VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

GENEROVÁNÍ KÓDU Z MODELŮ PETRIHO SÍTÍ

CODE GENERATION FROM OBJECT ORIENTED PETRI NETS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL CIBÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Cibák Michal**
Program: Informační technologie
Název: **Generování kódu z modelů Petriho sítí**
Code Generation from Object Oriented Petri Nets
Kategorie: Softwarové inženýrství

Zadání:

1. Prostudujte problematiku generování zdrojových kódů z modelů softwarového systému. Prostudujte koncept formalismu Objektově orientovaných Petriho sítí (OOPN).
2. Seznamte se s aktuálním řešením generování kódu z OOPN do jazyka Java a proveďte analýzu tohoto řešení ve vztahu k požadavkům.
3. Navrhněte úpravy aktuálního řešení, příp. navrhněte nové řešení, transformace modelů popsaných formalismem OOPN do programovacího jazyka Java (generování zdrojového kódu).
4. Implementujte nástroj pro generování zdrojových kódů z OOPN modelů, který bude respektovat navržené mechanismy transformace. Vytvořte sadu testovacích příkladů.
5. Analyzujte možné problémy a omezení spojená s transformací OOPN modelů do programovacích jazyků. Pro vybrané problémy formálně specifikujte jejich podstatu, důsledky a možná řešení.

Literatura:

- Krysztof Czarnecki, Ulrich Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional, 2000. ISBN-13: 978-0201309775
- YAKINDU Statechart Tools. https://www.itemis.com/en/yakindu/state-machine/
- O. Ringert, A. Roth, B. Rumpe, A. Wortmann: Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems. In: MORSE 2014 - 1st International Workshop on Model-Driven Robot Software Engineering

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kočí Radek, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Cieľom tejto práce je analyzovať aktuálne riešenie generátoru kódu z Objektovo orientovaných Petriho sietí zapísaných v jazyku PNTalk do jazyku Java, navrhnúť jeho úpravy a implementovať ich. Ako prvý bol analyzovaný celkový návrh a boli špecifikované chýbajúce časti, následne bol analyzovaný zdrojový kód prekladača a boli odhalené chyby. Boli implementované navrhnuté opravy chýb v prekladači a niektoré chýbajúce časti z návrhu boli implementované čiastočne v rámci prekladača.

Abstract

The goal of this thesis is to analyze the current solution of a code generator from Object oriented Petri nets written in PNTalk language to Java language, propose changes and implement them. First the overall scheme was analyzed and missing parts were specified, then the source code of the compiler was analyzed and errors were found. The proposed corrections of errors in the compiler were made and some missing parts from the scheme were partially implemented within the compiler.

Kľúčové slová

OOPN, Oobjektovo orientované Petriho siete, PNTalk, Java, prekladač, parser, skener

Keywords

OOPN, Object oriented Petri nets, PNTalk, Java, compiler, parser, scanner

Citácia

CIBÁK, Michal. *Generování kódu z modelů Petriho sítí*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Generování kódu z modelů Petriho sítí

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Radka Kočího, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Michal Cibák
11. mája 2022

Poďakovanie

Ďakujem vedúcemu tejto práce, Ing. Radkovi Kočímu, Ph.D., za poskytnuté konzultácie a ochotu, ktoré boli veľmi nápomocné pri jej vypracovaní.

Obsah

Kapitola 1

Úvod

Petriho siete sú jednoduchým formalizmom využívaným na modelovanie diskrétnych systémov. Ich výhodou je zrozumiteľná grafická podoba a dobrá formálna analyzovateľnosť. Ich použitie však nie je vhodné v rozsiahlejších a podrobných modeloch systémov. Na tento účel sú vhodnejšie obecné programovacie jazyky, ktorých vývoj viedol na zavedenie objektovej orientácie.

Vhodným spojením Petriho sietí a objektovej orientácie je možné získať pozitívne vlastnosti oboch prístupov. Takýchto pokusov bolo viacero, každý výsledok mal svoje výhode a nevýhody. Jedným z nich sú Objektovo orientované Petriho siete, skrátene OOPN, ktorých formálna definícia je inšpirovaná čistou objektovou orientáciou jazyka Smalltalk a je popísaná v dizertačnej práci docenta Janouška [2]. Ako konkrétna implementácia OOPN v systéme Smalltalk je v nej popísaný aj jazyk a systém PNTalk.

Snaha priniesť tento formalizmus do moderného programovacieho jazyka viedla na vytvorenie prekladača a simulátora v jazyku Java [1]. Táto práca na toto riešenie nadväzuje v snahe dostať ho bližšie do podoby finálneho produktu.

Kapitola 2

Aktuálne riešenie

Táto kapitola vychádza z práce "Generování kódu z modelů Petriho sítí"[1].

2.1 Štruktúra aplikácie

Aplikácia je rozdelená do 3 logických častí:

- Prekladač zdrojových súborov písaných v jazyku PNTalk, ktoré popisujú Objektovo orientované Petriho siete [2], do jazyka Java. Skladá sa z parsera a generátora, ktoré sú tvorené niekoľkými triedami, z toho najpodstatnejšie sú:
  - PNParser – riadi preklad zdrojového textu
  - PNScanner – získava požadované časti zdrojového textu
  - PNSyntaxValidator – kontroluje syntaktickú správnosť vybraných častí zdrojového textu
  - PNGenerator – generuje z vnútornej štruktúry Java súbory
- Knižnica simulátoru, ktorá zabezpečuje beh preloženého modelu.
- Spúšťacia aplikácia, ktorá vzniká spojením preloženého modelu a simulátora.

2.2 Štruktúra tried modelu

Podľa definície jazyka PNTalk sú navrhnuté triedy, ktoré reprezentujú model OOPN. Väčšina z nich je využívaná ako pre vnútornú reprezentáciu modelu v prekladači, tak aj v rámci vygenerovaných zdrojových súborov, s ktorými pracuje simulátor, niektoré sú však určené výhradne pre jedno alebo druhé. V nasledujúcich podkapitolách je uvedený ich bližší popis.

2.2.1 Triedy pre prekladač

V tejto časti sú predstavené triedy a rozhranie, ktoré sú využívané iba v prekladači.

PNClasses

PNClasses predstavuje vrchol hierarchie vnútornej reprezentácie OOPN v prekladači. V jej inštancii je uchovaný celý model ako zoznam všetkých tried modelu a meno hlavnej triedy.

PNClazz

PNClazz predstavuje jednu triedu z modelu. V jej inštancii sú uložené dáta potrebné pre vygenerovanie zdrojových súborov v jazyku Java. Meno triedy sa využíva na začiatku názvu príslušných súborov, meno rodičovskej triedy je potrebné pre riešenie prípadnej dedičnosti, objektová sieť predstavuje buď celú sieť, alebo len zmeny voči sieti v rodičovskej triede v prípade dedenia, a nakoniec sú uchované zoznamy metód, konštruktorov a synchrónnych portov, ktoré sú buď nové, alebo nahrádzajú definíciu z rodičovskej triedy.

IPNNet, PNNet, PNMethodNet a PNConstructor

IPNNet je rozhranie pre akýkoľvek typ siete. PNNet predstavuje sieť zloženú z miest a prechodov, ktoré sú v jej inštancii uložené v zoznamoch, využívaná je pre objektovú sieť. PNMethodNet predstavuje sieť metódy, pričom dedí z PNNet a v inštancii má naviac správu, na ktorú reaguje. PNConstructor je obdobný PNMethodNet, ale predstavuje sieť konštruktora.

2.2.2 Triedy pre simulátor

V tejto časti sú predstavené triedy a rozhranie, ktoré sú využívané iba v simulátore a vygenerovaných súboroch, s ktorými pracuje. V týchto súboroch je zdrojový kód triedy, ktorá dedí z niektorej z týchto tried.

IPN, PN, PNClass, PNMethod

IPN je rozhranie pre sieť. PN je abstraktná trieda s atribútmi pre množinu miest a zoznam prechodov, slúži ako základ pre triedy PNClass a PNMethod, ktoré z nej dedia a ktoré sú použité ako rodičovské triedy vo vygenerovaných súboroch. PNClass predstavuje triedu OOPN a ako atribúty pridáva mapy pre synchrónne porty a metódy, ktoré jej prislúchajú. PNMethod predstavuje metódu v triede OOPN a ako atribúty pridáva triedu OOPN, ktorej patrí, a zoznam jej argumentov.

2.2.3 Spoločné triedy

V tejto časti sú predstavené triedy a rozhrania, ktoré sú využívané aj pre vnútornú reprezentáciu zdrojového textu v prekladači, aj vo vygenerovaných súboroch a v simulátore.

PNMultiset a PNMultisetItem

Trieda PNMultiset odpovedá v OOPN jednej hrane medzi miestom a prechodom spolu s jej hranovým výrazom. Vyjadruje, z akého miesta má odobrať značky alebo do akého miesta ich naopak pridať, čomu odpovedajú aj jej atribúty. Objekt tejto triedy v sebe má inštanciu miesta, v ktorom vykonáva zmeny, a zoznam objektov typu PNMultisetItem, ktoré určujú, koľko akých značiek sa pridá alebo odstráni. V inštancii PNMultisetItem sú teda uložené dve značky, jedna vyjadruje množstvo a druhá typ.

PNExpression

Táto trieda predstavuje jeden výraz z akcie alebo stráže prechodu. Jej inštancia obsahuje zoznam správ, značku, ktorej sú posielané a prípadne zoznam premenných, do ktorých je výsledok uložený.

IPNMessage, PNMessageUnary, PNMessageBinary, PNMessageKey a PNMessageKeyPair

IPNMessage je rozhranie pre všetky typy správ. Správa vyjadruje, aká metóda alebo synchrónny port sa má zavolať a s akými argumentami. Trieda PNMessageUnary predstavuje unárnu správu, teda správu bez argumentov, v ktorej inštancii stačí uschovať selektor. Trieda PNMessageBinary predstavuje binárnu správu, teda správu s jedným argumentom, ktorá okrem selektora v inštancii má aj operand. Trieda PNMessageKey predstavuje kľúčovú správu, teda správu s potenciálne viacerými pomenovanými argumentami, ktorej inštancia obsahuje zoznam PNMessageKeyPair objektov. Tie obsahujú selektor a operand.

IPNToken, PNToken, PNTokenBoolean, PNTokenInteger, PNTokenCharacter, PNTokenString, PNTokenSymbol, PNTokenVariable, PNTokenPnObject, PNTokenArray

IPNToken je rozhranie pre všetky typy značiek. PNToken je abstraktná trieda, ktorá predstavuje obecný typ značky, má teda aj obecný atribút typu Object pre uschovanie hodnoty danej značky. Triedy, ktoré predstavujú konkrétny typ značky, z tejto abstraktnej triedy dedia. PNTokenBoolean má v tomto atribúte objekt typu boolean, pre PNTokenInteger to je integer, PNTokenCharacter využíva String, tak ako aj PNTokenString, PNTokenSymbol a PNTokenVariable, PNTokenPnObject využíva tiež typ String, ktorý vyjadruje názov triedy z OOPN, ale pridáva ešte atribút pre uschovanie inštancie danej triedy. PNTokenArray predstavuje zoznam, tie ale nie sú podporované.

PNPlace

Trieda PNPlace predstavuje miesto v sieti. V inštancii tejto triedy je uložené meno daného miesta, zoznam značiek, ktoré sa v ňom nachádzajú, a zoznam výrazov, ktoré slúžia ako inicializačná akcia, teda k počiatočnému značeniu ešte môžu pridať ďalšie značky.

PNTransition, PNCondition, PNGuard a PNAction

Táto trieda predstavuje prechod v sieti. V jej inštancii sa uchováva meno daného prechodu, vstupná, výstupná a obyčajná podmienka, akcia, stráž a mapa premenných, ktoré využíva simulátor. PNCondition má ako atribút zoznam PNMultiset objektov, teda vyjadruje, na ktoré miesta bude mať aký vplyv. PNGuard aj PNAction majú ako atribút zoznam výrazov a odpovedajú stráži a akcii prechodu.

PNSynchronousPort

PNSynchronouPort je trieda, ktorá predstavuje synchrónny port. Ako atribúty má správu, na ktorú reaguje, vstupnú, výstupnú a obyčajnú podmienku, stráž a zoznam argumentov, ktoré sú využívané v simulátori.

2.3 Implementácia

Správne spracovanie Objektovo orientovanej Petriho siete zapísanej v jazyku PNTalk a naplnenie navrhnutej štruktúry popísanej v časti 2.2 je základom pre to, aby mohol byť vygenerovaný kód v jazyku Java, ktorý odpovedá pôvodnej OOPN. Toto spracovanie zabezpečuje parser. Je dôležité vedieť, ako je implementovaný, aby bolo možné identifikovať

príslušné miesta, kde je potrebné doplniť chýbajúcu funkcionalitu, ako bolo popísané v časti 3.1.3. Výslednú vnútornú reprezentáciu takto spracovanej OOPN využije generátor pre vygenerovanie súborov v jazyku Java, čím končí úloha prekladača.

2.3.1 Scanner

Scanner je implementovaný ako abstraktná trieda so statickými metódami a stará sa o delenie zdrojového textu v jazyku PNTalk na časti podľa jeho štruktúry. Na rozdiel od typického scanneru [3] neobsahuje metódu, ktorá by vracala token odpovedajúci vždy prvej lexikálnej jednotke na vstupe. Namiesto toho je kód členený do metód odpovedajúcich štruktúre OOPN. Týmto metódam je v argumente predaná nimi očakávaná časť zdrojového textu, v ktorej vyhľadajú požadovanú časť alebo viaceré časti, uložia ich do zoznamu a vrátia v objekte typu PNScannerData, ktorý okrem tohto zoznamu obsahuje aj pôvodný reťazec z argumentu metódy upravený tak, že nájdené podreťazce sú z neho odstránené. Parser teda volá metódy scanneru podľa toho, ktorú časť zdrojového textu práve spracúva. Prvá metóda z celého zdrojového textu vystrihne deklaráciu hlavnej triedy. Zvyšný text, predstavujúci triedy OOPN, očakáva na vstupe druhá metóda, ktorá ho rozdelí na jednotlivé definície tried. Ďalšie metódy z definície triedy získajú jej hlavičku, objektovú sieť, metódy a synchrónne porty, iné zo siete získajú miesta a prechody, obdobne sú z prechodu získané podmienky, akcia a stráž. Princíp je rovnaký aj pre ostatné časti. Vyhľadanie požadovanej časti pracuje tak, že sú nájdené kľúčové slová, ktoré ju ohraničujú. Napríklad deklarácia hlavnej triedy začína kľúčovým slovom main a končí pred definíciou triedy, ktorá začína kľúčovým slovom class. Tá zas končí pred začiatkom definície ďalšej triedy. Jej hlavička je na začiatku a končí pred telom, ktoré môže obsahovať objektovú sieť, metódy, konštruktory a synchrónne porty, uvádzané ich príslušnými kľúčovými slovami. Obdobne je to aj pre ďalšie časti zdrojového textu, viac o kľúčových slovách je možné nájsť v syntaxi jazyka PNTalk v prílohe A. Funkcionalita vyhľadania a vystrihnutia podreťazca podľa kľúčových slov je implementovaná v dvoch obecných metódach, a to getDataByDelimiters a getContentAsListByKeywords, výber časti pred kľúčovým slovom je riešený individuálne v príslušných metódach. Metóda getDataByDelimiters zo zadaného reťazca vystrihne každú časť medzi dvoma poskytnutými oddeľovačmi, či medzi prvým oddeľovačom a koncom reťazca. Tieto oddeľovače by mali mať podobu regulárneho výrazu odpovedajúcemu jednému kľúčovému slovu alebo zoznamu kľúčových slov oddelených logickou operáciou or. Predpokladá sa, že prvý oddeľovač nebude zoznam, preto toto kľúčové slovo nie je ponechané ani v pôvodnom, ani vo výslednom vystrihnutom reťazci – nemalo by byť potrebné, keďže sa vie, že ním musel začínať. Metóda getContentAsListByKeywords je podobná, ale oddeľovače jej nie sú predávané v argumentoch, využíva zoznam kľúčových slov z abstraktnej triedy PNScannerConstants, v ktorej sú definované konštanty predstavujúce kľúčové slová a niektoré dôležité znaky podľa syntaxe jazyka PNTalk, ako aj regulárne výrazy predstavujúce vybrané zoznamy kľúčových slov. Je volaná tam, kde ako počiatočný a koncový oddeľovač môže byť viac kľúčových slov, teda pri získaní častí prechodu a synchrónneho portu. V jej prípade však musí byť počiatočné kľúčové slovo vo výslednom vystrihnutom reťazci ponechané, aby bolo možné neskôr identifikovať, o akú časť z niekoľkých možností sa v skutočnosti jedná. Okrem nich sú ešte v scanneri metódy na získanie prvej sekvencie nebielych znakov z reťazca, odstránenie prvého výskytu zadaného reťazca v reťazci (keďže sú reťazce v Jave nemenné, táto metóda vracia nový, upravený reťazec), získanie reťazca pred zadaným oddeľovačom a rozdelenie reťazca podľa bielych znakov (na rozdelenie sa nerátajú biele znaky, ktoré sú súčasťou PNTalkového literálu predstavujúceho reťazec).

2.3.2 Parser

Parser je implementovaný podľa návrhového vzoru singleton, Jeho kód je členený do metód tak, aby odpovedali štruktúre zdrojového textu v jazyku PNTalk. Na najvyššej úrovni je metóda spracúvajúca celý text, z ktorej sú volané metódy spracúvajúce jeho časti, ktoré volajú ďalšie metódy spracúvajú ich časti, až kým sa nezavolá metóda na najnižšej úrovni, v ktorej sa text ďalej nedelí. Každá metóda teda očakáva v argumente predaný určitý úsek zdrojového textu, ktorý spracuje a buď vráti nový objekt niektorej triedy popísanej v časti 3.2, ktorá prislúcha spracovávanému reťazcu, alebo naplní objekt, ktorý jej bol predaný ako argument. Získanie požadovanej časti reťazca je zabezpečené volaním príslušných metód zo scanneru. Tento podreťazec je potom buď predaný ďalšej metóde, alebo je rovno overená jeho syntaktická správnosť, a to buď volaním príslušnej statickej metódy z abstraktnej triedy PNSyntaxValidator pre literály, selektory správ a výrazy, alebo je kontrola vykonaná priamo v danej metóde pre iné časti. Niektoré reťazce môže byť potrebné rozdeliť na jednotlivé časti podľa určitého znaku, čo je vyriešené volaním metódy split s príslušným znakom. Podľa bodky sú od seba oddelené výrazy, podľa čiarky hranové výrazy a jednotlivé časti multisetov a podľa apostrofu počty značiek od ich hodnoty. Správnosť väčšiny kľúčových slov nie je kontrolovaná, pretože pokiaľ podľa nich bol určitý reťazec rozdelený, museli sa na danej pozícii nachádzať a teda aj byť správne. V prípade nájdenia chyby je vyhodená výnimka s dodatočnými informáciami. Spracovanie zdrojového textu začína v počiatočnej metóde, ktorej je predaný celý zdrojový text a jej výsledkom je naplnený objekt typu PNClasses, ktorý predstavuje túto Objektovo orientovanú Petriho sieť. V tomto texte sa vyhľadá deklarácia hlavnej triedy OOPN, predá sa metóde, ktorá overí jej syntaktickú správnosť, a zvyšok textu je rozdelený na definície jednotlivých tried. Každá je predaná metóde, ktorá sa postará o jej spracovanie a vráti objekt typu PNClazz. V nej sa vyhľadá hlavička, predá sa príslušnej metóde na jej spracovanie, rovnako pre objektovú sieť, metódy a synchrónne porty. Obdobne je to aj v týchto metódach a postupuje sa až dovtedy, kým sa nespracuje najmenšia časť, ktorej prislúcha niektorá trieda z časti 2.2.

Kapitola 3

Analýza aktuálneho riešenia

Táto kapitola sa venuje analýze riešenia predstaveného v kapitole 2.

3.1 Analýza návrhu

3.1.1 Generované súbory reprezentujúce model

Generované Java súbory sú na úpravu nepraktické. V prípade nutnosti zmeny je lepšie túto spraviť v PNTalk zdrojovom texte, z ktorého tieto súbory boli vygenerované, a preložiť ho znovu. Je preto dôležité, aby prekladač fungoval správne a vedel odhaliť všetky chyby, pretože nie je zaručené, že zdrojové texty v PNTalku budú generované nástrojom a teda správne, ale môžu byť ručne upravené.

3.1.2 Návrhový vzor singleton

Ako bolo spomenuté v časti 2.3.2, parser je implementovaný podľa tohto návrhového vzoru. Podľa literatúry [4] sa tento vzor okrem iného zvykne používať aj v prípadoch, keď by mohla byť daná trieda implementovaná ako abstraktná (nemožno vytvárať inštancie) a metódy ako statické, čo je tento prípad. Význam to ale nemá, pretože výhody tohto vzoru, ako je možnosť dynamicky vybrať jednu z možných implementácií alebo predísť problémom s nevhodným poradím inicialíácie, ktoré môžu nastať, pokiaľ je daná trieda prepojená s inými, sa v tomto prípade užitia neprejavia.

3.1.3 Chýbajúce časti

Fryc v závere svojej bakalárskej práce [1] uvádza, že v rámci jeho implementácie nie sú podporované všetky možnosti, ktoré sú súčasťou jazyka PNTalk.

Zo syntaxe sú to:

- Symbol temps, ktorý predstavuje dočasné premenné počiatočnej akcie miesta a akcie prechodu
- Symbol list, ktorý predstavuje zoznam v multimnožine
- Zátvorky vo výraze, ktoré určujú prednosť v postupnosti jeho vyhodnocovania
- Symbol cascade a jemu prislúchajúcu bodkočiarku, ktoré predstavujú postupnosť správ zaslaných úvodnému objektu vo výraze

- Symbol arrayconst a array, ktoré predstavujú typ pole

Ďalej je to dedičnosť tried. Dedenie základnej triedy PN je automatické, je braná ako knižnica metód nepopísaných OOPN a je dostupná pre každú triedu. Pri dedení z vlastnej triedy musí byť umožnené potomkovi predefinovať jednotlivé časti sietí, ktoré boli definované v jeho rodičovskej triede, a umožniť vrátiť sa k pôvodnej definícii predefinovanej metódy, čo aktuálne riešenie neumožňuje.

V kapitole o návrhu štruktúry tried tiež uvádza, že z implementácie boli vynechané konštruktory, ktoré je možné nahradiť metódami vykonávajúcimi ekvivalentnú činnosť. Z diagramu tried značiek a jeho popisu ďalej vyplýva, že číselné literály sú reprezentované triedou PNTokenInteger, teda je možné reprezentovať iba celé čísla, desatinné čísla nie sú podporované.

V kapitole s popisom fungovania simulátoru sú uvedené typy správ, ktoré simulátor podporuje. Pre zistenie, o ktoré konkrétne správy sa jedná, je nutné nahliadnuť do kódu simulátoru. Z unárnych správ sú to print a println, ktorým rozumie každý typ značky, new, ktorej rozumie značka typu PNTokenPnObject, a metódy a synchrónne porty z definície vlastnej triedy OOPN. Z binárnych správ sú to numerické operácie +, -, * a / (sčítanie, odčítanie, násobenie a celočíselné delenie), a booleovské operácie <, >, =, <=, >= a = (menší, väčší, rovný, menší alebo rovný, väčší alebo rovný, nerovný), ktorým rozumie značka typu PNTokenInteger. Kľúčové správy podporuje simulátor iba tie, ktoré sú v definícii vlastnej triedy OOPN. Implementované sú teda iba niektoré správy na demonštráciu funkčnosti simulátoru a je potrebné rozšíriť knižnicu podporovaných správ pre jednotlivé typy značiek.

3.1.4 Návrh úprav a doplnkov riešenia

Ako prvé treba doplniť syntaktickú kontrolu nepodporovaných častí, aby správny vstup nebol označený za nesprávny, ale radšej bolo oznámené, že daná syntaktická konštrukcia nie je podporovaná.

Následne treba doplniť chýbajúce typy značiek a ich spracovanie, konkrétne PNTokenArray, PNTokenList, PNTokenNil a PNTokenReal. Následne sa môže prejsť k implementácii ich podpory v generátore a simulátore.

Na záver je potrebné doplniť knižnicu metód ako pre staré, tak aj pre nové typy značiek.

3.2 Analýza kódu a testovanie

Popri komentovaní kódu bol zároveň manuálne kontrolovaný a bolo vykonaných niekoľko testov, ktoré viedli na odhalenie chýb popísaných v podkapitole nižšie.

3.2.1 Nedostatky

Pri vystrihávaní požadovaných častí z reťazca, ktorý už sám môže byť vystrihnutý z iného reťazca, je náročné sledovať, na akom mieste sa daná časť nachádza v ohľade k celému zdrojovému textu. Preto v prípade nájdenia chyby nie je možné v chybovej správe uviesť, kde v texte sa nachádza, iba o akú chybu sa jedná. Taktiež nie je z danej metódy možné dostať sa k informácii, ktoré sú dostupné na mieste, odkiaľ bola volaná. Napríklad v prípade výskytu výnimky v definícii objektovej siete nie je možné v priloženej správe poskytnúť informáciu o tom, v ktorej triede sa táto objektová sieť nachádza. Aby bolo možné poskytnúť dostatočné množstvo informácií na určenie pozície, kde bola daná chyba nájdená, bolo by nutné odchytávať všetky potenciálne výnimky a poskladať tak cestu od chyby až k triede.

Pri mnohých výnimkách však nie je vôbec priložená chybová správa, alebo neposkytuje dostatočnú informáciu o type chyby. So získavaním častí z reťazca je tiež spojená možná chyba, kde po vystrihnutí očakávaných častí v pôvodnom reťazci ostane nejaký obsah. Tento typ chyby nie je kontrolovaný.

V niektorých prípadoch záleží na poradí a počte častí daného reťazca, táto kontrola však nie je uskutočnená. Jedná sa o telo definície triedy, kde nepovinná definícia objektovej siete má byť ako prvá, a o telo definície synchrónneho portu a prechodu, kde poradie nepovinných častí je „podmienka, vstupná podmienka, stráž, akcia (iba v prípade prechodu) a výstupná podmienka“. Pri vystrihnutí časti reťazca medzi dvoma kľúčovými slovami sa stratí informácia o jej pozícií, preto je overenie nutné uskutočniť ako prvé. Pri rozdelení reťazca podľa viacerých kľúčových slov je možné poradie skontrolovať aj neskôr, pretože zoznam podreťazcov je zoradený rovnako, ako boli aj v pôvodnom reťazci. Počet jednotlivých častí je v oboch prípadoch možné zistiť z vráteného zoznamu všetkých nájdených podreťazcov.

Metódy scannera getDataByDelimiters, getContentAsListByKeywords a niektoré ďalšie vyhľadávajú v zadanom reťazci pomocou regulárneho výrazu zhodu s požadovaným kľúčovým slovom, avšak bez akýchkoľvek obmedzení. To znamená, že toto slovo môže byť nájdené aj uprostred iného identifikátora alebo ako súčasť PNTalkového literálu typu reťazec. V oboch prípadoch sa jedná o nesprávnu identifikáciu kľúčového slova.

Delenie reťazca podľa bielych znakov síce preskočí zhodu v PNTalkovov lieráli typu reťazec, pre typ znak už táto kontrola nie je vykonaná. Odstránením takejto medzery sa tento literál stáva neplatným, lebo úvodný znak $ nebude ničím nasledovaný. Rovnaká kontrola by mala byť prevedená aj v prípade, že sa reťazec delí podľa bodky, čiarky alebo apostrofu. K deleniu by nemalo dôjsť ani uprostred zátvoriek, keďže ale v riešení nie sú podporované zátvorky vo výrazoch, typ pole a ani zoznamy, jediné miesto, kde by mohol pri takomto delení nastať problém, je v podmienkach. V nich sú jednotlivé hranové výrazy uzatvorené v zátvorkách a oddelené čiarkou, kde jeden výraz predstavuje multiset, ktorého hodnoty sú tiež oddelené čiarkou. Pokiaľ teda multiset obsahuje viac hodnôt, pri oddeľovaní jednotlivých hranových výrazov dôjde zároveň aj k rozdeleniu tohto multisetu.

Kontrola správnosti výrazov, ako aj ich spracovanie, sa líši voči syntaxi jazyka PNTalk nie len v tom, že nie sú podporované zátvorky, ale bola prispôsobená schopnostiam simulátora tak, aby bolo možné demonštrovať jeho základnú funkcionalitu. Napríklad zložené správy sú rozdelené na postupnosť jednoduchých správ iba v prípade, že sa jedná o zloženú binárnu správu, zložená unárna správa nie je podporovaná.

S nepodporovanou dedičnosťou tried je spojená aj chýbajúca podpora pseudopremenných super a self. Zmena v prekladači však nie je nutná, na ich reprezentáciu môže byť použitá značka typu PNTokenVariable a spracovanie vyriešené v simulátore, kde by sa skontrolovalo meno premennej pred tým, než by sa s ňou začalo pracovať.

Napriek tomu, že značka typu boolean existuje a v simulátore je aj využívaná, v prekladači sa výskyt identifikátorov true a false nekontroluje a táto značka sa nevytvára. Namiesto toho sa vytvorí premenná s týmto menom.

Pri kontrole reťazca, či sa jedná o literál typu číslo, nie sú oproti popisu v syntaxi jazyka PNTalk akceptované žiadne biele znaky, číslo musí byť celé spolu. Pretože sa výrazy delia podľa medzier, kde je výnimka iba medzery v literáli typu reťazec, takéto číslo by bolo tiež rozdelené a teda aj nesprávne skontrolované. Tak, ako to vyplýva z názvu typu značky PNTokenInteger, napriek predošlej kontrole sú podporované iba celé čísla, a to vo formáte, ktorý akceptuje metóda parseInt() triedy Integer.

Z definície metódy či synchrónneho portu prislúchajúcej kľúčovej správe, kde selektor tejto správy je zložený z viacerých častí, je vygenerovanému súboru priradené meno iba podľa prvej časti selektora. Nie je teda možné mať unárnu správu s rovnakým menom, ako prvá časť selektora kľúčovej správy bez dvojbodky. Zároveň nie je možné mať dve kľúčové správy s rovnakým počiatočným selektorom.

Pri generovaní súborov prislúchajúcich jednotlivým triedam, metódam a synchrónnym portom sa z príslušnej zložky nevymazávajú pôvodné súbory. Pokiaľ by teda nastala situácia, že je pôvodný zdrojový kód OOPN zmenený tak, že sa odstráni niektorá časť, ktorej prislúcha vygenerovaný súbor, tento súbor tam ostane. Tým pádom dochádza k nekonzistencii medzi pôvodným a preloženým modelom.

3.2.2 Návrh úprav

Všetky chyby spomenuté v predošlej podkapitole je nutné opraviť, aby bola zaručená správna funkcia prekladača. Ďalej by mohla byť doplnená aj kontrola identifikátorov na miestach, kde sa vie, aký typ identifikátora tam byť môže alebo nie. Hlásené by to potom mohlo byť ako syntaktická chyba. Napríklad mená tried by mali začínať veľkým písmenom, takže id s malým počiatočným písmenom na tom mieste by bola chyba.

Kapitola 4

Implementácia

Ako prvý bod implementácie bola oprava spracovania zdrojového textu a kontrola syntaxe.

Aby metóda getDataByDelimiters nehľadala zhody uprostred iných slov, bol upravený regulárny výraz tak, aby nájdené slovo bolo akceptované na delenie len vtedy, ak bolo na začiatku reťazca alebo bol pred ním biely znak, a zároveň bol za ním biely znak alebo koniec reťazca. Následne ešte prebehne kontrola, či sa nájdené slovo nenachádza uprostred literálu typu reťazec pomocou novo pridanej metódy, ktorá kontroluje od začiatku reťazca počet apostrofov označujúcich jeho začiatok a koniec. Ďalej bola táto metóda zobecnená, aby aj prvý oddeľovač mohol byť zoznam kľúčových slov oddelený logickou operáciou or, tým pádom bolo potrebné pridať aj možnosť počiatočný oddeľovač ponechať vo vystrihnutom reťazci. Vďaka tejto úprave bolo možné nahradiť množstvo opakujúceho sa kódu z iných metód a zabezpečiť v nich správny výber podľa kľúčových slov.

Ďalej bola pridaná metóda getContentBeforeDelimiter, ktorá vykonáva rovnakú kontrolu, ale vracia obsah od začiatku reťazca po vybrané kľúčové slovo. Toto opäť viedlo k odstráneniu rovnakého kódu z niekoľkých metód a zabezpečeniu správneho výberu podľa kľúčového slova.

Nasledovalo pridanie metód, ktoré by pri delení reťazca podľa určitého znaku alebo podľa sekvencie bielych znakov rešpektovali zátvorky a literály typu string a charconst. Bolo potrebné byť opatrný pri kontrole charconst, pretože sa od aktuálneho znaku treba pozrieť o jeden doľava, či to nie je znak dolár, a pokiaľ áno, tak ešte o jeden doľava by už dolár byť nemal. To by pri zhode na začiatku reťazca viedlo k prístupu na znak pred začiatkom reťazca. Zároveň pri delení podľa bodky bolo nutné kontrolovať o jeden znak doprava, kde je vyžadovaný biely znak pre rozlíšenie bodky, ktorá by mohla byťž súčasťou čísla.

Ďalšie úpravy zahŕňali väčšinu metód v menšom či väčšom rozsahu, nakoľko sa v nich nachádzalo veľké množstvo logických chýb, ako v parseri, tak aj v scanneri. Mimo iné boli pri kontrole id na príslušné miesta pridané aj kontroly na vyhradené identifikátory, medzi ktoré patria aj kľúčové slová, a pokiaľ bolo na danom mieste možné určiť, aký typ identifikátora sa tam môže nachádzať, bola pridaná kontrola aj naň.

Nakoniec boli pridané chýbajúce typy značiek a pridaná funkcionalita na ich vytvorenie. Pokiaľ sa pri preklade zdrojového textu vyskytne reťazec prislúchajúci takejto značke, bude správne spracovaný a značka sa vytvorí, následne však bude vyhodená výnimka upozorňujúca na to, že generátor ani simulátor už s touto značkou pracovať nevedia.

Generátor ani simulátor neboli upravované.

Kapitola 5

Testovanie

Počas analýzy zdrojového kódu pôvodného riešenia boli vytvárané jednoduché zdrojové texty v jazyku PNTalk, ktorých účelom bolo demonštrovať odhalené nedostatky riešenia. Podľa odhalenej chyby bola navrhnutá Objektovo orientovaná Petriho sieť tak, aby sa pri jej spracovaní prejavila chyba prekladača. Tá mohla byť dvojakého typu:

- chyba v textovom popise danej OOPN nebola odhalená a preklad prebehol úspešne
- správny textový popis danej OOPN spôsobil chybu pri jeho preklade

Po odstránení príslušnej chyby bolo upravené riešenie otestované rovnakým spôsobom. Pôvodné riešenie malo vlastnú sadu príkladov, ktoré predstavovali slúžili na demonštrovanie úspešného prekladu a simulácie v aktuálnom stave daného riešenia. Tieto príklady boli opätovne preložené upraveným riešením a pri troch z nich, konkrétne basicInput5, intermediateInput3 a advancedInput6, bola odhalená chyba, kde definícia vstupnej podmienky predchádzala definíciu obyčajnej podmienky.

Kapitola 6

Záver

V rámci tejto práce bolo analyzované predošlé riešenie z bakalárskej práce Generování kódu z modelů Petriho sítí [1], identifikované jeho nedostatky a navrhnuté úpravy. Účelom úprav v prekladači bolo doplniť syntaktickú kontrolu a zabezpečiť korektné spracovanie zdrojového textu napísaného v jazyku PNTalk do vnútornej štruktúry, a tiež rozšíriť spracovanie o chýbajúce časti.

Boli implementované navrhnuté úpravy v triedach PNParser, PNScanner, PNSyntaxValidator a PNScannerConstants, a boli doplnené chýbajúce typy značiek tak, že po ich syntaktickej kontrole sa spracujú a následne je vyhodená výnimka, ktorá opozorňuje, že v nadväzujúcich častiach programu, teda v generátore a v simulátore, nie sú tieto značky podporované.

Z úprav syntaktickej kontroly zdrojového textu v jazyku PNTalk museli byť vynechané výrazy, ktorých implementácia odzrkadľuje schopnosti simulátora a nie syntax jazyka. Kontrola zvyšku syntaxe je úplná, ale v dôsledkom spôsobu, akým je prekladač implementovaný, je niekoľko špecifických požiadaviek na zdrojový text:

- číselné literály v sebe nesmú mať medzery
- za bodkou medzi výrazmi musí byť biely znak

V prípade ďalšej práce, v rámci ktorej by boli upravené ďalšie časti programu, môže byť potrebné mierne zasiahnuť aj do častí, ktoré sú výsledkom tejto práce. Z toho dôvodu je kód riadne komentovaný, využívané sú javadoc komentáre[1] pre jednotlivé metódy a obsahuje mnohé poznámky a vysvetlivky na miestach, kde to pre čitateľa môže byť dôležité, a je členený tak, aby bolo jednoduché nájsť požadované miesto, ktoré by vyžadovalo úpravu.

[1] https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

Literatúra

[1] Fryč, T. *Generování kódu z modelů Petriho sítí*. Brno, CZ, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21446/.

[2] Janoušek, V. *Modelování objektů Petriho sítěmi*. Faculty of Information Technology BUT, 2008. 164 s. ISBN 978-80-214-3749-4. Dostupné z: https://www.fit.vut.cz/research/publication/8831.

[3] Meduna, A. *Elements of compiler design*. Boca Raton: Auerbach Publications, 2008. ISBN 1-4200-6323-5.

[4] Pecinovský, R. *Návrhové vzory*. Vyd. 1. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

Príloha A

Syntax jazyka PNtalk

Popis syntaxe jazyka PNTalk Backus-Naurovou formou prebratý z dizertačnej práce "Modelování objektů Petriho sítěmi"[2]. Východiskový symbol je "classes".

```
classes: [class]* "main" id [class]*
class: "class" classhead [objectnet] [methodnet|constructor|sync]*
classhead: id "is a" id
objectnet: "object" net
methodnet: "method" message net
constructor: "constructor" message net
sync: "sync" message [cond] [precond] [guard] [postcond]
message: id | binsel id | [keysel id]+
net: [place|transition]*
place: "place" id"(" [initmarking] ")" [init]
init: "init" "{" initaction "}"
initmarking: multiset
initaction: [temps] exprs
transition: "trans" id [cond] [precond] [guard] [action] [postcond]
cond: "cond" id"(" arcexpr ")" ["," id"(" arcexpr ")"]*
precond: "precond" id"(" arcexpr ")" ["," id"(" arcexpr ")"]*
postcond: "postcond" id"(" arcexpr ")" ["," id"(" arcexpr ")"]*
guard: "guard" "{" expr3 "}"
action: "action" "{" [temps] exprs "}"
arcexpr: multiset
multiset: [n "`"] c ["," [n "`"] c]*
n: [dig]+ | id
c: literal | id | list
list: "(" [c ["," c]* ["|" [id | list] ]] ")"
temps: "|" [id]* "|"
unit: id | literal | "(" expr ")"
unaryexpr: unit [ id ]+
primary: unit | unaryexpr
exprs: [expr "."]* [expr]
expr: [id ":="]* expr2
expr2: primary | msgexpr [ ";" cascade ]*
expr3: primary | msgexpr
```

```
msgexpr: unaryexpr | binexpr | keyexpr
cascade: id | binmsg | keymsg
binexpr: primary binmsg [ binmsg ]*
binmsg: binsel primary
binsel: selchar[selchar]
keyexpr: keyexpr2 keymsg
keyexpr2: binexpr | primary
keymsg: [keysel expr2]+
keysel: id":"
literal: number | string | charconst | symconst | arrayconst
arrayconst: "#" array
array: "(" [number | string | symbol | array | charconst]* ")"
number: [-][[dig]+ "r"] [hexDig]+ ["." [hexDig]+] ["e"["-"][dig]+].
string: "'"[char]*"'"
charconst: "\$"char
symconst: "#"symbol
symbol: id | binsel | keysel[keysel]* | string
id: letter[letter|dig]*
selchar: "+" | "-" | "*" | "/" | "~" | "|" | "," | "<" | ">" | selchar2
selchar2: "=" | "&" | "\" | "@" | "%" | "?" | "!"
hexDig: "0".."9" | "A".."F"
dig: "0".."9"
letter: "A".."Z" | "a".."z"
char: letter | dig | selchar | "[" | "]" | "{" | "}" | "(" " | ")" | char2
char2: " " | "\^" | ";" | "\$" | "#" | ":" | "." | "-" | "`"
```

Príloha B

Obsah pamäťového média

- **xcibak00.pdf** - táto práca vo formáte PDF
- **readme.txt** - informácie o obsahu média
- **xcibak00.zip** - archív s adresárovou štruktúrou obsahujúcou zdrojové kódy z pôvodnej práce, kde sú nahradené tie, ktoré boli v rámci tejto práce upravené
- **original.zip** - archív s adresárovou štruktúrou obsahujúcou pôvodné zdrojové kódy z predošlej práce bez akýchkoľvek zmien
- **latex.zip** - archív so zdrojovými kódmi tejto písomnej práce