Technická Univerzita v Košiciach
Fakulta Elektrotechniky a Informatiky

Zvýrazňovanie syntaxe na základe definície meta-modelu

Matej Gagyi

Úvod

Úspešné dománové jazyky si vyžadujú:

- Vývoj programovacieho jazyka
- Vývoj podporných nástrojov

Pravidlá gramatiky musíme implementovať 2x

Definícia jazyka

```
1  @Parser(className = "yajco.exemple.sml.parser.StateMachineParser",
2  mainNode = "StateMachine",
3  tokens = {
4      @TokenDef(name = "ID", regexp = "[a-zA-Z][a-zA-Z0-9]*")
5  }
6  skips = {
7      @Skip("#.*\\n"), //comment
8      @Skip(" "),
9      @Skip("\\t"),
10     @Skip("\\n"),
11     @Skip("\\r")
12 })
13 package yajco.example.sml.model;
14
15 import yajco.annotation.config.Skip;
16 import yajco.annotation.config.TokenDef;
17
18 import yajco.annotation.config.Parser;
```

Zvýraznená syntax

```
1  # States of the state machine
2  state A
3  state B
4  state C
5
6  # Transitions of the state machine
7  trans A -> B
8  trans B -> C
9  trans A -> C
10
```

Cieľ práce

- Odstrániť duplikáciu úsilia automatizáciou
- Podpora viacerých editorov a knižníc
- Jednoduchá rozšíriteľnosť

Postup práce

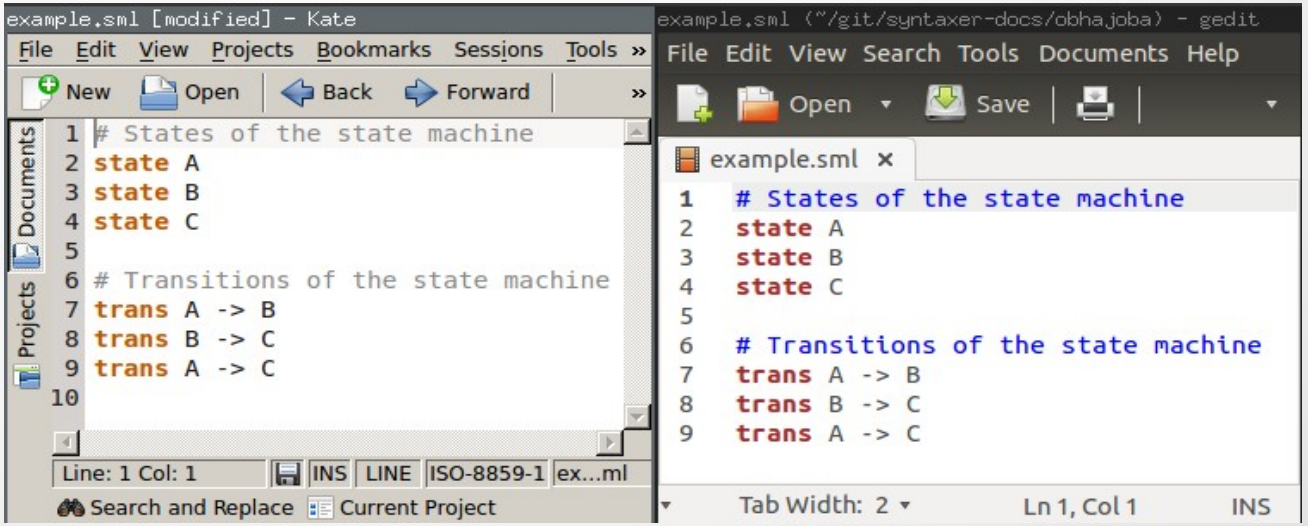
1) Výber editorov
2) Analýza definície syntaxe pre vybrané editori
3) Návrh modelu zvýraznenia syntaxe
4) Návrh procesu transformácie modelu
5) Návrh šablón podľa analyzovaných editorov
6) Vypracovanie textu

Postup práce – Výber editorov

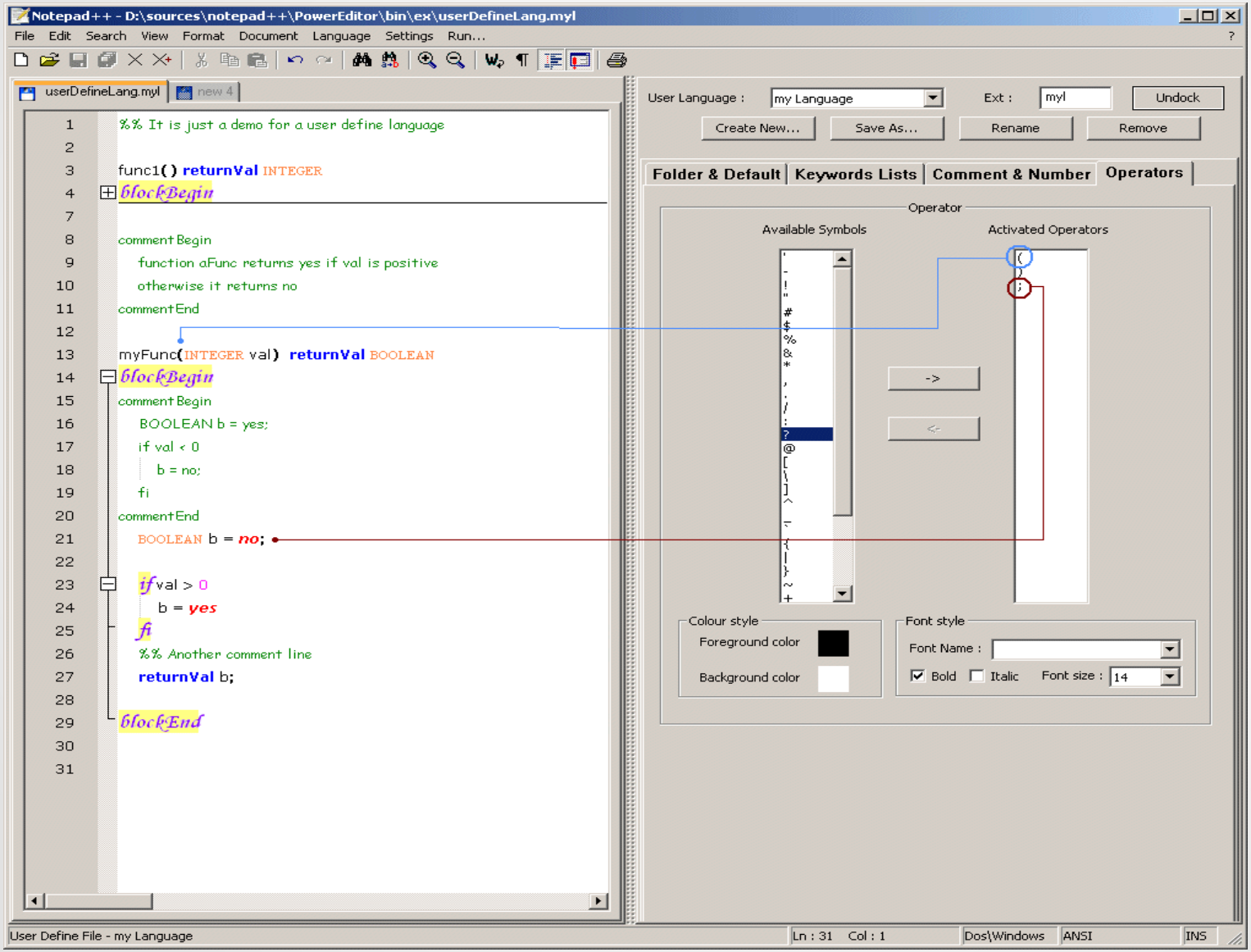
- Vinikajúca dokumentácia umožnila vyvinúť:
  - **Kate a Gedit pre Linux**
- Niektoré koncové definánila ície syntaxe si vyžadujú novú implementáciu transformácie modelu:
  - **Pygments** a **CodeMirror** pre Web
- Veľmi slabá dokumentácia zdržala vývoj pre populárny editor:
  - **Notepad++** pre Windows

Vypracovanie – Kate a Gedit

- **Kate** a **Gedit**
  - Množstvo dokumentácie, IRC kanál
  - Deklaratívny zápis syntaxe
  - Sila



Vypracovanie - Notepad++



Vypracovanie – Model zvýraznenia syntaxe

- Meta-model syntaxe a API YajCo
- Java AnnotationProcessor
- Rozdielnosť editorov

Vypracovanie – Model zvýraznenia syntaxe

- **Meta-model syntaxe a API YajCo**
  - Anotácie YajCO: @TokenDef, @Before
  - CompilerGenerator vs. PrinterGenerator
  - API triedy language

```
 9  public class Transition extends Declaration {
10
11      private String label;
12
13      private State source;
14
15      private State target;
16
17      @Before("trans")
18      @After(";")
19      public Transition(
20              @Token("ID") String label,
21              @Before(":")
22              @Token("ID")
23              @References(value = State.class, field = "source") String sourceLabel,
24              @Before("->")
25              @Token("ID")
26              @References(value = State.class, field = "target") String targetLabel) {
27          this.label = label;
28      }
```

Vypracovanie – Transformácia syntaxe

- Implementované sú nasledovné výstupy transformácie meta-modelu jazyka:
  - kľúčové slová
  - operátori, výrazy
  - Komentáre

Vypracovanie – Transformácia syntaxe

- Problémy nastali pri snahe návrhu pravidiel pre niektoré výstupy:
  - Konštanty – Čísla, rôzne typy, reťazce
  - Polia a komplexné typy
  - Funkcie, volania, identifikátori a menné priestory

Vypracovanie – Transformácia syntaxe

- **Konštanty** – alfanumerické znaky, podtrhovník, ziadne čísla
- **Operátori** – YajCo ponúka koncept @Operator
- **Komentáre** – Analýza regulárnych výrazov, najzložitejšia časť transformácie
- **Konštanty** – Rôznorodosť zápisu regulárnych výrazov, rôzne interpretácie v editoroch. Preddefinované pravidlá v niektorých editoroch by si vyzadovali Analyzátor RE
- **Funkcie a volania** – YajCo neponúka žiaden koncept funkcie a menne príestory @References tiež neposkytujú potrebnú informáciu

Vypracovanie – Transformácia syntaxe

- YajCo používa RE
- Cheme rozoznať komentár od operátora...
- RE generujú jazyk reťazcov
  - “Ako v nich rozoznáme rôzne elementy jazyka?”

Vypracovanie – Transformácia syntaxe

- Java podporuje regulárne výrazy z OS UNIX
- Syntaktická analýza môže využiť zovšeobecnenia
  - `Priklad: ab+ → abb*`

Vypracovanie – Transformácia syntaxe

- **Nevhodnosť reprezentácie**
  - Je náročné hľadať vlastnosti celého jazyka
    - “Aké znaky možu byť na indexe N?”
  - Traverz AST tento dopyt nerieší

Vypracovanie - Šablóny

- **Rozdielnosť editorov**
  - Deklaratívna konfigurácia vs. Imperatívny kód
  - Rozdielne správanie
    - Regulárne výrazy
    - Elementy gramatiky

```
 3  public class KateEditorStrategy extends GeneralEditorStrategy {
 4      private static final String ID = "kate";
 5      private static final String NAME = "Kate";
 6      private static final String TEMPLATE = "kate.vm";
 7      private static final String SUFFIX = "kate.xml";
 8
 9      public KateEditorStrategy() {
10          super(ID, NAME, TEMPLATE, SUFFIX);
11      }
12
13      @Override
14      protected String transformRegexp(String regexp, String type) {
15          return regexp.replaceAll("\\\\n\\)?\\$?$", "\\$");
16      }
17  }
18
```

Vypracovanie – Šablóny: Gedit

```
<definitions>
    <context id="keywords" style-ref="keyword">
      <keyword>state</keyword>
      <keyword>trans</keyword>
    </context>

    <context id="linecomment" style-ref="comment">
      <start>#</start>
      <end>$</end>
    </context>

    <context id="r">
      <include>
        <context ref="keywords"/>
        <context ref="linecomment"/>
      </include>
    </context>
  </definitions>
```

Vypracovanie – Šalbóny: Kate

```
<language name="YajCo yajco.example.sml.model" version="1.00" kateversion="2.4"
section="Sources" extensions="$globs">
  <highlighting>
    <list name="keywords">
      <item>trans</item>
      <item>state</item>
    </list>
    <contexts>
      <context attribute="Normal Text" lineEndContext="#pop" name="Normal Text">
        <keyword attribute="Keyword" context="#stay" String="keywords"/>
        <RegExpr attribute="Comment"  context="#stay" String="#.*$" />
      </context>
    </contexts>
    <itemDatas>
      <itemData name="Normal Text" defStyleNum="dsNormal"/>
      <itemData name="Keyword" defStyleNum="dsKeyword"/>
      <itemData name="Comment" defStyleNum="dsComment"/>
    </itemDatas>
  </highlighting>
</language>
```

Vypracovanie – Problémy

- **Java AnnotationProcessor**
  - Hook javac pre spracovanie annotácií
  - Napojenie na Maven

```
[INFO] ------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------
[INFO] Total time: 2.783s
```

Výsledky

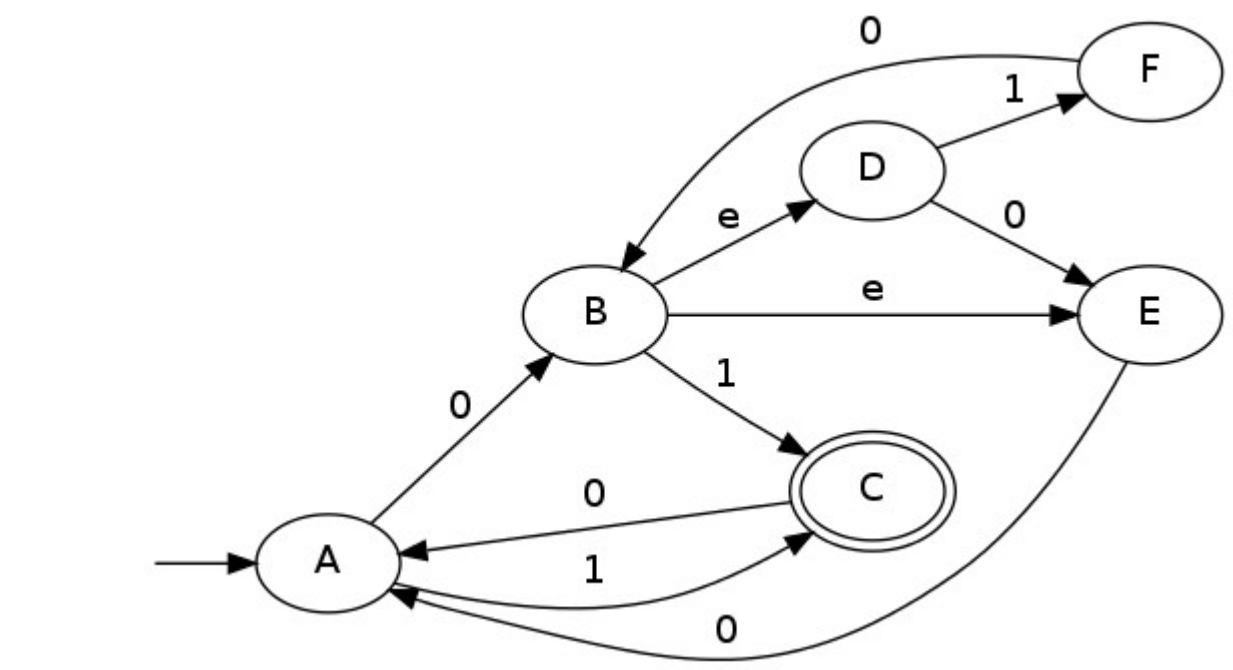
- Implementácia programu na základe zadania úlohy.
- Program podporuje editori Kate a Gedit a je ľahko rozšíriteľný
- Zdrojové kódy YajCo a Syntaxer su dostupné na internete:
  - https://github/com/yin/yajco
  - https://github.com/yin/syntaxer
- 6 ukážkových jazykov je dostupných na internete:
  - https://github.com/yin/yajco-examples

Výsledky

- Slabinou práce je text
- Práca nebola písana podľa pokynov pre vypracovanie záverečných prác (citácie, zdroje, prílohy).
- Chýbajúce prílohy – užívateľská a systémová príručka
- Nesplnenie jedného bodu zadávacieho listu:
  - Opis súborových formátov a logického usporiadania definícií syntaxu pre rôzne editori
  - Šalóny v zdrojových kódoch sú jediné miesto pre čerpanie informácií o týchto formátoch

Dakujem za pozornosť

Zadné vrátka – epsilon-NFA



Zadné vrátka – eNFA definícia

Definícia: Nedeterministický konečný automat s ε-prechodmi je pätica

$$\mathbf{A} = (\mathbf{K}, \mathbf{\Sigma}, \boldsymbol{\delta}, \mathbf{q_0}, \mathbf{F}) \quad \text{kde}$$

K je konečná množina stavov

Σ je konečná vstupná abeceda

$q_0 \in K$ je začiatočný stav

$F \subseteq K$ je množina akceptačných (koncových) stavov

$\delta : K \times \Sigma_\varepsilon \rightarrow P(K)$ je prechodová funkcia

a zároveň

$\varepsilon \notin K$

$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Zadné vrátka - NFA