FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Nástroj pro generování vývojových diagramů z podmnožiny jazyka C
Student:	Vojtěch Procházka
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem práce je vytvořit nástroj pro generování vývojových diagramů zejména pro vyjádření funkce mikroprogramového automatu.
Prostudujte existující řešení a vyberte vhodnou cílovou aplikaci pro zobrazení vývojového diagramu.
Důraz bude kladen na jednoduchou konverzi mezi jazykem C a vnitřní formou popisu v cílové aplikaci.
Navrhněte řešení pro automatické generování vývojových diagramů pomocí podmnožiny příkazů jazyka C.
Navržené řešení zrealizujte a otestujte.
Vytvořte několik ukázek možného popisu vývojového diagramu v jazyce C.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 3. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Nástroj pro generování vývojových diagramů z podmnožiny jazyka C

Vojtěch Procházka

Katedra číslicového návrhu
Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

10. května 2018

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Pavlu Kubalíkovi, Ph.D. za cenné rady a veškerou pomoc, kterou při tvorbě bakalářské práce poskytl. Velice si cením přístupu a ochoty při vedení.

Zároveň bych rád poděkoval mé rodině, a to hlavně mé sestře, za podporu a pomoc při tvorbě a korektuře.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2018 ……

Odkaz na tuto práci

Procházka, Vojtěch. *Nástroj pro generování vývojových diagramů z podmnožiny jazyka C*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá tvorbou nástroje pro generování vývojových diagramů, a to zejména pro vyjádření funkce mikroprogramového automatu. Účel nástroje je usnadnit a zrychlit tvorbu vývojových diagramů. Nástroj je psaný v jazyce C++ a generuje Python skript pro použití v programu Dia.

Klíčová slova vývojové diagramy, mikroprogramované automaty, porovnání, C++, Dia, tvorba diagramů

Abstract

In this thesis, we provide a tool for generating flowcharts from a subset of C language, especially for microprogrammed state machine. The purpose of the tool is to provide simpler and faster creation of flowcharts. The tool is written in the language of C++ and generates a Python script for use in the application Dia.

Keywords flowcharts, microprogrammed state machine, comparison, C++, Dia, creation of flowcharts

Obsah

Seznam obrázků

Seznam ukázek kódu

Úvod

Tvoření různých diagramů a grafů je dnes již běžná praxe při mnoha činnostech. Pro popsání množiny stavů se využije například graf přechodů stavového automatu, pro popsání vývoje mikrokontroléru lze využít vývojového diagramu. Diagramy lze samozřejmě kreslit ručně rovnou na papír, ale v dnešní době můžeme využít možností počítače a přenechat mu určitou část tohoto procesu. Mezi hlavní výhody tvoření diagramu rovnou na počítači patří jednodušší opravování chyb, kopírovatelnost a rozšířitelnost.

Tato bakalářská práce si bere jako hlavní cíl navrhnout a zrealizovat nástroj, který ulehčí tvorbu vývojových diagramů, a to zejména pro vyjádření funkce mikroprogramového automatu. Nástroj má podobu překladače, který má na vstupu diagram popsaný podmnožinou jazyka C. Nástroj potom tento diagram vnitřně uloží a vygeneruje Python skript do open-source programu Dia. Formát vstupního jazyka je navržený tak, aby popsání vývojového diagramu bylo co nejjednodušší, a navíc rozšíření takového diagramu bylo velmi snadné. Hlavní důvod, proč je vstupní formát podmnožinou jazyka C, je jednoduché naučení se tohoto formátu pro člověka, který zná C.

Jelikož je na ČVUT FIT jazyk C první vyučovaný programovací jazyk, dá se tedy předpokládat, že pro studenta ČVUT FIT bude mít tento nástroj praktické využití v různých předmětech během studia. Toto je hlavní důvod, proč jsem si vybral toto téma, jelikož při průchodu studiem jsem si několikrát sám pro sebe poznamenal, že by se nějaký takový nástroj hodil, aby mi usnadnil studium a ušetřil čas.

Dalším cílem této práce je prozkoumat existující řešení tvorby vývojových diagramů na základě určitých kritérií. Tato kritéria jsou například jednoduchost vytvoření diagramu dle velikosti diagramu (malý, střední, velký), typ licence, složitost rozšíření diagramu (o jeden nebo více stavů) a subjektivní obtížnost vytvoření diagramu pro naprostého nováčka s daným nástrojem. Nutno podotknout, že většina porovnávaných nástrojů je založena graficky a žádný z nich nepodporuje jednoduše popsatelný vstup v jakémkoliv jazyce.

Struktura práce

Práce se zabývá nejdříve analýzou současných řešení a jejich porovnáním a pokračuje popisem návrhu nástroje, kde jsou rozebrány hlavní myšlenky při tvorbě takového nástroje. Poté je popsaná realizace samotného nástroje. Součástí práce je uživatelská příručka, ve které je mimo jiné i několik ukázek, v nichž je zobrazena funkce a užitečnost nástroje. Zároveň tyto ukázky mohou sloužit k rychlému pochopení vstupního formátu.

Kapitola 1

Cíle práce

Bakalářská práce si pokládá jako hlavní cíle:

1. Prozkoumat existující řešení kreslení diagramů.
2. Na základě prozkoumání zvolit vhodnou aplikaci, ve které bude zobrazen vygenerovaný diagram.
3. Navrhnout vhodný formát pro popis diagramu.
    - Tento formát by měl být podmnožinou jazyka C.
4. Napsat nástroj, který provede konverzi mezi daným formátem a vnitřní reprezentací diagramu jako graf.
5. Tento nástroj také provede konverzi mezi vnitřní reprezentací diagramu a vstupním formátem cílové aplikace, ve které bude diagram zobrazen.
6. K nástroji musí být sepsána uživatelská příručka, kde bude dostatečně vysvětleno, jak s ním zacházet.
7. Nástroj by měl být uživatelsky přívětivý.

V souhrnu jde tedy hlavně o to, aby nástroj nebyl napsán jen pro účely splnění bakalářské práce, ale aby byl zároveň použitelný při běžné tvorbě diagramů, a to obzvláště se zaměřením na studenty ČVUT FIT.

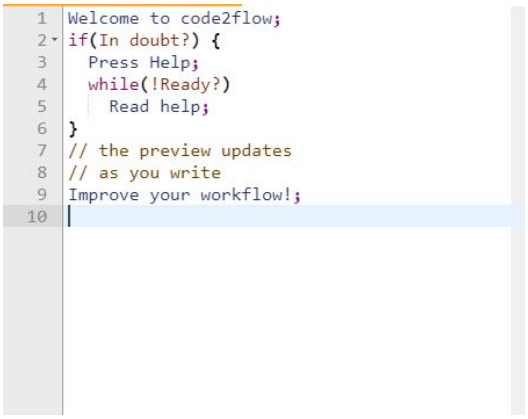
Kapitola 2

Analýza existujících řešení

V kapitole jsou popsány a srovnány existující nástroje pro kreslení diagramů. Cílem kapitoly je nejenom seznámit se s dosavadními řešeními, ale i pomoci při výběru cílové aplikace pro účely bakalářské práce. U konce kapitoly je umístěna tabulka, ve které jsou shrnuty poznatky a pozorování.

2.1 Code2flow.com

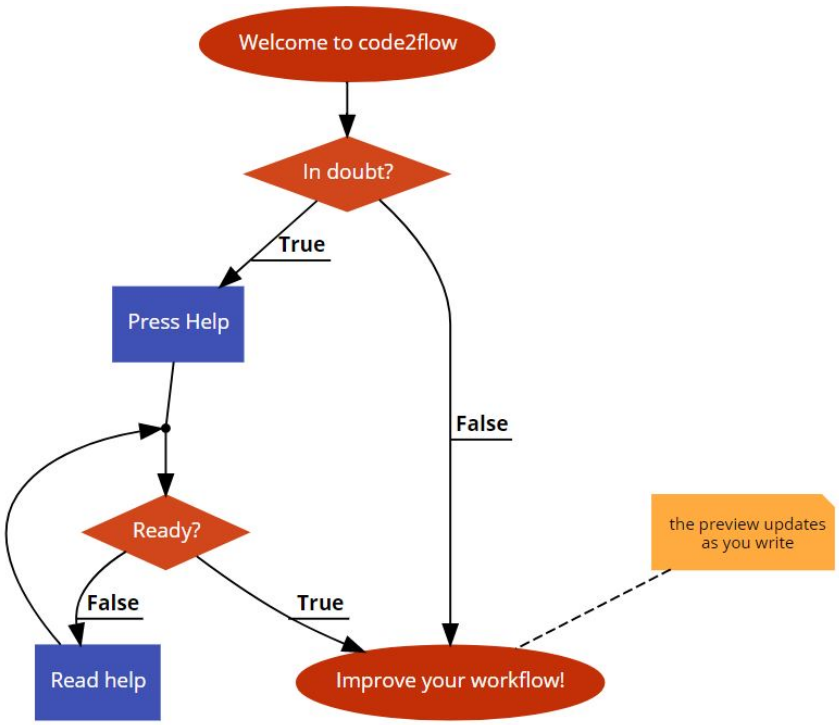
Code2flow[1] je nástroj, jenž je z porovnávaných aplikací kapitoly nejvíce podobný samotnému nástroji, který jsem v rámci bakalářské práce vytvořil. Uživatel píše pseudokód a po jeho přeložení stránka zobrazí příslušný vývojový diagram. Z limitovaného počtu možností lze graficky upravit výsledný diagram, avšak posouvat jednotlivými objekty dle libosti není možné. Pseudokód jako takový je jednoduchý na pochopení, ale kvůli této jednoduchosti může být složitější popsat diagram správně. Čím větší diagram, tím více lze poznat užitečnost aplikace.



Obrázek 2.1: Code2flow, okno s pseudokódem

Dá se využívat verze zdarma, jež je dostatečná na to, aby si člověk udělal představu o užitečnosti. Existují i placené verze s měsíčním poplatkem. Placené verze jsou rozšířené o možnosti ukládání diagramů, podporu a odstranění code2flow watermarky. Nejdražší verze stojí skoro $3600 ročně.

Navíc v den testování tohoto nástroje nefungovalo správně stažení diagramu jako PNG. Pozadí se udělalo černé a spolu s černými šipkami to udělalo diagram absolutně nečitelný a k ničemu. Přiložený obrázek diagramu je vytvořený print screenem a následným oříznutím.



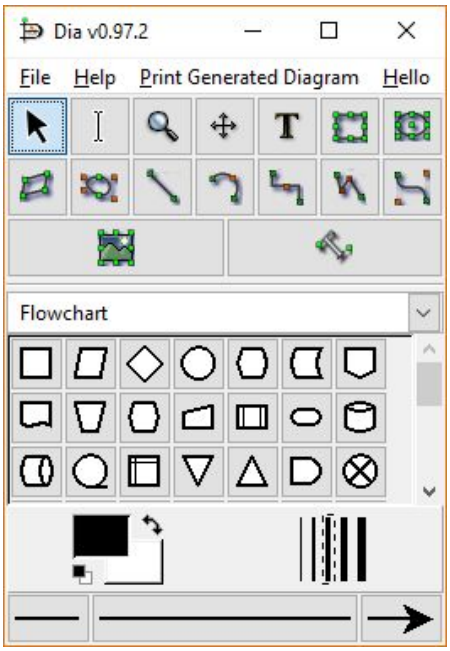
Obrázek 2.2: Code2flow, okno s diagramem

2.2 Dia a PyDia

V této sekci bude rozebráno tvoření diagramů v programu Dia. V Dia lze tvořit diagramy dvěma způsoby; buď nakreslit rovnou pomocí grafického nástroje, anebo diagram popsat v Pythonu a spustit jako skript, čímž se vykreslí. Rozšíření Dia pluginem o Python se nazývá PyDia.

2.2.1 Dia

Dia v dnešní době nikoho svou grafickou podobou neohromí, ale když člověk uváží, že první verze byla vydána ještě pro Windows 98, uvědomí si, že na tehdejší dobu to musel být velmi dobrý nástroj. UI je velmi jednoduché a zároveň intuitivní. Snad jediná neintuitivní věc, s kterou jsem se setkal, byla nutnost přepnout do režimu kurzoru Text pro editaci textu uvnitř objektu.

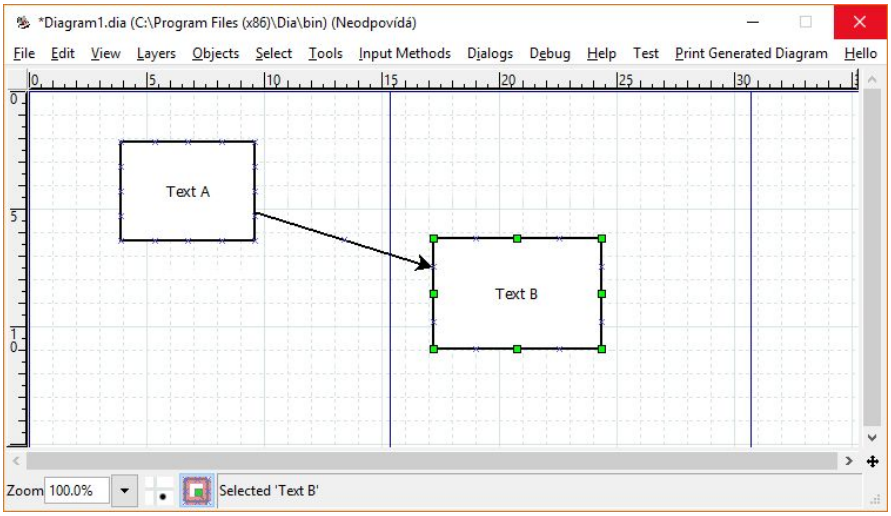


Obrázek 2.3: Dia, okno s nástroji

Velkou výhodou Dia je možnost jednoduchého importování vlastních objektů. V kompetencích Dia je mimo jiné například i UML, objekty vhodné pro výuku počítačových sítí a další. Samotná tvorba diagramů probíhá tak, jak by člověk očekával. Akorát celý proces občas působí nešikovně, až zastarale — obzvláště, když má člověk možnost to porovnat například s Draw.io, viz pozdější sekce této kapitoly.

I přes to, že je možnost uložit si diagram v mnoha formátech, pouze pár z nich funguje správně tak, aby diagram po exportování a následném importování si zachoval všechny funkcionality. Ani jeden z těchto formátů není po otevření v textovém editoru čitelný člověkem, tudíž se nedá samotné Dia využít pro psaní diagramu pseudokódem.

Aplikace je licencovaná pod GPL verze 2 [2], tedy — pro naše potřeby — zdrojové kódy jsou přístupné a veškeré produkty vycházející z Dia musí mít přístupné zdrojové kódy. Diagramy vytvořené v Dia jsou plně majetkem toho, kdo je vytvořil.



Obrázek 2.4: Dia, okno s diagramem

2.2.2 PyDia

Psaní vývojových diagramů pomocí Pythonu pro program Dia je velmi neintuitivní a komplikované. Bez dokumentace k PyDia[3] je to prakticky nemožné, a dokonce tato dokumentace není ani součástí webové stránky k Dia[4]. Pro představu složitosti poslouží následující ukázka.

```
obj0, h1obj0, h2obj0 = (
        dia.get_object_type("Flowchart - Box").create(10, 8) )
layer.add_object(obj0)
obj0.properties["text"] = "OED OEAB"
```

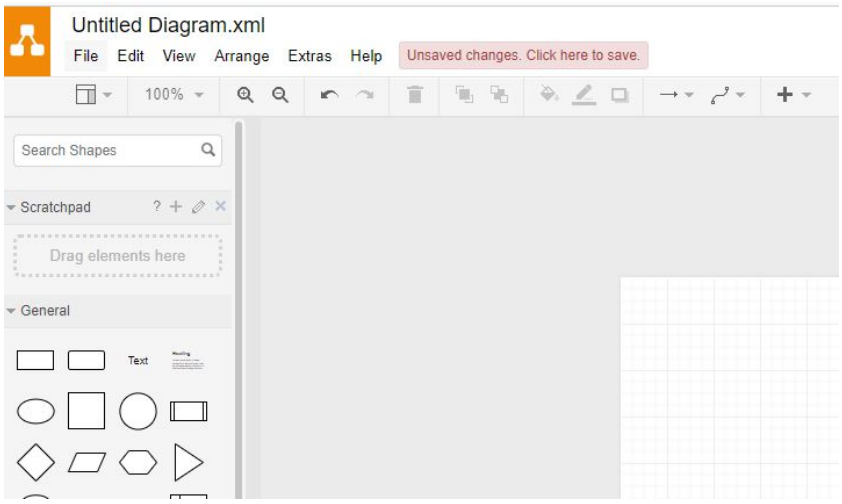
Ukázka kódu 2.1: PyDia, vytvoření objektu a vložení textu

Tento kód pouze vytvoří na pozici [10,8] obdélník a vloží do něj text „OED OEAB". Z tohoto je ihned patrné, že psát pomocí PyDia diagram o jakékoliv velikosti je spíše za trest a každý raději vezme do ruky papír a tužku a nakreslí diagram ručně. Ovšem i přes všechny tyto negativy je tu jedno velké pozitivum. Diagramy napsané v PyDia lze nadále editovat pomocí samotné aplikace Dia. Pokud bude vygenerovaný diagram nedostačující, lze nadále využít všech nástrojů Dia pro jeho upravení a doladění.

Ačkoliv je pro člověka nemyslitelné psát PyDia kód, tento proces je možné programově automatizovat. Program, který má vnitřně uložený diagram, bez větších problémů vytvoří PyDia kód. Proto je PyDia, spolu s Dia, zvolenou cílovou aplikací pro zobrazení vývojového diagramu zadaného v podmnožině jazyka C.

2.3 Draw.io

Tento nástroj hned na první pohled zaujme svou grafickou podobností ke Google Docs, a to především Google Drawings, což je veliká výhoda — již u prvního použití si člověk připadá ve známém prostředí, a proto se může pustit do práce ihned, aniž by se musel dlouze seznamovat s aplikací.

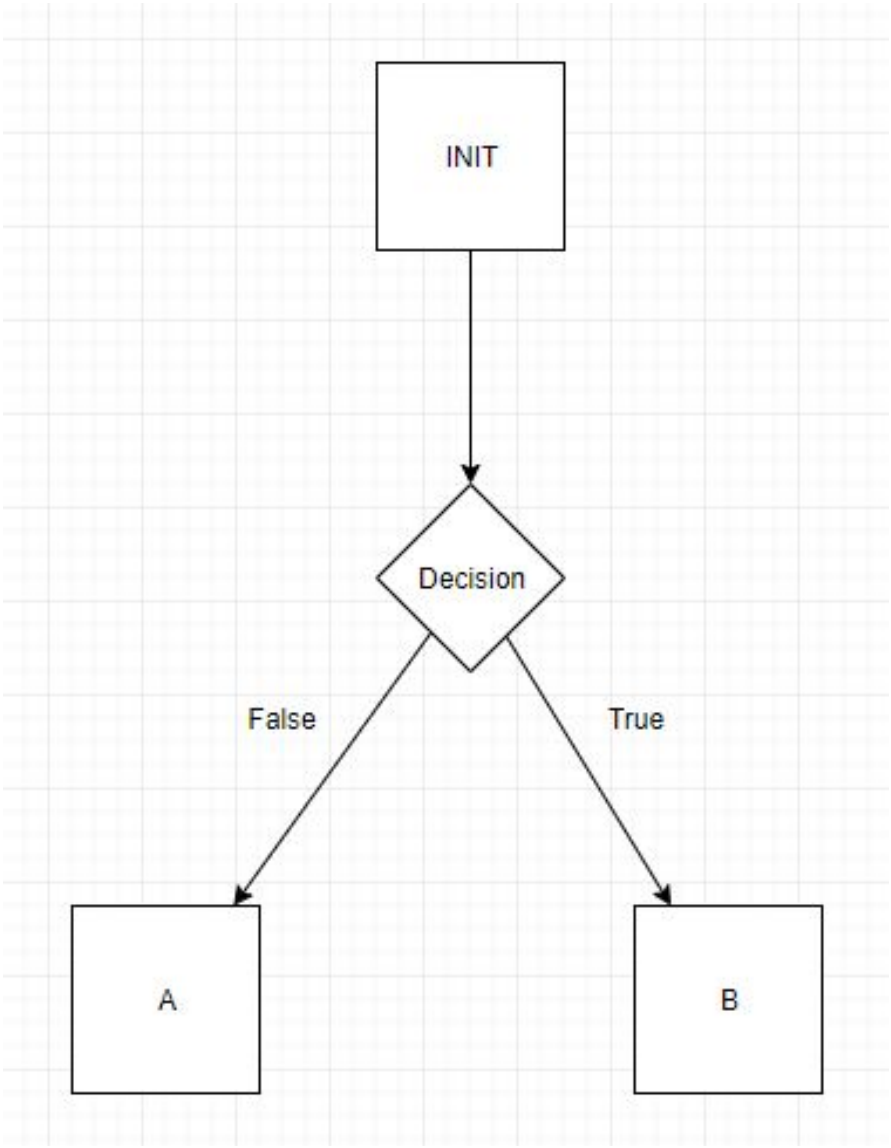


Obrázek 2.5: Draw.io, výřez UI

Hned po vytvoření malého diagramu jsem si zalíbil tuto aplikaci — nástroje jsou velmi intuitivní a velké množství možností po stranách indikuje, že pokud by člověk chtěl vytvořit i složitější a větší diagram, bude mít k dispozici jakýkoliv nástroj, který by ho mohl napadnout. Vytvořené diagramy lze jak stáhnout, tak uložit do Google Drive pro opětovné otevření a pokračování v práci.

Ačkoliv ukládaný formát je XML, ukládá se pouze ID diagramu, tedy pro člověka nesrozumitelný text, tudíž se nedá použít pro psaní v pseudokódu.

Aplikace je úplně zdarma, jak pro osobní, tak i korporátní použití, a dokonce je i open-source.



Obrázek 2.6: Draw.io, okno s diagramem

2.4 Microsoft Visio

Microsoft Visio je placený, méně známý produkt od známé společnosti Microsoft. Se zaměřením na kreslení diagramů a tvoření vektorové grafiky je z produktů Microsoft jednoznačnou volbou pro tvorbu diagramů. Bohužel, jelikož je Visio placené (s cenami začínajícími u 3000 Kč), nemohl jsem si aplikaci vyzkoušet, ale i přes to považuji za vhodné ji zmínit.

2.5 Tabulka porovnání existujících řešení

Hodnoceno počtem bodů; 10 = nejlepší, 0 = nejhorší. Visio je z výše zmíněných důvodů vynecháno.

Kategorie	**Code2flow**	**Dia**	**PyDia**	**Draw.io**
Jednoduchost vytvoření malého diagramu	4	8	2	9
Jednoduchost vytvoření středně velkého diagramu	6	7	1	8
Jednoduchost vytvoření velkého diagramu	8	4	1	5
Typ licence	Free Trial	GPLv2	GPLv2	OSS
Složitost rozšíření diagramu o jeden stav	10	7	3	8
Složitost rozšíření diagramu o více stavů	8	4	3	6
Obtížnost vytvořit diagram pro nováčka	6	7	0	9
Součet bodů	42	37	10	45

Tabulka 2.1: Tabulka porovnání existujících řešení

Z tabulky vychází Draw.io jako nejvhodnější nástroj pro kreslení diagramů. I přes nízké hodnocení je PyDia ideální cílovou aplikací pro potřeby bakalářské práce. Důvody byly popsány v sekci PyDia.

Kapitola 3

Analýza a návrh programového řešení

3.1 Zvolené řešení

Po uvážení veškerých požadavků a cílů bakalářské práce jsem zvolil jako řešení nástroj, jehož specifikace jsou následující:

1. Vstupní formát je vytvořený z podmnožiny jazyka C.
   - C je první programovací jazyk vyučovaný na ČVUT FIT.
   - Vzhledem k tomu, že nástroj je určen hlavně pro studenty, má nástroj větší šanci na uchycení právě díky podobnosti k C.
2. Zvolená cílová aplikace je Dia s pluginem PyDia.
   - Ačkoliv je PyDia nepoužitelný při tvorbě diagramů pro člověka, vygenerovat příslušný Python kód v programu není problém.
   - Po nahrání diagramu do Dia lze diagram nadále editovat ve WYSIWYGu, jako kdyby byl vytvořený v Dia.
3. Aplikace je psaná v C++.
   - Čistě z důvodu, že jsem s tímto jazykem nejvíce obeznámen.
4. Aplikace obsahuje lexikální analýzu vstupního formátu.
   - Díky tomu je zaručená lehká rozšiřitelnost vstupního formátu o další funkcionality.
5. Vstupní i výstupní kódy jsou v souborech.
6. Během překladu je v konzoli vypisován průběh, případně jsou vypsána smysluplná chybová hlášení.

3.2 Základní požadavky nástroje

Aby nástroj získal skutečné uplatnění, je potřeba, aby tento nástroj splňoval určité požadavky. Splněním těchto požadavků se sice nezíská jistota, že nástroj bude využíván, ale pokud by nebyly splněny, nástroj nemá dlouhodobě šanci na uchycení.

Požadavky:

1. bezchybnost,
2. dostupnost nástroje,
3. jednoduché k naučení,
4. lehké vložení nového stavu doprostřed diagramu,
5. možnost pokračovat v práci na již dříve vytvořeném diagramu,
6. srovnatelná nebo rychlejší tvorba diagramů než existující nástroje,
7. vytvořená uživatelská příručka usnadňující naučení se s nástrojem.

Požadavky není složité dodržet, a proto jsou při návrhu nástroje vzaty v potaz bez výjimky. Ačkoliv bod ohledně rychlosti tvorby diagramů se složitě měří, patří mezi nejdůležitější z požadavků. Žádný student, jenž potřebuje vytvořit diagram pro potřeby jakéhokoliv předmětu, nebude ochotný věnovat čas nástroji, který není efektivní.

3.3 Vývojové prostředí

Zvolené vývojové prostředí je Microsoft Visual Studio Enterprise 2017. Studenti ČVUT mají přístup k plné verzi zdarma. Visual Studio je dobré pro vývoj různých jazyků, mezi ty důležitější patří C++, C# a ASP.NET framework. Ideální je při vývoji aplikací pro Windows, např. i díky debuggeru, který umožňuje pustit program krok po kroku v uživatelsky přívětivém prostředí, čímž pomáhá k rychlému hledání chyb v programu.

Visual Studio lze rozšířit o různé pluginy, a tím si ještě více zpříjemnit vývoj. Využívám plugin VsVim, který rozšiřuje Visual Studio o zkratky textového editoru Vim, jenž je pro Linux. Mezi významné klávesové zkratky patří 'dd' — má stejný efekt, jako kdyby uživatel označil celý řádek a stisknul klávesovou zkratku Ctrl+x. Používáním pluginu si šetřím čas při psaní kódu a je to pro mě pohodlnější.

3.4 Návrh formátu vstupního kódu

Při navrhování formátu je potřeba nejdříve rozmyslet, jaké vlastnosti musí daný kód mít. V tomto případě jsou identifikovány hlavní vlastnosti:

1. Rychlé k pochopení.
2. Podmnožinou jazyka C.
3. Schopný popsat diagram, jeho stavy a přechody.
4. Možnost přidat k stavům a přechodům textový popis.

Identifikováním těchto požadovaných vlastností se zpřesňuje podoba výsledného formátu. Dobré je vzít v úvahu, jak jsou popsány konečné automaty, když se je snaží programátor popsat jazykem C. Nejpoužívanější způsoby jsou buď příkazem `switch`, anebo tabulkou. Formát by šlo navrhnout tak, aby připomínal popis automatu tabulkou, ovšem celkový dojem by nebyl úplně „céčkový", a proto zbývá popis pomocí příkazu `switch`.

Při popisu konečného automatu příkazem `switch` hlavní část programu vypadá zhruba takto:

```
#define q0 0
#define q1 1
#define q2 2
...

#define a0 0
#define a1 1
...

int main () {
        int stav = 0;
        while(1){
                int ch = readInput();
                switch(stav){
                case q0:
                        switch(ch){
                        case a0:
                                stav = q1;
                                break;
                        case a1:
                                ...
                                break;
                        }
                break;
                case q1:
                        switch(ch){
                        case a0:
                                stav = q3;
                                break;
                        case a1:
                                ...
                                break;
                        }
                break;
                ...
                }
        }
        ...
}
```

Ukázka kódu 3.1: Popis konečného automatu příkazem switch v jazyce C

Kde `q0, q1, …, qn` reprezentuje stavy a `a0, a1, …, an` reprezentuje jednotlivé symboly vstupní abecedy. Vstupní abeceda může reprezentovat různé akce, například a0 může představovat vhození pětikoruny do automatu a a1 zmáčknutí tlačítka storno. Takovýto formát je jednoduchý pro popsání; pro každý vstupní symbol (ch) a stav se na základě typu vstupního symbolu (ch) udělá příslušná akce. Kód je jednoznačně členěný — čtení následujícího symbolu, vnější příkaz `switch` dle stavu, vnitřní příkaz `switch` dle symbolů

— a dlouhodobě udržitelný, jelikož lze často jednoduše dohledat, v kterém stavu při jaké akci nastala chyba.

Když budeme vycházet z tohoto formátu, ihned můžeme položit několik základních kamenů vstupního formátu:

1. Hlavní popis přechodů bude ve (vnějším) příkazu `switch` dle stavu.
2. Popis přechodů bude podřazený (vnějšímu) příkazu `switch` dle stavu. Příkaz `switch` se dá také napsat jako `if ()`, `else if ()` a `else`. Z podstaty tvoření diagramů toho využijeme a místo vnitřního příkazu `switch (ch)` bude použito klíčové slovo `if`.
3. Definování stavů a všech pomocných typů proběhne v části kódu nad hlavní funkcí.
4. Před příkazem `switch` dle stavu proběhne přiřazení počátečního stavu do stavové proměnné.

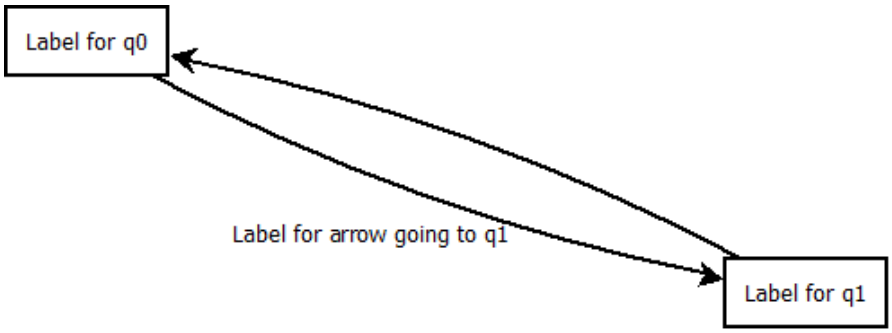
Tímto je navržena struktura vstupního formátu. Dále je potřeba rozhodnout náležitosti specifické pro tvorbu diagramů:

1. Určení tvaru a popisu stavů.
   - Tvary, jež budou v kódu použity, je potřeba nejdříve deklarovat typu `TYPE`.
   - Těmto tvarům je potřeba nadále přiřadit stavy.
   - Každý stav má svůj název, který je použit pouze v kódu, a také má svůj popis — text, jenž se zobrazí uvnitř stavu.
2. Stavová proměnná.
   - Jméno proměnné jde určit deklarací typu `FSM`.
   - Před hlavním příkazem `switch` je potřeba přiřadit počáteční stav.
3. Přechody mezi stavy.
   - Pokud je v hlavním příkazu `switch` přiřazen do stavové proměnné jiný stav, vygeneruje se tím šipka (přechod) z aktuálního stavu do přiřazeného stavu. Popis k šipce lze přidat pomocí `if (popis šipky)`.
4. Možnost určit v kódu pozici vygenerovaného stavu.
   - Zavedením proměnných pro řádek a sloupec.
   - Jména proměnných si může uživatel určit sám deklarováním proměnné typu `ROWTYPE` a `COLUMNTYPE`.

5. Definovaní typu šipky.

   - Dia umožňuje různé typy šipek, sice je dle mého názoru `Arc` nejlepší typ šipky pro kreslení grafu, dát tuto možnost uživateli neuškodí.
   - Jméno proměnné, do níž lze přiřadit typ šipky, může uživatel určit deklarováním proměnné typu `ARROWTYPE`.

Na následující ukázce bude tento formát zobrazen a řádně okomentován, aby bylo jasné, co se provádí. Nejdříve ovšem výsledný graf:



Obrázek 3.1: Velmi malý diagram

Kód generující tento diagram:

```
//Deklarace typu --- bude alespon jeden stav s typem Box
TYPE Box;
/*
Pro typ Box budou dva stavy, q0 a q1 a tyto stavy budou
mit definovane popisy
*/
Box = { {q0,"Label for q0"},{q1,"Label for q1"} };

//Deklarace promenne pro typ sipek
ARROWTYPE arrow;
//Deklarace promenne uchovavajici aktualni stav
FSM state;
//Deklarace promenne pro urceni sloupecku
COLUMNTYPE col;
//Deklarace promenne pro urceni radku
ROWTYPE row;

int main() { //int main () je pro zachovani pocitu jazyka C
    //Nastaveni sipek jako Arc
    arrow = Arc;
    //Nastaveni q0 jako pocatecniho stavu
    state = q0;
```

```
    /*
    promenne pro radek a sloupecek maji defaultni hodnoty 0
    a nelze je v teto casti prenastavit (lze u prvniho stavu)
    */

    //Hlavni prikaz switch dle stavu
    switch (state) {
    /*
    Kdyz je v C napsany case ..., rozumi se tim vetsinou
    "Kdyz stav je ...". V pripade vstupniho formatu je lepsi
    toto cist jako "Ze stavu ..."
    */
    case q0:
        /*
        Nasledujici radek lze precist takto:
            Pri prechodu (ze stavu q0) do stavu q1
            pridej k sipce popis (definovany v zavorkach)
        */
        if (Label for arrow going to q1) state = q1;
        break; //break je pro zachovani pocitu jazyka C
    case q1:
        /*
        Zde neni if() cast, a proto sipka, ktera se vytvori
        ze stavu q1 do stavu q0, nebude mit zadny popis.
        */
        state = q0;
        /*
        Nastavenim sloupecku na nejakou hodnotu je nastavena
        X souradnice. Jelikoz radky jsou na autoinkrementu
        (neni-li dano jinak), nemusime souradnici radku
        nastavovat.
        */
        col = 2;
        break;
    }
    return 0;//return 0 je pro zachovani pocitu jazyka C
}
```

Ukázka kódu 3.2: Popis velmi malého diagramu vstupním formátem

3.4.1 Porušení podmnožiny jazyka C

Podmínka v příkazu `if` — `if (Label for arrow going to q1) state = q1;` — porušuje zadání práce "Navrhněte řešení pro automatické generování vývojových diagramů pomocí podmnožiny příkazů jazyka C". Splnit tuto podmínku není nijak obtížné, podmínka v příkazu `if` by se změnila na `if (arrow == "Label for arrow going to q1") state = q1;` . Ovšem pro podstatu práce — vytvořit použitelný nástroj — je lepší, když podmínka zůstane nesplněna a navržený formát zůstane ve tvaru, který je příjemnější pro uživatele.

3.5 Popis výstupního kódu v Pythonu

Jak již bylo zmiňováno v subsekci PyDia, samotné tvoření diagramů pomocí PyDia je až zbytečně složité. Avšak pro potřeby práce je i přesto potřeba znát, jak se diagramy pomocí PyDia tvoří. Základním krokem je nalézt, alespoň nějakou dokumentaci[3]. Citovaná dokumentace sice není perfektní, ale i tak pomůže k pochopení.

Pro potřeby práce je potřeba umět vytvořit pomocí PyDia:

1. objekt s definovaným tvarem,
2. šipky,
3. napojení šipek mezi objekty,
4. vložení textu do objektu,
5. textové pole, s nímž lze volně pohybovat,
6. nastavit barvu okrajů objektu na bílou (pro vytvoření neviditelného objektu, z kterého / na který lze napojovat šipky).

Všechny body tohoto seznamu budou demonstrovány v následujících ukázkách.

Vytvoření objektu a vložení textu do objektu: Vytvoření textového pole se provádí úplně stejně, akorát místo `Flowchart - Box` je potřeba napsat `Standard - Text`.

```
#vytvori novy objekt typu "Flowchart - Box" na pozici 5,3
#a vrati tri polozky: objekt, handle1 a handle2
obj0, h1obj0, h2obj0 = (
    dia.get_object_type("Flowchart - Box").create(5, 3) )
#vlozi objekt do vrstvy, ve ktere se zrovna pracuje
layer.add_object(obj0)
#vlozi do objektu text
obj0.properties["text"] = "Label for q0"
```

Ukázka kódu 3.3: Vytvoření objektu a vložení textu

Vytvoření a napojení šipek: Samotná tvorba šipek je stejná jako tvorba jakéhokoliv jiného objektu, ovšem napojování šipek je jiné. Každý objekt v Dia má několik napojovacích bodů. Pokud se šipka napojí na všechny body objektu, dosáhne se stejného efektu, jako kdyby se v Dia natáhla šipka doprostřed objektu.

```
#vytvoreni sipky a vlozeni do vrstvy
obj1, h1obj1, h2obj1 = (
    dia.get_object_type("Standard - Arc").create(0,0) )
layer.add_object(obj1)

#napojeni vsech napojovacich bodu objektu 0
#na pocatek sipky
for i in obj0.connections:
    h1obj1.connect(i)
#napojeni vsech napojovacich bodu objektu 0
#na konec sipky
for i in obj2.connections:
    h2obj1.connect(i)
#nastaveni vlastnosti pocatecni/koncove sipky
#prvni hodnota je tvar sipky u napojeni
#druha hodnota je sirka, treti delka
obj1.properties["start_arrow"] = (0,.5,.5)
obj1.properties["end_arrow"] = (22,.5,.5)
```

Ukázka kódu 3.4: Vytvoření a napojení šipek

Nastavení barvy okrajů objektu: Nastavení barvy se provádí podobně jako vkládání textu. Položka `line_colour` však nepřijímá text, nýbrž trojici desetinných čísel, reprezentující hodnoty RGB. Akorát místo běžné škály 0-255 je tato škála mezi 0-1. Pomocí desetinných míst lze dosáhnout jiné barvy.

```
obj0.properties["line_colour"]=(1.,1.,1.)
```

Ukázka kódu 3.5: Nastavení barvy okrajů objektu:

Velmi malý diagram v Pythonu: Na následující ukázce jsou vzaty znalosti z předchozích ukázek, dány dohromady a sestaveny tak, aby to byl plně funkční příklad. Části kódu, které nebyly v předchozích ukázkách představeny, jsou řádně okomentovány. Výsledný diagram si lze prohlédnout v předchozí sekci Návrh formátu vstupního kódu.

```
# -*- coding: utf-8 -*-
import dia #importnuti knihovny dia

#funkce, ktera je volana pri stisku tlacitka
def mydia_callback(data, flags) :
    #vytvoreni noveho okna s~diagramy
    diagram = dia.new("GeneratedGraph.dia")
    #zobrazeni okna s~diagramy
    diagram.display()
    #nastaveni dat na data diagramu
    data = diagram.data
    #nastaveni vrstvy, do niz jsou vkladany objekty
    layer = data.active_layer

    obj0, h1obj0, h2obj0 = (
        dia.get_object_type("Flowchart - Box").create(5, 3) )
    layer.add_object(obj0)
    obj0.properties["text"] = "Label for q0"

    obj2, h1obj2, h2obj2 = (
        dia.get_object_type("Flowchart - Box").create(25, 10) )
    layer.add_object(obj2)
    obj2.properties["text"] = "Label for q1"

    obj1, h1obj1, h2obj1 = (
        dia.get_object_type("Standard - Arc").create(0,0) )
    layer.add_object(obj1)
    for i in obj0.connections:
        h1obj1.connect(i)
    for i in obj2.connections:
        h2obj1.connect(i)
    obj1.properties["start_arrow"] = (0,.5,.5)
    obj1.properties["end_arrow"] = (22,.5,.5)
    obj4, h1obj4, h2obj4 = (
         dia.get_object_type("Standard - Text").create(12, 7) )
    layer.add_object(obj4)
    obj4.properties["text"] = "Label for arrow going to q1"

    obj3, h1obj3, h2obj3 = (
         dia.get_object_type("Standard - Arc").create(0,0) )
    layer.add_object(obj3)
    for i in obj2.connections:
        h1obj3.connect(i)
    for i in obj0.connections:
        h2obj3.connect(i)
    obj3.properties["start_arrow"] = (0,.5,.5)
    obj3.properties["end_arrow"] = (22,.5,.5)

    #tento cyklus projde vsechny objekty v~dane vrstve
```

```
    #a obnovi je --- bez tohoto by se sipky zobrazovaly
    #nespravne
    for i in layer.objects:
        diagram.update_connections(i)

#prida tlaciko do listy s~nastroji pro spusteni funkce
dia.register_action("Print Generated Diagram",
    "_Print Generated Diagram",
    "/ToolboxMenu/", mydia_callback)
#prida tlaciko do okna s~diagramy pro spusteni funkce
dia.register_action("Print Generated Diagram",
    "_Print Generated Diagram",
    "/DisplayMenu/", mydia_callback)
```

Ukázka kódu 3.6: Popis velmi malého diagramu v Pythonu

3.6 Možnosti zlepšení do budoucna

Ačkoliv nástroj funguje a plní svůj účel, lze ho vylepšit. Hlavní oblasti pro zlepšení jsou:

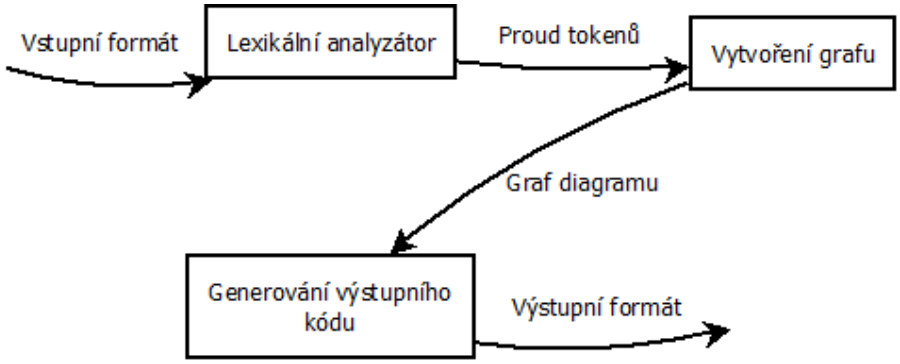
1. GUI pro zadávání vstupu.
2. Kreslení diagramu přímo v nástroji, bez využití softwaru třetí strany.
3. Podpora všech tvarů a šipek, které Dia nabízí.

Kapitola 4

Realizace

4.1 Architektura programu

Nejdůležitější části architektury jsou znázorněny na následujícím obrázku.



Obrázek 4.1: Architektura nástroje

Hlavní funkce programu je; nejdříve se otevře soubor obsahující kód ve vstupním formátu, potom je tento kód přeložen lexikálním analyzátorem na proud tokenů, z kterých se následně vytvoří graf. Z tohoto grafu je posléze vygenerován Python kód ve formátu vstupu pro PyDia. Jednotlivé části budou blíže rozebrány v následujících sekcích.

4.2 Lexikální analýza

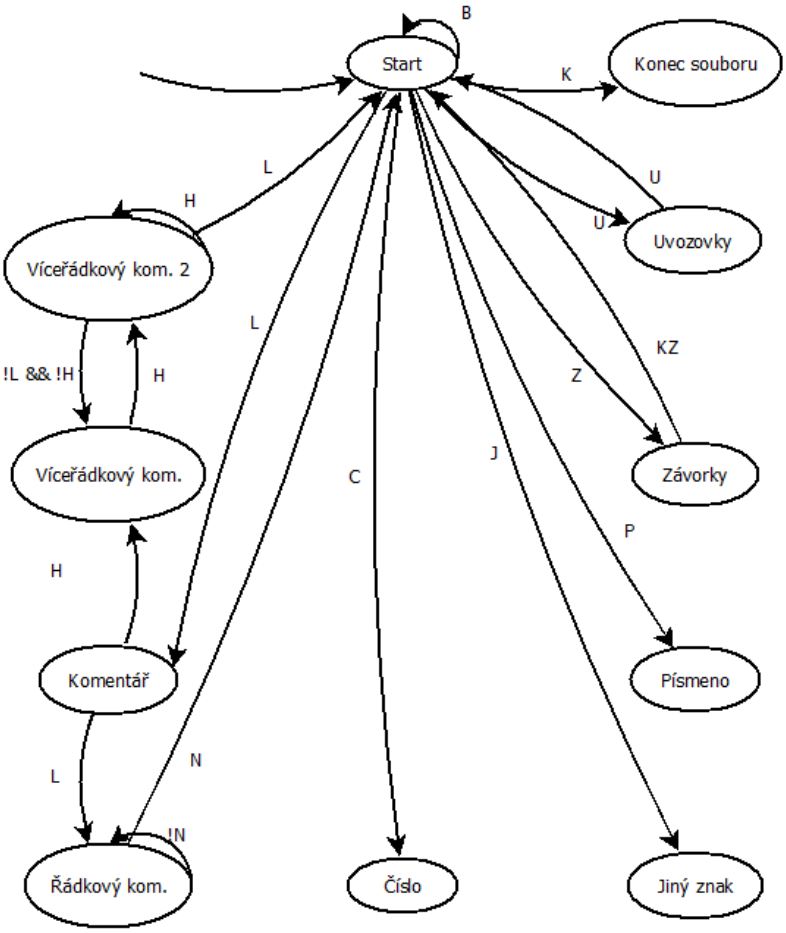
Lexikální analyzátor (dále jen Lexan) je v tomto nástroji ručně napsaný bez použití jakýchkoliv nástrojů pro automatické generování. V programu je implementovaný ve třídě `Lexan`. Lexan má za účel vzít sekvenci znaků na vstupu a jeho výstupem je sekvence tokenů. Při překladu vstupního kódu jsou ignoro-

vány bílé znaky, které nejsou součástí řetězce. Jako bílé znaky jsou považovány takové znaky, jež mají ASCII kód 0-32. Mezi ně patří mezera, tabulátor i nový řádek. Také jsou ignorovány veškeré komentáře. Tokeny, které jsou výstupem, jsou implementovány jako třída `LexToken`. Tokeny obsahují symbol tokenu a případně atribut — buď číslo, anebo řetězec související s daným tokenem. Na následující tabulce lze vidět reprezentaci lexikálních elementů. Formátování tabulky je inspirováno předmětem BI-PJP[5].

Syntaxe	**Lexikální symbol**	**Atribut**
identifikátor	t_ident	řetěz znaků
číslo	t_numb	celé číslo
=	t_equals	
{	t_lcurly	
}	t_rcurly	
;	t_semicolon	
:	t_colon	
,	t_comma	
řetězec uvnitř uvozovek	t_string	řetěz znaků
řetězec uvnitř závorek	t_parstring	řetěz znaků
TYPE	kw_type	
FSM	kw_fsm	
ARROWTYPE	kw_arrowtype	
SWITCH	kw_switch	
CASE	kw_case	
BREAK	kw_break	
IF	kw_if	
RETURN	kw_return	
INT	kw_int	
MAIN	kw_main	
ROWTYPE	kw_rowtype	
COLUMNTYPE	kw_columntype	

Tabulka 4.1: Tabulka reprezentace lexikálních elementů

Na přiloženém obrázku je znázorněna práce Lexanu. Tento automat není úplný, jinak by byl nepřehledný. I když není úplný, je z něj zřejmé, jak Lexan pracuje.



Obrázek 4.2: Konečný automat lexikálního analyzátoru

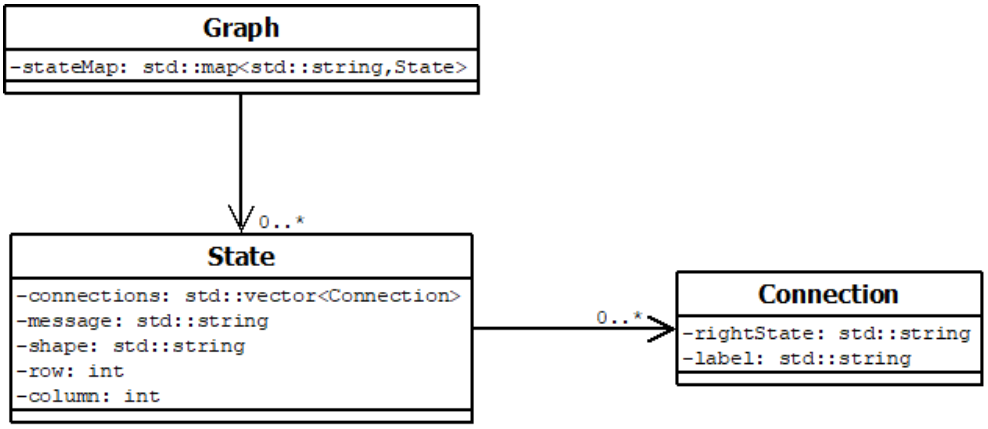
Vstupní znak	Zkratka	Vysvětlivky
'a'..'z', '_', 'A'..'Z'	P	písmeno nebo podtržítko
'/'	L	lomítko
'"'	U	uvozovka
'('	Z	levá závorka
')'	KZ	pravá závorka
'0'..'9'	C	číslo
'\0'..' '	B	bílý znak
'=', '{', '}', ':', ';', ','	J	jiný znak
EOF	K	konec souboru
'\n'	N	nový řádek
'*'	H	hvězdička

Tabulka 4.2: Legenda k automatu

4.3 Vnitřní reprezentace grafu

4.3.1 Architektura grafu

Graf je uložený v jednoduché struktuře, jak je znázorněno na obrázku:



Obrázek 4.3: Zjednodušené zobrazení architektury grafu

Obrázek není kompletní — neobsahuje všechny metody a parametry, které zobrazené třídy mají — ale pro potřeby představení uložení grafu je dostatečný. Graf je uložený v mapě stavů, jejímž klíčem je jméno stavu. Tyto stavy si dále drží polohu, tvar a popis. Každý stav pak může mít přechody. Přechody si navíc pamatují jméno pravého stavu a popis přechodu.

4.3.2 Tvorba grafu z proudu tokenů

Tato část programu se dá považovat za jádro, avšak popsat zde detailně, jak se z proudu tokenů vytvoří graf, by bylo velmi dlouhé a nepřehledné. Tvorba probíhá v konečném automatu s konečnými podautomaty a celkový počet stavů je okolo 50 — odtud plyne předpoklad o nepřehlednosti detailního popisu tvorby. Vstupem pro automat jsou jednotlivé tokeny z proudu. Překlad probíhá v pěti hlavních stavech: Před deklarací typů stavů, před `int main ()` funkcí, před příkazem `switch`, uvnitř příkazu `switch` a po příkazu `switch`.

Stručný popis jednotlivých stavů z automatu hlavní úrovně:

1. Před deklarací typů stavů
   - Počáteční stav automatu.
   - Tento stav končí deklarací typů stavů.

   - Zároveň během tohoto stavu lze deklarovat `FSM`, `ARROWTYPE`, `COLUMNTYPE` a `ROWTYPE`.

2. Před `int main ()` funkcí

   - V tomto stavu jsou plněny stavy s názvy, popisy a tvarem do mapy stavů.
   - Zároveň během tohoto stavu je zapotřebí deklarovat `FSM`, `ARROWTYPE`, `COLUMNTYPE` a `ROWTYPE`, pokud se tak již nestalo v předchozím stavu.
   - Stav končí, když začíná `int main ()` funkce.

3. Před příkazem `switch`

   - Během tohoto stavu se nastaví počáteční stav do stavové proměnné a typ šipky do korespondující proměnné.
   - Po přečtení příkazu `switch` končí tento stav.

4. Uvnitř příkazu `switch`

   - Tento stav obstarává čtení a ukládání přechodů mezi stavy v grafu.
   - Přečtením pravé složené závorky začíná následující stav.

5. Po příkazu `switch`

   - Poslední stav pouze čeká na uzavírací formality; `return 0;` a pravou složenou závorku uzavírající `main` funkci.

4.4 Generování Python kódu pro graf

Samotné generování Python kódu je velmi jednoduché, když člověk zná formát vstupu pro PyDia, viz sekce Popis výstupního kódu v Pythonu. Jelikož se prakticky jedná jen o vypsání vnitřní reprezentace grafu ve formátu PyDia do souboru, považuji za zbytečné toto více rozebírat.

Kapitola 5

Testování

Hlavní testování tohoto programu proběhlo v sérii tří testů: v každém z nich bylo cílem vytvořit předem daný diagram pomocí nástroje a porovnat očekávání s výsledky. Zároveň stojí za zmínku, že veškeré diagramy v práci (až na diagramy v kapitole Analýza existujících řešení) byly vytvořené nástrojem, čímž byly provedeny další testy, které v této kapitole nejsou zmíněny.

5.1 Test: Malý diagram

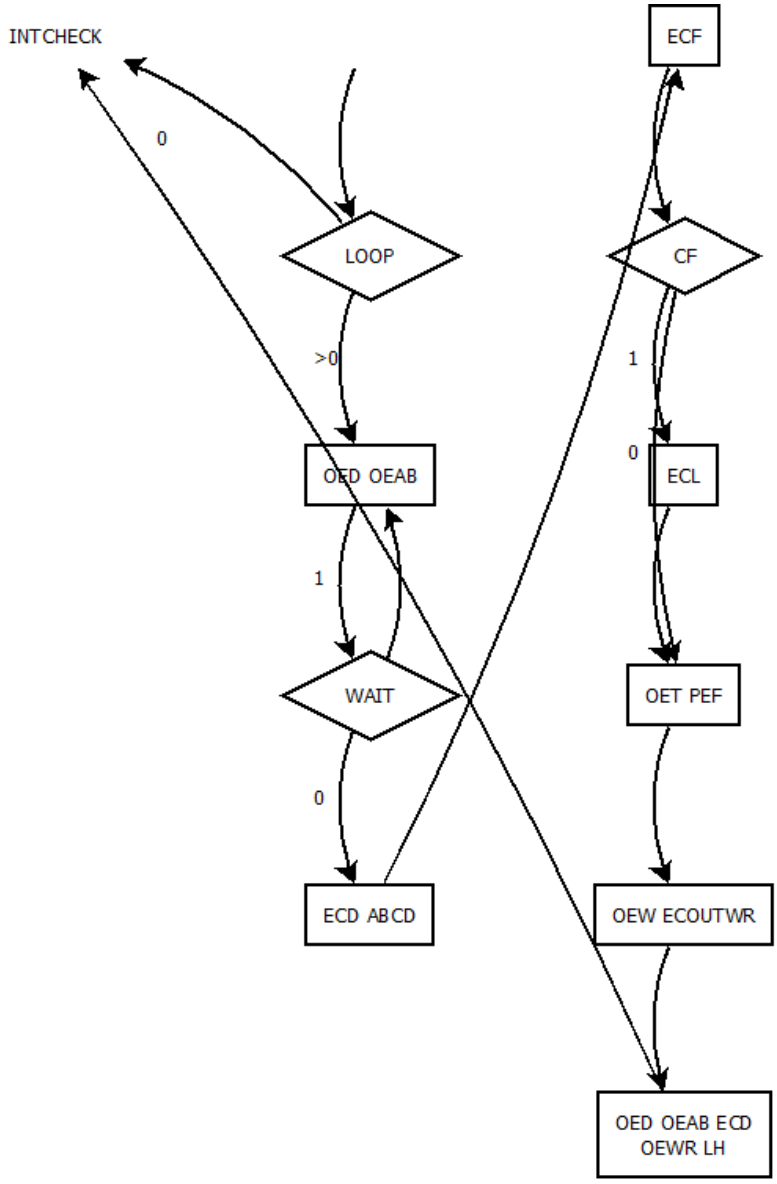
Cílem je vytvořit malý diagram o přibližně 10 stavech. Diagram by měl být výřezem z vývojového diagramu. Stavy by tedy měly obsahovat popisky a v diagramu by se měl nacházet alespoň jeden stav s více přechody. Stavy s více přechody by měly mít navíc popis u přechodů.

```
TYPE Diamond, Box, Empty;

Diamond  = {{LP, "LOOP"},{CF, "CF"},{WT, "WAIT"}};
Box = {
    {OED, "OED␣OEAB"},{ECD, "ECD␣ABCD"},{ECF, "ECF"},
    {ECL, "ECL"},{OET, "OET␣PEF"},{OEW, "OEW␣ECOUTWR"},
    {OED2, "OED␣OEAB␣ECD\nOEWR␣LH"}};
Empty = {{INIT, ""},{INTCHECK, "INTCHECK"}};


FSM state;
ARROWTYPE arrow;
ROWTYPE row;
COLUMNTYPE col;

int main () {

    state = INIT;
    arrow = Arc;
```

```
    switch ( state ) {
        case INIT:
            state = LP;
            col = 1;
            break;
        case LP:
            if ( 0 ) state = INTCHECK;
            if ( >0 ) state = OED;
            break;
        case OED:
            state = WT;
            break;
        case WT:
            if ( 1 ) state = OED;
            if ( 0 ) state = ECD;
            break;
        case ECD:
            state = ECF;
            break;
        case ECF:
            state = CF;
            row = 0;
            col = 2;
            break;
        case CF:
            if ( 0 ) state = OET;
            if ( 1 ) state = ECL;
            break;
        case ECL:
            state = OET;
            break;
        case OET:
            state = OEW;
            break;
        case OEW:
            state = OED2;
            break;
        case OED2:
            state = INTCHECK;
            break;
        case INTCHECK:
            col = 0;
            row = 0;
            break;
    }
    return 0;
}
```

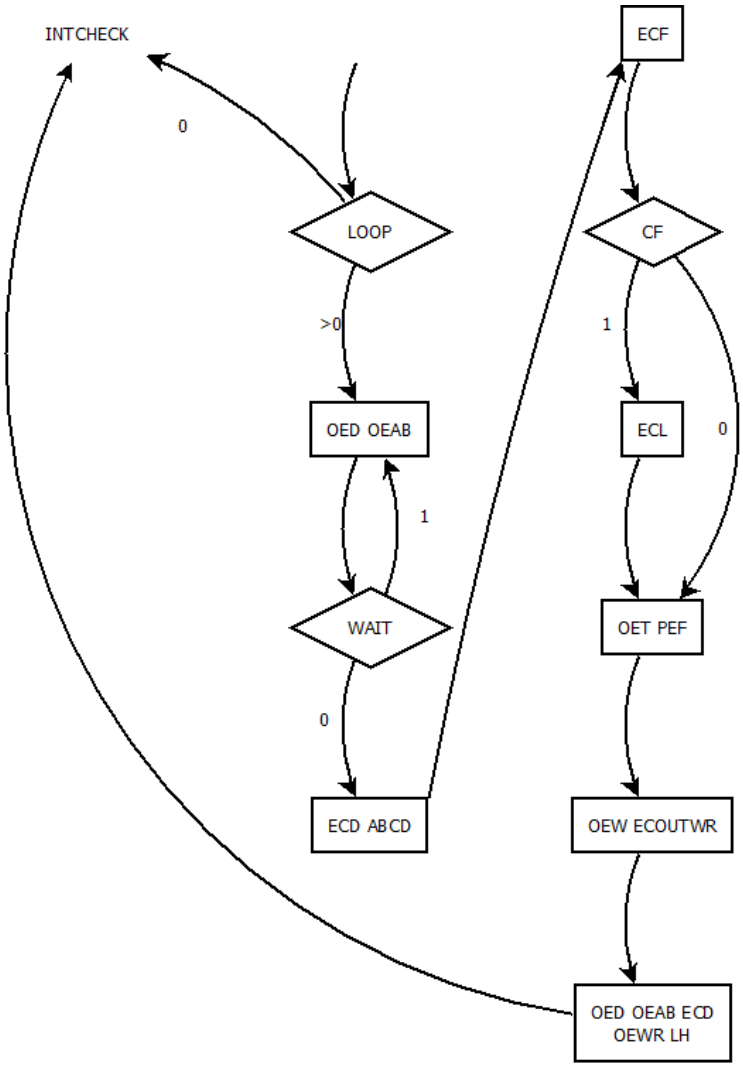
Ukázka kódu 5.1: Vstup pro malý diagram

Tento kód, když se po překladu do správného formátu spustí v Dia, vytvoří následující diagram.



Obrázek 5.1: Test, malý diagram bez úprav

Jak je vidět, vytvořený diagram je skoro až nepoužitelný. Ovšem díky tomu, že diagram lze nadále editovat pomocí Dia WYSIWIGu, stačí pouhých 20 vteřin ladících úprav a diagram je rázem použitelný.



Obrázek 5.2: Test, malý diagram po úpravách

Z předchozího obrázku je vidět, že takovýto diagram je dostatečně dobrý k prezentování. Je na něm na první pohled vidět informace, kterou se diagram snaží předat a není potřeba hádat/tipovat, co má diagram představovat.

5.2 Test: Středně velký diagram s hodně přechody mezi stavy

V tomto testu jsou zkoušeny schopnosti nástroje při tvoření diagramu s hodně přechody mezi stavy. Z mé zkušenosti vyplývá, že takovéto diagramy jsou často velmi chaotické, pokud se kreslí na papír. Jedná se o automat představující jednoduchý automat na jízdenky, kam se dají vhazovat mince 1, 2,

5, 10, 20 Kč, zmáčknout tlačítko pro tisk jízdenky v hodnotě 6, 8, 12 Kč, anebo kliknout STORNO. Jelikož je vstupní kód pro automat velmi dlouhý, některé stavy ze vstupního kódu jsou vynechány, neboť jsou velmi podobné těm předchozím/následujícím, a díky tomu to bude alespoň trochu přehlednější. Popisy a0, ..., a8 reprezentují jednotlivé akce (a0 je STORNO, a1, ..., a5 jsou pro vhazování mincí a a6, ..., a8 reprezentují tisk jízdenek).

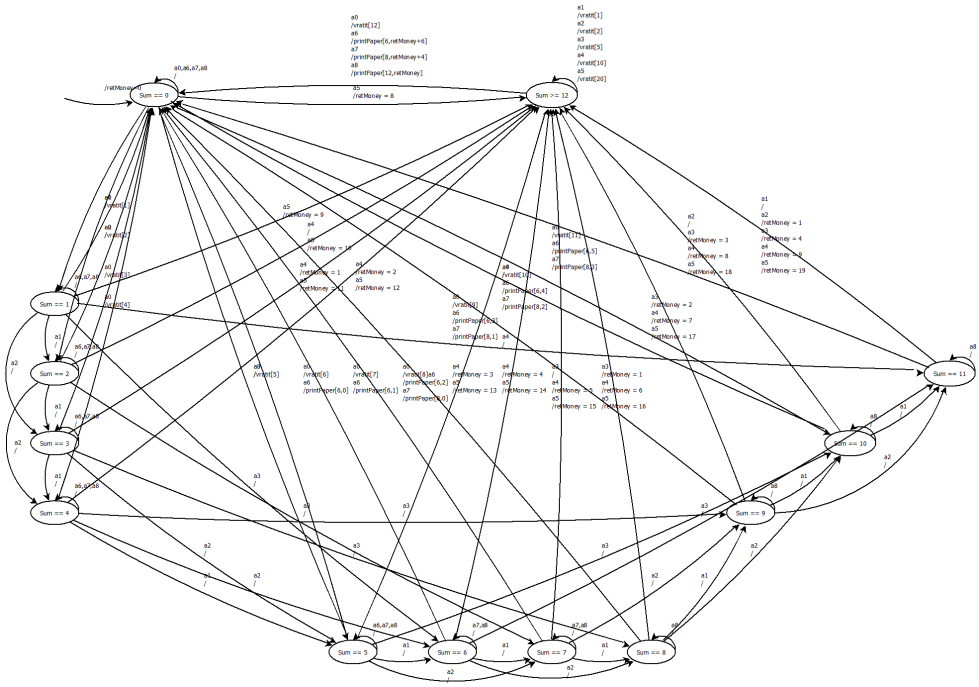
```
TYPE Ellipse, Empty;

Ellipse = {{q0,"Sum == 0"},{q1,"Sum == 1"},{q2,"Sum == 2"},
{q3,"Sum == 3"},{q4,"Sum == 4"},{q5,"Sum == 5"},
{q6,"Sum == 6"},{q7,"Sum == 7"},{q8,"Sum == 8"},
{q9,"Sum == 9"},{q10,"Sum == 10"},{q11,"Sum == 11"},
{q12,"Sum >= 12"}
};
Empty = {{INIT,""}};


ARROWTYPE arrow;
FSM state;
ROWTYPE row;
COLUMNTYPE col;

int main (){
   state = INIT;
   arrow = Arc;
   switch(state){
      case INIT:
         col = 1;
         if (/retMoney=0) state = q0;
         break;
      case q0:
         col = 2;
         row = 0;
         if (a0,a6,a7,a8\n/) state = q0;
         if (a1\n/) state = q1;
         if (a2\n/) state = q2;
         if (a3\n/) state = q5;
         if (a4\n/) state = q10;
         if (a5\n/retMoney = 8) state = q12;
         break;
      case q1:
         col = 1;
         row = 3;
         if (a0\n/vratit(1)) state = q0;
         if (a1\n/) state = q2;
         if (a2\n/) state = q3;
         if (a3\n/) state = q6;
         if (a4\n/) state = q11;
```

```
        if (a5\n/retMoney = 9) state = q12;
        if (a6,a7,a8\n/) state = q1;
        break;
    case q2:
        //zkráceno
    case q3:
        //zkráceno
    case q4:
        //zkráceno
    case q5:
        col = 4;
        row = 8;
        if (a0\n/vratit(5)) state = q0;
        if (a1\n/) state = q6;
        if (a2\n/) state = q7;
        if (a3\n/) state = q10;
        if (a4\n/retMoney = 3\na5\n/retMoney = 13) state = q12;
        if (a6,a7,a8\n/) state = q5;
        break;
    case q6:
        col = 5;
        row = 8;
        if (a0\n/vratit(6)\na6\n/printPaper(6,0)) state = q0;
        if (a1\n/) state = q7;
        if (a2\n/) state = q8;
        if (a3\n/) state = q11;
        if (a4\n/retMoney = 4\na5\n/retMoney = 14) state = q12;

        if (a7,a8\n/) state = q6;
        break;
    case q7:
        //zkráceno
    case q8:
        //zkráceno
    case q9:
        //zkráceno
    case q10:
        col = 9;
        row = 5;
        if (a0\n/vratit(10)\na6\n/printPaper(6,4)\n
            a7\n/printPaper(8,2)) state = q0;
        if (a1\n/) state = q11;
        if (a2\n/\na3\n/retMoney = 3\na4\n/retMoney = 8\n
            a5\n/retMoney = 18) state = q12;
        if (a8\n/) state = q10;
        break;
    case q11:
        //zkráceno
    case q12:
```

```
            col = 6;
            row = 0;
            if (a0\n/vratit(12)\na6\n/printPaper(6,retMoney+6)\n
                a7\n/printPaper(8,retMoney+4)\na8\n
                /printPaper(12,retMoney)) state = q0;
            if (a1\n/vratit(1)\na2\n/vratit(2)\na3\n/vratit(5)\n
                a4\n/vratit(10)\na5\n/vratit(20)) state = q12;
            break;
    }
    return 0;
}
```

Ukázka kódu 5.2: Vstup pro středně velký diagram s hodně přechody mezi stavy

Je důležité podotknout, že víceřádkové formátování je v ukázce pouze pro potřeby přehlednosti. Pokud by se takovýto kód spustil, vygenerované popisy šipek by obsahovaly navíc všechny bílé znaky (tabulátory a nové řádky), zde použité pro formátování. Několik stavů je zkrácených, neboť je zřejmé, jak zhruba vstupní kód vypadá — velká část je kopírovatelná.



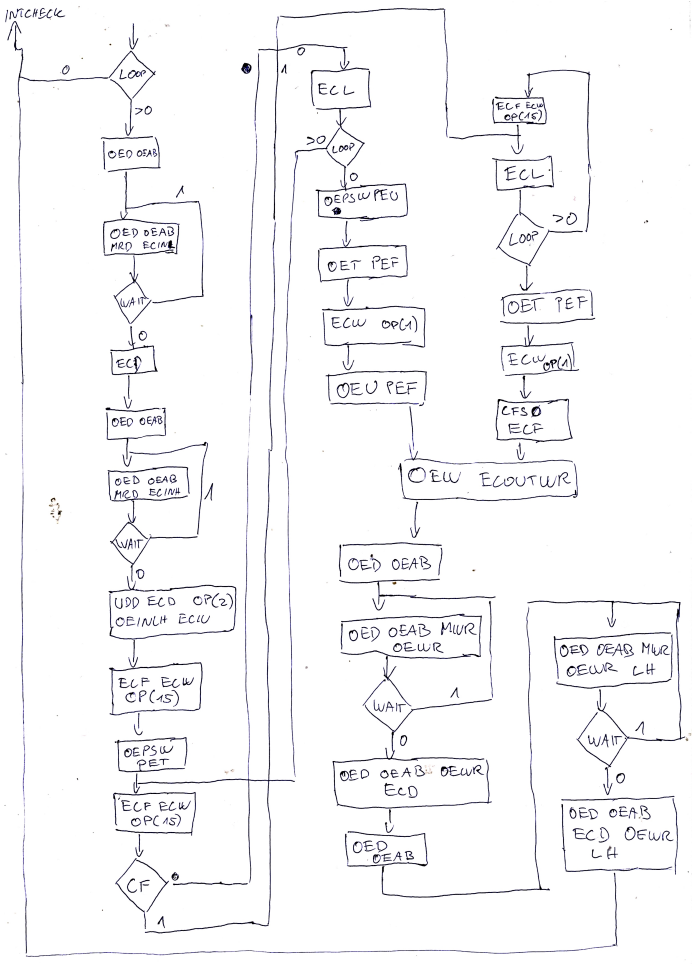
Obrázek 5.3: Test, miniatura středně velkého diagramu

Jelikož papír formátu A4 je nevhodným médiem pro zobrazení tohoto diagramu, lze ho sem dát pouze ve zmenšené podobě. Z diagramu, jehož doladění zabralo přibližně 2 minuty, jde i přesto pár informací získat — text je sice

nečitelný, ale jde si jednoduše představit, že ve správné velikosti by čitelný byl — text nepřesahuje skrz sebe, u přechodů je jasně vidět, kde je začátek a kde konec. Zároveň tento test otestoval, že nástroj zvládá i tvorbu přechodů počínající a končící ve stejném stavu.

5.3 Test: Velký diagram s málo přechody mezi stavy

Tento test je založený na reálném diagramu tvořeným pro předmět BI-JPO. Při tvorbě právě tohoto diagramu mi došlo, že kreslení velkých diagramů na papír není zcela praktické, neboť by stačila jedna chyba, a musel bych dělat celý diagram od začátku.



Obrázek 5.4: Vývojový diagram pro předmět BI-JPO

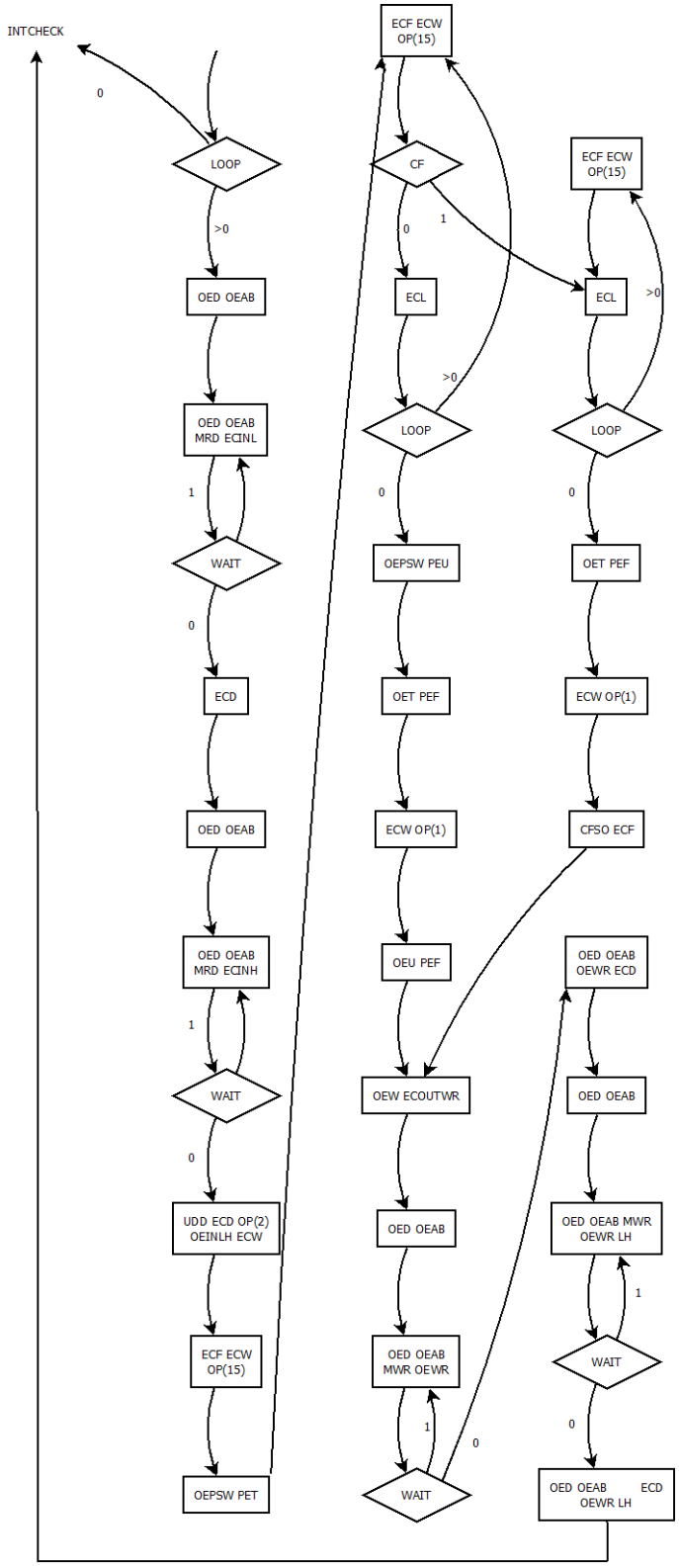
Tím se ovšem nabízí jako ideální diagram pro testování nástroje. Jak lze vidět, v reálném vývojovém diagramu je sice hodně stavů, ale z jakéhokoliv stavu jsou maximálně dva přechody do jiného stavu.

```
TYPE Diamond, Box, Empty;

Diamond = {{LP,"LOOP"},{WT,"WAIT"},{WT2,"WAIT"},
{CF,"CF"},{LP2,"LOOP"},{LP3,"LOOP"},{WT3,"WAIT"},
{WT4,"WAIT"}};
Box = {
    {OED,"OED␣OEAB"},{OED2,"OED␣OEAB\nMRD␣ECINL"},
{ECD,"ECD"},{OED3,"OED␣OEAB"},
{OED4,"OED␣OEAB\nMRD␣ECINH"},
{UDD,"UDD␣ECD␣OP(2)\nOEINLH␣ECW"},{ECF,"ECF␣ECW\nOP(15)"},
{OEPSW,"OEPSW␣PET"},{ECF2,"ECF␣ECW\nOP(15)"},
{ECL,"ECL"},{OEPSW2,"OEPSW␣PEU"},{OET,"OET␣PEF"},
{ECW,"ECW␣OP(1)"},{OEU,"OEU␣PEF"},{ECF3,"ECF␣ECW\nOP(15)"},
{ECL2,"ECL"},{OET2,"OET␣PEF"},{ECW2,"ECW␣OP(1)"},
{CFSO,"CFSO␣ECF"},{OEW,"OEW␣ECOUTWR"},
{OED5,"OED␣OEAB"},{OED6,"OED␣OEAB\nMWR␣OEWR"},
{OED7,"OED␣OEAB\nOEWR␣ECD"},{OED8,"OED␣OEAB"},
{OED9,"OED␣OEAB␣MWR\nOEWR␣LH"},
{OED10,"OED␣OEAB␣␣ECD\nOEWR␣LH"}
};
Empty = {{INIT,""},{INTCHECK,"INTCHECK"}};


FSM state;
ARROWTYPE arrow;
ROWTYPE row;
COLUMNTYPE col;

int main () {

    state = INIT;
    arrow = Arc;

    switch ( state ) {
       case INIT:
          col = 1;
          state = LP;
          break;
       case LP:
          if ( 0 ) state = INTCHECK;
          if ( >0 ) state = OED;
          break;
       case OED:
          state = OED2;
          break;
```

```
        case OED2:
            state = WT;
            break;
        case WT:
            if (1) state = OED2;
            if (0) state = ECD;
            break;
        //zkráceno
        case ECF3:
            col = 3;
            row = 1;
            state = ECL2;
            break;
        //zkráceno
        case ECF2:
            col = 2;
            row = 0;
            state = CF;
            break;
        //zkráceno
        case OED7:
            col = 3;
            row = 7;
            state = OED8;
            break;
        //zkráceno
        case OED10:
            state = INTCHECK;
            break;
        case INTCHECK:
            col = 0;
            row = 0;
            break;
    }
    return 0;
}
```

Ukázka kódu 5.3: Vstup pro velký diagram s málo přechody mezi stavy

Kvůli množství stavů a relativní nezajímavosti kódu byly některé úseky zkráceny. Z kódu, jenž byl ponechán, je možné si všimnout, že hlavní část kódu se velmi opakuje, prakticky buď posun do následujícího stavu, anebo návěstí s dvěma možnými přechody. Nejdéle zabralo psaní definice stavů a hlavně vyplňování popisu stavu. Čas strávený vyplňováním popisu stavu by ovšem byl stejně dlouhý při jakémkoliv jiném způsobu tvoření diagramů. Naštěstí nástroj při pokusu o deklaraci stavu, který již byl deklarován, vypíše chybové hlášení. Vzniklých chybových hlášení jsem využil a během chvíle stavy v konfliktu očísloval.



Obrázek 5.5: Nástrojem generovaný vývojový diagram pro předmět BI-JPO

Závěrečná úprava (včetně změny typu šipky vedoucí na `INTCHECK`) zabrala méně než 3 minuty, což je přijatelný čas vzhledem k tomu, že diagram vypadá velmi prezentovatelně — pokud by se vešel na stránku. Ovšem i přes to, že se nevejde do této práce v celé svojí kráse, lze usoudit, že nástroj má své využití. Diagramy vypadají velmi dobře, vzhledem k času stravéném na vytvoření.

Všechny testy proběhly bez problému a nástroj pracoval dle očekávání.

Závěr

Nástroj pro tvorbu vývojových diagramů se povedlo úspěšně vytvořit. Všechny cíle práce, a dokonce i mé požadavky definované v kapitole 3 byly úspěšně splněny. Vstupní formát pro nástroj je srozumitelný a generování diagramů probíhá svižně. Velice zajímavým podnětem pro mne byla lexikální analýza, se kterou se během tvorby nástroje setkal. Díky ní lze jednoduše nástroj rozšířit o různé funkcionality, např. rozšíření vstupního formátu, aby podporoval vkládání komentářů, jak to podporuje jazyk C, zabralo pouhých 10 minut. Bez ní by to zabralo několikanásobně déle.

Již při prvních testech bylo zřejmé, že nástroj bude velice užitečný, jelikož sám jsem ho začal používat ihned, jakmile to bylo možné. Všechny diagramy v práci byly tvořeny pomocí tohoto nástroje. Z toho je zřejmé, že nástroj má své opodstatnění. Samotné tvoření bylo rychlejší, a hlavně pro programátora přirozenější než pouhé přenášení objektů v kreslícím programu.

Ačkoliv je nástroj již v použitelném stavu, je několik možností, jak ho ještě vylepšit. Mezi hlavní možnosti vylepšení patří vlastní grafické generovaní výsledného diagramu. Kdyby výstupem programu byl rovnou hotový diagram, určitě by to bylo příjemnější na používání, než nutnost přenášet skript do programu Dia. Dalším velkým zlepšením by určitě bylo změnit způsob zadávání kódu diagramu — od jednoduché aplikace, kde by se zvolil soubor s kódem, až po vlastní textový editor.

Literatura

[1] code2flow - online interactive code to flowchart converter. [online], listopad 2016, [cit. 2018-04-17]. Dostupné z: https://code2flow.com/app

[2] GNU GENERAL PUBLIC LICENSE v2.0. [online], duben 2007, [cit. 2018-04-18]. Dostupné z: http://dia-installer.de/doc/gpl-2.0.html

[3] Python: module dia. [online], únor 2010, [cit. 2018-01-24]. Dostupné z: https://projects-old.gnome.org/dia/pydia.html

[4] Dia Diagram Editor. [online], duben 2007, [cit. 2018-04-03]. Dostupné z: http://dia-installer.de

[5] Lexikální analyzátor. [online], únor 2013, [cit. 2018-05-01]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-PJP/lectures/02/start

[6] How to use Dia with Python on Windows. [online], červenec 2013, [cit. 2018-05-03]. Dostupné z: http://dia-installer.de/howto/python_win32/index.html.en

Dodatek A

Uživatelská příručka

Tato příručka k nástroji pro generování vývojových diagramů z podmnožiny jazyka C obsahuje návod, jak zprovoznit tento nástroj na Windows, popsání vstupního formátu, zmíní možnosti pro diagramy, které lze s nástrojem vytvořit, a na konci budou příklady nástroje v provozu.

A.1 Instalace

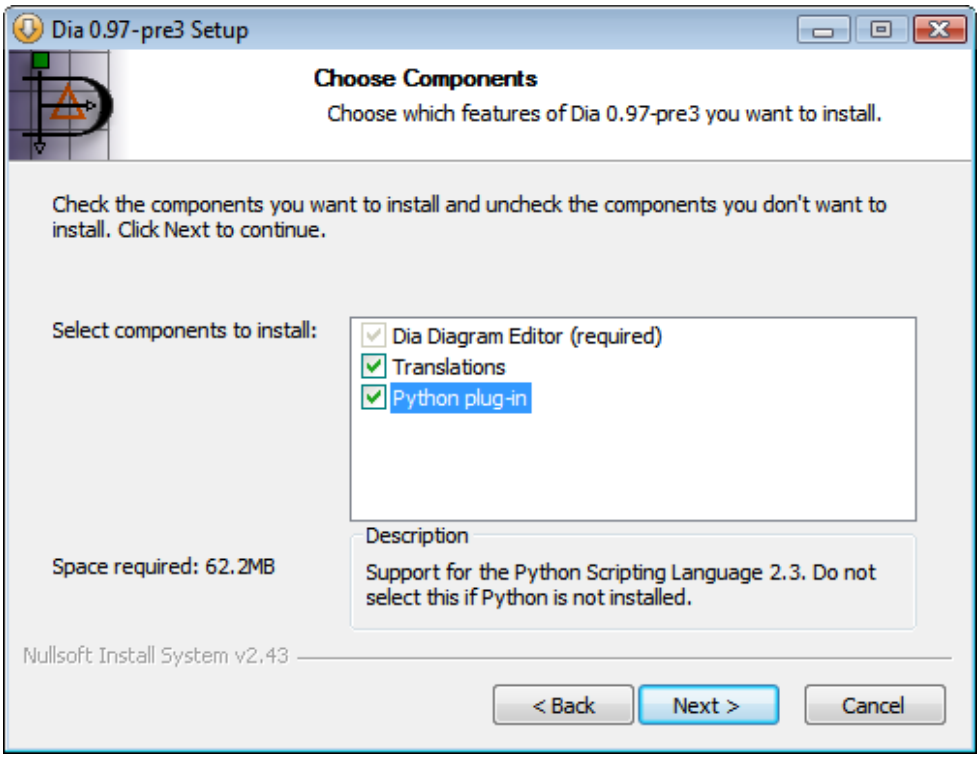
Ačkoliv je instalace Dia s Pythonem dobře popsána na jejich oficiálních stránkách [6, `http://dia-installer.de/howto/python_win32/index.html.en`], zkrácená verze je obsažena i v této příručce.

1. Instalace Python 2.3

   - Dia pro Windows podporuje pouze verzi 2.3.
   - Na přiloženém médiu se nachází v adresáři instalace se jménem Python-2.3.5.exe

2. Instalace Dia

   - Při výběru komponent je nutné zaškrtnout políčko "Python plug-in".
   - Na přiloženém médiu se nachází v adresáři instalace se jménem dia-setup-0.97.2-2-unsigned.exe



Obrázek A.1: Instalace Dia

3. Instalace PyCairo 1.0.2

   - Na přiloženém médiu se nachází v adresáři instalace se jménem pycairo-1.0.2-1.win32-py2.3.exe

4. Instalace PyGtk 2.8.6

   - Na přiloženém médiu se nachází v adresáři instalace se jménem pygtk-2.8.6-1.win32-py2.3.exe

5. Nahrání nástroje do domovského adresáře Dia.

   - Tento krok není nutný, ale velmi doporučený.

   - Na přiloženém médiu v adresáři exe všechny tři soubory (FlowchartGenerator.exe, generatedgraph.py, graphinput.c) zkopírovat do domovského adresáře Dia.

A.2 Použití

Jakmile je instalace dokončená, stačí do souboru `graphinput.c` vložit Váš kód ve vstupním formátu, spustit nástroj a ujistit se, že vygenerovaný soubor `generatedgraph.py` se nachází v domovském adresáři Dia. Při následujícím spuštění Dia bude na horní liště okna s diagramy/nástroji tlačítko `Print Generated Diagram`. Po stisknutí tohoto tlačítka se vytvoří nové okno s Vaším diagramem.

A.3 Vstupní formát

Pro následující ukázku platí, že uvnitř hranatých závorek jsou nepovinné části a [0-9]* symbolizuje kladné celé číslo. Celý vstupní formát má takovouto podobu:

```
TYPE type1  [, type2, .. , type n] ;

type1  = {{state1,"state1text"}
[,{state2,"state2text"}, .. ,{staten,"statentext"}]    };
[
type2  = {{state(n+1),"state(n+1)text"}
[,{state(n+2),"state(n+2)text"}, .. ,
   {state(n+n),"state(n+n)text"}]    };
..
typen  = {{state(n*n+1),"state(n*n+1)text"}
[,{state(n*n+2),"state(n*n+2)text"}, .. ,
   {state(n*n+n),"state(n*n+n)text"}]    };
]

FSM fsmVariable;
ARROWTYPE arrowtypeVariable;
ROWTYPE rowtypeVariable;
COLUMNTYPE columntypeVariable;

int main () {

   fsmVariable = initialState;
   arrowtypeVariable = shapeOfTheArrows;

   switch ( fsmVariable ) {
      case stateNum:
         [
         [if (labelForArrow) ]  fsmVariable = stateNum;
         ..
         [if (labelForArrow) ]  fsmVariable = stateNum;
         ]
         [columntypeVariable = [0-9]*; ]
         [rowtypeVariable = [0-9]*; ]
```

```
            break;
        }
        return 0;
}
```

Ukázka kódu A.1: Vstup pro velký diagram s málo přechody mezi stavy

Na první pohled se může zápis zdát nepřehledný. Hned však při pohledu na první příklad to bude jasné. Je důležité zmínit, že **pokud není specifikováno příslušnými proměnnými pro pozici sloupečku a řádek jinak**, platí, že následující stav má **stejný sloupeček** jako stav předchozí a **řádek o jedna zvětšený**.

A.4 Možnosti diagramů

1. Povolené typy stavů:
   - Empty, Diamond, Box, Ellipse jsou nejběžnějšími typy, pro něž je tento nástroj vytvořen.
   - Typ Empty je ve skutečnosti stejný jako typ Box, akorát má bílé okraje — jeho využití je jako skrytý stav například při tvorbě šipky vedoucí do počátečního stavu.
   - Další typy, které nástroj podporuje: Parallelogram, Display, Document, Preparation, Terminal, Merge, Extract, Delay.
2. Povolené typy šipek:
   - PolyLine, Arc, Line.
   - Arc jsou zahnuté šipky, PolyLine a Line si jsou podobné — rovné šipky, ke kterým lze ve WYSIWIGu přidělávat rohy a tím je zahnout.
3. Omezení:
   - Pro každou levou závorku v textu u přechodu u šipek v `if (text)` musí existovat závorka pravá.
   - Popis u stavu nesmí obsahovat uvozovky ["].

A.5 Příklady

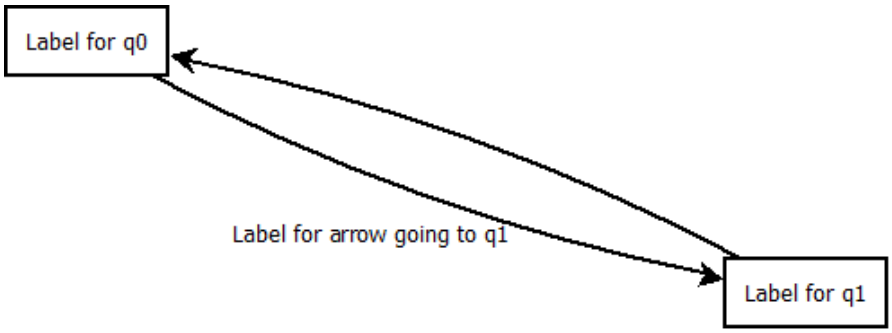
A.5.1 Velmi malý diagram

Prvním příkladem je velmi malý diagram. Zde si čtenář může povšimnout hlavně kostry vstupního formátu, vzhledem k malému počtu stavů. Jedná se o stejný diagram jako v kapitole Návrh formátu vstupního kódu bakalářské práce, včetně doprovodných komentářů v kódu.

```
//Deklarace typu ---- bude alespon jeden stav s typem Box
TYPE Box;
/*
Pro typ Box budou dva stavy, q0 a q1 a tyto stavy budou
mit definovane popisy
*/
Box = { {q0,"Label for q0"},{q1,"Label for q1"} };

//Deklarace promenne pro typ sipek
ARROWTYPE arrow;
//Deklarace promenne uchovavajici aktualni stav
FSM state;
//Deklarace promenne pro urceni sloupecku
COLUMNTYPE col;
//Deklarace promenne pro urceni radku
ROWTYPE row;

int main() { //int main () je pro zachovani pocitu jazyka C
    //Nastaveni sipek jako Arc
    arrow = Arc;
    //Nastaveni q0 jako pocatecniho stavu
    state = q0;

    /*
    promenne pro radek a sloupecek maji defaultni hodnoty 0
    a nelze je v teto casti prenastavit (lze u prvniho stavu)
    */

    //Hlavni prikaz switch dle stavu
    switch (state) {
    /*
    Kdyz je v C napsany case ..., rozumi se tim vetsinou
    "Kdyz stav je ..."  V pripade vstupniho formatu je lepsi
    toto cist jako "Ze stavu ..."
    */
    case q0:
        /*
        Nasledujici radek lze precist takto:
            Pri prechodu (ze stavu q0) do stavu q1
            pridej k sipce popis (definovany v zavorkach)
        */
```

```
        if (Label for arrow going to q1) state = q1;
        break; //break je pro zachovani pocitu jazyka C
    case q1:
        /*
        Zde neni if() cast, a proto sipka, ktera se vytvori
        ze stavu q1 do stavu q0, nebude mit zadny popis.
        */
        state = q0;
        /*
        Nastavenim sloupecku na nejakou hodnotu je nastavena
        X souradnice. Jelikoz radky jsou na autoinkrementu
        (neni-li dano jinak), nemusime souradnici radku
        nastavovat.
        */
        col = 2;
        break;
    }
    return 0;//return 0 je pro zachovani pocitu jazyka C
}
```

Ukázka kódu A.2: Popis velmi malého diagramu vstupním formátem



Obrázek A.2: Příklad velmi malého diagramu

A.5.2 Malý diagram

```
TYPE Diamond, Box, Empty;

Diamond  = {{LP,"LOOP"},{CF,"CF"},{WT,"WAIT"}};
Box = {
    {OED,"OED␣OEAB"},{ECD,"ECD␣ABCD"},{ECF,"ECF"},
    {ECL,"ECL"},{OET,"OET␣PEF"},{OEW,"OEW␣ECOUTWR"},
    {OED2,"OED␣OEAB␣ECD\nOEWR␣LH"}};
Empty = {{INIT,""},{INTCHECK,"INTCHECK"}};


FSM state;
ARROWTYPE arrow;
ROWTYPE row;
COLUMNTYPE col;

int main () {

    state = INIT;
    arrow = Arc;

    switch ( state ) {
        case INIT:
            state = LP;
            col = 1;
            break;
        case LP:
            if ( 0 ) state = INTCHECK;
            if ( >0 ) state = OED;
            break;
        case OED:
            state = WT;
            break;
        case WT:
            if ( 1 ) state = OED;
            if ( 0 ) state = ECD;
            break;
        case ECD:
            state = ECF;
            break;
        case ECF:
            state = CF;
            row = 0;
            col = 2;
            break;
        case CF:
            if ( 0 ) state = OET;
            if ( 1 ) state = ECL;
```

```
            break;
        case ECL:
            state = OET;
            break;
        case OET:
            state = OEW;
            break;
        case OEW:
            state = OED2;
            break;
        case OED2:
            state = INTCHECK;
            break;
        case INTCHECK:
            col = 0;
            row = 0;
            break;
    }
    return 0;
}
```

Ukázka kódu A.3: Příklad, vstup pro malý diagram

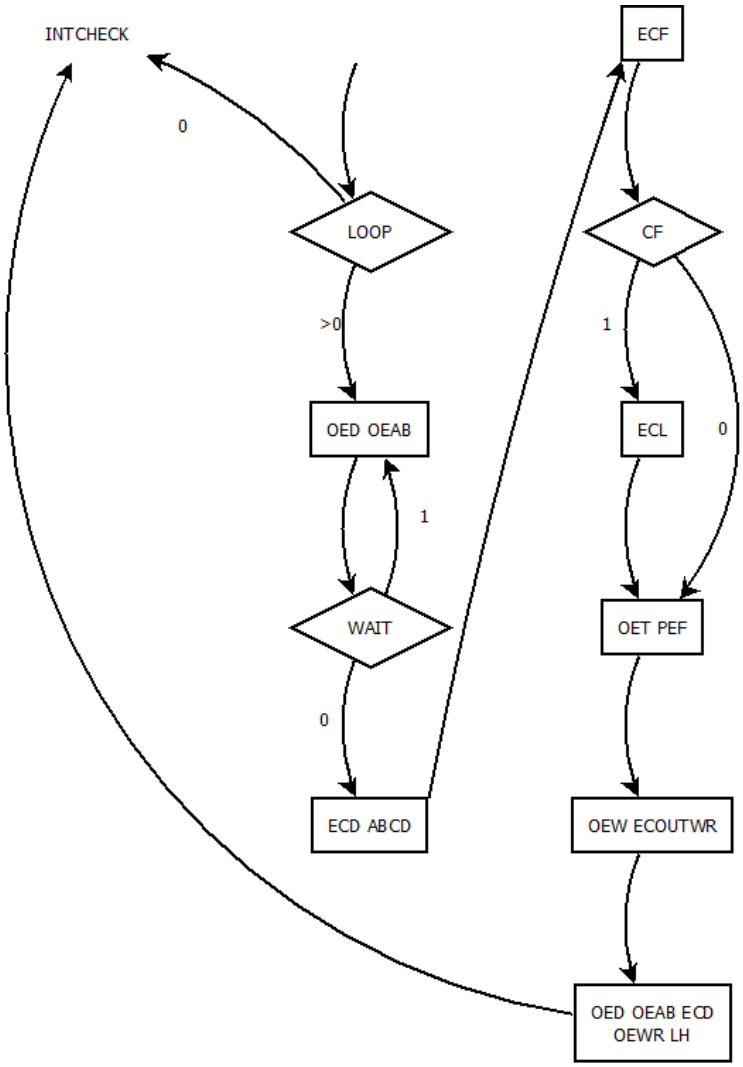
Na tomto příkladu lze krásně vidět všechny důležité funkcionality nástroje:

1. využití skrytého stavu,

2. používání nastavování pozice,

3. rozdělení popisu u stavu OED2 na více řádků pomocí '\n'.

Výsledný diagram:



Obrázek A.3: Příklad, malý diagram

Dodatek B

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible Markup Language

OSS Open-source software

UI User interface

WYSIWYG What You See Is What You Get

RGB Red Green Blue

Dodatek C

Obsah přiloženého média

Médium obsahuje jak zdrojové kódy aplikace, tak i celou práci a instalační soubory pro nainstalování Dia s Python pluginem pro správný chod aplikace.

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou nástroje
  - `generatedgraph.py` ..... ukázka vygenerovaného Python scriptu
  - `graphinput.c` ..... ukázka souboru se vstupním kódem
  - `FlowchartGenerator.exe` ..... nástroj
- `src` ..... repozitář impementace
  - `FlowchartGenerator` ..... adresář se zdrojovými kódy impementace
  - `FlowchartGenerator.sln` ..... Visual Studio projekt
- `text` ..... adresář se zdrojovými kódy textu práce
  - `bp_prochvoj.tex` ..... zdrojová forma práce ve formátu LaTeX
  - `bp_prochvoj.pdf` ..... text práce ve formátu PDF
- `instalace` ..... adresář s různými instalačními soubory
  - `dia-setup-0.97.2-2-unsigned.exe` ..... instalační soubor pro Dia
  - `pycairo-1.0.2-1.win32-py2.3.exe` ... instalační soubor pro PyCairo
  - `pygtk-2.8.6-1.win32-py2.3.exe` ..... instalační soubor pro PyGtk
  - `Python-2.3.5.exe` ..... instalační soubor pro Python