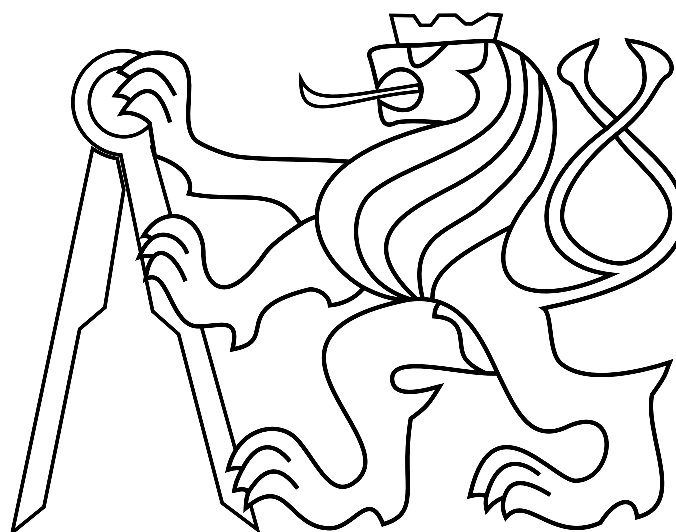
České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídicí techniky



DIPLOMOVÁ PRÁCE

Verifikace programů pro PLC
PLC program verification

Autor : Bc. Ondřej Maslikiewicz

Vedoucí práce : Ing. Pavel Burget, Ph.D.

2017

Prohlášení

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

V Praze, dne ………………………… ……………………………………….
Podpis

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Maslikiewicz Ondřej**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Verifikace programů pro PLC**

Pokyny pro vypracování:

1. Prozkoumejte možnosti exportu a následného zpracování programu, symbolů a datových struktur pro vybrané PLC (např. Siemens Simatic) v odpovídajícím vývojovém prostředí.
2. Proveďte analýzu použitých proměnných kvůli zjištění příčinných souvislostí.
3. Sestavte formální popis chování programu PLC automaticky nebo poloautomaticky na základě jeho analýzy ve formátu vhodném pro použití vybraným nástrojem pro verifikaci vlastností systémů diskrétních událostí.
4. Verifikujte vybrané vlastnosti programu a navrhněte způsob efektivního zpracování výsledků verifikace zpět do původního vývojového prostředí pro PLC.

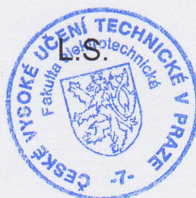
Seznam odborné literatury:

[1] C. Baier, J.- P. Katoen, Principles of Model Checking. MIT Press, Cambridge, Massachusetts. 2008

[2] M. Saeed AbouTrab, Michael Brockway, Steve Counsell, Robert M. Hierons, Testing Real-Time Embedded Systems using Timed Automata based approaches, Journal of Systems and Software, Volume 86, Issue 5, May 2013, Pages 1209-1223

Vedoucí: Ing. Pavel Burget, Ph.D.

Platnost zadání: do konce letního semestru 2017/2018



L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze, dne 30. 11. 2016

Poděkování

Zde bych rád poděkoval svému vedoucímu pracé Ing. Pavlu Burgetovi, Ph. D. za jeho pomoc a vedení při tvorbě této práce. Jeho pomoc byla velkým přínosem při psaní této práce i při tvorbě samotného nástroje pro převod a kontrolu PLC programu.

Táke bych rád poděkoval své rodině za podporu nejen při psaní této práce, ale i během celého studia.

Anotace

Tato práce se zabývá analýzou a kontrolou PLC programu. Její součástí je nástroj pro automatické generování binárních rozhodovacích diagramů a stavových automatů pro proměnné, které jsou použity v PLC programu. Hlavním cílem práce je převést program napsaný v Ladder Diagramu přímo do BDD a stavového automatu. Kontrola programu je provedena převedením do UPPAAL a následnou verifikací.

Klíčová slova

PLC (Programovatelný logický kontrolér), PLC program, Binární rozhodovací diagram, Stavový automat, Verifikace, Analýza

Abstract

This thesis is about analyzing and verification of PLC program. There is also an instrument for automatical generating of binar decision diagram and for state machines for variables which are used in PLC program. Main result of thesis is translation of program in Ladder Diagram into BDD and state machine. Verification of the program is made by translation of state machine into UPPAAL.

Key words

PLC (Programmable Logic Controller), PLC program, Binary Decision Diagram, State machine, Verification, Analysis

Obsah

1 Úvod **1**

1.1 Motivace 1

1.2 Virtuální zprovoznění 3

1.2.1 Simulační nástroje 4

1.2.2 Princip virtuálního zprovoznění 5

1.2.3 Propojení simulace a PLC 5

2 PLC **7**

2.1 Programovací jazyky 8

3 Nástroj **9**

4 Parser STL **10**

4.1 Export zdrojových souborů 11

4.2 Princip převodu programu na logické výrazy 13

4.3 Podporované příkazy STL 14

4.3.1 AND 15

4.3.2 AND not 15

4.3.3 OR 16

4.3.4 OR not 16

4.3.5 Set a Reset 17

4.3.6 Náběžná a sestupná hrana 17

4.4 Časovače 18

4.4.1 ODT 19

4.4.2 ODTS 19

4.4.3 PULSE 20

4.4.4 PEXT 20

4.4.5 OFFDT 21

4.4.6 Závorky 21

5 BDD 23
5.1 Použití při analýze PLC kódu 23
5.2 BDD pro jednotlivé logické operace 24
5.2.1 AND 25
5.2.2 AND not 25
5.2.3 Set Reset a detekce náběžné hrany 26
5.2.4 Časovače 27
5.3 Příklady složitějších výrazů 27
5.4 Dosazování proměnných 30
6 Stavový automat 30
6.1 Přiřazení 32
6.1.1 Jednoduché přiřazení 32
6.1.2 Vícenásobné přiřazení 33
6.2 Set a Reset 35
6.2.1 Set 35
6.2.2 Reset 36
6.2.3 Set a Reset 37
6.3 Časovače 37
6.3.1 ODT a ODTS 38
6.3.2 Více časovačů 39
7 Verifikace PLC programu 40
7.1 Simulace PLC v UPPAAL 41
7.2 Převod PLC programu 42
7.3 Model reálného světa 42
7.4 Verifikace 43
7.5 Verifikace PLC programu otočného stolu 43
7.5.1 Stavový automat pro otočný stůl 44
7.5.2 Převod programu do UPPAAL 45

7.5.3 Verifikace programu pro ovládání otočného stolu 47

8 Závěr **48**

9 Literatura **50**

A Obsah přiloženého CD **51**

B Program pro otočný stůl **52**

Seznam obrázků

1 Menu: Spuštění nástroje pro generování zdrojového kódu 11
2 Výběr zdrojového souboru 12
3 Zdroj pro generování 13
4 Zdrojový soubor 14
5 Logická operace *AND* 15
6 Logická operace *AND not* v LD 15
7 Logická operace *OR* v LD 16
8 Logická operace *OR not* v LD 16
9 Operace *Set* a *Reset* 17
10 Detekce náběžné a sestupné hrany 18
11 Zapojení časovače ODT 19
12 Zapojení časovače ODTS 19
13 Zapojení časovače PULSE 20
14 Zapojení časovače PEXT 20
15 Zapojení časovače OFFDT 21
16 Větvení programu vyžadující závorky pomocí *OR* 22
17 Větvení programu vyžadující závorky pomocí *AND* 22
18 BDD pro logickou operaci *AND*(*Network 1 c.svg*) 25
19 BDD pro logickou operaci *AND not*(*Network 1 c.svg*) 26

20 BDD pro *Set* a detekci náběžné hrany(*Network 1 Set c.svg*) 26
21 BDD pro časovač ODT(*Network 1 c.svg*) 27
22 Složitější struktura operací 28
23 Složitější struktura operací - BDD 29
24 Slučování BDDl: z leva BDD pro c, BDD pro f před sloučením, BDD pro f po sloučení 31
25 Stavový automat pro přiřazení 32
26 Stavový automat pro vícenásobné přiřazení 34
27 Stavový automat pro operaci *Set* 35
28 Stavový automat pro operaci *Reset* 36
29 Stavový automat pro operaci *Set* a *Reset* 37
30 Stavový automat pro časovač ODT 38
31 Stavový automat pro časovač ODTS 39
32 Příklad zapojení časovačů 40
33 UPPAAL stavový automat pro PLC 41
34 Ukázka spojení automatů (PLC - Sken I - Net 1) 42
35 UPPAAL stavový automat otočného stolu 45
36 UPPAAL systém - konec cyklu PLC 45
37 Stavový automat pro proměnnou *run* 46
38 Stavový automat pro proměnnou *rev* 46
39 Stavový automat pro proměnou *SafetyStop* 46
40 UPPAAL stavový automat pro časovač *ODT* s operací *reset* na výstupu . . 47
41 UPPAAL model náhodné změny hodnoty vstupu 48
42 LAD PLC program pro otočný stůl 52

Seznam zkratek

- PLC - Programmable Logic Controller (Programovatelný logický automat)
- IL - Instruction List
- LD - Ladder Diagram
- STL - Statement List
- BDD - Binární rozhodovací diagram (Binary decision diagram)
- ODT - On-delay timer
- ODTS - Retentive on-delay time

1 Úvod

Cílem této diplomové práce je vytvořit nástroj pro analýzu PLC programu. Takovýto nástroj v podstatě neexistuje a není jiný způsob analýzy programu, než jeho podrobné zkoumání pouze ze zdrojových kódů. Taková analýza je ale velmi zdlouhavá a náročná. Nástroj si klade za cíl formálně zobrazit PLC program do jednodušší formy a tím s úkolem analýzy pomoci.

Největším přínosem této práce je automatický převod PLC programu na logické výrazy, respektive na BDD a následná automatická tvorba stavového automatu.

Nástroj nabízí podporu PLC od společnosti Siemens, konkrétně PLC program psaný ve vývojovém prostředí Step7. Podporovaným jazykem je Ladder Diagram (LD). Program napsaný v LD musí být nejprve v prostředí Step7 převeden na formu zápisu pomocí STL.

1.1 Motivace

Hlavní motivací pro vytvoření nástroje určeného k analýze PLC programu není snaha o analýzu a verifikaci vlastního programu, ale kódu, který napsal někdo jiný. Přínosem automatické (poloautomatické) tvorby binárních rozhodovacích stromů (BDD) a stavového automatu je snadné určení vzájemné návaznosti proměnných a jejich vlivu na program.

Verifikace PLC programu je zapotřebí hned v několika případech:

- při navázání na práci někoho jiného
- při kontrole již provedené práce
- při opravení chyb

Ve všech případech je zapotřebí kódu porozumět a pochopit ho. Pro úplné pochopení programu je zapotřebí porozumět vnějšímu chování programu, tzn. jak se program chová k okolnímu světu, ale i vnitřní struktuře celého programu. Tento nástroj je vytvořen za účelem pomoci při této práci.

Vnější chování PLC programu není ve většině případů těžké odpozorovat pomocí testování různých kombinací vstupů. Z následného pozorování na výstupech lze usuzovat o

funkci programu. K tomu pochopení poslouží také specifikace, podle které je daný program napsán. Ve specifikaci je přímo aplikace, kterou má PLC vykonávat, popsána. Takto sice získáme potřebnou informaci o tom, jak program reaguje na vstupy, respektive co by měl vykonávat, ale již nezjistíme nic o tom, jak je program vykonáván. Pro úplné pochopení programu je zapotřebí nahlédnout na vnitřní strukturu. Bez tohoto náhledu není možné s programem dále pracovat, upravovat ho, měnit, nebo opravovat.

Pochopení vnitřního fungování programu je o něco složitější úkol, než pochopení vnějšího chování, respektive funkce programu. U porozumění vnitřní struktury v podstatě neexistuje způsob, jak jednoduše projít kód a pochopit jeho strukturu. Jediným spolehlivým způsobem, je jeho analýza řádek po řádku. To ovšem může být zdlouhavý a náročný proces, neboť takové procházení kódu je většinou mnohem náročnější než jeho psaní. Nejen, že člověk musí být naprosto seznámen s fungováním všech funkcí, funkčních bloků a operací, ale zároveň musí být schopen pochopit myšlenkový proces autora kódu.

Při procházení kódu by měly být nápomocny autorovy komentáře. Ne vždy je tato pomoc k dispozici. Stává se, že program naprosto neobsahuje komentáře, nebo jsou zmatečné a zavádějící.

V případě, že program nemá žádné komentáře, je nutné při následné analýze vycházet pouze z programu jako takového. Pochopení myšlenkového procesu autora je pak velmi složité. Každý programátor má při řešení daného problému svoji vizi o tom, jak daný problém vyřešit. Je možné, že ne pro každého bude použitý postup srozumitelný. Následný proces analýzy programu je pak velmi pomalý a neefektivní. Druhá možnost je, že autor sice kód vybavil komentářem, ale ten je zavádějící. Toto je možná ještě horší případ. U takového komentáře dojde k zavedení jistého předsudku o tom, co daná část programu vykonává. Při analýze takového programu pak člověk musí nejen pochopit samotný program, ale také musí přijít na chybu v úvaze, která vznikala aplikací nepřesného komentáře.

Ne vždy je absence komentářů nutně chybou a nedbalostí programátora. I když by měl každý programátor ctít etiku o komentování své práce, ne vždy je toto dodrženo. Komentáře by programátor měl psát i pro svou vlastní potřebu. V případě zpětného upravování a opravování staršího kódu musí sám autor pochopit vlastní myšlenkový proces. Jsou případy,

kdy je možné absenci komentáře pochopit. Takovým případem je například psaní programu pod nějakým vnitro-firemním standardem, kdy programátor může nabýt dojmu, že komentář je zbytečný. I ostatní programátoři ve firmě jsou se standardem seznámeni, a tudíž je předpoklad, že program bez komentáře bude pro ostatní stále srozumitelný. Program napsaný za pomoci nějakého standardu bývá okomentován velmi stroze. Tato skutečnost by úplně nevadila, kdyby program četl pouze člověk z dané firmy. To je sice předpoklad, na kterém v podstatě vznikl i daný vnitro-firemní standard, ale existuje případ, kdy tento předpoklad neplatí.

Danou situací je virtuální zprovoznění (viz 1.2). U virtuálního zprovoznění se k PLC kódu dostává i člověk z externí firmy, provádějící toto zprovoznění. Ten samozřejmě nemůže znát kompletně všechny standardy, které si někdo ve své firmě nastavil a absence komentářů je pro něj závažný problém. Kompletní znalost standardu by mohla být i na škodu, neboť pomocí virtuálního zprovoznění se dá přijít nejen na chybu v programu, ale i na chyby ve standardu jako takovém. Nutnost porozumění cizímu programu za účelem virtuálního zprovoznění je hlavní motivací pro vytvoření nástroje, který s touto nesnadnou a zdlouhavou prací pomůže.

1.2 Virtuální zprovoznění

Virtuální zprovoznění je relativně nový způsob jak ověřit celý koncept stroje, linky, nebo výrobní haly. Krokem před samotným virtuálním zprovozněním je simulace vyvíjeného stroje, navrhované pracovní linky, nebo rozvržení celé výrobní haly. Testování na úrovni simulace je již v mnoha firmách běžnou činností. Rozdělení simulace na simulaci stroje, linky a haly nebylo zvoleno náhodně. Jedná se o klasické rozdělení podle simulačních nástrojů od společnosti Siemens. Samozřejmě, že existují i jiné nástroje na simulování, ale vzhledem k následnému virtuálnímu zprovoznění se hodí použití právě produktu Siemens. Tyto nástroje jsou již více či méně přizpůsobeny k propojení s PLC. Nástroje pro simulaci se především liší úrovní abstrakce reálného světa. Od nejnižší abstrakce jsou to nástroje Mechatronics Concept Designer, Process Simulate a Plant Simulation.

1.2.1 Simulační nástroje

Pro návrh a následnou simulaci stroje je vhodné použít Mechatronics Concept Designer (MCD), respektive NX. NX je CAD software a MCD je jeho nadstavba. Díky nejnižší úrovni abstrakce reálného světa je možné simulovat i fyzikální působení mezi součástkami. Toho je dosaženo použitím herního jádra, které umožňuje dané výpočty provést. MCD má možnost simulovat fyzikální vlastnosti pevných nedeformovatelných těles, což znamená, že lze simulovat působení síly, tření, kolizí a následných reakcí. Jinak řečeno MCD kromě simulace kinematiky nabízí i možnost simulovat dynamiku. To je při návrhu nového stroje vcelku užitečným zjednodušením a zrychlením práce. Vzhledem k většímu počtu nutných výpočtů při simulaci není MCD vhodným řešením pro realizaci větších projektů. Za větší projekt se dá považovat výrobní linka.

Pro simulace výrobních linek a robotických buněk se hodí Process Simulate (PS). Tento nástroj již není vystavěn nad herním jádrem. Je zde použita vyšší úroveň abstrakce, která neobsahuje prostředky pro modelování vzájemného fyzikálního působení těles. Modely v Process Simulate mají svoji kinematiku. Dynamika je zde zastoupena pouze časem, za který se daný pohyb stane, nebo jeho rychlostí. U některých modelů lze ovlivnit i maximální zrychlení. Tato úroveň abstrakce umožňuje zároveň simulovat více strojů najednou. To se hodí především pro plánování a následnou simulaci výrobních linek. V tomto případě nás tedy především zajímá, jak stroje kooperují, nikoliv jak každý jednotlivý stroj přesně funguje.

Velkým přínosem Process Simulate je integrace robotických kontrolérů. Díky nim je snadné simulovat robotické buňky. Roboti v takové simulaci jsou řízeni téměř stejně jako jejich reálný vzor. Samotný Process Simulate nabízí možnost použití defaultního kontroléru. Ten je schopný vyřešit kinematickou úlohu i pro složitější roboty. Výsledkem použití oficiálního kontroléru výrobce robotů je v podstatě hotový robotický program. Ten pak stačí pouze do reálného robota nahrát a není nutné program psát přímo u robota a následně ho pak zdlouhavě ladit. Další integrovanou funkcí je simulace lidské práce. Zde je možné ověřit nejen proveditelnost dané operace, ale i zatížení daného pracovníka a případné následky na jeho zdraví. Pro virtuální zprovoznění nabízí Process Simulate nejjednodušší

propojení s PLC.

Poslední úrovní abstrakce simulace je Plant Simulation. Tento nástroj je určen k simulaci celé výrobní haly. Objekty již nejsou simulovány svojí kinematikou, ale četností výskytu jednotlivých událostí v daném časovém intervalu. V Plant Simulation lze simulovat i dlouhodobý běh linky za zlomek času. Tím přináší cenné statistické informace potřebné k optimálnímu rozvržení výrobní linky a také pro optimalizaci výrobního plánu.

Z každého nástroje bez ohledu na to, jak velká abstrakce byla použita, lze pro zjednodušení práce použít i výsledky simulace. Přínosem je samozřejmě například hotový robotický program, optimalizace výroby nebo ověření konceptu fungování nového stroje. Tyto výsledky ovšem nezjednodušují přímo zprovoznění dané linky.

1.2.2 Princip virtuálního zprovoznění

Běžné zprovoznění ve smyslu oživení například robotické buňky probíhá až na hotové stavbě. Vzhledem k ověřování a oživení PLC programu je to poměrně pozdě. Programátor musí program ladit přímo na místě, kde má továrna stát. Je to tedy zdlouhavý proces s prací mimo kancelář, což je pro zaměstnavatele dražší, než kdyby programátor mohl program odladit už v kanceláři. Samozřejmě i pro samotného programátora bude výhodnější si většinu věcí ověřit ještě před výjezdem na stavbu. Pro propojení simulace a PLC programu je vhodné použít virtuální zprovoznění. To znamená, že reálný PLC program neřídí skutečnou linku, ale pouze její simulaci.

1.2.3 Propojení simulace a PLC

Propojení simulovaného světa s PLC je možné několika způsoby. Pro připojení reálného PLC k simulačním nástrojům lze použít například OPC. Tato varianta je použitelná de facto pro libovolné PLC, které se dá připojit k PC. OPC s sebou ovšem nese i několik nevýhod. Tou hlavní je nutnost změny projektu pro PLC. Není možné simulovat periferie připojené k PLC, což s sebou nese nutnost změny HW konfigurace PLC. S touto změnou souvisí i nutnost přemapování vstupů a výstupů do vnitřní paměti PLC. Tyto změny s sebou nesou riziko zavlečení chyby a virtuální zprovoznění není poté obrazem toho reálného.

Zároveň odpadá možnost testování safety části programu.

V případě použití PLC komunikujícího po Profinetu, nebo Profibusu lze použít Simba box. Zde se budu nadále věnovat variantě s Profinetem, u Profibusu je to podobné. Simba je zařízení, které na Profinetové síti simuluje kompletní zařízení. Tzn. z pohledu PLC je jedno, jestli bude komunikovat s reálným zařízením, nebo pouze se simulovaným zařízením pomocí Simba boxu. Díky simulaci i zařízení, které se mají připojit k PLC, není nutné měnit HW konfiguraci v PLC projektu. Další výhodou Simba boxu je podpora Profisafe. To přináší výhodu ponechat PLC program tak, jak má být použit s reálnou linkou a otestovat ho celý včetně funkční bezpečnosti (safety). Simulace zařízení pomocí Simba boxu probíhá pouze na úrovni Profinetu jako takového. To znamená, že každé zařízení je simulováno jako I/O prvek. Funkcionalitu zařízení je třeba simulovat v nástrojích, ve kterých je simulován reálný svět. Simba box je nejlépe propojitelný s Process Simulate. Zde existuje přímé spojení, a je to pro řádné virtuální zprovoznění nejlepší řešení. K MCD a Plat Simulation lze Simbu připojit přes Simit. Sice je zapotřebí další software, ale Simit nabízí lepší možnosti simulace chování HW zařízení.

Poslední podporovanou variantou propojení je spojit simulační nástroj se simulovaným PLC. Ve vývojových prostředích Step7 v5.5 a TIA Portal je možné simulovat PLC. Toto simulované PLC se pak dá přímo propojit např. s Process Simulate. Bohužel zde chybí podpora safety. To by se ovšem mělo změnit v novém simulátoru v TIA Portalu. PLC Sim Advance je schopný simulovat i safety prvky a jeho ambice jsou nahradit SIMBA box a celé virtuální zprovoznění provést bez HW prvků.

Účelem virtuálního zprovoznění je tedy odladit celý koncept, např. robotické linky. K tomu je zapotřebí nejen znalost simulačních nástrojů, ale i programování pro PLC. Virtuální zprovozňovatel propojí oba světy a snaží se odladit všechny chyby vzniklé buď nepochopením specifikací, nebo jejich špatnou implementací. Samozřejmě k tomu, aby mohl odhalit a opravit chyby v PLC programu, potřebuje programátor takový program co nejlépe zanalyzovat. K analýze programu je možné použít právě nástroj, který je předmětem této práce.

2 PLC

Programovatelný logický automat PLC je v podstatě průmyslový počítač určený k řízení stroje nebo linky v reálném čase. Oproti běžnému PC se liší především zpracováváním programu v cyklech. Kromě fungování v cyklech se PLC liší svým přizpůsobením pro zpracování průmyslových signálů, tzn. digitální a analogové vstupy a výstupy. Historicky měly původní PLC nahradit řídicí zapojení s reléovou logikou. V dnešní době toho díky výkonným procesorům zvládají daleko více. V PLC se projevil stejný trend jako u osobních počítačů, a to použití vícejádrových procesorů. Proto není problém na dnešních PLC v podstatě bez problému provozovat WEB server, OPC server, VNC. Některá PLC dokonce umožňují běh normálním operačním systémům, například Windows 7 embedded. Samozřejmě, že si PLC stále zachovává původní úkol, kterým je řízení výrobního procesu nebo pracovního stroje. Na PLC je z tohoto ohledu kladen velký nárok na stabilitu a bezpečnost programu.

PLC se dá rozdělit podle struktury své stavby na kompaktní a modulární. U kompaktního PLC je vše v jednom zařízení. Procesor, vstupy, výstupy jsou umístěny v jednom zařízení a jeho možnosti rozšíření jsou velmi omezeny. Modulární systém se skládá z jednotlivých částí, které zastupují určité funkce systému jako například modul s CPU, vstupně výstupní modul nebo jiné speciální funkce. Tyto moduly je poté třeba propojit. Z hlediska moderního návrhu řídicího systému u větších projektů je modulární řešení výhodné. Takový řídicí systém je postaven přímo na míru konkrétní aplikaci. Dále je možné vstupně výstupní modul umístit na jiné místo než je samotné CPU. To samozřejmě snižuje množství nutných vodičů. Vstupně výstupní modul je možné umístit co nejblíže zdroji vstupu, nebo potřebným akčním členům. Od I/O modulu do PLC poté vede pouze vodič některé z průmyslových sběrnic, například PROFINET, PROFIBUS, POWERLINK ...

PLC cyklus je proces zpracování kódu. Na začátku každého cyklu je načtení vstupů. Tato hodnota je uložena do paměti PLC, čímž je zaručena její neměnnost po dobu zpracovávání programu. Poté následuje zpracování samotného programu. Během vykonávání programu se výstupní hodnota ukládá pouze do paměti PLC. Výstupy jsou zapsány na konci cyklu. Použitím cyklického zpracování programu je zajištěna bezpečnost při vykoná-

vání programu. Nemůže se stát, že by pro polovinu programu měly vstupy jiné hodnoty, než pro jeho zbytek. Tato nesrovnalost by mohla mít za následek havárii celého programu, a tím pádem konec řízení. V některých aplikacích by taková situace mohla mít katastrofální následky.

2.1 Programovací jazyky

PLC se dá programovat hned v několika jazycích. Většina PLC má implementovány jazyky, které podléhají normě IEC EN 61131-3. Podle této normy je možné použít následující programovací jazyky.

Ladder Diagram (LD) je grafický jazyk, který nahrazuje reléové schéma. Tento jazyk je velmi populární. Vzhledem k tomu, že původní funkcí PLC bylo nahradit právě zapojení, která byla realizována pomocí relé, není se čemu divit. I v dnešní době je Ladder Diagram velmi populární a rozšířený jazyk. Je vcelku jednoduchý na pochopení. Jedná se o nejnižší programovací jazyk z normy. Na úkor jednoduchosti a grafické reprezentace nejsou všechny úkony jednoduše proveditelné. Na většinu úkonů i přesto tento jazyk stačí.

Function Block Diagram (FBD) je též grafický jazyk. Je velmi názorný a jeho použití je poměrně jednoduché a rychlé. V podstatě jde o spojování funkčních bloků. FBD se hodí pro rychlou a jednoduchou tvorbu programu na úrovni konceptu. Celý program je možno poskládat v podstatě z již předpřipravených funkcí a funkčních bloků. FBD je velmi abstraktní a na rychlé pochopení jednoduchý.

Instruction List (IL) je textový jazyk na nejnižší úrovni. Díky textové podobě lze pomocí tohoto jazyku vytvářet i poměrně složité algoritmy. To je ovšem na úkor čitelnosti a srozumitelnosti takového kódu.

Structured Text (ST) je programovací jazyk vyšší úrovně. Tento jazyk umožňuje velmi efektivně vytvořit i složitější algoritmy. Hodí se především pro tvorbu jednotlivých funkčních bloků. Tyto bloky je poté jednoduché použít v jiné části programu. Nejjednodušší použití je v FBD, samozřejmě je takové bloky možné použít i v jiných programovacích jazycích.

Sequential Function Chart (SFC) je grafický programovací jazyk, který je přímo určený

pro tvorbu sekvenčních automatů. Stavový automat se v tomto programovacím jazyce vytváří velmi jednoduše. SFC je založen na principu akce, která je vykonávána v určitém stavu, a přechodu mezi stavy.

U některých výrobců PLC se programovací jazyky mohou lehce lišit. Jsou jinak pojmenované, nebo je lehce odlišná syntaxe. Velký rozdíl se také dá najít mezi grafickými reprezentacemi. Základní myšlenka daného programovacího jazyka ovšem zůstává stejná u každého výrobce PLC.

Nástroj vytvořený v této práci je určen pro PLC od společnosti Siemens, respektive pro programy psané v prostředí Step7.

3 Nástroj

Celý nástroj je naprogramován v Pythonu. Python je skriptovací programovací jazyk, mezi jehož přednosti patří především jednoduchost na naučení a pevná syntaxe. Tou je zaručena vysoká čitelnost kódu. Jelikož se jedná o skriptovací jazyk, je zapotřebí k běhu interpret, který je dostupný pro každý běžný operační systém. Díky tomu je možné de facto stejný program pustit na libovolném počítači. Potřebné změny v takovém multiplatformním systému jsou pouze v části programu, který je přímo závislý na operačním systému. V případě nástroje na tvorbu BDD je to práce se soubory. Operační systémy na bázi Microsoft Windows a UNIX se liší v cestě k daným souborům, implementace proběhla pouze pod OS Linux. Vývojová prostředí pro programování PLC bývají většinou dostupná pouze pro OS Windows. Je tedy v plánu nástroj rozšířit o podporu dalších operačních systému, především pak MS Windows. Rozšiřitelnost na jiné operační systémy byl i jedním z důvodů volby Pythonu jako jazyka pro vývoj celého nástroje. Dalším důvodem je velká rozšířenost a popularita Pythonu. Díky tomu existuje velké množství knihoven a hotových řešení. To se týká například i tvorby BDD z logického výrazu. Knihovna, kterou jsem použil, je dostupná na [2].

Program je dělen na několik částí. První je převod PLC programu na formu, se kterou je možné dále pracovat. Jedná se o parser textu na jednotlivé logické výrazy s rozšířením o časovače. Druhá část se zabývá převodem logického výrazu na BDD. Ten je poté ve

formě obrázku uložen do příslušného adresáře. V této části je i řešení slučování jednotlivých networků. V poslední části je řešen převod programu na stavový automat. Stavový automat je vybudován z pohledu hodnoty proměnné na každém řádku. Z takového automatu lze snadno určit, jakou hodnotu by daná proměnná měla mít v každé části programu. Ve stavovém automatu jsou narozdíl od BDD lépe implementovány i časovače.

4 Parser STL

Při tvorbě nástroje pro tvorbu BDD a stavového automatu z PLC programu jsem se rozhodl pro podporu programovacího jazyka Ladder Diagram, respektive Instruction List. Vzhledem k tomu, že nástroj podporuje PLC Siemens Simatic, jedná se o jazyk STL. V programovacím nástroji Step7 je možné generovat zdrojové kódy pro program napsaný v LD. Této možnosti jsem využil, neboť LD je stále velmi populární a rozšířený programovací jazyk a bylo tedy vhodné začít s podporou právě u něho.

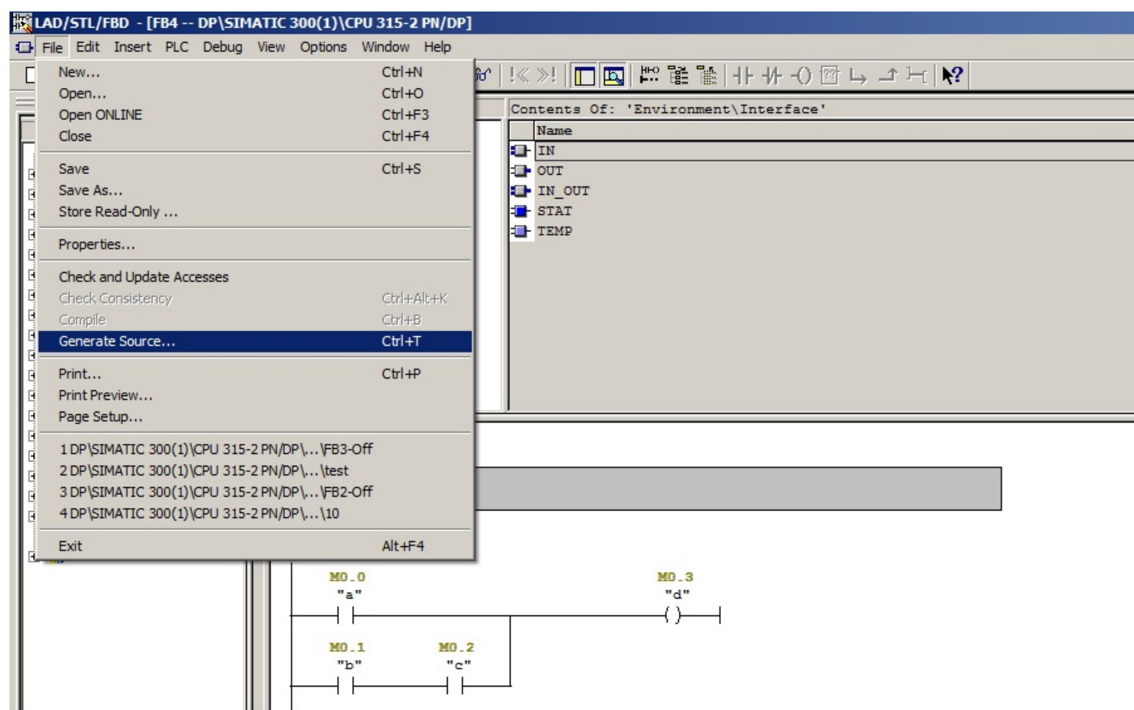
Zdrojový kód pro LD je ve Stepu převeden automaticky do instruction listu, respektive STL. Toto je velmi výhodné. STL je dobře popsán a má pevně stanovenou strukturu. Toho se dá využít pro počítačové zpracování PLC programu. U zápisu jiného jazyka by bylo nutné nejprve určit, jak je daný program ukládán. Tento problém mají především grafické programovací jazyky. Ty si v podstatě každý výrobce PLC může ukládat ve svém formátu. Tento formát ve většině případů nebude mít veřejně přístupnou dokumentaci, možná nebude ani možné soubor z projektu exportovat. Určitou možností je použití PLCopen XML, což je univerzální způsob zápisu programů pro PLC, který již někteří výrobci začínají podporovat. Jeho využití pro účely této práce je námětem na její další rozšíření. Díky možnosti exportu programu psaném v Ladder Diagramu přímo do zdrojového kódu ve tvaru STL je automatické načtení programu realizovatelné.

Automatický převod PLC programu na logické výrazy je největším přínosem této práce. Jako vstupní soubor je použit automaticky vygenerovaný textový soubor. Tento soubor obsahuje zdrojové informace k programu napsaném v Ladder Diagramu. Tento program je automaticky v prostředí Step7 převeden do jazyka STL, který víceméně odpovídá standardnímu jazyku IL dle IEC 61131-3. Textový soubor zachovává členění po jednotlivých

networkcích jako je tomu u LD. Toto dělení je zachováno i pro převod na logické výrazy.

4.1 Export zdrojových souborů

Prostředí Step7 nabýzí možnosti okamžitého přepnutí zobrazení programu v LD, STL, nebo FB. Využitá funkce pro tuto práci je automatické generování zdrojových souborů. Generátor zdrojových souborů se nachází v *Menu: File - Generate Source ... (Ctrl + T)* - viz obrázek 1

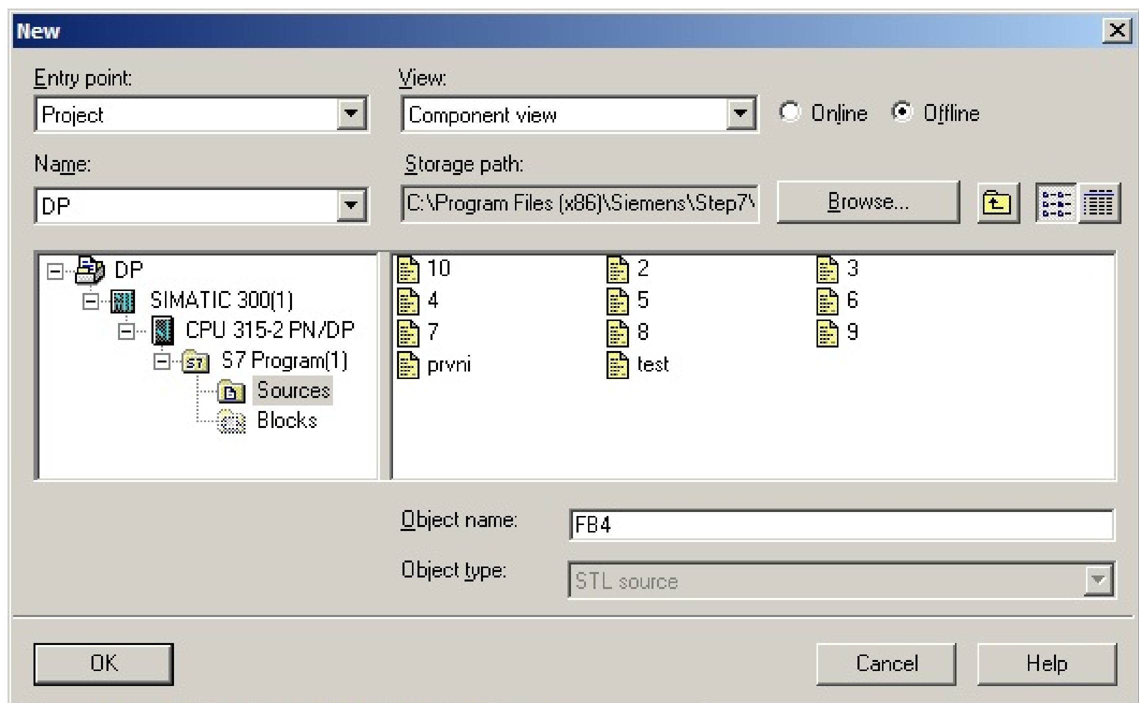


Obrázek 1: Menu: Spuštění nástroje pro generování zdrojového kódu

Po spuštění nástroje pro generování zdrojového kódu se otevře okno, kde je možné vytvořit nový zdrojový soubor, nebo pro generování použít již existující - viz obrázek 2

Po vybrání souboru, do kterého se má zdrojový kód uložit, je nutné zvolit, z čeho se má kód vygenerovat. V následujícím okně (viz obrázek 3) je tedy nutné vybrat zdroj pro generování.

Po vygenerování kódu je daný soubor dostupný v architektuře projektu ve složce *Source*. Tento soubor se dá přímo ve Stepu7 otevřít a případně i editovat. Zdrojový kód lze z editoru

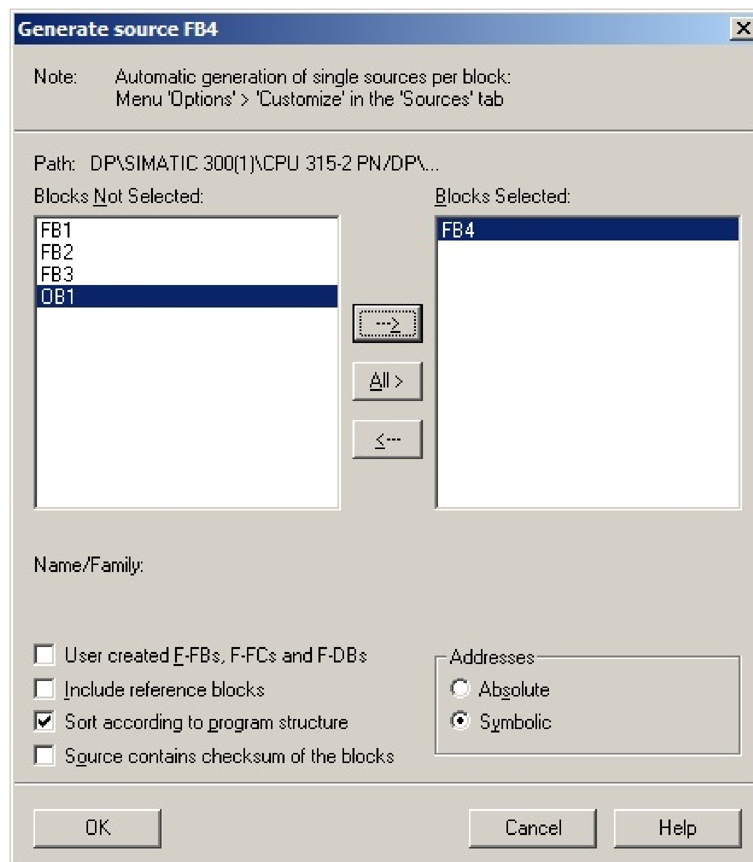


Obrázek 2: Výběr zdrojového souboru

ve Step7 kopírovat a následně uložit do libovolného textového souboru a ten poté nahrát do nástroje pro automatickou tvorbu BDD a stavového automatu. Druhá možnost je přímo použít vygenerovaný soubor, který se nachází ve složce projektu cestak k souboru. V mém případě je:

```
Siemens\Step7\S7Proj\Dp\s7asrcom\00000001\xxxxx.AWL
```

Místo *xxxx.AWL* je číslo zdrojového souboru. Soubory jsou číslovány podle stáří vzniku. Nejnovější soubor má nejvyšší číslo v hexadecimální formě zápisu.



Obrázek 3: Zdroj pro generování

4.2 Princip převodu programu na logické výrazy

Jak již bylo řečeno, pro převod PLC programu na logické výrazy, je využito dělení programu na jednotlivé networky. Z textového souboru jsou nejprve odstraněny všechny nepotřebné řádky. To, co v takovém souboru poté zůstane, jsou pouze řádky network a pod ním STL přepis daného řádku. Soubor je následně rozdělen na jednotlivé networky. Každý network (řádek) je poté předán části nástroje, která se stará přímo o převod na logické výrazy.

V PLC programu se mohou vyskytovat libovolné proměnné. Nástrojem jsou podporovány jak symbolické adresy například $c = M0.2$, tak přímo adresy bez symbolického názvu. U adres bez symbolického názvu bylo nutné změnit značení. Jelikož nástroj pro kreslení grafů [3] nepodporuje mezery, tečky a jiné speciální znaky, je zavedeno následující značení.

Každá mezera je nahrazena znakem _. Struktura *Q 0.0* říká, že se jedná o nultý bit v nultém bytu na výstupním rozsahu adres. Pro nástroj je tento zápis pozměněn na *Q_ 0[0]*. Číslo v závorce *[]* je pořadí bitu v daném bytu, ten je reprezentován číslem před závorkou. Takže například *I 10.5* je v nástroji zapsáno jako *I_ 10[5]*.

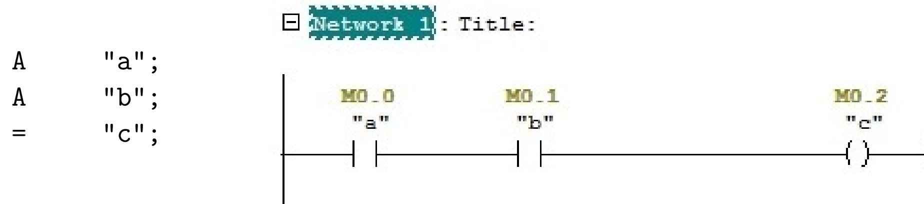
Obrázek 4: Zdrojový soubor

4.3 Podporované příkazy STL

Mezi hlavní podporované operace patří především logické operace. S logickými operacemi není nutné provádět žádné dodatečné změny, neboť je lze rovnou převést na logický výraz. Podporované logické operace jsou:

4.3.1 AND

Logická operace *AND* se v jazyce STL zapíše jako:



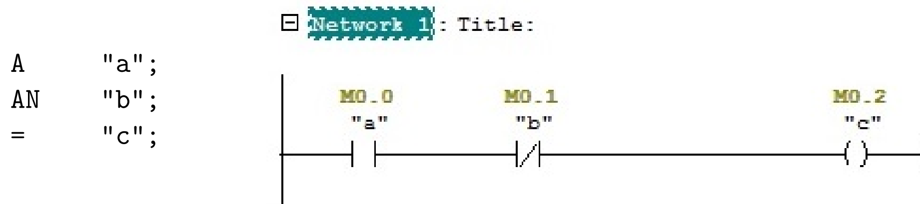
Obrázek 5: Logická operace *AND*

Přepis na logický výraz poté vypadá jako:

$$c = a \ \& \ b$$

4.3.2 AND not

Obdobou logické operace *AND* je operace *AND not*.



Obrázek 6: Logická operace *AND not* v LD

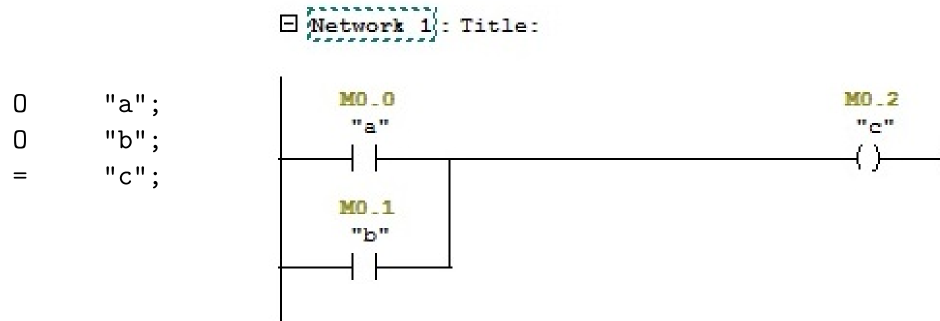
AND not se v logickém výrazu zapíše jako:

$$c = a \ \& \ \sim b$$

Kde $\sim$ je znak pro negaci.

4.3.3 OR

Další z podporovaných operací OR.



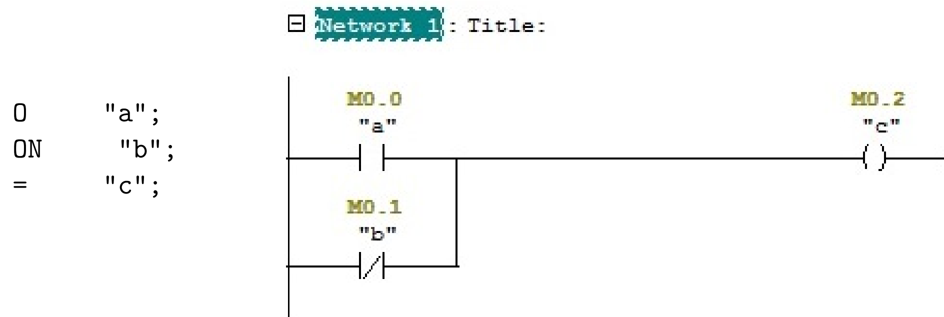
Obrázek 7: Logická operace OR v LD

Přepis do logického výrazu poté má tvar:

$$c = a \mid b$$

4.3.4 OR not

Samozřejmě, jako je tomu u AND, existuje u OR jeho negovaná podoba.



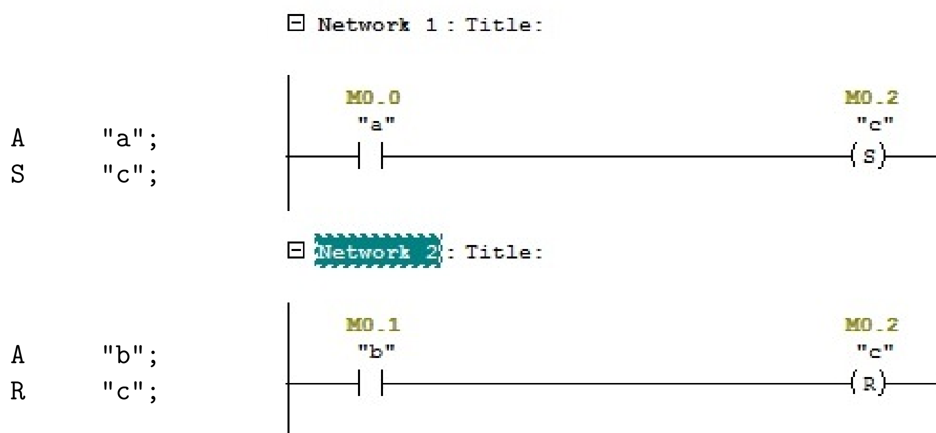
Obrázek 8: Logická operace OR *not* v LD

Přepis do logického výrazu vypadá následovně:

$$c = a \mid \sim b$$

4.3.5 Set a Reset

Pro operace *Set* a *Reset* bylo nutné vymyslet jednoduchou formu zápisu, která by se dala interpretovat jako přiřazení. Interperetace přiřazením je důležitá pro následnou tvorbu BDD. Proto byly operace *Set* a *Reset* implementovany přímo jako přiřazení, kdy je ale místo proměnné c použita proměnná *Set* c respektive *Reset* c. Výraz za $=$ poté není to, co má být do proměnné přiřazeno, ale podmínka pro danou operaci.



Obrázek 9: Operace *Set* a *Reset*

Přepis do logického výrazu vypadá poté následovně:

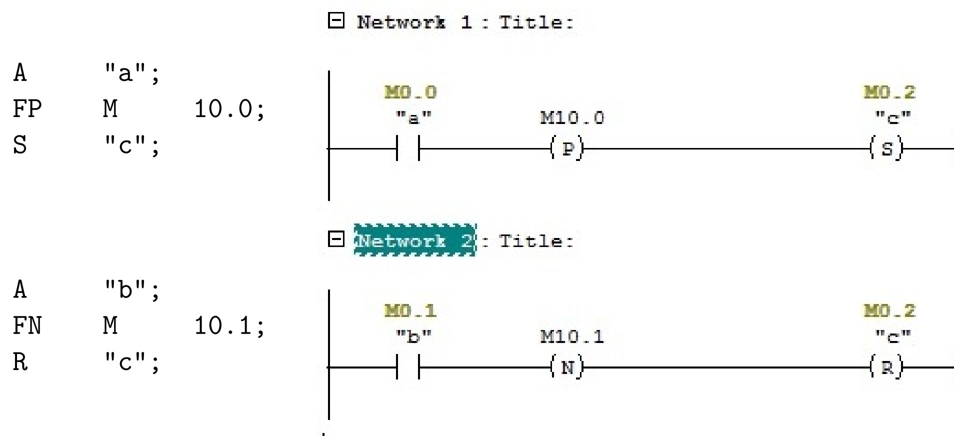
$$Set\ c = a$$

$$Reset\ c = b$$

4.3.6 Náběžná a sestupná hrana

Další příkazy, které byly do nástroje implementovány, jsou náběžná a sestupná hrana. U těchto operací bylo znovu nutné navrhnout, jak je převést na logické výrazy s přiřazením. To jsem nakonc vyřešil přidáním logické proměnné reprezentující operaci. De facto to odpovídá způsobu, jakým je detekce náběžné respektive setupné hrany zakreslena v LD. Nejprve je proměnná, která má být použíta pro detekci hrany. Samozřejmě, že je možné použít i

kombinaci proměnných. Poté následuje operace pro detekci hrany. Ta má velmi podobný způsob zakreslení jako operace *AND*. Tato podobnast byla využita a zápis detekce hrany je tedy logickým součinem výrazu před operací detekce hrany a proměnné, která reprezentuje detekci hrany.



Obrázek 10: Detekce náběžné a sestupné hrany

Přepis do logického výrazu vypadá poté následovně:

$$Set\ c = a\ \&\ RE_M10[0]$$

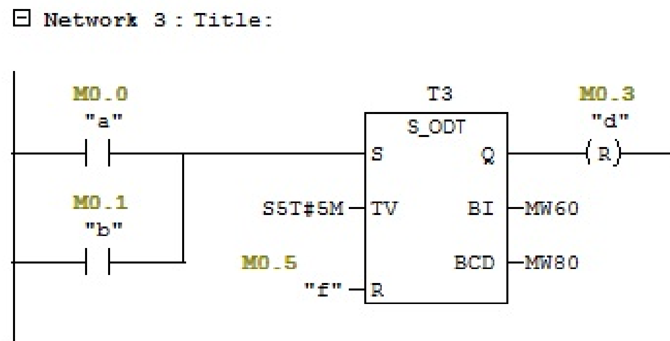
$$Reset\ c = b\ \&\ FE_M10[1]$$

4.4 Časovače

Mezi další podporované operace patří časovače. Při integraci časovačů jsem využil podobného postupu jako u integrace detekce náběžné a sestupné hrany. Každý časovač je v logickém výrazu reprezentován jako speciální proměnná. K této proměnné je přidán i výraz pro reset časovače. Ten se následně v BDD sice nezobrazuje, protože daný popis proměnné by byl moc dlouhý, ale je použit ve stavovém automatu.

4.4.1 ODT

```
A(      ;
O       "a";
O       "b";
)       ;
L       S5T#5M;
SD      T       3;
A       "f";
R       T       3;
L       T       3;
T       MW     60;
LC      T       3;
T       MW     80;
A       T       3;
R       "d";
```



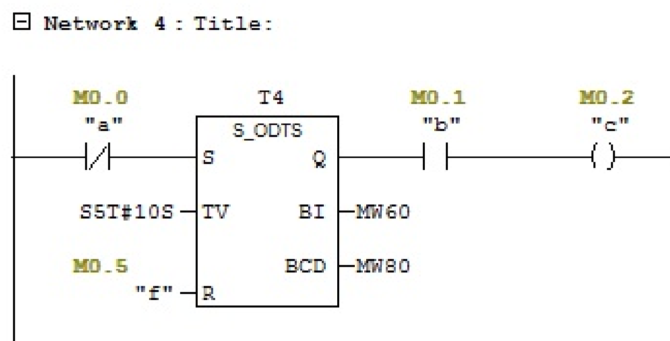
Obrázek 11: Zapojení časovače ODT

Přepis do logického výrazu vypadá poté následovně:

$$Reset\ d = (a \mid b)\ \&\ SD_T3_5M$$

4.4.2 ODTS

```
A(      ;
AN      "a";
L       S5T#10S;
SS      T       4;
A       "f";
R       T       4;
L       T       4;
T       MW     60;
LC      T       4;
T       MW     80;
A       T       4;
)       ;
A       "b";
=       "c";
```



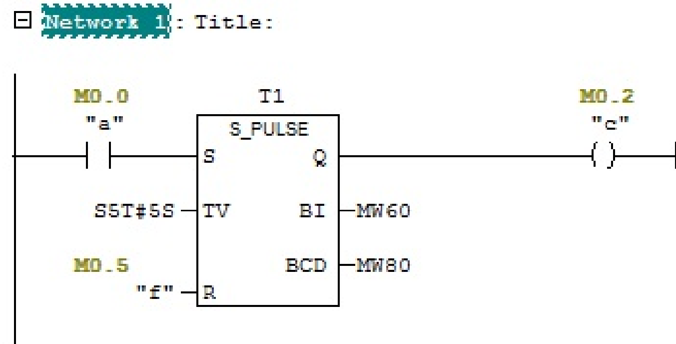
Obrázek 12: Zapojení časovače ODTS

Přepis do logického výrazu vypadá poté následovně:

$$c = \sim a\ \&\ SS_T4_10S\ \&\ b$$

4.4.3 PULSE

```
A      "a";
L      S5T#5S;
SP     T       1;
A      "f";
R      T       1;
L      T       1;
T      MW     60;
LC     T       1;
T      MW     80;
A      T       1;
=      "c";
```



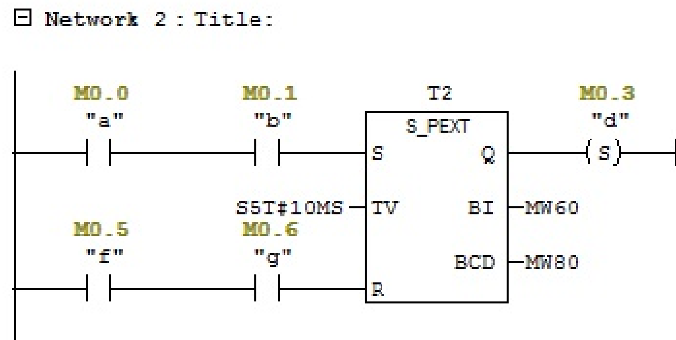
Obrázek 13: Zapojení časovače PULSE

Přepis do logického výrazu vypadá poté následovně:

$$c = a\ \&\ SP_T0_5S$$

4.4.4 PEXT

```
A      "a";
A      "b";
L      S5T#10MS;
SE     T       2;
A      "f";
A      "g";
R      T       2;
L      T       2;
T      MW     60;
LC     T       2;
T      MW     80;
A      T       2;
S      "d";
```



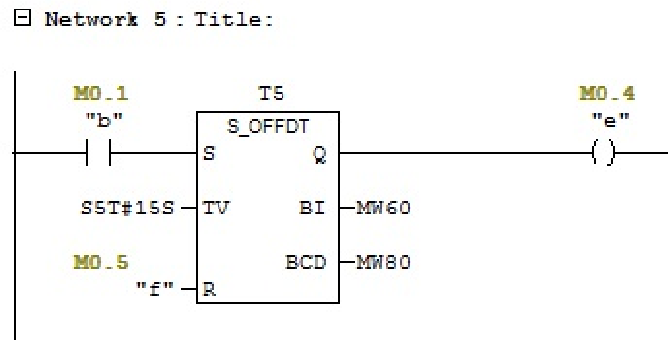
Obrázek 14: Zapojení časovače PEXT

Přepis do logického výrazu vypadá poté následovně:

$$Set\ d = a\ \&\ b\ \&\ SE_T2_10MS$$

4.4.5 OFFDT

```
A     "b";
L     S5T#15S;
SF    T      5;
A     "f";
R     T      5;
L     T      5;
T     MW    60;
LC    T      5;
T     MW    80;
A     T      5;
=     "e";
```



Obrázek 15: Zapojení časovače OFFDT

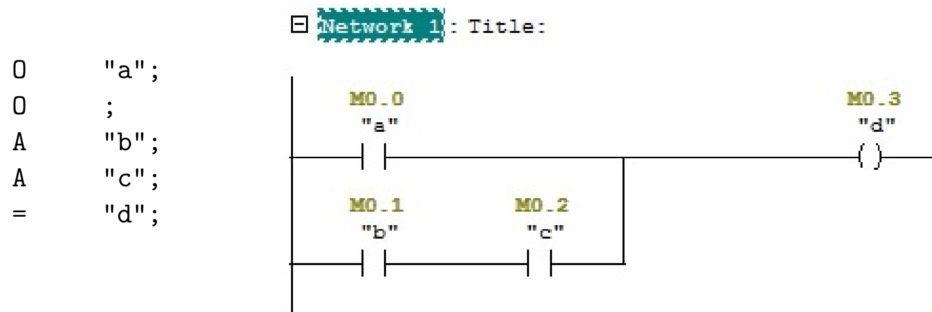
Přepis do logického výrazu vypadá poté následovně:

$$e = b \ \& \ SF_T5_15S$$

4.4.6 Závorky

Narozdíl od jednotlivých příkazů je převod závorek mnohem složitější záležitost. Závorky v programu vznikají větvením v Ladder Diagramu. Takové větvení není nic neobvyklého a bylo tedy nutné závorky imlementovat.

V jazyce STL je několik možností jak psát závorky. Použití daného způsobu záleží na tom, co za závorkou následuje. Jde- li o závorku kvůli složitějšímu výrazu v logickém součtu, jako například na obrázku 16 , tento program pak má následující přepis do STL:

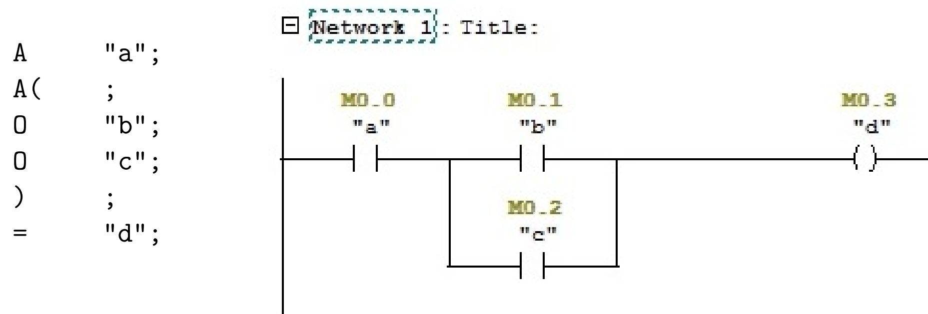


Obrázek 16: Větvení programu vyžadující závorky pomocí *OR*

Přepis do logické funkce vypadá následovně:

$$d = a \mid (b \;\&\; c)$$

Druhou možností jak otevřít závorku je pomocí operace *AND*. Takový příklad je na obrázku 17. Tento program pak má následující přepis do STL:



Obrázek 17: Větvení programu vyžadující závorky pomocí *AND*

Přepis do logické funkce vypadá následovně:

$$d = a \;\&\; (b \mid c)$$

5 BDD

Binary Decision Diagram je způsob, jak zakreslit logický výraz pomocí grafu. Jedním ze způsobů, jak takový diagram sestavit, je vyjít z binárního stromu. Binární strom obsahuje všechny možné kombinace proměnných. Jde v podstatě o grafické zobrazení tabulky všech možných kombinací. Pro snadnou analýzu se binární strom pro svoji rozsáhlost moc nehodí. Proto se binární strom zjednodušuje postupným slučováním. BDD je tedy v podstatě sloučený binární strom. V BDD se každá proměnná vyskytuje pouze jednou. Diagram je poté mnohem jednodušší pro snadnou analýzu.

5.1 Použití při analýze PLC kódu

Pro jednoduchou a rychlou analýzu PLC programu se dá použít Binary Decision Diagram (BDD). Pro tvorbu takového diagramu je zapotřebí vycházet z logického výrazu, a tak je nutné nejprve program převést na logické výrazy. Každý network v Ladder Diagramu se dá považovat za řádek programu, který odpovídá zakreslení příslušného reléového schématu. Tento způsob programování je grafickým znázorněním logického výrazu a je ho tedy třeba převést do podoby, ze které se dá BDD postavit. Více o tom pojednává část věnující se exportu programu do vytvořeného nástroje a jeho následného převodu na logické výrazy.

Program, který je možné převést do BDD, může obsahovat logické fukce AND, NAND, OR, NOR a NOT. Dále je implementováno rozšíření na náběžnou a sestupnou hranu a v programu se mohou vyskytovat časovače.

Logické výrazy vzniklé převodem jednotlivých řádků PLC programu (viz výše) se zobrazí jako BDD. Použitá knihovna (viz [2]) podporuje zjednodušování logických výrazů a jejich interpretaci pomocí BDD. Následně lze takto vytvořený BDD zobrazit i graficky (viz [3]).

Z BDD lze poté velmi rychle a snadno přijít na vzájemné závislosti mezi jednotlivými proměnnými, které jsou v programu použity. Vytvoření takového diagramu pouze pro jednu řádku programu by nepřineslo velké výhody. V případě několikanásobného použití proměnných by bylo nutné studovat několik obrázků zároveň. Takovým případem může být například situace, kdy na prvním řádku do proměnné a bude něco uloženo a na

desátém řádku bude tato hodnota použita ve výrazu přiřazeném do proměnné b. Aby bylo možné určit, které proměnné se podílejí na výsledné hodnotě b, bylo by třeba nejprve projít diagram pro proměnnou a, tento výsledek si pamatovat a následně ho ručně aplikovat do analýzy diagramu pro b.

Pro lepší použití nástroje je implementována funkce, která na příslušných místech nahradí proměnnou a za výraz, který byl do proměnné přiřazen. Výsledný BDD pro proměnnou b obsahuje tedy i výraz pro a. Tím je docíleno úplného vykreslení závislostí všech proměnných, které se podílejí na výsledné hodnotě proměnné b. Velmi rychle a jednoduše se díky tomu dá říci, jaká musí být splněna podmínka, aby daná proměnná nabývala hodnotu logická jedna nebo nula.

Toto se hodí například při virtuálním zprovoznění. V situaci, kde je třeba na výstupu nastavit určitou hodnotu pro odblokování pohybu je třeba zjistit, jaká musí být kombinace na vstupu, aby výstup nebyl blokován. Jedná se například o blokování robotů, respektive splnění podmínek pro začátek práce. Jednou z takových podmínek může být dostatečně vysoký tlak vzduchu. Většina těchto start podmínek není v simulačních nástrojích implementována a virtuální zprovozňovatel si je musí sám nastavit. V případě, že sám není autorem PLC programu, potřebuje rychle a efektivně zjistit potřebnou kombinaci vstupů pro uvolnění výstupu.

Dalším přínosem BDD je automatické zjednodušení logického výrazu, ke kterému dojde ještě před vykreslením diagramu. Z takového grafu je poté snadné určit, jestli jsou všechny operace relevantní. Jinak řečeno, může se stát, že tímto způsobem dojde k odhalení zbytečných částí programu. Příkladem je třeba použití operace OR, kde jeden z operandů je vždy pravdivý. V takovém případě je druhý operand, respektive celý OR, zbytečný. Program tedy poté na základě prostudování BDD může být upraven. Upravením dojde ke zjednodušení celého programu a zároveň se zvýší čitelnost takového kódu.

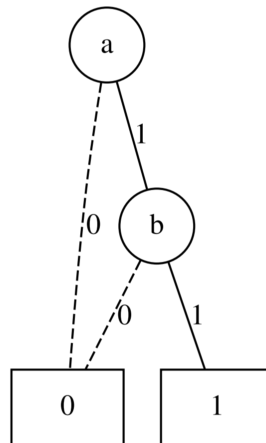
5.2 BDD pro jednotlivé logické operace

V této kapitole je názorně zobrazeno, jak BDD vypadá pro jednotlivé elementární logické operace. Pro většinu operací, které jsou popsány v kapitole 4.3 o podporovaných operacích

jazyka STL, je vytvořen jednoduchý graf. Je samozřejmé, že pro takto jednoduché příklady by BDD nedávalo velký smysl. Jde pouze o ukázkové příklady.

Každý BDD, který je pomocí nástroje vygenerován, je následně uložen. Jméno souboru je tvořeno podle následujícího klíče. Na prvním místě stojí, ze kterého networku (řádku) je daný výraz vytvořen. Druhá část názvu je samotná proměnná, do které se přiřazuje. Například : *Network 1 c.svg*. Pokud se nejedná o přiřazení, ale o operace *Set* a *Reset*, je před danou proměnnou klíčové slovo *Set* respektive *Reset*.

5.2.1 AND

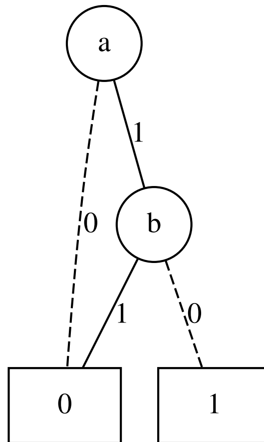


Obrázek 18: BDD pro logickou operaci *AND*(*Network 1 c.svg*)

Z obrázku 18 je na první pohled patrné, že proměnná c nabývá hodnoty log. jedna pouze v případě, že proměnná a je v log. jedna a zároveň proměnná b je v log. jedna. Jedná se o BDD pro část programu, která je popsána v 4.3.1.

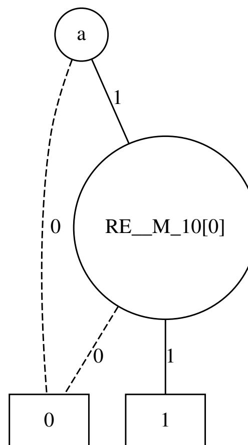
5.2.2 AND not

Na obrázku 19 je vidět, jak vypadá BDD pro operaci *AND not*. Proměnná c je v logické jedničce v případě, že a je v log. jedna a zároveň b je v log. nula. Jedná se o BDD pro případ popsaný v 4.3.2.



Obrázek 19: BDD pro logickou operaci *AND not*(*Network 1 c.svg*)

5.2.3 Set Reset a detekce náběžné hrany

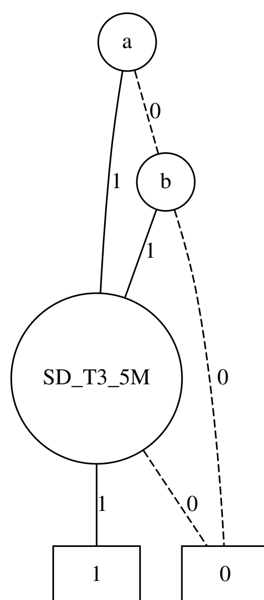


Obrázek 20: BDD pro *Set* a detekci náběžné hrany(*Network 1 Set c.svg*)

Na obrázku 20 je vidět, jak vypadá BDD pro detekci náběžné hrany. Jak již bylo popsáno výše, je využito postupu, kdy detekce je zadefinována jako nová speciální proměnná. Pro náběžnou hranu to je tedy *RE_*. Za _ následuje proměnná, do které je uložena hodnota výrazu před operací z minulého cyklu. Dále se tento graf vztahuje k operaci *Set*. Na první pohled se nijak neliší od běžného přiřazení. Liší se pouze ve jménu automaticky vygenerovaném souboru.

5.2.4 Časovače

Znázornění každého časovače v BDD vypadá téměř stejně. Jediný rozdíl je v písmenném označení daného časovače. Je použito stejného triku jako u detekce hran, tedy přidá se speciální proměnná, která reprezentuje danou funkci. V BDD jsou časovače pouze znázorněny pro informaci, že v kódu časovač je. Z BDD nelze nijak usuzovat o jeho přesném vlivu na danou proměnnou. Způsob, jakým je časovač zapojen lze vyčíst ze stavovéhou automatu (viz níže).



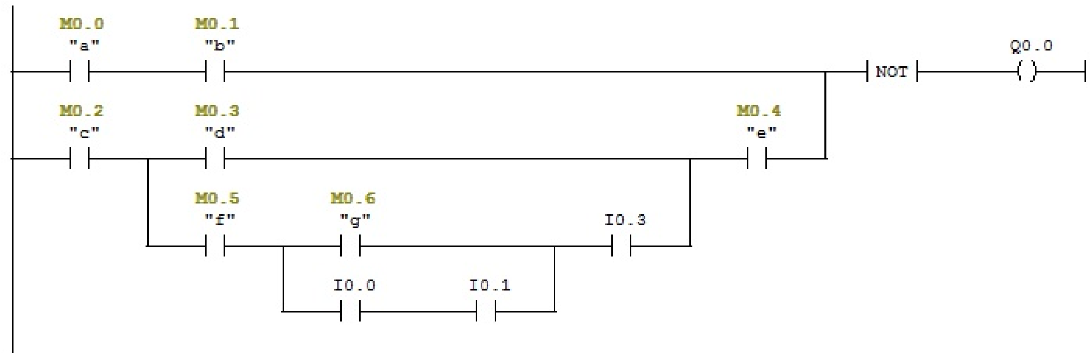
Obrázek 21: BDD pro časovač ODT(*Network 1 c.svg*)

Z BDD pro časovač (obrázek 21) lze zjistit o jaký časovač se jedná, v tomto případě *(SD = ODT)*, který časovač to je (*T3*), a také jeho čas (*5 min*). Z BDD tedy víme, že proměnná c je ovlivněna časovačem zpožděného sepnutí s časem 5 minut.

5.3 Příklady složitějších výrazů

Jednoduché části programu je vcelku snadné převést na BDD. Diagram je poté velmi malý a snadno interpretovatelný. Nástroj se ovšem dá použít i pro složitější struktury příkazů. Takový příklad je na obrázku 22. U takto složitého řádku kódu je již vcelku obtížné určit

všechny kombinace, aby se proměnná *Q0.0* rovnala logické jedničce.



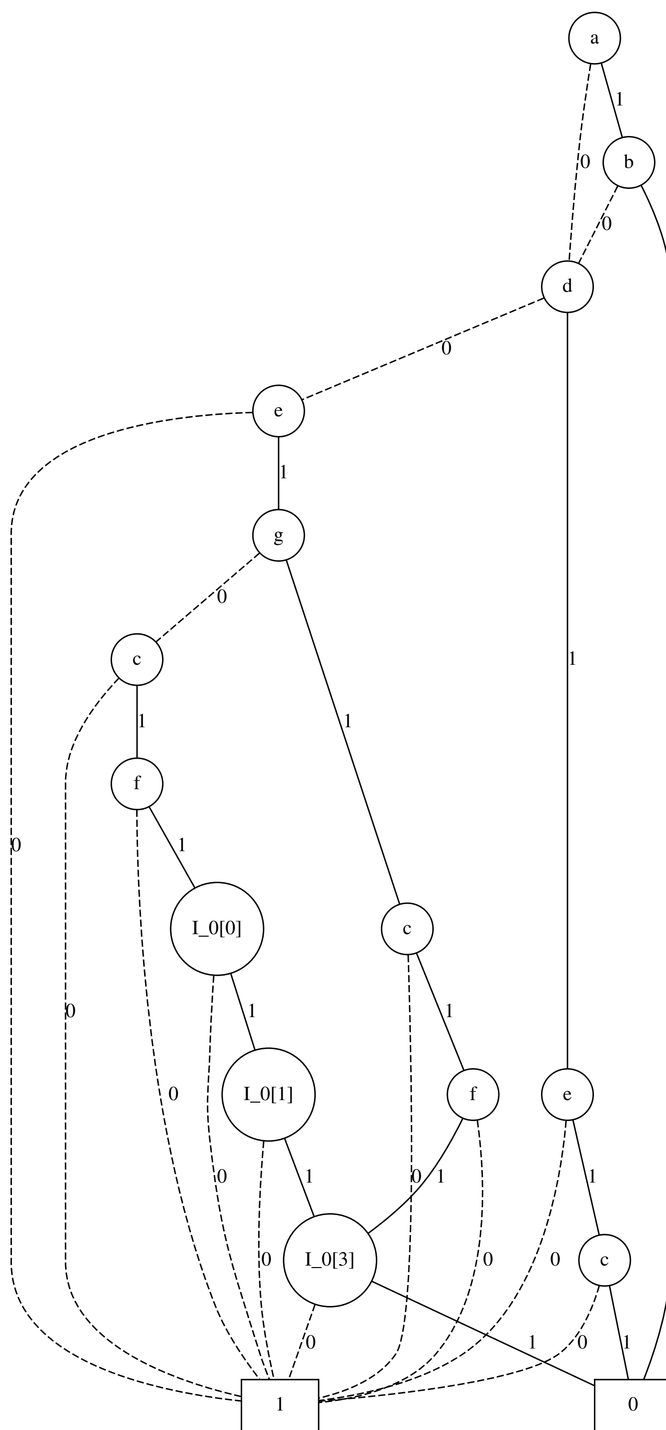
Obrázek 22: Složitější struktura operací

Takováto struktura se pomocí logického výrazu dá zapsat jako:

$$Q_0[0] = \sim (a \ \& \ b \mid (c \ \&(d \mid (f \ \& \ (g \mid (I_0[0] \ \& \ I_0[1])) \ \& \ I_0[3])) \ \& \ e))$$

BDD pro tento komplikovanější případ je na obrázku 23. Z grafu lze snadno a přehledně určit potřebné kombinace pro hodnotu logická jedna na výstupu *Q_ 0[0]*. V tomto případě se jedná například o kombinace:

- $a = 0, \ e = 0$
- $a = 1, \ b = 0, \ c = 0$
- $a = 0, \ d = 0, \ e = 1, \ g = 0, \ c = 1, \ f = 1, \ I_0[0] = 1, \ I_0[1] = 1, \ I_0[3] = 0$
- ...



Obrázek 23: Složitější struktura operací - BDD

5.4 Dosazování proměnných

V okamžiku, kdy se jedna proměnná vyskytuje jak na straně přiřazení, tak v některém z výrazů, je vhodné danou proměnnou v logickém výrazu rovnou nahradit daným výrazem.

Příklad takového programu:

```
NETWORK
TITLE =

      A     "a";
      A     "b";
      =     "c";
NETWORK
TITLE =

      O     "c";
      O     "e";
      =     "f";
```

Přepis do logických výrazů vypadá následovně:

$$c = a \ \& \ b$$

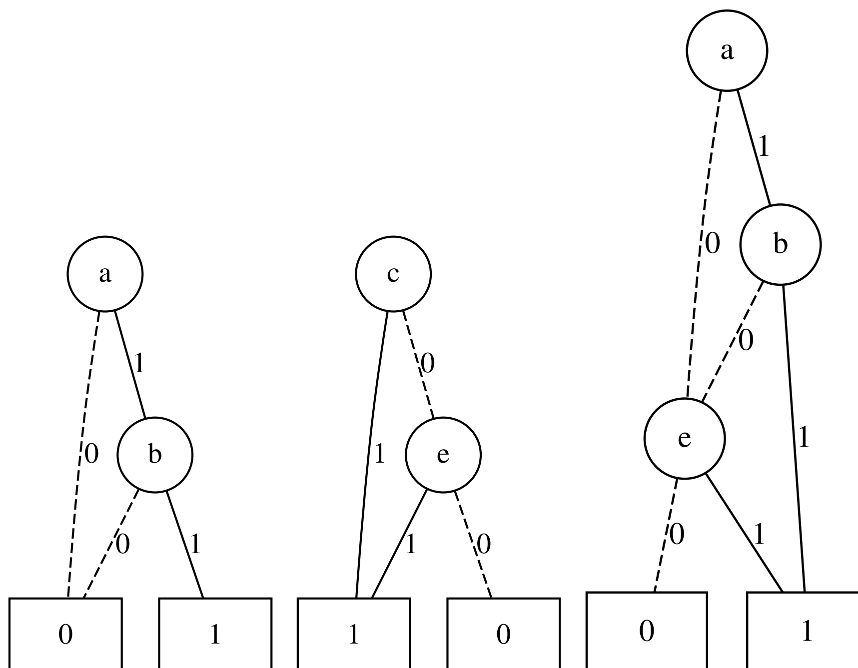
$$f = c \mid e$$

BDD pro tento program je vidět na obrázku 24.

Po automatickém dosazení celého výrazu, kterému je rovna proměnná c, je výsledný BDD pro f na obrázku 24. Z takového grafu se výsledná hodnota pro f snadno určí. V takto jednoduchém případě to není úplně třeba, ale ve složitějších programech se tato funkcionalita hodí. Zároveň se může ukázat, že daný výraz je zbytečně složitý a některé větve v něm nemají žádný vliv a je je tedy možné odstranit a tím program zjednodušit.

6 Stavový automat

Poslední, nikoliv v důležitosti, částí nástroje je automatická tvorba stavového automatu. Z BDD sice lze získat informace o tom, jaké proměnné se na hodnotě logického výrazu podílí, ale již nelze určit přesnou posloupnost těchto událostí. Pro přesné zmapování, jak probíhá vývoj dané proměnné, bylo nutné implementovat zobrazení v podobě stavového automatu.



Obrázek 24: Slučování BDDl: z leva BDD pro c, BDD pro f před sloučením, BDD pro f po sloučení

Stavový automat, který je automaticky vytvořen tímto nástrojem, je vytvořen vzhledem ke hodnotám, které může daná proměnná nabývat v určité části programu. Každá proměnná, do které se přiřazuje nějaký výraz, nebo se vyskytuje v operaci *Set* nebo *Reset*, má svůj vlastní stavový automat.

Každá binární proměnná má možnost v daný okamžik nabývat hodnoty logická nula, nebo logická jedna. Za časový okamžik se v PLC programu dá považovat daný Network, na kterém se proměnná vyskytuje na straně přiřazení. Z tohoto rovnou plyne, že stavový automat se dělí časově na jednolivé Networky a stavově na hodnotu, kterou proměnná v daný okamžik nabývá.

Podstatnou výhodou stavového automatu je možnost lépe zakreslit časovače. Jak již bylo patrné z popisu BDD, časovače nelze moc dobře reprezentovat v logických výrazech jako takových. Jejich reprezentace je pouze přibližná a má omezenou informativní schopnost. Ze stavového automatu je mnohem jasnější, jak je daný časovač použit. Kromě

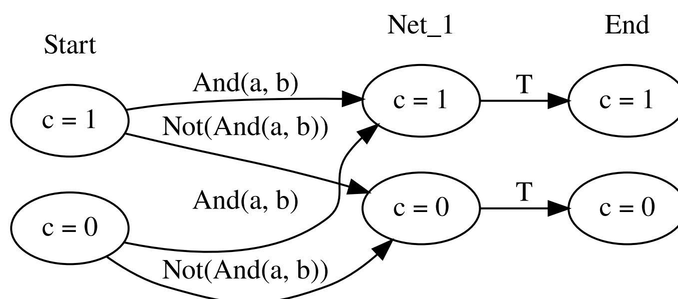
logického výrazu, který slouží pro spuštění časovače, je možné zobrazit i výraz sloužící pro reset časovače. Zároveň je naznačeno, který časovač reaguje na úroveň signálu nebo na jeho náběžnou hranu.

Pro to, aby ve stavovém automatu bylo zahrnuto veškeré fungování časovačů, by bylo nutno vytvořit stavové automaty jednolivých časovačů. Toto v nástroji zatím není implementováno. Je zaveden předpoklad, že každý, kdo by takový nástroj mohl používat, je s fungováním časovačů seznámen. Jinak řečeno, pro pochopení jak program funguje, stačí říct, o jaký časovač se jedná, jaký je jeho čas, jaká je podmínka startu, popřípadě jaká je podmínka resetu. Tyto informace by každému, kdo nástroj pro analýzu PLC programu použije, měly stačit pro zjednodušení práce. Pro automatickou analýzu by ovšem stavové automaty jednotlivých časovačů byly nutností.

6.1 Přiřazení

6.1.1 Jednoduché přiřazení

Na obrázku 25 je vidět jednoduchý stavový automat pro jednu operaci přiřazení. Jedná se o logickou operaci *AND*, která je popsána v kapitole 4.3.1. Příslušný BDD k tomuto programu je v kapitole 5.2.1.



Obrázek 25: Stavový automat pro přiřazení

Stavový automat je rozdělen do tří částí. Z leva do prava jsou to *Start*, *Net_1* a *End*. Každá část reprezentuje určitý časový okamžik. Binární proměnná c může být v každém čase v jednom ze dvou stavů. Stavy jsou rozděleny podle hodnoty dané proměnné

a to na $c = 0$ a $c = 1$.

V první části *Start* je znázorněn stav, ve kterém je proměnná na začátku PLC cyklu. Program obsahuje pouze jedno přiřazení do proměnné c a to na *Network 1*. Proto je druhý časový okamžik pojmenován jako *Net_1*. Poslední část je v tomto případě *End* a má význam konce PLC cyklu.

Mezi stavy z různých časových okamžiků existuje možnost přechodu. Každý přechod je aktivní pouze při splnění příslušné pdmínky. V tomto případě přechod mezi stavy v *Start* a stavy v *Net_1* odpovídá logickému výrazu před přiřazením do proměnné c. Z podmínek přechodu je patrné, že přiřazená hodnota v *Net_1* nezáleží na hodnotě proměnné v předchozím časovém okamžiku. To je také důvod, proč z obou stavů $c = 0$ a $c = 1$ v čase *Start* vede stejné propojení se stavy v následném časovém okamžiku.

V tomto konkrétním případě je možné do stavu $c = 1$ přejít s podmínkou *AND(a,b)*. Tato podmínka reprezentuje situaci zobrazenou na obrázku 5 v kapitole 4.3.1 a jedná se o logický výraz:

$$c = a \ \& \ b$$

Podmínka pro přechod do stavu $c = 0$ je negací podmínky pro přechod do $c = 1$, tj. *Not(AND(a,b))*. Jinak řečeno podmínka přechodu do stavu $c = 0$ u přiřazení je nesplnění podmínky pro přechod do stavu $c = 1$.

Poslední přechod mezi stavy proměnné v časových okamžicích Net_1 a *End* je pouze znázornění, jaká hodnota se do proměnné uloží na konci PLC cyklu. Konec cyklu je zde označen pomocí T. V tomto přechodu se nejedá o podmínku typu $T = 1$, ale jde pouze o označení konce cyklu. Stav proměnné z konce cyklu je použit jako stav v časovém okamžiku *Start* v následujícím cyklu PLC.

6.1.2 Vícenásobné přiřazení

Vícenásobným přiřazením se rozumí situace, kdy do jedné proměnné je hodnota přiřazena na několika řádcích programu.

Příklad takového kódu:

```
NETWORK
TITLE =

      A     "a";
      A     "b";
      =     "c";
NETWORK
TITLE =

      O     "d";
      O     "e";
      =     "c";
```

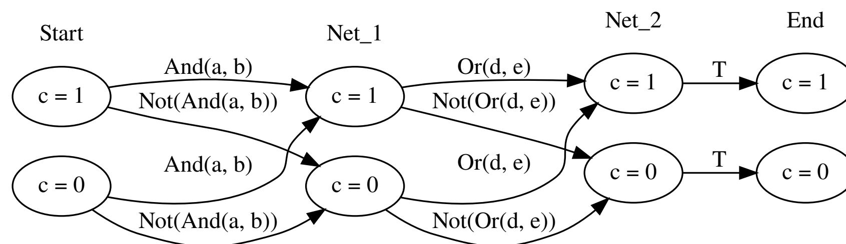
Nejprve se do proměnné přiřadí hodnota:

$$c = a \ \& \ b$$

a následně je hodnota přepsána:

$$c = d \mid e$$

Stavový automat pro takovouto strukturu programu je na obrázku 26.



Obrázek 26: Stavový automat pro vícenásobné přiřazení

Tento stavový automat má čtyři časové okamžiky *Start*, *Net_ 1*, *Net_ 2* a *End*. Hodnota proměnné mezi danými časovými okamžiky se řídí podle poslední změny. Tzn. mezi *Net_ 1* a *Net_ 2* je hodnota c rovna výrazu:

$$c = a \ \& \ b$$

v časovém okamžiku vykonávání programu v části *Network 1*.

V této ukázce programu je patrné, že hodnota je přepsána okamžitě. V takovém případě je namístě otázka, jestli má první přiřazení vůbec nějaký smysl. V reálném programu může ovšem jít o větší časové rozmezí mezi jednotlivými přiřazeními. Poté může mít aktuální hodnota vliv na jinou část programu. V případě, že tomu tak není a ze stavového automatu je vidět okamžitý přepis hodnoty proměnné, je možné takovou část programu podrobit pečlivější analýze.

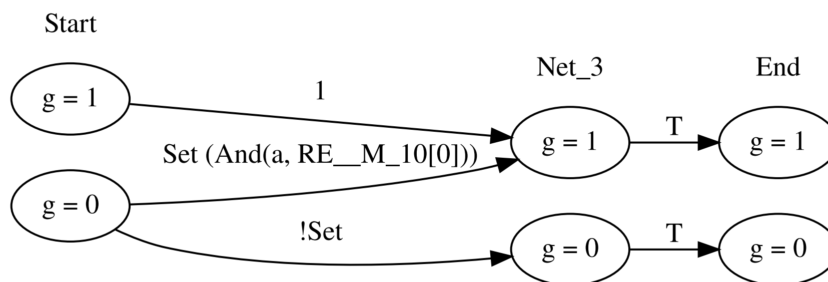
6.2 Set a Reset

6.2.1 Set

Pro operaci *Set* se stavový automat trochu liší. Na obrázku 27 je vidět stavový automat pro část programu:

```
NETWORK
TITLE =

      A     "a";
      FP    M     10.0;
      S     "g";
```



Obrázek 27: Stavový automat pro operaci *Set*

Zápis přechodu mezi stavy pomocí operace *Set* se ve stavovém automatu zapíše jako *Set()*. V závorkách je uvedena podmínka pro provedení operace. Z automatu operace *Set* je patrné, že přechod ze stavu $g = 1$ z předchozího časového okamžiku je vždy přímo do

stavu $g = 1$ v aktuálním časovém okamžiku. Jinak řečeno je-li hodnota proměnné logická jedna, operace *Set* na ni nemá žádný vliv.

Podmínka přechodu *!Set* je negace podmínky *Set()*. Pokud tedy ze stavu $g = 0$, v časovém okamžiku *Start*, nenastane podmínka $Set(And(a, RE_10[0]))$ v časovém okamžiku *Net_3* přejde se do stavu $g = 0$

V této konkrétní podmínce je vidět i formát zápisu detekce náběžné hrany. $And(a, RE_M10[0])$ jedná se o stejný způsob zápisu jako je v 5.2.3.

Před *RE_* je logický výraz, na který má být operace pro vyhodnocení náběžné hrany použita. Kdyby byl v řádku kódu ještě nějaký výraz za detekcí náběžné hrany, podmínka by vypadala následovně : $And(a, RE_M10[0], b)$. Proměnná b v takovém případě nemá vliv na detekci náběžné hrany a tudíž do podmínky vstupuje pouze jako úrovňové vyhodnocení.

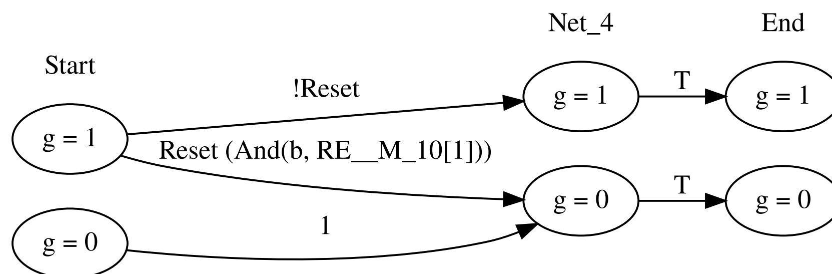
6.2.2 Reset

Obdobou operace *Set* je *Reset*.

```
NETWORK
TITLE =

      A     "b";
      FP    M     10.1;
      R     "g";
```

Stavový automat pro tuto část programu je na obrázku 28.



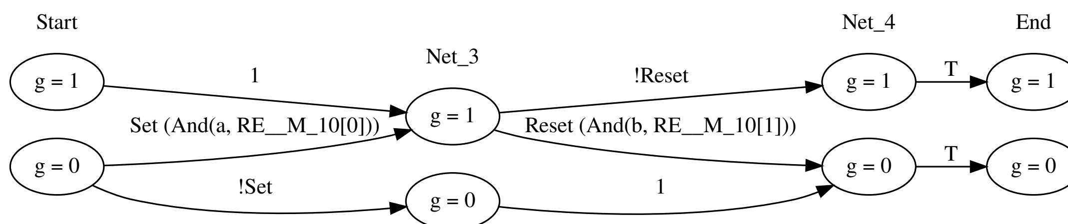
Obrázek 28: Stavový automat pro operaci *Reset*

U operace *Reset* je to se stavy opačně, než je tomu u příkazu *Set*. Nachází-li se proměnná

ve stavu logická nula, není možné, aby pomocí operace *Reset* přešla do logické jedničky. Zápis podmínky pro operaci *Reset* je obdobný, jako je tomu u *Set*, tedy *Reset(podmínka)*.

6.2.3 Set a Reset

V programu, ve kterém se vyskytuje operace *Set*, se z pravidla bude nacházet i *Reset*. Příkladem může být třeba program se stavovým automatem na obrázku 29. Jedná se o kombinaci předešlých samostatných operací *Set* a *Reset*.



Obrázek 29: Stavový automat pro operaci *Set* a *Reset*

Pomocí stavového automatu, ve kterém se bude nacházet pouze operace *Set* nebo *Reset*, lze nalézt i chyby v programu. Jestliže v programu bude pouze operace *Set*, tak po splnění její podmínky nebude existovat cesta, aby se proměnná vrátila do stavu logické nuly. V případě, že toto není úmyslem, jedná se o chybu. Taková chyba nemusí být nalezena při testování programu.

V okamžiku, kdy se chyba v programu nachází v části kódu, který se neprovádí v každém cyklu, je velmi náročné takovou chybu odhalit. Může se stát, že chyba nastane až po nasazení řídicího programu na samotnou linku. Takové případy jsou samozřejmě nežádoucí. Stavový automat může v hledání takových chyb pomoci, aby mohly být včas odstraněny.

6.3 Časovače

U časovačů je stavový automat neúplný. Nejprve by bylo zapotřebí vytvořit přesné stavové automaty daných časovačů, a ty poté použít pro vykreslení celého automatu vztahujícího

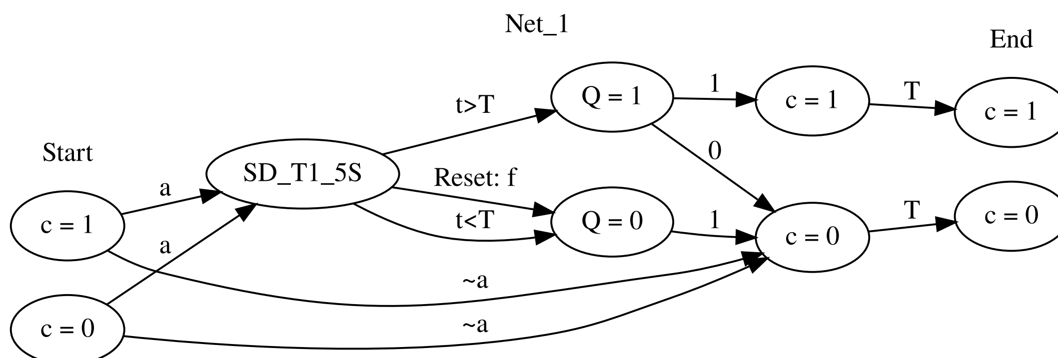
se k programu, kde je časovač použit. Úplné zakreslení časovačů by však mělo negativní vliv na výslednou přehlednost a čitelnost stavového automatu.

Implementace časovačů do stavového automatu, která byla použita, předpokládá předchozí znalost vnitřního fungování konkrétních časovačů. V takovém případě je výsledný stavový automat zjednodušený. Toto zjednodušení může napomoci lepší čitelnosti, ovšem klade nároky na znalosti časovačů. Informace, které lze zatím ze stavového automatu o časovači získat, jsou především o jaký časovač se jedná a jaké jsou jeho parametry. Dále pak podmínky pro spuštění a jsou zde naznačeny i podmínky výstupů společně s resetem časovače.

Vzhledem k tomu, že v podobě stavového automatu se časovače od sebe moc neliší, nebudou zde uvedeny všechny implementované časovače.

6.3.1 ODT a ODTS

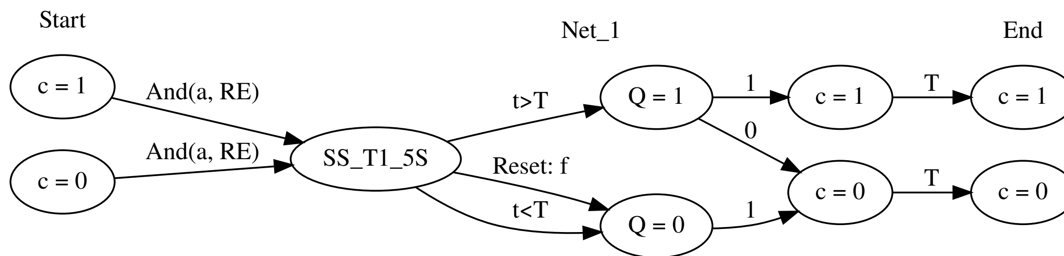
Na obrázcích 30 a 31 je vidět rozdíl mezi časovačem, který potřebuje pro běh trvale úroveň na vstupu a časovačem, který se spustí na náběžnou hranu. V obou případech se jedná o zpožděné sepnutí.



Obrázek 30: Stavový automat pro časovač ODT

Na obrázku 30 je znázorněn stavový automat pro časovač *ODT*. Pro běh časovače je zapotřebí úroveň logická jedna na vstupu časovače. Proto z obou stavů v časovém okamžiku *Start* vedou dva přechody. V případě, že $a = 1$, časovač běží. Pokud tomu tak není, je v

časovém okamžiku *Net_ 1* rovnou do proměnné c přiřazena logická nula. Stavy $Q = 1$ a $Q = 0$ jsou vnitřní stavy časovače a znázorňují jeho hodnotu na výstupu. Přechod mezi stavem *SD_ T1_ 5S* a jednotlivými vnitřními stavy se řídí podle uplynulého času. Podmínka $t > T$ je splněna, pokud je uplynulý čas časovače *T1* větší, než parametrem nastavený čas T, v tomto případě $5s$.



Obrázek 31: Stavový automat pro časovač ODTS

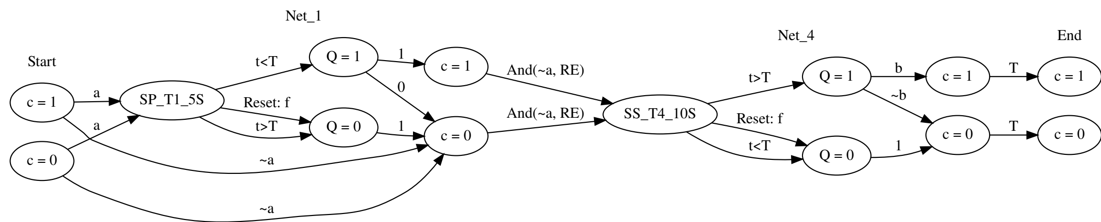
Na rozdíl od časovače *ODT* reaguje časovač *ODTS* (obrázek 31) na hranu. Stavový automat programu používajícího takový časovač je patřičně zjednodušen. Do podmínky běhu časovače je přidáno klíčové slovo *RE*. Ze stavů v časovém okamžiku *Start* nejsou záměrně nakresleny přechody rovnou do stavu $c = 0$ v časovém okamžiku *Net_ 1*. Podmínku pro tento přechod by bylo složité zapsat a zároveň by svou velikostí narušovala kompaktnost celého grafu. V podmínce by muselo být zohledněno spuštění časovače v některém z předchozích cyklů v časovém okně od posledního restartu časovače do současnosti.

Při interpretaci takového automatu je tedy zapotřebí rozlišit situaci, kdy časovač byl spuštěn (start podmínka je napsána v příslušném přechodu), nebo jeho spuštění nenastalo. Pokud časovač nebyl spuštěn, je jeho vnitřní stav $Q = 0$. Tomu pak odpovídá příslušný stav proměnné. V případě, že časovač běží, záleží na uplynulém čase.

6.3.2 Více časovačů

Na obrázku 32 je znázorněn stavový automat pro vícenásobné použití časovače. Jde v podstatě o totéž, jako je tomu u vícenásobného přiřazení, nebo u operací *Set* a *Reset*.

Tento stavový automat je tedy možné interpretovat následovně. V programu na prvním



Obrázek 32: Příklad zapojení časovačů

řádku (*Network 1*) je použit časovač *PULSE*. Podmínka pro spuštění časovače je $a = 1$. Vnitřní stav časovače je roven logické jedničce od okamžiku spuštění, až do chvíle, než uplyne nastavený čas ($5s$). Po uplynutí času přechází vnitřní stav časovače do logické nuly. Další možnost, jak se časovač dostane do vnitřního stavu logická nula, je změna na vstupu časovače nebo použití resetu. Proměnné je na konci řádku rovnou přiřazena výstupní hodnota časovače.

Tuto hodnotu má proměnná c až do čtvrtého řádku programu (*Network 4*). Na tomto řádku je spuštěn časovač *ODTS*. Podmínkou pro jeho spuštění je náběžná hrana na negaci proměnné a, tedy sestupná hrana na proměnné a. Pokud časovač nebyl spuštěn, má jeho vnitřní stav hodnotu logická nula. Po spuštění má vnitřní stav hodnotu logická nula až do okamžiku $t = 10s$. V tomto okamžiku přechází vnitřní stav na hodnotu logická jedna, kde setrvává až do okamžiku resetu časovače náběžnou hranou na proměnné *f*. Vnitřní stav časovače *T4* je přes logickou operaci *AND* s proměnnou b přiřazen do proměnné c. Proměnná c má tuto hodnotu až do konce PLC cyklu.

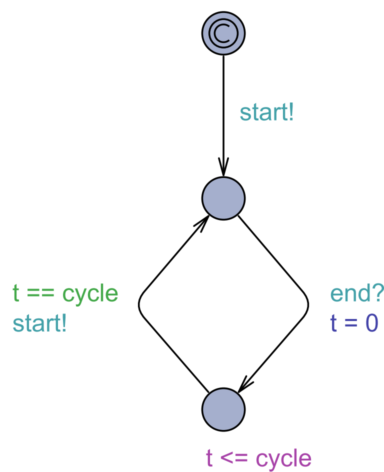
7 Verifikace PLC programu

Verifikace je ověření vlastností daného programu. Jedná se o způsob, jak formálně rozhodnout o správnosti fungování daného programu. Pro verifikaci se dají použít různé nástroje, v této práci je použit nástroj UPPAAL, který je dostupný na [5].

7.1 Simulace PLC v UPPAAL

Pro správnou verifikaci PLC programu v UPPAAL bylo zapotřebí vytvořit časový automat, který bude napodobovat funkci PLC. PLC pracuje v cyklech a jednotlivé programové příčky se provádějí sekvenčně. Pro napodobení této činosti PLC byla použita synchronizace časových automatů.

První automat je simulace PLC jako řídicího prvku. Model je zobrazen na obrázku 33.

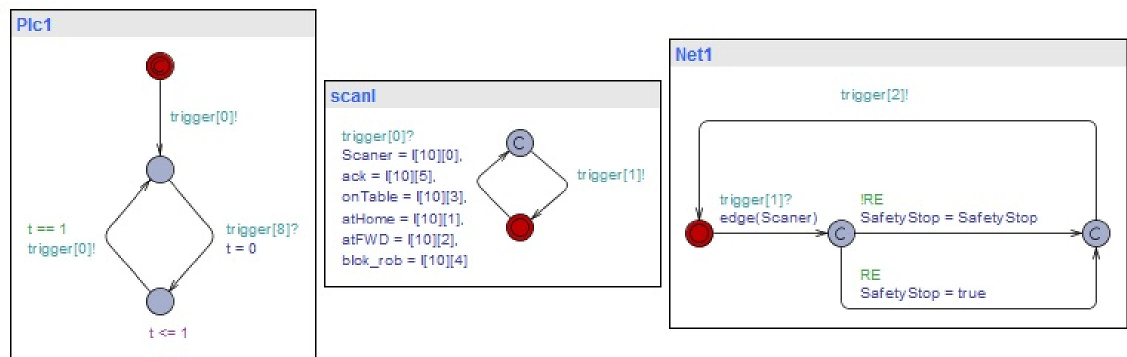


Obrázek 33: UPPAAL stavový automat pro PLC

Tento automat simuluje dobu trvání jednoho cyklu. V tomto případě je použit model, kdy každý cyklus trvá jednu časovou jednotku. Automat má dva stavy, které reprezentují začátek cyklu a jeho průběh. Při přechodu ze stavu reprezentujícím začátek cyklu je aktivován následný časový automat prostřednictvím synchronizačního signálu *start*.

První automat v cyklu PLC je naskenování vstupů. Tento časový automat má za úkol uložit do vnitřních proměnnných hodnoty, které jsou na vstupech *I*. Automat modeluje skutečné naskenování vstupů v reálném PLC. Po naskenování proměnných následují stavové automaty jednotlivých programových příček.

Po provedení poslední programové příčky v daném programu je spuštěn automat reprezentující zápis proměnných na výstupy *Q*. Po zápise se řízení předává zpět automatu PLC, který uzavře cyklus a okamžitě přechází do nového cyklu. V některých případech je



Obrázek 34: Ukázka spojení automatů (PLC - Sken I - Net 1)

zapotřebí po zápisu proměných spustit speciální časový automat, který modeluje fyzický svět. Toto je použito i v příkladu, který je popsán v 7.5.

7.2 Převod PLC programu

Každý programový řádek daného PLC programu je převeden na samostatný časový stavový automat. K tomuto převodu je vhodné použít stavové automaty, které jsou automaticky vytvořeny nástrojem. Narozdíl od těchto automatů je zapotřebí do UPPAAL přesně namodelovat i fungování časovačů, detekci náběžné a sestupné hrany a operace *Set* a *Reset*.

7.3 Model reálného světa

Kromě stavového automatu pro samotný PLC program je pro ověření jeho správnosti zapotřebí modelovat i reálný svět. Takový stavový automat má za úkol modelovat chování reálného zařízení, kterým má být PLC řizeno. Stavové automaty musí přesně popisovat chování zařízení, a to ve všech stavech, ve kterých se dané zařízení může nacházet.

Před spojením automatu PLC programu a automatu reálného zařízení, je zapotřebí automat pro chování reálného zařízení verifikovat samostatně. Výsledek verifikace PLC programu by měl být nezatížen chybami, které vznikly při modelování reálného zařízení.

7.4 Verifikace

Nástroj UPPAAL je snadné použít pro ověřování vlastností stavových automatů. Často ověřovanou vlastností je neuváznutí automatu. V UPPAAL je možné procházet přes všechny možné stavy. Lze tedy ověřit to, že automat má v každém stavu alespoň jeden dosažitelný následný stav. Tímto je zaručeno, že nedojde k uváznutí automatu představujícího PLC program. Takový stav může nastat například při použití operace *Set* bez příslušné párové operace *Reset*. Danou proměnnou pak lze pouze nastavit na logickou jedničku a už neexistuje možnost návratu do logické nuly.

Další ověřování vlastností se už týká přímo funkčnosti. Typicky při splnění určitých podmínek má nastat příslušná akce. Ověřit je možné dosažení určitého stavu alespoň v jednom případě při splnění podmínek. Dosažitelnost je závislá i na tom, v jakém stavu se zrovna automat nachází. Je možnost provést test i na dosažitelnost při splnění podmínek nezávisle na aktuálním stavu. Jinak řečeno, v případě splnění daných podmínek, je vždy provedena příslušná akce, odpovídající některému ze stavů.

Při nesplnění nadefinovaného ověření se dá pomocí nástroje UPPAAL přesně vysledovat posloupnost událostí, které vedly k jeho nesplnění. Na základě tohoto průchodu, lze zpětně navrhnout změny PLC programu. Po úpravě v PLC je samozřejmě nutné provést adekvátní úpravu i v UPPAAL modelech, resp. modely znovu vygenerovat. Pro takto upravený program je vhodné všechny testy provést znovu, protože se může stát, že zásahem do programu již nebudou předchozí splněné testy splnitelné.

Tímto interativním postupem lze docílit maximální spolehlivosti programu. Bezchybnost takového programu je ovšem závislá i na kvalitě modelů jak samotného programu, tak i vnějšího světa. Pro odladění všech chyb je tedy zapotřebí co možná nejpřesnějších modelů.

7.5 Verifikace PLC programu otočného stolu

Pro názornou ukázku verifikace PLC programu byl vytvořen jednoduchý program pro ovládání otočného stolu (obrázek v příloze B). Otočný stůl slouží, k založení dílu do linky. Obsluha založí díl na jedné straně stolu, ten se poté otočí na druhou stranu, kde na dílu

pracuje robot. Po vykonání práce robot díl ze stolu odebere a stůl je uvolněn pro otočení zpět do původní polohy.

Otočný stůl má dvě polohy *Home* a *Forward(Fwd)*. Mezi těmito polohami se stůl může otáčet při splnění určitých podmínek. V obou případech se stůl nesmí otočit, když je narušena bezpečnost přítomností obsluhy v pracovním prostoru stolu. Narušení bezpečnosti je reprezentováno proměnnou *SafetyStop*. Dále se stůl nesmí otáčet, když je pohyb blokován robotem. Tj. reprezentováno proměnnou *blok_rob*. Z pozice *Home* se stůl do pozice *Fwd* může otočit, když je na stole detekován předmět *onTable = 1* a když se stůl nachází v příslušné poloze *atHome*. Otočení z polohy *Fwd* do polohy *Home* je povoleno, když na stole nic není a stůl se nachází v poloze *atFwd*.

PLC používá pro ovládání stolu dva výstupy: *run* pro zapnutí pohonu stolu a *rev* pro otočení smyslu otáčení. V obou případech otočení stolu musí být proměnná *run* v logické jedničce po celou dobu vykonávání pohybu. Není nutné proměnnou nulovat přesně při dojezdu do polohy. Zastavení stolu je řešeno mechanickým koncovým spínačem. Pro začátek pohybu je ovšem důležitá náběžná hrana. Pro pohyb z *Fwd* do *Home* je nutné, aby proměnná *rev* byla v logické jedničce. Otočení stolu trvá předem definovaný čas. Po uplynutí tohoto času je tedy nutné proměnnou *run* vynulovat.

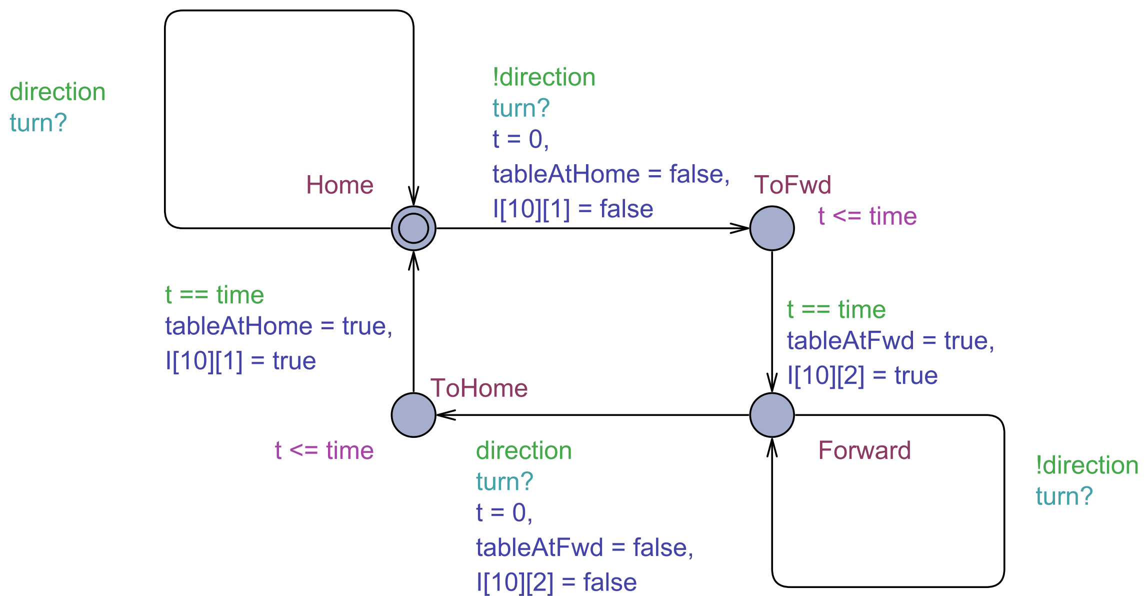
Celý program lze vidět na obrázku v příloze B.

7.5.1 Stavový automat pro otočný stůl

Pro možnost verifikace fungování PLC programu bylo nutné nejprve v UPPAAL namodelovat chování otočného stolu. To představuje model reálného světa a tvoří vlastně specifikaci požadovaného chování stolu, které bude implementováno prostřednictvím programu v PLC. Stavovový automat pro otočný stůl je znázorněn na obrázku 35.

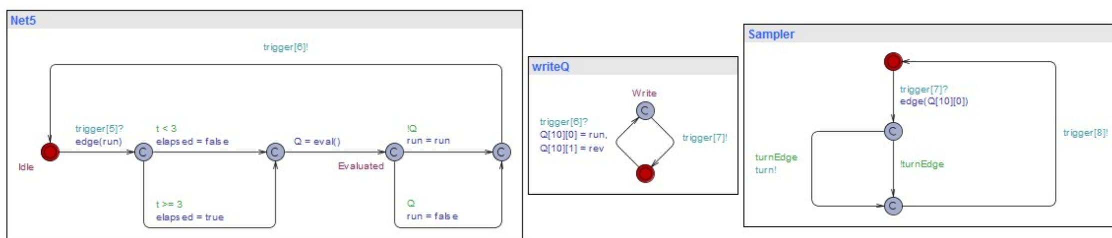
Z modelu je patrné, že otočný stůl se může nacházet ve čtyřech stavech. Dva stavy jsou koncové polohy, do kterých se má stůl otočit, tedy poloha *Home* a *Forward*. Stavy *toHome* a *toFwd* znázorňují přejezd stolu z jedné polohy do druhé.

Přechod z koncového stavu do stavu reprezentujícího pohyb je možný pouze v okamžiku splnění podmínky správného směru otáčení a při detekované náběžné hraně na proměnné



Obrázek 35: UPPAAL stavový automat otočného stolu

run. Tato detekce je provedena samostatným stavovým automatem *Sampler*, který bylo nutné umístit do PLC cyklu (obrázek 36).

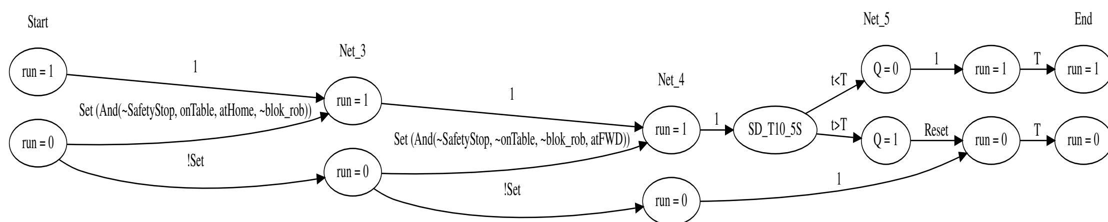


Obrázek 36: UPPAAL systém - konec cyklu PLC

7.5.2 Převod programu do UPPAAL

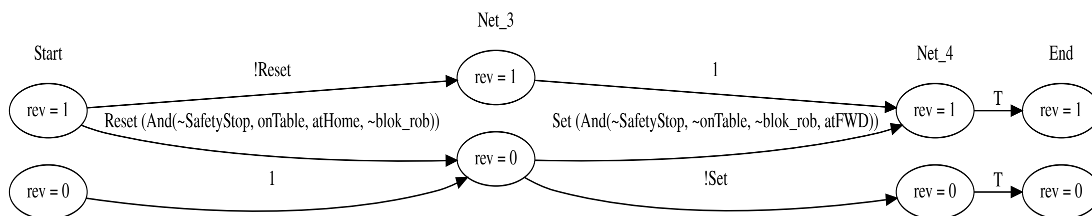
Celý program pro řízení stolu bylo nutné převést do UPPAAL stavových automatů. K převodu byly použity nástrojem automaticky vygenerované stavové automaty pro jednotlivé proměnné, které mají vliv na funkci otočného stolu. Pro proměnou *run* je automat na obrázku 37.

Reverzace otočného stolu se řídí stavovým automatem, který je znázorněn na



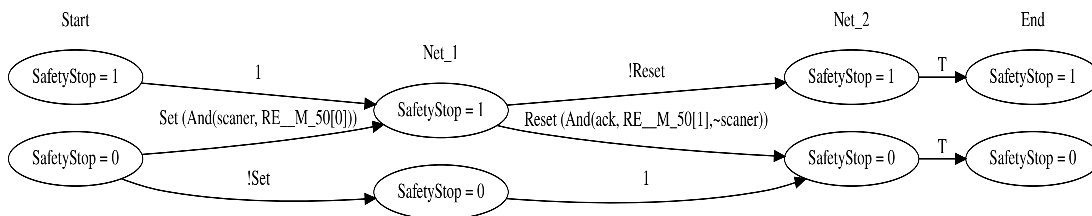
Obrázek 37: Stavový automat pro proměnnou *run*

obrázku 38.



Obrázek 38: Stavový automat pro proměnnou *rev*

Bezpečnostní blokování rozjezdu (obrázek 39) je provedeno pouze zjednodušeně. V tomto případě se nejedná o bezpečnost jako takovou. Proměná *SafetyStop* není deklarována jako bezpečnostní proměnná. Celý program je v jednom bloku a není dělen na běžnou a safety část. Jde tedy pouze o jednoduchý příklad, ve kterém není možné *SafetyStop* považovat za bezpečnostní zastavení.

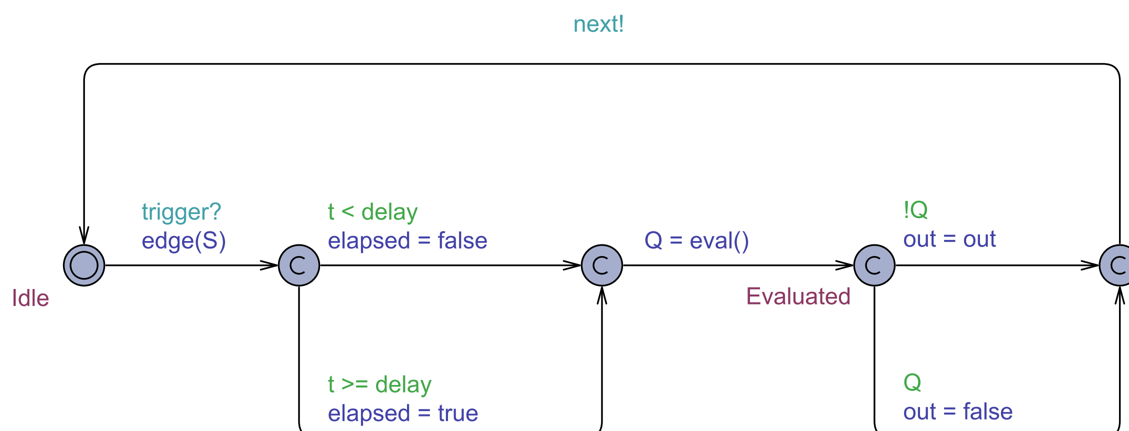


Obrázek 39: Stavový automat pro proměnou *SafetyStop*

Ze stavových automatů je patrné, že v Networcích 1 a 2 je *set* respektive *reset* proměnné

SafetyStop. V obou případech je zde použita detekce náběžné hrany. Hodnota proměnné *rev* je ovlivněna v Networcích 3 a 4 a hodnotu proměnné *run* je možné měnit v Networcích 3, 4 a 5. Program je k nahlédnutí v příloze B

Na *Network 5* je použit *ODT* časovač, který je nutné namodelovat. V UPPAAL narozdíl od stavových automatů generovaných nástrojem, je nutné časovač namodelovat celý. Model stavového automatu pro použití časovače *ODT* je v tomto případě vidět na obrázku 40. Na výstupu časovače je navíc použita operace *reset*. V případě normálního přiřazení by automat byl lehce pozměněn. Hodnota proměnné za časovačem by byla rovna hodnotě výstupu časovače *Q*.



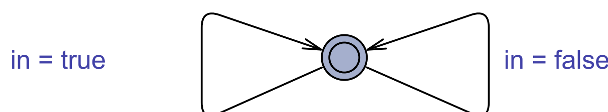
Obrázek 40: UPPAAL stavový automat pro časovač *ODT* s operací *reset* na výstupu

7.5.3 Verifikace programu pro ovládání otočného stolu

Pro celý systém automatů bylo ověřeno několik vlastností. Kromě samotného stavového automatu PLC programu a otočného stolu jsou ve výsledném systému zařazeny automaty pro náhodnou změnu vstupních proměnných (obrázek 41) tak, aby bylo možné prověřit všechny možné kombinace vstupů.

Model má pouze jeden stav a dva přechody. Přechody nejsou nijak podmíněné a v UPPAAL se tedy provedeny náhodně. Při přechodu dojde k zapsání hodnoty *true*, respektive *false* do příslušné proměnné reprezentující vstup.

Jak první bylo nutné ověřit funkcionalitu stavového automatu pro otočný stůl. Testy



Obrázek 41: UPPAAL model náhodné změny hodnoty vstupu

pro otočný stůl jsou nezávislé na PLC cyklu a je zde testováno, zda se stůl je schopný otočit a jestli se otáčí správným směrem.

Další testy se už přímo týkají fungování PLC programu. Hlavním testem kromě nezaseknutí systému automatů bylo testování, zda je možné stůl řízený PLC programem otáčet z jedné koncové polohy do druhé. Dále bylo testováno, jestli se stůl otočí pokaždé, když jsou splněny patřičné podmínky. Zde došlo k selhání testu při snaze otočit se z pozice *Fwd* do pozice *Home*. Podmínky byly splněny, ale stůl se neotočil. UPPAAL umožňuje podrobnou analýzu sledu událostí, které vedly k selhání testu. Po důkladném prostudování nastalé situace bylo zjištěno, že automat pro otáčení stolu byl nesprávně zapojen. Proměnná *rev* nebyla propojena s vnitřní proměnou stavového automatu otočného stolu *direction*.

Hledání chyby patří mezi hlavní úskalí verifikace pomocí UPPAAL. Je třeba správně navrhnout pravidla pro ověřování vlastností. A při nesplnění určitého pravidla je zapotřebí rozhodnout, jestli se jedná o chybu stavového automatu, propojení stavových automatů, nebo chybu v definici pravidla. V tomto případě nastala chyba v propojení automatů.

8 Závěr

V rámci této práce byl vytvořen nástroj pro automatickou tvorbu BDD a stavových automatů. BDD i stavový automat je generován vzhledem k hodnotám binární proměnné.

Z BDD je možné snadno určit vzájemnou souvislost mezi všemi proměnnými napříč celým programem a je určen především pro snadné stanovení výsledné hodnoty proměnné na konci PLC cyklu. Motivací pro tvorbu BDD byla i možnost snadno zjistit, jaké kombinace proměnných v programu jsou zapotřebí pro logickou jedničku respektive nulu u dané proměnné.

Narozdíl od BDD je stavový automat vygenerován nejen vzhledem k hodnotě pro-

měnné, ale i času, kdy ke změně hodnoty může dojít. Stavový automat názorně zobrazuje, jak probíhá zapisování hodnot do proměnné, a to v každé části programu, kde je možné proměnnou změnit. Stavový automat je patřičně zjednodušen pro snažší ruční analýzu. Z takto vygenerovaného stavového automatu se dá sestavit model pro UPPAAL. Pomocí systému stavových automatů v UPPAAL lze provést automatickou verifikaci PLC programu.

Jak pro BDD, tak i stavový automat je důležité převést PLC program do zpracovatelné podoby. Velkým přínosem této práce je možnost automaticky převést PLC program napsaný v Ladder Diagramu na logické výrazy. S logickými výrazy se pak nadále pracuje a je pomocí nich vygenerován BDD a stavový automat.

Závěr práce je věnován verifikaci PLC programu. Práce se nezabývá přímoautomatický generováním stavových automatů z programu PLC pro UPPAAL za účelem verifikace. Pozornost je věnována spíše samotnému principu zapojení verifikace PLC programu do procesu vývoje.

Vhodným pokračováním této práce by bylo doplnění nástroje o automatické generování stavových automatů nebo automatické generování testů pro program, kdy je stavový prostor příliš velký a pro úplnou verifikaci kvůli tomu nevhodný.

Literatura

[1] E. P. Enoiu, D. Sundmark, P. Pettersson. *Model-Based Test Suite Generation for Function Block Diagrams using the UPPAAL Model Checker* IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, 2013

[2] *Python knihovna EDA [online] [cit. 4-1-2017]*
`http://pyeda.readthedocs.io/`

[3] *Open source graph visualization software [online] [cit. 4-1-2017]*
`http://www.graphviz.org/`

[4] *Statement List (STL) for S7-300 and S7-400 Programming [online] [cit. 4-1-2017]*
`https://cache.industry.siemens.com/dl/files/446/45523446/att_79269/v1/s7awl__b.pdf`

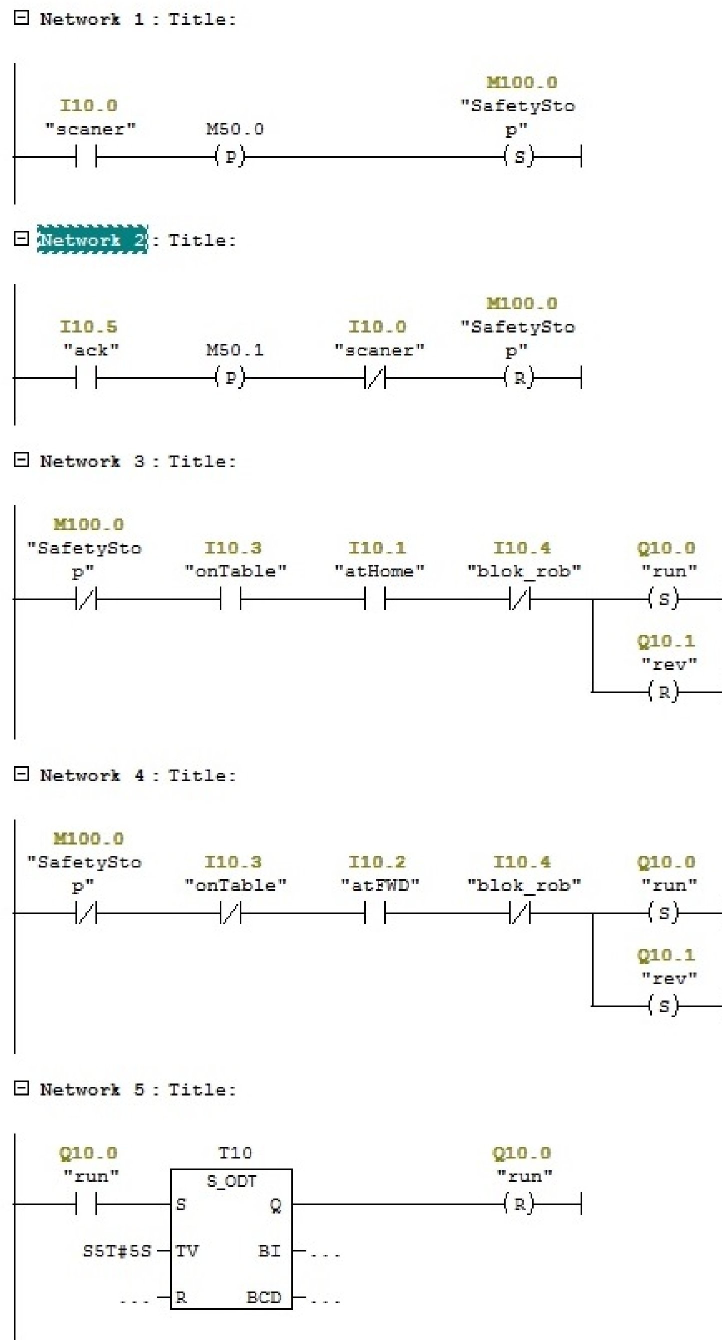
[5] *UPPAAL [online] [cit. 4-1-2017]*
`http://www.uppaal.org/`

[6] G. Behrmann, A. David,K. G. Larsen *A Tutorial on Uppaal 4.0 [online] [cit. 4-1-2017]*
`http://www.uppaal.com/admin/anvandarfiler/filer/uppaal-tutorial.pdf`

A Obsah přiloženého CD

- Text této práce v digitální podobě
- Zdrojové kódy nástroje pro tvorbu BDD a stavového automatu
- Projekt pro verifikaci programu otočného stolu v UPPAAL

B Program pro otočný stůl



Obrázek 42: LAD PLC program pro otočný stůl

STL PLC program pro otočný stůl :

```
NETWORK
TITLE =
      A     "scaner";
      FP    M     50.0;
      S     "SafetyStop";
NETWORK
TITLE =
      A     "ack";
      FP    M     50.1;
      AN    "scaner";
      R     "SafetyStop";
NETWORK
TITLE =
      AN    "SafetyStop";
      A     "onTable";
      A     "atHome";
      AN    "blok_rob";
      S     "run";
      R     "rev";
NETWORK
TITLE =
      AN    "SafetyStop";
      AN    "onTable";
      A     "atFWD";
      AN    "blok_rob";
      S     "run";
      S     "rev";
NETWORK
TITLE =
      A     "run";
      L     S5T#5S;
      SD    T     10;
      NOP   0;
      NOP   0;
      NOP   0;
      A     T     10;
      R     "run";
```