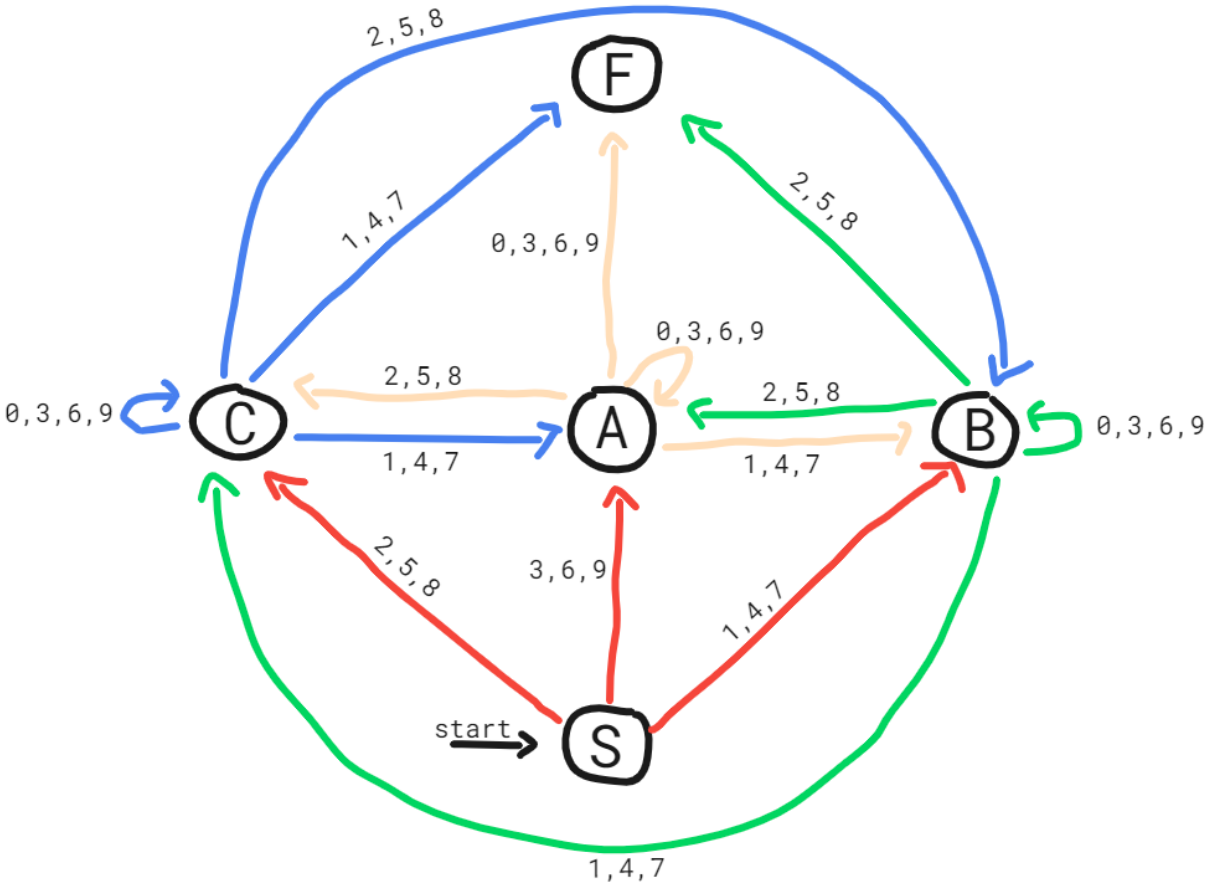
→ P:

➢ **S -> 0 | 3 | 6 | 9 | 3A | 6A | 9A | 1B | 4B | 7B | 2C | 5C | 8C**

➢ **A -> 0 | 3 | 6 | 9 | 0A | 3A | 6A | 9A | 1B | 4B | 7B | 2C | 5C | 8C**

➢ **B -> 2 | 5 | 8 | 0B | 3B | 6B | 9B | 2A | 5A | 8A | 1C | 4C | 7C**

➢ **C -> 1 | 4 | 7 | 0C | 3C | 6C | 9C | 1A | 4A | 7A | 2B | 5B | 8B**



- **Explanation:**
- *Automaton is divided into three groups. The first, the so-called the neutral group contains the numbers: 0, 3, 6, 9. I call it neutral because these digits do not affect the result in any way, i.e., when they are inserted into any number, we know that they are divisible by 3 and we do not need to deal with them anymore. Next, we have two groups, the first to which the numbers belong: 1, 4, 7. The second to which the numbers belong: 2, 5, 8. These groups are divided in the machine so that they complement each other. This means that if we start with the number "2", we can supplement it with a number from the opposite group and always achieve a result divisible by the number 3, e.g. 2 + (1 | 4 | 7) is either 3, 6 or 9 and we get neutral numbers. On the contrary, if we started with the number "1", we can supplement it with the numbers 2, 5, 8, which also create neutral numbers for us. Finally, it should be mentioned that if we can "loop" one group three times, we also get a neutral number (3x1, 3x5, 3x7...)*