VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PŘEVOD LTL FORMULÍ S OMEZENÝMI OPERÁTORY DO AUTOMATŮ S ČÍTAČI

TRANSLATION OF LTL WITH BOUNDED REPETITION TO AUTOMATA WITH COUNTERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ALEXANDRA SLEZÁKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Mgr. LUKÁŠ HOLÍK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Studentka: **Slezáková Alexandra**
Program: Informační technologie
Název: **Převod LTL formulí s omezenými operátory do automatů s čítači**
Translation of LTL with Bounded Repetition to Automata with Counters
Kategorie: Algoritmy a datové struktury

Zadání:

1. Nastudujte existující algoritmy pro překlad LTL to konečných automatů.
2. Adaptujte vhodný algoritmus pro překlad LTL s omezenými operátory do automatů s čítači. Uvažujte fragment LTL, kde slovo není modelem formule tehdy, když některý jeho prefix je v jistém regulárním jazyce protipříkladů.
3. Implementujte algoritmus a otestujte na praktických formulích (například formulích poskytnutých firmou Honeywell).
4. Srovnejte velikost automatů generovaných existujícími algoritmy a Vašich automatů s čítači.
5. Volitelně, také můžete hledat způsoby, jak optimalizovat velikost výsledného čítačového automatu.

Literatura:

- Lukáš Holík, Ondřej Lengál, Olli Saarikivi, Lenka Turoňová, Margus Veanes, Tomáš Vojnar: Succinct Determinisation of Counting Automata via Sphere Construction (Technical Report). CoRR abs/1910.01996 (2019)

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, částečně 2 a 3.

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

Vedoucí práce: **Holík Lukáš, Mgr., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Táto práca rieši prevod LTL formulí s obmedzenými operátormi (temporálne operátory, ktoré majú obmedzenú platnosť na určitý počet krokov) do automatu s čítačmi. Použitím klasických metód je automatová reprezentácia takýchto formulí veľká, pretože veľkosť automatu môže byť exponenciálna k hornej hranici obmedzenia. Prezentujeme konštrukciu, ktorá vytvára automat nezávislý na obmedzení.

Práca predstavuje riešenie problému pomocou čítačového Büchi automatu. Čítače zaisťujú nevytváranie podobných stavov a prechodov, čo vedie k zmenšeniu veľkosti automatu. Naša metóda je implementovaná a experimentálne overená. Počet stavov automatu pre formule s veľkým obmedzením operátora redukujeme niekoľkonásobne v porovnaní s klasickými metódami.

Abstract

This work solves translation of LTL with bounded repetition (temporal operators that have limited validity for a certain number of steps) to automaton with counters. Using classical methods, the automaton representation of such formulas is large, because the size of automaton may be exponential to the upper limit of the bounded repetition. We present a construction that creates the automaton independent from repetition.

The work represents a solution to the problem using the Büchi automaton with counters. The counters ensure that similar states and transitions are not created, which leads to a reduction in the size of the automaton. Our method is implemented and experimentally verified. We reduce the number of states of the automaton for formulas with large bound of operators several times in comparison to classical methods.

Kľúčové slová

preklad LTL formulí, LTL formule s obmedzenými operátormi, čítačový Büchi automat, lineárna temporálna logika, overovanie modelov

Keywords

translation of LTL formulas, LTL with bounded repetition, counting Büchi automaton, linear temporal logic, model checking

Citácia

SLEZÁKOVÁ, Alexandra. *Převod LTL formulí s omezenými operátory do automatů s čítači*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Lukáš Holík, Ph.D.

Převod LTL formulí s omezenými operátory do automatů s čítači

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Mgr. Lukáša Holíka, Ph.D. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Alexandra Slezáková
4. júna 2020

Poďakovanie

Ďakujem pánovi Mgr. Lukášovi Holíkovi, Ph.D. za užitočné pripomienky, ochotu a usmernenie pri písaní tejto bakalárskej práce. Taktiež ďakujem Prof. Ing. Tomášovi Vojnarovi, Ph.D. a Ing. Ondřejovi Lengálovi, Ph.D. za odborné konzultácie.

Obsah

Kapitola 1

Úvod

V dnešnej dobe je používanie počítačových systémov rozšírené v rôznych oblastiach, ktoré si kladú vysoké nároky na ich vlastnosti a predovšetkým správnosť. Zlyhanie takéhoto systému by mohlo viesť k finančnej strate, v horšom prípade ohrozeniu zdravia.

V praxi sa pre korektnosť systémov používa testovanie alebo simulácia. Tento spôsob overovania môže byť časovo a finančne náročný. Rovnako vyžaduje aj odbornú spôsobilosť zamestnanca. Týmto nárokom je možné zamedziť použitím metód formálnej verifikácie. Jednou z týchto metód je overovanie modelov oproti modelom temporálnej logiky.

Pri overovaní modelov sa kontroluje, či zadaný model odpovedá špecifikácii. Verifikovaný model je modelovaný ako konečný automat a špecifikácia je formalizovaná napísaním vlastností temporálnej logiky [4]. Na overovanie je možné použiť už vytvorené nástroje. Avšak problémom niektorých nástrojov je neschopnosť spracovať dlhé formule, prípadne vytváranie veľkých automatov.

Táto práca, na ktorej čiastočne spolupracujeme s firmou Honeywell, je zameraná na vytvorenie čítačového Büchi automatu z LTL formulí s obmedzenými operátormi. Súčasné riešenie prevádza tento typ formulí do rozsiahleho Büchi automatu s počtom stavov, ktorý odpovedá obmedzeniu formule. Automat vytvorený pre zložitejšiu formulu s vysokým číslom obmedzenia môže pozostávať až z niekoľko desiatok stavov. Príkladom takejto formule môže byť $\mathbf{G}_{\{=20\}}\varphi$, kde φ musí platiť 20 krokov. Cieľom je previesť formule od firmy Honeywell na automaty s menším počtom stavov v porovnaní s existujúcimi nástrojmi použitím čítača. Negácia formule umožní odvodenie jednoduchších temporálnych a booleovských operátorov z tých zložitejších, čo taktiež vedie k zmenšeniu veľkosti automatu.

Použitím klasických metód automat pre negovanú formulu $\mathbf{F}(!\text{f\_ice\_sys} \wedge \mathbf{G}_{\{=120\}}\text{art})$ z praxe pozostáva z viacej ako 120 stavov. Zo 120 stavov je možné vykonať prechod prijatím ľubovoľných slov. Použitím nami navrhnutého algoritmu by bol vytvorený čítačový Büchi automat, ktorý by pozostával z jedného takéhoto stavu a k navyšovaniu hodnoty čítača by dochádzalo pri prijímaní ľubovoľných slov, kým by sa nerovnalo obmedzeniu operátora. Použitím čítača môže dôjsť k zredukovaniu počtu stavov na 3.

Pri vytváraní automatov z obecných formulí môžu nastať problémy, kedy kombinácia niektorých operátorov vedie k veľkému automatu. Takáto kombinácia spôsobuje neustále zanorovanie a s jedným čítačom si nevystačíme. Keďže sú naším cieľom formule od Honeywell a tie majú určitú obmedzenú formu, môžeme sa sústrediť len na tú. Prevod formulí v tejto forme do automatov sme implementovali a potvrdilo sa, že pri použití čítačov dokážeme generovať menšie automaty.

Nasledujúca kapitola je zameraná na zhrnutie základných vlastností lineárnej temporálnej logiky, overovania modelov a prevodu LTL formulí do automatov. Ďalšia kapitola

sa venuje návrhu riešenia vrátane vytvárania Büchi automatov pre konkrétne LTL formule a následnej implementácii. Na záver sa práca venuje porovnávaniu veľkostí automatov vygenerovaných súčasnými nástrojmi a naším programom pre formule poskytnuté firmou Honeywell.

Kapitola 2

Teoretický základ

Cieľom tejto kapitoly je definovať lineárnu temporálnu logiku v sekcii 2.1 a overovanie modelov v sekcii 2.2. Sekcia 2.3 popisuje zhodnotenie súčasných riešení prevodu LTL formulí do automatu.

2.1 Lineárna temporálna logika

Preklad lineárnej temporálnej logiky je intenzívne študovaná oblasť už od staroveku. Prvýkrát bola predstavená Amirom Pnuelim v roku 1977 [21]. LTL overovanie modelu je rozšírená a plne automatizovaná technika používaná na overenie špecifikácie daného systému. Temporálna logika rozširuje výrokovú a predikátovú logiku spôsobmi, ktoré umožňujú odkazovanie na nekonečné správanie reaktívneho systému. Ponúka matematicky presné zápisy na vyjadrenie vlastností behov systému, takzvané LT vlastnosti.

Základné časové modality, ktoré sa vyskytujú v temporálnej logike zahŕňajú tieto operátory:

□ „always“ **G** (teraz a navždy v budúcnosti)
◊ „eventually“ **F** (nakoniec, napokon v budúcnosti)

Syntax

V tejto sekcii sú popísané pravidlá, podľa ktorých môže byť LTL formula zostavená. Základnými prvkami formule sú atomické výroky, logické operácie ako konjunkcia, negácia a časová modalita **X**(„next“). **X**–modalita je operátor unárneho prefixu a vyžaduje LTL formulu ako argument. Formula $\mathbf{X}\varphi$ udržuje súčasný stav, ak nasledujúci stav platí φ. U–modalita je binárny operátor infixu a ako argument vyžaduje dve LTL formule. Zápis pre formulu $\varphi_1 \,\mathsf{U}\, \varphi_2$ znamená, že niekedy v budúcnosti platí φ_2 a dovtedy platí φ_1 [2].

Definícia 1 Formule lineárnej temporálnej logiky nad konečnou množinou atomických výrokov (tvrdení, ktoré musia byť pravdivé alebo nepravdivé) AP sú sformulované podľa nasledujúcej gramatiky:

$$\varphi := \mathit{true} \mid a \mid \neg a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \mid \mathbf{X}\,\varphi \mid \mathbf{F}\,\varphi \mid \mathbf{G}\,\varphi$$

kde $a \in AP$.

Poradie operátorov je dané nasledovne. Unárne operátory sú silnejšie ako tie binárne. Negácia a časový operátor „next“ sú rovnako silné.

S použitím booleovských spojek $\wedge$ a $\neg$ je zaistená plná sila výrokovej logiky. Ďalšie booleovské spojky ako disjunkcia $\vee$, implikácia $\rightarrow$ a ekvivalencia $\leftrightarrow$ je možné odvodiť nasledovne [17]:

$$\begin{array}{lcl} \varphi_1 \vee \varphi_2 & \stackrel{\text{def}}{=} & \neg\ (\ \neg\ \varphi_1 \wedge \neg\ \varphi_2\ ) \\ \varphi_1 \rightarrow \varphi_2 & \stackrel{\text{def}}{=} & \neg\ \varphi_1 \vee \varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 & \stackrel{\text{def}}{=} & (\ \varphi_1 \rightarrow \varphi_2\ ) \wedge (\ \varphi_2 \rightarrow \varphi_1\ ) \\ \mathbf{F}\ \varphi & \stackrel{\text{def}}{=} & \neg\ \mathbf{G}\ \neg\ \varphi \\ & \vdots & \end{array}$$

Kombinovaním časových modalít **F** a **G** vzniknú nové časové modality:

$\mathbf{G}\ \mathbf{F}\ \varphi$	„nekonečne často φ“
$\mathbf{F}\ \mathbf{G}\ \varphi$	„nakoniec navždy φ“

Sémantika

Formuly LTL vyjadrujú vlastnosti cesty. Sémantika LTL formule φ je definovaná ako jazyk *Words*(φ), ktorý obsahuje všetky nekonečné slová nad abecedou 2^{AP} vyhovujúce φ. Každej LTL formule odpovedá vždy jedna konkrétna lineárna temporálna vlastnosť. Množinu LT vlastností popisujúcich typy chovania delíme nasledovne [2]:

- Invariantná vlastnosť predstavuje najjednoduchší typ, jej splnenie je závislé len na platnosti invariantu nad každým stavom zvlášť. Invariantom rozumieme formulu jednoduchej logiky nad množinou atomických výrokov s konjunkciou a negáciou.
- Vlastnosť bezpečnosti (safety) vyjadruje, že v systéme nenastane chyba, pričom porušenie takejto vlastnosti nastane vždy v konečnom čase. Formálna definícia vlastnosti je určená množinou všetkých nedovolených konečných prefixov, ktoré túto vlastnosť porušujú. Každá safety formula môže byť prevedená na safety automat. Operátor **G** je používaný pre vlastnosti bezpečnosti. Formula $\mathbf{G}p$ je platná, ak p platí vo všetkých stavoch cesty. Protipríkladom vlastnosti bezpečnosti je cesta stavov, kde posledný stav je v rozpore s vlastnosťou [4].
- Vlastnosť živosti (liveness) vyjadruje, že vždy bude možné vykonať niečo dobré. Formálna definícia vlastnosti vyžaduje, aby konečný prefix mohol byť rozšírený na nekonečnú postupnosť. Protikladom živých vlastností v najjednoduchšej forme je cesta k cyklu, ktorá neobsahuje požadovaný stav. Takýto cyklus predstavuje nekonečné množstvo ciest, ktoré nikdy nedosiahnu určitý stav [4].

Definícia 2 (Interpretácia nad slovami) Nech σ je slovo, kde i-te písmeno slova označujeme ako $\sigma[i]$, pre $i \geq 0$. Nech φ je LTL formula nad AP. LT vlastnosť indukovaná φ je

$$\mathit{Words}(\varphi) = \{\ \sigma \in (2^{AP})^{\omega}\ \mid\ \sigma \models \varphi\ \}$$

kde relácia vyplývania $\models\ \subseteq (2^{AP})^{\omega} \times \text{LTL}$ je najmenšia relácia s nasledujúcimi vlastnosťami:

$$
\begin{array}{lcll}
\sigma & \models & \mathit{true} & \\
\sigma & \models & a & \text{iff} \quad a \in \sigma[0] \\
\sigma & \models & \neg a & \text{iff} \quad a \notin \sigma[0] \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ a } \sigma \models \varphi_2 \\
\sigma & \models & \varphi_1 \vee \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ alebo } \sigma \models \varphi_2 \\
\sigma & \models & \neg\varphi & \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models & \mathbf{F}\varphi & \text{iff} \quad \exists K \in \mathbb{N} : \sigma_k \models \varphi \\
\sigma & \models & \mathbf{G}\varphi & \text{iff} \quad \forall K \in \mathbb{N} : \sigma_k \models \varphi
\end{array}
$$

2.2 Overovanie modelov

Overovanie modelu je automatizovaná technika, ktorá vzhľadom na konečne stavový model systému a formálnu vlastnosť (vyjadrenú v temporálnej logike) systematicky kontroluje, či je vlastnosť v danom stave v modeli pravdivá alebo nepravdivá. Je to účinná technika na identifikáciu potenciálnych chýb návrhu a zvyšuje dôveru v správnosť návrhu systému. Bežne sa používa v oblasti hardvéru a softvéru: pri overovaní mikroprocesorov alebo softvéru vo vesmírnom priestore; v bezpečnostných protokoloch; v sektore dopravy (vlaky) [20]. Typickým príkladom konečného systému sú digitálne sekvenčné obvody a komunikačné protokoly [4].

Digitálny sekvenčný obvod je typ obvodu, ktorý pozostáva z kombinačnej logiky a pamäťových prvkov. Nazýva sa sekvenčný, pretože operácie sa vykonávajú postupne [7]. Komunikačné protokoly sú súbormi hardvérových alebo softvérových pravidiel, ktoré musia koncové body komunikácie dodržiavať, aby si mohli vymieňať informácie [15].

Keďže sa predpokladá konečnosť systému, sémantika systému je zvyčajne nejaký konečný automat, ktorý môže byť reprezentovaný *Kripkeho štruktúrou*.

Definícia 3 *Kripkeho štruktúra* M je štvorica [4]:

$$M = (S, I, T, L), \text{ kde}$$

- S je množina *stavov*,
- $I \subseteq S$ je množina *počiatočných stavov*,
- $T \subseteq S \times S$ je *prechodová funkcia*,
- $L : S \to P(A)$ je *označovacia funkcia*, kde A je množina *atomických výrokov* a *P(A)* označuje potenčnú množinu nad A.

Kripkeho štruktúra je štandardná reprezentácia skúmaného systému. Vytváranie takejto štruktúry z danej syntaktickej reprezentácie nemusí byť niekedy ľahké. Konkrétne, veľkosť opisu systému a veľkosť stavového priestoru môže byť odlišná. Napríklad, pri modelovaní sekvenčného obvodu s bránami a klopnými obvodmi môže byť stavový priestor exponenciálne väčší ako popis systému [4].

Jazyk Kripkeho štruktúry M, označený $L(M)$, je množina všetkých postupností slov $a_0 a_1 \ldots$, pre ktoré existuje beh $s_0 s_1 \ldots$, kde $(s_i, s_{i+1}) \in T$ a $L(s_i) = a_i$ pre všetky $i \geq 0$ [6]. Formula φ je platná, keď $L(M) \subseteq Words(\varphi)$.

Príklad 1. Predpokladajme vzájomné vylúčenie dvoch konkurenčných procesov pre zdieľaný zdroj [4]. Pseudokód pre tento príklad sa nachádza na obrázku 2.1. Vychádzame z predpokladu, že procesy sú vykonávané na jedinej výpočetnej jednotke. Čakací príkaz prepne

proces do režimu spánku. Ak sú všetky procesy v režime spánku, plánovač musí nájsť čakaciu podmienku, ktorá znovu aktivuje odpovedajúci proces. V prípade nesprávnosti všetkých čakacích podmienok sa systém zastaví.

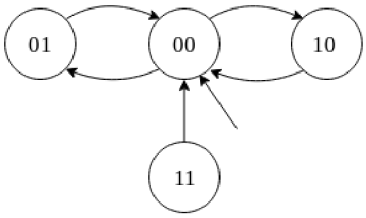
```
          process A                                  process B
              forever                                    forever
A.pc = 0          wait for B.pc = 0        B.pc = 0          wait for A.pc = 0
A.pc = 1          access shared resource   B.pc = 1          access shared resource
              end forever                                end forever
          end process                                process
```

Obr. 2.1: Prevzaté z [4]: Pseudokód pre dva procesy A a B

Každý proces má dva programové čítače s hodnotami 0 a 1, kde 1 predstavuje kritickú časť. Stav systému je dvojica programových čítačov, ktoré môžu byť zakódované ako binárny vektor $s \in S = \{0,1\}^2$ dĺžky dva. Prechodová funkcia pozostáva z niekoľkých možných prechodov podľa nasledujúcich pravidiel: ďalší stav s' je počiatočný stav 00, pokiaľ súčasný stav už nie je počiatočným stavom. Počiatočný stav môže prejsť tam a späť ku 01 aj 10. To znamená, že prechodová funkcia T môže byť reprezentovaná ako nasledujúca sada bitových reťazcov:

$$\{0100, 1000, 1100, 0001, 0010\}$$

Grafické znázornenie tohto príkladu vo forme Kripkeho štruktúry sa nachádza na obrázku 2.2. Počiatočný stav má vstupnú hranu bez zdroja. Ostatné hrany zodpovedajú jednému z piatich prechodov. Nedosiahnuteľný stav 11 je možné odstrániť až po vykonaní analýzy dosiahnuteľnosti, ktorá označí všetky dosiahnuteľné stavy. Preto je sekvencia 11, 00, 10,... platná cesta Kripkeho štruktúry.



Obr. 2.2: Prevzaté z [4]: Kripkeho štruktúra pre dva procesy

Systém z príkladu spĺňuje vlastnosť bezpečnosti v tom zmysle, že tieto dva procesy sa riadia vzájomnou vlastnosťou vylučovania, ktorá hovorí o tom, že najviac jeden proces môže byť v kritickej oblasti. Negácia tejto vlastnosti je, že stav, v ktorom sú oba procesy v kritickej oblasti, nie je dosiahnuteľný.

2.2.1 LTL overovanie modelov založené na automatoch

Aj keď je Kripkeho štruktúra tradičným spôsobom pri overovaní modelov, my budeme pracovať s prechodovým systémom (transition system) TS a LTL formulou φ. Problémom je overenie $TS \models \varphi$, ktoré popisuje splnenie formule vo všetkých slovách prijímaných TS. Prechodové systémy sú veľké a manuálne overovanie $TS \models \varphi$ je zložité, preto sú overovacie nástroje žiadúce a ponúkajú plne automatizovanú analýzu prechodového systému [2].

Vo všeobecnosti, prechodové systémy sa používajú na opis procesov s konfiguráciami reprezentujúcimi stavy a prechody. V podstate sa jedná o Büchi automat, ktorý je definovaný nižšie, kde všetky stavy sú implicitne koncové. Každá LTL formula φ môže byť reprezentovaná Büchi automatom, ktorý rozširuje koncept konečného automatu o nekonečné slová.

Vytvorený automat pre formulu je možné použiť na overovanie modelov. Základný algoritmus pre LTL overovania modelov funguje nasledovne:

1. formula je prevedená do Büchi automatu A_ξ, ktorý akceptuje jazyk negovanej formule, teda $Words(\neg\xi)$,
2. vytvorí sa produkt TS a automatu A_ξ (produkt reprezentuje prienik jazyka TS a A_ξ),
3. overí sa, či je jazyk produktu prázdny. TS splňuje ξ práve vtedy, keď má produkt prázdny jazyk.

V tejto práci sa zameriavame hlavne na krok 1, prevod formule ξ na automat.

Alternatívou k overovaniu modelov je run-time monitoring. Pokiaľ je ξ safety vlastnosť, potom je možné detekovať porušenie vlastnosti za behu systému. Vlastnosť je porušená, pokiaľ produkt prejde do koncového stavu.

Prevod LTL formulí do Büchi automatov je veľmi preskúmaná oblasť, kompletný prehľad nie je predmetom tejto práce. Sekcia bližšie popisuje dve základné techniky prevodu LTL formulí do Büchi automatu, a to prevod do generalizovaného nedeterministického Büchi automatu a transformácia Büchi automatu založeného na prechodoch na Büchi automat.

Prevod do generalizovaného nedeterministického Büchi automatu

Rozdiel medzi generalizovaným Büchi automatom a Büchi automatom je v prijímajúcej podmienke. Automat prijíma slovo, ak je aspoň jeden koncový stav z každej množiny koncových stavov navštevovaný nekonečne často. V generalizovanom nedeterministickom Büchi automate môžu existovať stavy s viacerými možnosťami prechodu pre rovnaký symbol [22].

Definícia 4 *Generalizovaný nedeterministický Büchi automat* (GNBA) je pätica [2]:

$$\mathcal{G} = (Q, \Sigma, \delta, Q_0, \mathcal{F}), \text{ kde}$$

- Q je konečný *stavový priestor*,
- Σ je *abeceda*,
- $Q_0 \subseteq Q$ je množina *počiatočných stavov*,
- $\delta : Q \times \Sigma \to 2^Q$ je *prechodová funkcia*,
- $\mathcal{F}$ je (pravdepobne prázdna) podmnožina 2^Q.

Prvky $\mathcal{F}$ sa nazývajú prijímajúce množiny. Prijímaný jazyk $\mathcal{L}_\omega(\mathcal{G})$ pozostáva zo všetkých nekonečných slov v $(2^{AP})^\omega$, ktoré majú aspoň jeden nekonečný beh $q_0 q_1 q_2 \ldots$ v $\mathcal{G}$ taký, že pre každú prijímajúcu množinu $F \in \mathcal{F}$ existuje nekonečne veľa indexov i, kde $q_i \in F$.

V nasledujúcej časti je opísaná jedna zo základných konštrukcií GNBA pre formulu.

Definícia 5 (Uzáver φ) Uzáver LTL formule φ je množina *closure(*φ*)* obsahujúca všetky podformule ψ danej formule a ich negácie $\neg\psi$. Uzáver je definovaný podľa nasledujúcich pravidiel [2]:

- $\varphi \in \mathit{closure}(\varphi)$
- $\varphi_1 \wedge \varphi_2 \in \mathit{closure}(\varphi) \rightarrow \varphi_1, \varphi_2 \in \mathit{closure}(\varphi)$
- $\varphi_1 \vee \varphi_2 \in \mathit{closure}(\varphi) \rightarrow \varphi_1, \varphi_2 \in \mathit{closure}(\varphi)$
- $X\ \varphi_1 \in \mathit{closure}(\varphi) \rightarrow \varphi_1 \in \mathit{closure}(\varphi)$

Množina formulí $B \subseteq \mathit{closure}(\varphi)$ je nazývaná *základná*, ak B je množina všetkých formulí $\psi \in \mathit{closure}(\varphi)$.

Definícia 6 (Základná množina formule) $B \subseteq \mathit{closure}(\varphi)$ je základná, ak je v súlade s výrokovou logikou, maximálne a lokálne v súlade s operátorom until [2].

1. B je v súlade s výrokovou logikou, napríklad pre všetky $\varphi_1 \wedge \varphi_2, \psi \in \mathit{closure}(\varphi)$:
   - $\varphi_1 \wedge \varphi_2 \in B \leftrightarrow \varphi_1 \in B$ a $\varphi_2 \in B$
   - $\psi \in B \rightarrow \neg\psi \notin B$
   - true $\in \mathit{closure}(\varphi) \rightarrow$ true $\in B$
2. B je lokálne v súlade s operátorom until, napríklad pre všetky $\varphi_1 \mathsf{U} \varphi_2 \in \mathit{closure}(\varphi)$:
   - $\varphi_2 \in B \rightarrow \varphi_1 \mathsf{U} \varphi_2 \in B$
   - $\varphi_1 \mathsf{U} \varphi_2 \in B$ a $\varphi_2 \notin B \rightarrow \varphi_1 \in B$
3. B je maximálne, napríklad pre všetky $\psi \in \mathit{closure}(\varphi)$:
   - $\psi \notin B \rightarrow \neg\psi \in B$

Pri konštrukcii GNBA nad 2^{AP} pre zadanú LTL formulu φ predpokladáme, že formula obsahuje operátory $\wedge, \neg$, $\mathbf{X}$ a U a odvodené operátory $\vee, \rightarrow$, $\mathbf{G}$, $\mathbf{F}$. Základná myšlienka konštrukcie $\mathcal{G}_\varphi$ je nasledovná:

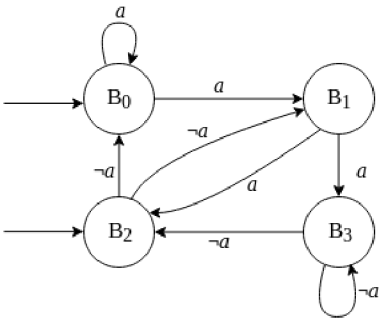
Teorém 1. *Pre akúkoľvek LTL formulu φ existuje GNBA $\mathcal{G}_\varphi$ nad abecedou 2^{AP} taký, že platí:*

- *Words(φ)* $= \mathcal{L}_\omega\ (\mathcal{G}_\varphi)$
- $(\mathcal{G}_\varphi)$ *môže byť skonštruované v čase a priestore* $2^{\mathcal{O}(|\varphi|)}$
- *počet prijímajúcich stavov $\mathcal{G}_\varphi$ je ohraničený* $\mathcal{O}(|\varphi|)$

Príklad 2. Príklad je zameraný na vytvorenie GNBA pre formulu $\varphi = \mathbf{X}a$. Stavy automatu patria do základnej množiny formule obsiahnutej v $\mathit{closure}(\varphi) = \{a, \mathbf{X}a, \neg a, \neg\mathbf{X}a\}$. Stavový priestor Q pozostáva z nasledujúcich základných množín:

$$\begin{array}{lllllll} B_0 & = & \{a, \mathbf{X}a\}, & B_1 & = & \{a, \neg\mathbf{X}a\} \\ B_2 & = & \{\neg a, \mathbf{X}a\}, & B_3 & = & \{\neg a, \neg\mathbf{X}a\}. \end{array}$$

Počiatočnými stavmi automatu sú základné množiny podľa pravidiel v definícii 6, tj. $Q_0 = \{B_0, B_2\}$. Po prijatí $\{a\}$ je možný prechod do B_0 aj do B_1, keďže oba tieto stavy obsahujú $\{a\}$. Rovnako je možné vykonať prechod z B_1 do B_2 a B_3. Totožne urobíme prechody aj pre ostatné stavy a výsledný GNBA sa nachádza na obrázku 2.3.



Obr. 2.3: Prevzaté z [2]: Generalizovaný nedeterministický Büchi automat pre $\varphi = \mathbf{X}a$

Transformácia Büchi automatu založeného na prechodoch

Generalizovaný Büchi automat založený na prechodoch (TGBA) nad abecedou Σ je Büchi automat s označeniami prechodov a generalizovanou akceptačnou podmienkou. Giannakopoulou a Lerda v práci [14] uvádzajú, že označenie prechodov namiesto stavov vedie k zmenšeniu veľkosti Büchi automatu, ktorý vznikol transformáciou TGBA.

Definícia 7 *Generalizovaný Büchi automat založený na prechodoch* je pätica [19]:

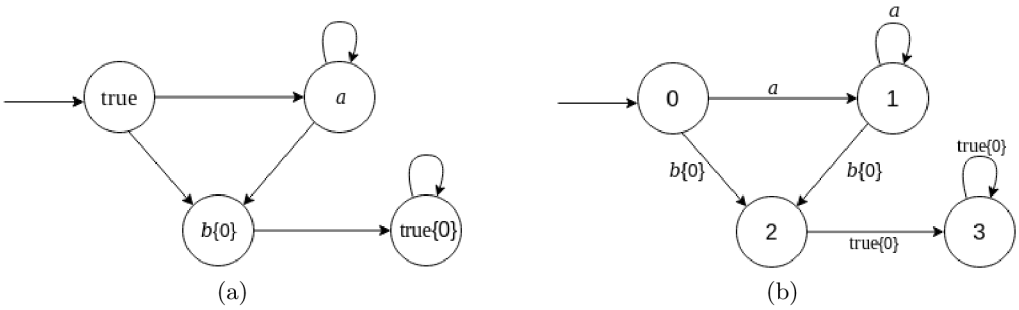
$$\mathrm{A} = (Q, \Sigma, \delta, I, F), \text{ kde}$$

- Q je konečná množina *stavov*,
- Σ je konečná *abeceda*,
- $\delta \subseteq Q \times (2^{\Sigma} \backslash \{\emptyset\}) \times 2^{F} \times Q$ je *prechodová funkcia*, kde každý prechod je neprázdna množina písmen abecedy a množina akceptačných podmienok.
- $I \subseteq Q$ je množina *počiatočných stavov*,
- $F = \{f_1, f_2, \ldots, f_n\}$ je konečná množina *akceptačných podmienok*.

Jazyk generalizovaného Büchi automatu založeného na prechodoch Prechod A je v tvare $(q_i, \alpha, F_i, q_{i+1})$, kde q_i je zdrojový stav a q_{i+1} je cieľový stav. α je podmienka, prechod môže byť vykonaný zo stavu q_i do stavu q_{i+1}, ak dôjde k prečítaniu písmena v α. F_i je množina akceptačných podmienok označených na tomto prechode. Množina prechodov označených f_i sa označuje ako $AccSet(f_i)$.

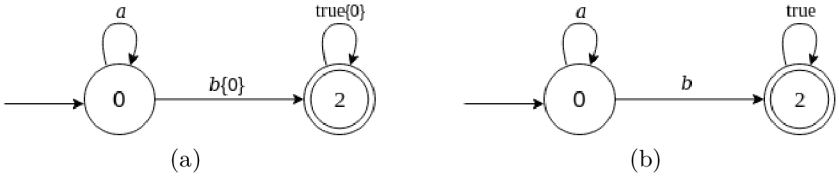
Beh σ nad slovom $u = u_0 u_1 \ldots \in \Sigma^{\omega}$ je nekonečná postupnosť $q_0, q_1, \ldots$ v Q^{ω} taká, že $q_0 \in I$ a $\forall_i \geq 0, \exists \alpha_i \in \Sigma'$, kde $u_i \in a_i$ a $(a_i, q_{i+1}) \in \delta(q_i)$. Beh σ je prijímajúci, ak pre každé $1 \leq j \leq n$ používa nekonečne veľa prechodov z $AccSet(f_j)$ [19].

Oproti GBA, ktoré majú označené stavy, TGBA má označenie na svojich prechodoch. Okrem toho aj akceptačná množina obsahuje prechody namiesto stavov [14]. Napríklad, automat na obrázku 2.4a je generovaný pre formulu $\varphi = a \mathbin{\mathsf{U}} b$. V tomto automate sú označené stavy a je možné ho transformovať do TGBA, ktorý sa nachádza na obrázku 2.4b. Označenie $\{0\}$ značí, že stav alebo prechod patrí do prijímacej sady 0.



Obr. 2.4: Prevzaté z [14]: Transformácia automatu na TGBA

Spojením stavov s rovnakým označením získavame TGBA na obrázku 2.5a. V ďalšom kroku je možný prevod na Büchi automat s označenými prechodmi a koncovým stavom 2.5b.



Obr. 2.5: Prevzaté z [14]: Transformácia TGBA na Büchi automat s koncovým stavom

2.2.2 LTL overovanie modelov s obmedzenými operátormi

Techniku popísanú v tejto časti prvýkrát navrhol Biere a spol. v roku 1999 [3]. Nerieši problém zložitosti overovaného modelu, ale experimenty ukázali, že dokáže vyriešiť veľa prípadov, kedy nie je možné použiť techniku založenú na binárne rozhodovacích diagramoch (BDD) [4]. BDD sú orientované acyklické grafy, ktoré uchovávajú zmenšenú formu pravdivostnej tabuľky booleovskej funkcie a využívajú slepé prehľadávanie do šírky, pri ktorom spotrebúvajú veľa pamäte [8]. Najväčšou výhodou obmedzeného overovania modelov (BMC) v porovnaní s BDD je priestorová efektívnosť, kedy niektoré booleovské funkcie nie je možné kódovať ako BDD [18].

Základnou myšlienkou overovania modelov s obmedzenými operátormi je hľadanie protipríkladu vo výpočte, ktorého dĺžka je obmedzená nejakým celým číslom k. Ak nie je nájdená žiadna chyba, k je inkrementované až do momentu dosiahnutia obmedzenia. Takéto obmedzenie je nazývané *prahom úplnosti* [4].

Pri riešení problémov je hľadaný plán, t.j. postupnosť krokov na vykonanie nejakej úlohy. V BMC je hľadanie plánu obmedzené na cesty s vopred určenými hranicami. Keďže LTL formule sú definované nad všetkými cestami, protipríkladom bude nájdenie stopy, ktorá im odporuje. Ak nájdeme takúto stopu, nazývame ju svedkom. Protipríkladom k $M \models \mathbf{F}q$ je, či existuje svedok pre $\mathbf{G}\neg q$. Budeme predpokladať, že formula je v negovanej normálnej forme, v ktorej sa negácie môžu vyskytovať len pred atomickými výrokmi. Túto formu formule získame použitím De-Morganových zákonov.

Podľa práce A.Biera [4] môžeme vychádzať z predpokladu, že aj keď prefix cesty je konečný, mohol by predstavovať nekonečnú cestu. Ak existuje cyklus, kedy sa z posledného

stavu dostaneme do niektorého z predchádzajúcich stavov 2.6b, takzvaný *spätný cyklus*, jedná sa o nekonečnú cestu.

(a) žiadny cyklus (b) *(k,l)*-cyklus

Obr. 2.6: Prevzaté z [4]: Prípady pre cestu s obmedzením

Pre definovanie sémantiky overovania modelu s obmedzenými operátormi sa používa k-cyklus, kedy sa predpokladá konečnosť prefixu cesty. Na určenie platnosti formule sú využité len prvé $k+1$ stavy. Ak je cesta k-cyklom, udržiava sa pôvodná LTL sémantika, pretože všetky informácie o tejto (nekonečnej) ceste sú obsiahnuté v dĺžke predpony k [4].

2.3 Zhodnotenie súčasných riešení

Súčasné nástroje pre prevod LTL formulí vytvárajú rozmerné automaty, kde je počet stavov závislý na čísle, ktoré odpovedá obmedzeniu. Pri zložitejšom systéme môže mať takýto automat niekoľko desiatok stavov, pričom niektoré zo stavov sú podobné. Sekcia bližšie popisuje existujúce nástroje.

SPIN

SPIN je veľmi populárny nástroj pre overovanie správnosti softvéru, ktorý bol vyvinutý v Bell Labs v skupine Unix vo výskumnom stredisku Computing Sciences Research Center, počnúc rokom 1980. Môže byť plne použitý pre overovanie modelov, ktorých vlastnosti sú vyjadrené pomocou LTL syntaxi.

Využíva programovací jazyk Promela, ktorý zahŕňa sekvenčné konštrukcie inšpirované príkazom Dijkstra [11], vplyvy komunikačných štruktúr a výrazy prevzaté z C. Pomocou extraktoru je možné odvodiť model Promela z kódu napísanom v jazyku C a overiť ho. Na kontrolu, či model programu vyhovuje určitej vlastnosti, sa vykonáva optimalizované slepé prehľadávanie do hĺbky [24].

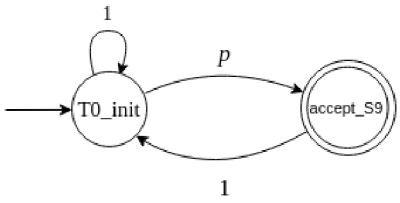
SPIN ponúka prevod LTL formulí do PROMELA never claim, ktoré sú používané na špecifikáciu konečného alebo nekonečného správania systému, ku ktorému by nikdy nemalo dôjsť [16]. Príklad PROMELA never claim pre LTL formulu $\mathbf{GF}p$ sa nachádza na obrázku 2.7a. Automat vygenerovaný pre túto formulu je na obrázku 2.7b.

Tauriainen and Heljanko vo svojej práci testovali verzie SPINu na náhodne vygenerovaných formulách [25]. LTL formule pozostávali typicky zo 4 až 7 symbolov s 5 atomickými výrokmi. Výsledok SPINu 3.3.9 bol neúspešný v prípade, že vstupná formula obsahovala operátor **X** alebo booleovské konštanty. Vo verzii 3.3.10 došlo k optimalizácii, kedy sa zmenšila veľkosť automatu. Práve táto verzia bola o niečo nestabilnejšia a zlyhala pri generovaní automatu.

Pre účely našej práce sa nepodarilo zistiť, ako previesť LTL formulu s obmedzenými operátormi do PROMELA syntaxi pomocou nástroja SPIN a možnosti `-f` ako tomu bolo v prípade formule na obrázku 2.7a.

```
$ spin -f '[]<>p'
never  {    /* []<>p */
T0_init:
     do
     :: ((p)) -> goto accept_S9
     :: (1) -> goto T0_init
     od;
accept_S9:
     do
     :: (1) -> goto T0_init
     od;
}
```

(a) PROMELA never claim pre $\mathbf{GF}p$



(b) Büchi automat pre $\mathbf{GF}p$

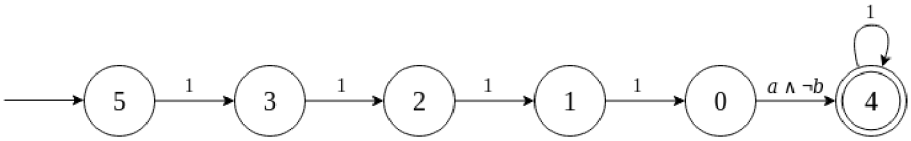
Obr. 2.7: Prevod LTL formule v nástroji SPIN

SPOT

SPOT je C++ knižnica pre overovanie modelov. Jednou zo špecifických vlastností je, že sa spolieha na Büchi automaty založené na prechodoch, ktoré umožňujú lepšie preklady LTL formulí.

Duret-Lutz a Poitrenaud vo svojej práci [12] uvádzajú, že SPOT dopĺňa algoritmus implementovaný pomocou BDD, ako ho predložil Couvreur [9], obohatený o techniky, ktoré predložili Sebastiani a Tonetta [23] s cieľom vytvoriť viac deterministický automat. SPOT taktiež implementuje ďalší algoritmus Couvreura [10], ktorého náplňou nie je vytvoriť malý automat. Tento automat je možné znázorniť pomocou BDD a jeho reprezentácia je efektívna na používanie.

SPOT dokáže vytvárať automaty pre LTL formule s obmedzenými operátormi. Avšak tieto automaty majú veľa stavov s rovnakými prechodmi. Ako príklad, na obrázku 2.8 sa nachádza Büchi automat pre $\varphi = \mathbf{F}_{\{=4\}}(a \wedge \neg b)$. Automat pozostáva až zo 6 stavov: z počiatočného a koncového stavu a ďalších štyroch stavov, ktorých počet odpovedá obmedzeniu daného temporálneho operátora.



Obr. 2.8: Büchi automat pre $\varphi = \mathbf{F}_{\{=4\}}(a \wedge \neg b)$

V prípade temporálneho operátora **G** s opakovaním, SPOT zostrojil totožný automat s automatom pre operátor **F** s opakovaním. Tieto dva operátory s opakovaním označuje ako syntaktický cukor pre rôzne typy opakovania operátora **X**. Napríklad, $\mathbf{F}_{\{=5\}}p$ je skrátený zápis formule $\mathbf{XXXXX}p$. Avšak v prípade operátora **G** by tomu tak nemalo byť.

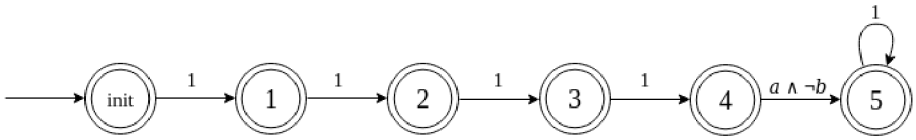
SPOT bol otestovaný aj na formulách od firmy Honeywell. Tie však obsahujú aritmetické a porovnávacie operátory, ktoré SPOT vyhodnotil ako chybové.

LTL2BA

LTL2BA softvér bol napísaný Denisom Oddoux a upravený Paulom Gastinom. Softvér je založený na práci *Fast LTL to Büchi Automata Translation* [13] . Denis Oddoux a Paul Gastin prezentujú algoritmus, ktorý generuje Büchi automat z LTL formule. Tento algorit-

mus vytvára veľmi slabý alternujúci co-Büchi automat a ten je následne transformovaný do Büchi automatu.

Rovnako ako nástroj SPIN, tak ani LTL2BA nevytvára automaty s opakovaním temporálnych operátorov, ale podporuje temporálny operátor **X**, pomocou ktorého môžeme formulu prepísať. Vytvorený automat je veľmi podobný s automatom vygenerovaným nástrojom SPOT, avšak všetky stavy má označené ako koncové. Ďalej nepodporuje atomický výrok s veľkým začiatočným písmenom.



Obr. 2.9: Büchi automat pre $\varphi = \mathbf{XXXX}(a \wedge \neg b)$

LTL3BA

LTL3BA softvér je založený na LTL2BA spomenutom vyššie. Modifikácie softvéru LTL2BA sú opísané v práci *LTL to Büchi Automata Translation: Fast and More Deterministic* a viedli k rýchlejšiemu prekladu a menším automatom [1].

Keďže LTL3BA je založený na LTL2BA, zápis temporálneho operátora s opakovaním nepodporuje. Otestovaná bola rovnaká formula ako pri LTL2BA a automat mal iba jeden koncový stav ako automat na obrázku 2.8. Odlišoval sa iba v číslovaní stavov.

Záver zhodnotenia súčasných riešení

Hlavným dôvodom vzniku tejto práce je, že uvedené nástroje pre prevod LTL formulí majú veľké problémy s obmedzenými operátormi. V niektorých prípadoch nebolo možné zostrojiť automat, pretože obmedzenie bolo vysoké. Z toho plynie, že je potrebný náš nový algoritmus.

Kapitola 3

Popis vlastnej práce

Cieľom práce je vytvoriť menšie automaty zredukovaním počtu stavov a pridaním čítača pre obmedzené operátory. Pri niektorých obecných formulách môžu nastať problémy pri prevode na automat, kedy temporálne operátory spôsobia zanorenie a vznikajú veľké automaty. Z tohto dôvodu je implementácia zameraná predovšetkým na prevod formulí poskytnutých firmou Honeywell formy $\mathbf{G}(\Psi)$. Pre Ψ platí nasledujúca gramatika:

$$\begin{array}{lcl} \Psi & := & \mathbf{G}(\varphi) \mid \mathbf{F}(\varphi) \mid \mathbf{X}(\varphi) \mid \mathbf{G}_{\{=n\}}(\varphi) \mid \mathbf{F}_{\{=n\}}(\varphi) \mid \mathbf{X}_{\{=n\}}(\varphi) \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi \\ \varphi & := & a \mid \neg\, a \mid a \wedge a \end{array} \tag{3.1}$$

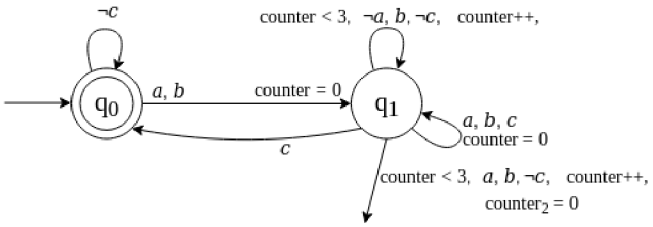
kde φ je stavová formula, t.j. formula bez temporálnych operátorov a n je celé kladné číslo odpovedajúce obmedzeniu. Po negácii $\mathbf{G}(\Psi)$ získavame $\xi = \mathbf{F}(\Psi')$[1], kde Ψ' je formula formy Ψ z gramatiky 3.1.

Vytvorený automat A_ξ je možné použiť ako run-time monitor spomenutý v časti 2.2.1. Tento automat prijíma negovanú vlastnosť, kedy sa zamedzí používaniu operátora $\mathbf{G}$, ktorý sa nachádza na začiatku formule a operátora until U. Tieto operátory by mohli viesť k vzniku veľkého automatu. Ďalej sa vyhneme používaniu implikácie, ktorej negáciu je možné vyjadriť konjunkciou, kde platí:

$$\neg(A \rightarrow B) \equiv (A \wedge \neg B)$$

Príklad 3. Príklad je zameraný na problém, ktorý vzniká pri operátore $\mathbf{G}$ a U. Obrázok 3.1 ukazuje automat pre formulu $\Psi = \mathbf{G}(a \rightarrow (b\ \mathsf{U}_{\{=3\}}\ c))$. Operátor $\mathsf{U}_{\{=3\}}$ s obmedzením 3 znamená, že b bude prečítané práve trikrát a následne dochádza k prečítaniu c. Avšak, spoločne s operátorom $\mathbf{G}$ vznikne zanorenie, ktoré môže viesť k vzniku veľkého automatu s viacerými čítačmi. Spomenuté zanorenie je naznačené pri opätovnom prečítaní a alebo b. Musí sa spustiť nová inštancia, čo znamená, že je potrebné inicializovať nový čítač $counter_i$ pre $i > 1$ a k pôvodnému automatu sa pridajú podobné stavy ako stavy q_0 a q_1.

[1] Formule poskytnuté firmou Honeywell sú jednoduchšie a nepoužitie operátora $\mathbf{G}$ využijeme na optimalizáciu pri vytváraní produktového automatu opísaného v sekcii 3.3.3.



Obr. 3.1: Automat pre $\Psi = \mathbf{G}(a \rightarrow (b \bigcup_{\{=3\}} c))$

Sekcia 3.1 vysvetľuje negáciu formulí. Ďalšia časť 3.2 popisuje prevod LTL formulí na Büchi automat. Na základe získaných poznatkov je v časti 3.3 popísaný spôsob implementácie.

3.1 Negácia formulí

Pre negáciu temporálnych operátov platí pravidlo duality, ktoré ukazuje, že operátor $\mathbf{X}$ je duálny voči sebe [2, str. 248]:

$$\begin{aligned} \neg\mathbf{G}\varphi &\equiv \mathbf{F}\neg\varphi \\ \neg\mathbf{F}\varphi &\equiv \mathbf{G}\neg\varphi \\ \neg\mathbf{X}\varphi &\equiv \mathbf{X}\neg\varphi \\ \neg\mathbf{G}_{\{=k\}}\varphi &\equiv \mathbf{F}_{\{=k\}}\neg\varphi \\ \neg\mathbf{F}_{\{=k\}}\varphi &\equiv \mathbf{G}_{\{=k\}}\neg\varphi \\ \neg\mathbf{X}_{\{=k\}}\varphi &\equiv \mathbf{X}_{\{=k\}}\neg\varphi \end{aligned}$$

Negácia formule $\mathbf{G}(\Psi)$ sa vykonáva v dvoch krokoch. V prvom kroku sa získa $\mathbf{F}(\neg\Psi)$ a následne pomocou DeMorganových zákonov dôjde k vytvoreniu Ψ' v negovanej normálnej forme, v ktorej sa negácie môžu vyskytovať iba pred atomickými výrokmi.

$$\neg(\mathbf{G}(\Psi)) \equiv \mathbf{F}\neg(\Psi) \equiv \mathbf{F}(\Psi')$$

Ψ' je booleovská kombinácia, z ktorej je zostrojený automat v sekcii 3.2. Pre logické operácie konjunkciu $\wedge$ a disjunkciu $\vee$ sa zostavia automatové konštrukcie pre zjednotenie a prienik, ktoré sa dajú zobecniť na automaty s čítačmi. Následne sa vytvorí automat pre $\mathbf{F}(\Psi')$, ktorý môže na začiatku prečítať ľubovoľné množstvo znakov a nedeterministicky prejsť do počiatočného stavu automatu pre Ψ'.

Príklad 4. Príklad vysvetľuje negáciu formule $\xi = \mathbf{G}(\mathbf{F}a \rightarrow \mathbf{G}b)$ sformulovanej podľa spomenutej gramatiky 3.1. Negácia prebieha v dvoch krokoch za použitia pravidla duality. V prvom kroku dochádza k vytvoreniu $\mathbf{F}\neg(\mathbf{F}a \rightarrow \mathbf{G}b)$. V druhom kroku sa negácia vyskytuje iba pred atomickými výrokmi $\mathbf{F}(\mathbf{F}a \wedge \mathbf{F}\neg b)$.

3.2 Konštrukcia automatu pre formulu

K prevodu negovanej formule na automat s čítačmi dochádza v prípade, že formula obsahuje obmedzené operátory. V opačnom prípade je automat vytvorený pomocou automatových konštrukcií pre prienik a zjednotenie. Automaty vytvárame indukciou ku štruktúre formule.

V tejto sekcii je nižšie popísaná formálna definícia čítačového Büchi automatu a jazyku, ktorý prijíma. Ďalej sú popísané dôležité operácie, ktoré používame pri vytváraní automatu.

V časti 3.2.1 je popísaná konštrukcia automatu pre LTL formulu bez obmedzených operátorov. Časť 3.2.2 približuje prevod LTL formule s obmedzenými operátormi na čítačový Büchi automat.

Čítačový Büchi automat

Definícia 8 *Čítačový Büchi automat* je šestica:

$$A = (Q, C, \Sigma, \delta, I, F), \text{ kde}$$

- Q je konečná množina *stavov*,
- C je konečná množina *čítačov*,
- Σ je konečná *vstupná abeceda*,
- I je *množina počiatočných stavov*,
- $F \subseteq Q$ je množina *koncových stavov*,
- δ je množina *prechodov* v tvare (q, a, φ, f, r), kde $q \in Q$ je *zdrojový stav*, $r \in Q$ je *cieľový stav*, $a \in \Sigma$ je *vstupný symbol*, φ je *podmienka* a f je *aktualizácia čítača*.

Podmienka φ je konjunkciou *atomických podmienok* v tvare $c == d$, $c \geq d$, $c == k$, $c \geq k$ alebo $k * c \leq l * d$, kde $c, d \in C$ sú čítače a $k, l \in \mathbb{N}$ je konštanta. Aktualizácia čítača môže byť v tvare $c := k$ alebo c++, kde $c \in C$ a $k \in \mathbb{N}$, f môže pozostávať z najviac jednej aktualizácie čítača obsahujúcej c pre každé $c \in C$.

Jednoduchý čítačový Büchi automat je automat, v ktorom sa nedá vykonať prechod z koncového stavu do nekoncového. Naša konštrukcia vytvára z jednoduchých formulí jednoduché Büchi automaty.

Jazyk čítačového Büchi automatu. *Konfigurácia* automatu A je dvojica (q, v), kde $q \in Q$ a v je *protihodnota* $v : C \to \mathbb{N}$, ktorá priraďuje každému čítaču hodnotu z $\mathbb{N}$. *Počiatočná konfigurácia* je konfigurácia (q_0, v_0), kde $v_0(c) = 0$ pre všetky $c \in C$ a $q_0 \in I$.

Hovoríme, že konfigurácia (q', v') je *a-následník* konfigurácie (q, v), ak δ obsahuje prechod (q, a, φ, f, r) s nasledujúcimi vlastnosťami:

- φ je uspokojené vo v, kde uspokojenie podmienky je definované obvyklým spôsobom.
- v' je získané z v aktualizáciou hodnôt čítača podľa f. A to, c++ znamená zvyšovanie hodnôt c, $c := k$ je nastavenie hodnoty c na konštantu k, a ak f neaktualizuje čítač c, potom hodnota c ostáva nezmenená.

Beh A nad slovom $a_1 a_2 a_3 \ldots \in \Sigma^\omega$ je postupnosť konfigurácií (q_0, v_0), (q_1, v_1), (q_2, v_2), ..., kde (q_0, v_0) je počiatočná konfigurácia a pre každé $i \geq 0$, (q_i, v_i) je a_i*-následník* (q_{i-1}, v_{i-1}). Beh je *prijímajúci*, ak nekonečne často dôjde k navštíveniu jedného z koncových stavov. *Jazyk* A, označený $L(A)$, je množina slov, pre ktoré má A prijímajúci beh.

Konštrukcia produktu pre zjednotenie a prienik

Táto sekcia opisuje konštrukciu produktu vzniknutého prienikom alebo zjednotením jazykov nad čítačovým Büchi automatom.

Nech $A_1 = (Q_1, C_1, \Sigma, \delta_1, q_0^1, F_1)$ a $A_2 = (Q_2, C_2, \Sigma, \delta_2, q_0^2, F_2)$ sú 2 čítačové Büchi automaty. Predpokladáme bez straty všeobecnosti, že $C_1 \cap C_2 = \emptyset$ a $Q_1 \cap Q_2 = \emptyset$ (ak prienik nie je prázdny, čítače a stavy jedného z automatov môžu byť jednoducho premenované).

Automat prijímajúci prienik jazykov A_1 a A_2 je automat $A = (Q, C, \Sigma, \delta^\times, F)$, kde $Q = Q_1 \times Q_2$, $C = C_1 \cup C_2$, $F = \{(q_1, q_2) \in Q \mid q_1 \in F_1 \wedge q_2 \in F_2\}$ a $\delta^\times$ je prechodová funkcia, ktorá je produktom prechodových funkcií a obsahuje prechod $((q_1, q_2), a, \varphi_1 \wedge \varphi_2, f_1 \cup f_2, (r_1, r_2))$, ak a iba ak existujú prechody $(q_1, a, \varphi_1, f_1, r_1) \in \delta_1$ a $(q_2, a, \varphi_2, f_2, r_2) \in \delta_2$.

Automat prijímajúci zjednotenie jazykov A_1 a A_2 je automat $A = (Q, C, \Sigma, \delta, F)$, kde $Q = Q_1 \cup Q_2$, $C = C_1 \cup C_2$, $F = F_1 \cup F_2$ a $\delta = \delta_1 \cup \delta_2$.

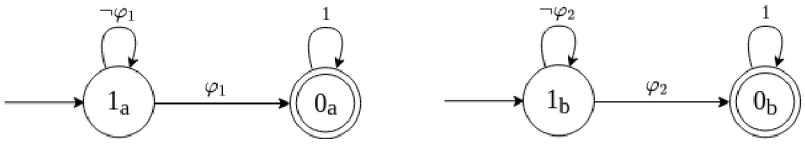
Spomenutý prienik automatov nefunguje pre všeobecný čítačový Büchi automat, ale správne funguje za predpokladu, že A_1 a A_2 sú jednoduché čítačové Büchi automaty. Automaty vytvorené z našich obmedzených temporálnych operátorov odpovedajú tejto vlastnosti, ako aj automaty vytvorené z ich booleovských kombinácií. Konštrukcia by sa mohla ľahko zobecniť na štandardnú všeobecnú konštrukciu prieniku Büchi automatu (ktorá sa pri splnení koncového stavu jedného alebo druhého vstupného automatu strieda medzi dvoma verziami δ).

3.2.1 Automaty pre LTL formule bez obmedzenia

Rovnako ako pri negácii, tak aj pri vytváraní automatu sa postupuje v niekoľkých krokoch. V prvom kroku dochádza k vytvoreniu automatov pre podformule. Tieto automaty väčšinou pozostávajú z maximálne 2 stavov a nie sú ťažké na konštrukciu. Sekcia popisuje vytvorenie automatových konštrukcií pre prienik a zjednotenie.

Prienik automatov

Majme negovanú formulu $\xi = \mathbf{F}(\Psi)$ sformulovanú podľa gramatiky 3.1, kde Ψ je $\mathbf{F}\varphi_1 \wedge \mathbf{F}\varphi_2$, φ_1 a φ_2 sú stavové formule. V prvom kroku sa vytvoria automaty pre podformule $\mathbf{F}\varphi_1$ a $\mathbf{F}\varphi_2$. Podformula $\mathbf{F}\varphi_1$ opisuje, že φ_1 sa objaví aspoň raz. Podobne to platí aj pre druhú podformulu, ktorá opisuje prijatie φ_2. Obrázok 3.2 zobrazuje automaty pre obe podformule. Hrana s označením 1 značí prijatie ľubovoľných slov.



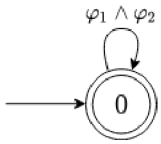
(a) Büchi automat $M1$ pre $\mathbf{F}\varphi_1$ (b) Büchi automat $M2$ pre $\mathbf{F}\varphi_2$

Obr. 3.2: Büchi automat pre fragmenty formule Ψ

V prípade, že by Ψ pozostávalo iba z temporálneho operátora $\mathbf{G}$ a booleovského operátora AND, automaty pre podformule by sa nevytvárali kvôli platnosti pravidla distributivity [2, str. 248]:

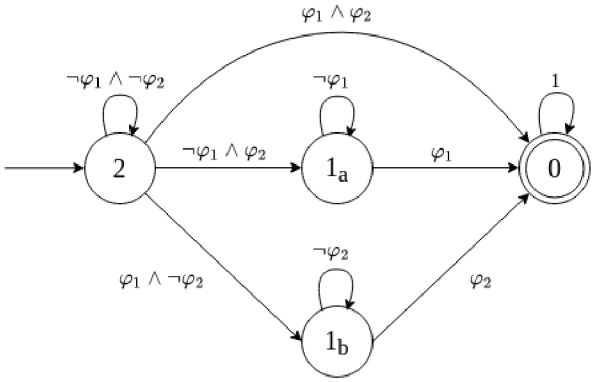
$$\mathbf{G}\varphi_1 \wedge \mathbf{G}\varphi_2 \equiv \mathbf{G}(\varphi_1 \wedge \varphi_2)$$

Z tohto dôvodu bude automat na obrázku 3.3 pre formulu $\mathbf{G}(\varphi_1 \wedge \varphi_2)$ tvorený z rovnakého počtu stavov a prechodov ako pre formulu $\mathbf{G}\varphi_1 \wedge \mathbf{G}\varphi_2$.



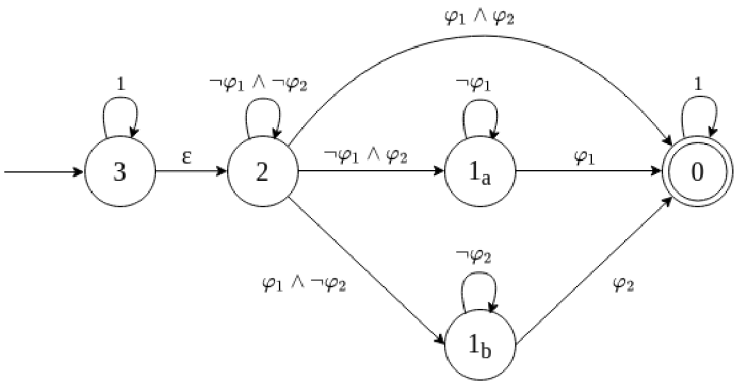
Obr. 3.3: Pravidlo distributivity
$\mathbf{G}\varphi_1 \wedge \mathbf{G}\varphi_2 \equiv \mathbf{G}(\varphi_1 \wedge \varphi_2)$

V druhom kroku je automat vytvorený pomocou automatových konštrukcií pre prienik spomenutých v časti 3.2. Ak $M1$ a $M2$ sú automaty, ich prienikom sú stavy, ktoré sú stavmi $M1$ a $M2$. Prechodové funkcie ostávajú rovnaké a koncové stavy sú všetky stavy, ktoré sú koncovými v $M1$ a $M2$. Automat je zobrazený na obrázku 3.4.



Obr. 3.4: Büchi automat M pre $\Psi = \mathbf{F}\varphi_1 \wedge \mathbf{F}\varphi_2$

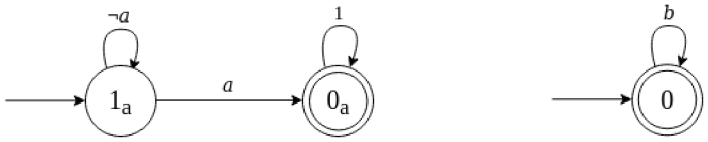
V poslednom kroku sa vytvorí automat pre $\mathbf{F}(\Psi)$ zobrazený na obrázku 3.5, ktorý môže prijať ľubovoľné množstvo slov a následne nedeterministicky prejsť do počiatočného stavu automatu M pre Ψ.



Obr. 3.5: Büchi automat pre $\xi = \mathbf{F}(\Psi)$

Príklad 5. Príklad popisuje vytvorenie Büchi automatu pre konkrétnu formulu s operátorom $\mathbf{F}$ a $\mathbf{G}$. Nech $\Psi = \mathbf{F}a \wedge \mathbf{G}b$ je negovaná LTL formula. Formula pozostáva z dvoch rozdielnych

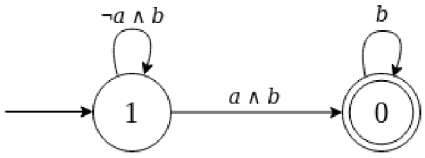
temporálnych operátorov. Automat pre podformulu $\mathbf{F}a$ sa nachádza na obrázku 3.6a, kde sa očakáva objavenie a aspoň raz. Obrázok 3.6b zobrazuje automat ako pre fragment $\mathbf{G}b$.



(a) Büchi automat $M1$ pre $\mathbf{F}a$ (b) Büchi automat $M2$ pre $\mathbf{F}b$

Obr. 3.6: Büchi automat pre fragmenty formule Ψ

Büchi automat vytvorený prienikom automatov pre fragmenty sa nachádza na obrázku 3.7. Koncový stav je pre oba automaty rovnaký, počiatočným stavom je stav automatu pre fragment $\mathbf{F}a$. Platnosť stavovej formule b sa očakáva vždy a preto je súčasťou každého prechodu.



Obr. 3.7: Büchi automat pre $\Psi = \mathbf{F}a \wedge \mathbf{G}b$

Zjednotenie automatov

Rovnako ako pri prieniku automatov 3.2.1 je automat vytvorený v 3 krokoch.

Majme negovanú formulu $\xi = \mathbf{F}(\Psi)$ sformulovanú podľa gramatiky 3.1, kde Ψ je $\mathbf{G}\varphi_1 \vee \mathbf{G}\varphi_2$, φ_1 a φ_2 sú stavové formule. Formula je tvorená dvoma fragmentmi $\mathbf{G}\varphi_1$ a $\mathbf{G}\varphi_2$. Fragment $\mathbf{G}\varphi_1$ opisuje, že φ_1 platí neustále. Podobne to platí aj pre druhý fragment. Obrázok 3.8 zobrazuje automaty pre oba fragmenty.

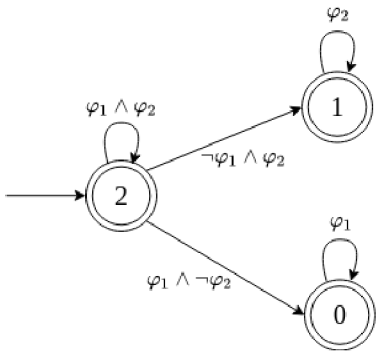
(a) Büchi automat $M3$ pre $\mathbf{G}\varphi_1$ (b) Büchi automat $M4$ pre $\mathbf{G}\varphi_2$

Obr. 3.8: Büchi automaty pre fragmenty formule $\mathbf{G}\varphi_1$ a $\mathbf{G}\varphi_2$

Ak by Ψ pozostávalo iba z temporálneho operátora $\mathbf{F}$ a booleovského operátora OR, k vytvoreniu automatov pre fragmenty nedôjde, pretože tu platí pravidlo distributivity [2, str. 248]:

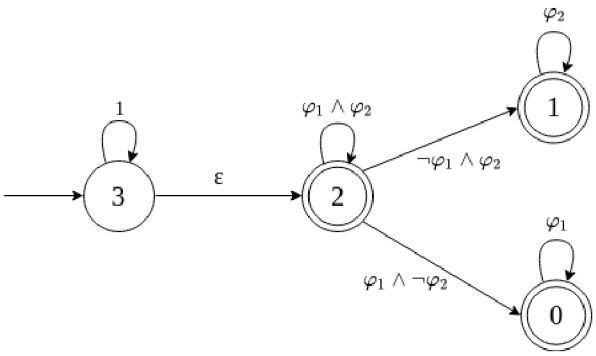
$$\mathbf{F}a \vee \mathbf{F}\neg b \equiv \mathbf{F}(a \vee \neg b)$$

V 2. kroku vznikne automat zjednotením $M3$ a $M4$ podľa 3.2. Predpokladáme, že porušenie platnosti formule nastane splnením $\mathbf{G}\varphi_1$ alebo $\mathbf{G}\varphi_2$, prípadne oboch naraz. Preto automat na obrázku 3.9 má všetky stavy koncové.



Obr. 3.9: Büchi automat pre $\mathbf{G}\varphi_1 \vee \mathbf{G}\varphi_2$

V poslednom kroku sa vytvorí automat pre $\mathbf{F}(\Psi)$ zobrazený na obrázku 3.5, ktorý môže prijať ľubovoľné množstvo slov a následne nedeterministicky prejsť do počiatočného stavu automatu pre Ψ 3.10.



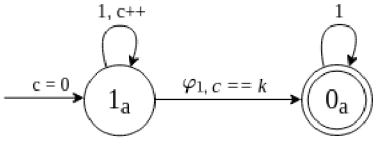
Obr. 3.10: Büchi automat pre $\xi = \mathbf{F}(\Psi)$

3.2.2 Automaty pre LTL formule s obmedzením

Pri prevode LTL formule s obmedzenými operátormi vytvárame čítačový Büchi automat. Formule poskytnuté firmou Honeywell majú rovnaké obmedzenie temporálnych operátorov takmer vo väčšine prípadov, čo umožňuje zjednodušiť formulu a vytvoriť automat s malým počtom stavov. Konštrukcia automatu je objasnená na obecnom príklade.

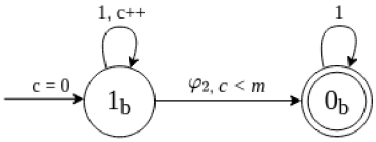
Majme negovanú formulu $\xi = \mathbf{F}(\Psi)$, kde Ψ je $\mathbf{G}_{\{=k\}}\varphi_1 \wedge \mathbf{F}_{\{=m\}}\varphi_2$, φ_1 a φ_2 sú stavové formule, $k, m \in \mathbb{N}$ sú konštanty, kde $k \leq m$. V prvom kroku sa vytvoria automaty pre fragmenty formule.

Pre fragment $\mathbf{G}_{\{=k\}}\varphi_1$ platí, že počas k krokov má dôjsť k prijatiu φ_1. V prípade nepoužitia negácie by bol automat vytvorený pre $\mathbf{F}_{\{=k\}}\neg\varphi_1$ a vieme, že po k krokoch od začiatku by malo byť prijaté $\neg\varphi_1$. Keďže pracujeme s negáciou formule, zisťujeme, kedy sa poruší platnosť formule. To znamená, že ak po k krokoch dôjde k prijatiu φ_1, môžeme povedať, že formula nie je platná, pretože sa očakáva platnosť $\neg\varphi_1$, ale došlo k prijatiu negácie. Obrázok 3.11 ukazuje vytvorený automat pre $\mathbf{G}_{\{=k\}}\varphi_1$. Čítač má označenie c.



Obr. 3.11: Čítačový Büchi automat pre $\mathbf{G}_{\{=k\}}\varphi_1$

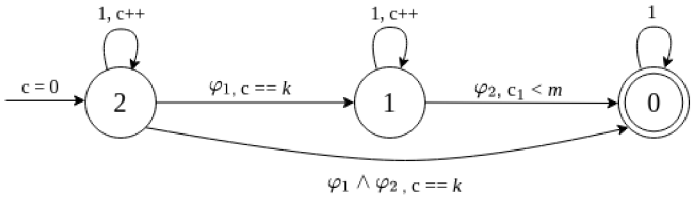
Obrázok 3.12 zobrazuje automat pre fragment $\mathbf{F}_{\{=m\}}\varphi_2$. Fragment vznikol negáciou $\mathbf{G}_{\{=m\}}\neg\varphi_2$, ktorý hovorí o platnosti $\neg\varphi_2$ počas m krokov. To znamená, že v prípade prijatia φ_2 do m krokov je formula neplatná.



Obr. 3.12: Čítačový Büchi automat pre $\mathbf{F}_{\{=m\}}\varphi_2$

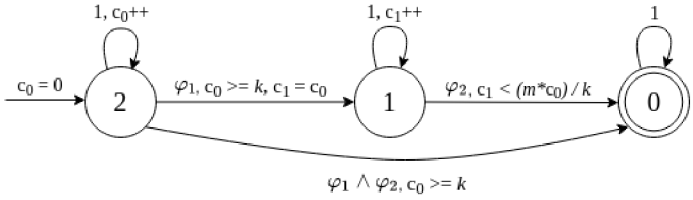
V počiatočnom stave automatu dochádza k inicializácii čítača na hodnotu 0. V tomto stave automat prijíma ľubovoľné slová a navyšuje sa hodnota čítača o 1. Prijímanie ľubovoľných slov je naznačených hodnotou 1 na prechode.

Zobrazený automat 3.13 vznikol automatovou konštrukciou pre prienik.



Obr. 3.13: Čítačový Büchi automat pre $\Psi = \mathbf{G}_{\{=k\}}\varphi_1 \wedge \mathbf{F}_{\{=m\}}\varphi_2$

Základná konštrukcia automatu pre $\mathbf{F}(\Psi)$ by mala byť rovnaká ako v prípade LTL formulí bez obmedzenia 3.2.1. Aby sme sa vyhli technickým komplikáciám s ϵ-*prechodmi* pri vytváraní produktu formule so systémom, ktorý popisujeme v sekcii 3.3.3, volíme alternatívnu konštrukciu, ktorá ϵ-*prechody* nevytvára. Pre $\mathbf{F}(\Psi)$ platí, že niekedy v budúcnosti musí nastať moment, kedy platí Ψ. To znamená, že k prijatiu φ_1 môže dôjsť po $k + x$ krokoch, kde $x \geq 0$. Z tohto dôvodu je potrebné upraviť aj druhú podmienku, aby pomer hodnôt medzi čítačmi ostal zachovaný. Čítačový Büchi automat pre $\xi = \mathbf{F}(\Psi)$ sa nachádza na obrázku 3.14.



Obr. 3.14: Čítačový Büchi automat pre $\xi = \mathbf{F}(\Psi)$

V prípade, že obmedzenie viacerých temporálnych operátorov rôzneho typu je rovnaké, dôjde k prepísanie formule podľa pravidiel distributivity a s operátorom **F** sa zachádza rovnako ako s operátorom **G**. Pri operátore $\mathbf{G}_{\{=k\}}$ je prechod vykonaný za podmienky, že hodnota čítača je rovná obmedzeniu k. Pri operátore $\mathbf{F}_{\{=m\}}$ je prechod možný v prípade, že hodnota čítača je menšia ako obmedzenie m. Hodnota čítača nemôže byť rovná a zároveň aj menšia ako obmedzenie.

Príklad 6. Príklad bližšie opisuje vytvorenie čítačového Büchi automatu pre formulu ξ s rovnakým obmedzením temporálnych operátorov odlišných typov.
Nech $\xi = \mathbf{F}((\mathbf{G}_{\{=2\}}a \wedge \mathbf{F}_{\{=2\}}b) \wedge \mathbf{G}_{\{=2\}}\ \mathbf{G}_{\{=2\}}\neg d)$ je negovaná LTL formula. Formulu môžeme prepísať nasledovne:

$$\xi = \mathbf{F}((\mathbf{G}_{\{=2\}}a \wedge \mathbf{F}_{\{=2\}}b) \wedge \mathbf{G}_{\{=4\}}\neg d)$$

Opakovanie pri operátore **G** je možné sčítať. Pre operátory bez opakovania platí pravidlo idempotencie [2, str. 248]:

$$\begin{aligned} \mathbf{GG}\varphi &\equiv \mathbf{G}\varphi \\ \mathbf{FF}\varphi &\equiv \mathbf{F}\varphi \end{aligned}$$

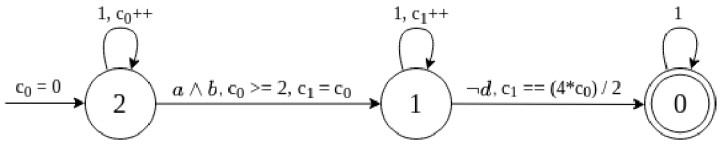
Pri temporálnom operátore **G** a **F** je rovnaké opakovanie. S operátorom **F** sa zachádza rovnako ako s operátorom **G** a podformule sa prepíšu na $\mathbf{G}_{\{=2\}}(a \wedge b)$ podľa pravidla distributivity.

Pre jednotlivé podformule sa vytvoria automaty zobrazené na obrázku 3.15.

(a) Čítačový Büchi automat pre $\mathbf{G}_{\{=2\}}(a \wedge b)$ (b) Čítačový Büchi automat pre $\mathbf{G}_{\{=4\}}\neg d$

Obr. 3.15: Čítačový Büchi automaty pre podformule

Rovnakým spôsobom ako pri obecnej formuli je vytvorený čítačový Büchi automat 3.13, ktorý má zmenené podmienky na prechodoch.



Obr. 3.16: Čítačový Büchi automat pre $\mathbf{F}((\mathbf{G}_{\{=2\}}(a \wedge b) \wedge \mathbf{G}_{\{=4\}}\neg d)$

3.3 Implementácia

Táto kapitola opisuje implementáciu prevodu LTL formulí s obmedzenými operátormi do automatu s čítačmi v jazyku C++. Vytvorený nástroj LTL2CBA je zameraný predovšetkým na prevod formulí poskytnutých firmou Honeywell sformulovaných podľa gramatiky

3.1. LTL2CBA načíta formulu vo forme reťazca, pre ktorú vytvorí reprezentáciu v podobe syntaktického stromu. Ten použije na vytvorenie negácie opäť vo forme reťazca, ktorý prevedie do stromu použitého na vytvorenie automatu pre negovanú formulu. Časť 3.3.1 popisuje spôsob, akým sa vytvorí negácia formule. Ďalej v časti 3.3.2 je popísaný prevod LTL formule na automat. Nakoniec je v 3.3.3 priblížený spôsob vytvárania produktového automatu pre vytvorený automat a skúmaný systém.

3.3.1 Vytvorenie negácie

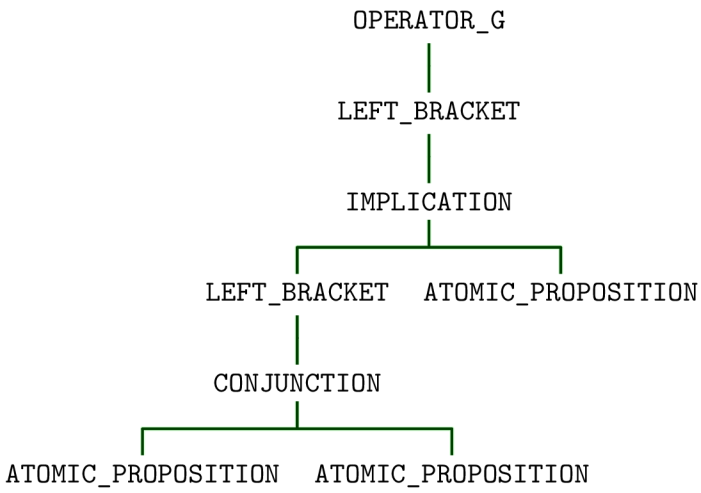
Podľa sekcie 3.1 vytvárame automat, ktorý prijíma negovanú vlastnosť, kedy formulu negujeme v dvoch krokoch. V tejto časti je popísaný spôsob reprezentácie formule ako stromu, ktorý je použitý na vytvorenie negácie.

Spracovanie

LTL formula je vo forme reťazca, ktorý reprezentuje vlastnosti systému. Prevod reťazca na syntaktický strom zaisťuje trieda `parse_formula`, ktorá prechádza jednotlivé znaky v reťazci a určuje ich typ: temporálne operátory, atomické výroky, zátvorky a booleovské operátory. LTL formule od firmy Honeywell obsahujú aritmetické a porovnávacie operátory a funkcie z knižníc C++. Tieto funkcie sú označené ako kľúčové slová a vyžaduje sa použitie zátvoriek.

Pre reprezentáciu formule slúži trieda `syntax_tree`, ktorá formulu chápe ako strom, kde v listoch sú atomické výroky, zátvorky alebo operátory. Zátvorky a booleovské operátory určujú, na ktoré miesto v strome bude pridaný nový uzol. V prípade kombinácie viacerých booleovských operátorov je potrebné použitie zátvoriek. Každý uzol si uchováva informácie o svojom type, opakovaní, negácii, potomkoch a obsahuje ukazateľa na predchádzajúci uzol.

Príklad 7. Príklad slúži na znázornenie syntaktického stromu pre formulu $\mathbf{G}((a \wedge \neg b) \rightarrow c)$. Formula obsahuje dva odlišné booleovské operátory, takže prednosť jednotlivých operátorov určujú zátvorky. Zobrazené sú len typy uzla jednotlivých častí formule:



Pre pravú zátvorku nie je vytváraný uzol. Informácie o výskyte pravej zátvorky sú ukladané vo forme príznaku v uzli ľavej zátvorky. Pre znak G je vytvorený uzol reprezentujúci operátor **G**. Všetky temporálne operátory je potrebné písať s veľkým písmenom, aby bolo možné odlíšiť operátor od atomického výroku.

Trieda `syntax_tree` obsahuje aj metódy na mazanie celého stromu alebo len jedného uzla. Tieto metódy sú využívané predovšetkým pri vytváraní automatu. Metóda `new_tree` vytvára z pôvodného stromu strom, ktorý na začiatku obsahuje uzly s operátorom **G**, ak sú prítomné, napríklad pre strom z formule $\mathbf{F}a \vee \mathbf{G}b \vee \mathbf{F}c$ je vytvorený strom reprezentujúci formulu $\mathbf{G}a \vee \mathbf{F}b \vee \mathbf{F}c$. Poprehadzovanie operátorov je možné z dôvodu platnosti vlastnosti komutativity pre logické operácie, kedy nezáleží na poradí operandov.

Takto vytvorený syntaktický strom je použitý na vytvorenie negácie v negovanej normálnej forme. Pomocou funkcie `negate_formula` v triede `formula` je zostavený reťazec, ktorý je negáciou pôvodnej LTL formule. Pri vytváraní stromu z negovanej formule dochádza k upraveniu formule ako bolo popísané v príklade 6. Taktiež sa formula upravuje podľa pravidiel distributivity [2, str. 248], kde platí:

$$\begin{aligned} \mathbf{G}a \wedge \mathbf{G}b &\equiv \mathbf{G}(a \wedge b) \\ \mathbf{F}a \vee \mathbf{F}b &\equiv \mathbf{F}(a \vee b) \\ \mathbf{X}a \wedge \mathbf{X}b &\equiv \mathbf{X}(a \wedge b) \\ \mathbf{X}a \vee \mathbf{X}b &\equiv \mathbf{X}(a \vee b) \end{aligned}$$

3.3.2 Prevod LTL formule na automat

Na prevod LTL formule na Büchi automat je implementovaný algoritmus podľa sekcie 3.2. Pre jednotlivé fragmenty formule sú vytvorené automaty a následne pomocou automatových konštrukcií pre zjednotenie a prienik je vytvorený Büchi automat pre $\mathbf{F}(\Psi)$. V tejto sekcii je bližšie popísané vytváranie automatov pre rôzne kombinácie temporálnych a booleovských operátorov.

V triede `automaton` je implementovaná metóda `create` na vytvorenie automatu. Implementovaný je prevod formulí s obmedzením temporálnych operátorov aj bez neho.

Z pôvodnej formule je zostavený strom, z ktorého je možné vytvoriť negovanú formulu v podobe reťazca. Tento reťazec je následne prevedený na syntaktický strom použitý pri konštrukcii automatu. Keďže pri disjunkcii a konjunkcii sa negujú oba jednoduché výroky a následne sú spojené operátorom odpovedajúcim negácii, pri implikácii sa predpoklad ponechá a tvrdenie sa zneguje. Práve z dôvodu, kedy sa pri implikácii neguje len tvrdenie, je opätovne vytvorený strom pre negáciu, čím sa predišlo zložitosti programu v podobe rozsiahlych vetiev s podmienenými príkazmi.

Dôležitou časťou tejto metódy je inicializácia zoznamu uzlov na spracovanie. V triede `parseast` sa vytvorí zoznam, ktorý obsahuje jednotlivé fragmenty formule. V prípade, že formula obsahuje rôzne booleovské operátory alebo booleovský operátor s viacej ako dvoma potomkami, tak v závislosti od počtu temporálnych operátorov a atomických výrokov sú vytvorené označenia prechodov.

LTL formule bez obmedzených operátorov Pri vytváraní reprezentácie formule ako stromu sa nastavuje príznak, či formula pozostáva z obmedzených operátorov. Ak formula takéto operátory nemá, pri konštrukcii automatu je dodržaný postup zo sekcie 3.2.1. V prípade temporálneho operátora **F** v príklade 5 obrázok 3.6a vzniká hrana s negáciou atomického výroku, kedy sa nevykonáva prechod do iného stavu a hrana s nenegovaným atomickým výrokom. Atomický výrok operátora **G** ostáva nezmenený. Pre značenie týchto hrán sa uchovávajú reťazce, ktoré reprezentujú atomické výroky.

Po vytvorení označení hrán sa prechádza k vytvoreniu automatu. Pre každé označenie je vytvorený stav, ak takýto stav ešte neexistuje a je označený ako zdrojový. Každý stav

si uchováva informácie o označení, prvej a poslednej odchádzajúcej hrane a príznak, či je tento stav aj koncovým stavom. Následne je hľadaný cieľový stav podľa jednoznačného identifikátora uzla. Pri vytváraní označenia prechodov má uzol s negovaným atomickým výrokom rovnaký identifikátor ako uzol nenegovaného atomického výroku zadanej formule. Identifikácia uzlov umožňuje nájdenie zdrojového a cieľového stavu. Hrany si uchovávajú informácie o zdrojovom a cieľovom stave, opakovanie a názov, ktorý je tvorený neprázdnou množinou písmen abecedy.

LTL formule s obmedzenými operátormi V sekcii 3.2.2 je opísaný spôsob, ako sa vytvárajú automaty pre formulu s obmedzenými operátormi. Pri implementácii bol dodržaný postup konštrukcie čítačového Büchi automatu.

V prvom kroku sa vytvárajú automaty pre fragment podobne ako pri formulách bez obmedzenia. Rozdielom je pridanie čítača na jednotlivé prechody. V počiatočnom stave dochádza k inicializácii čítača a následne je vytvorená hrana, ktorá umožní prijatie ľubovoľných slov, reprezentovaných hodnotou 1 (true). Ďalšia hrana z počiatočného stavu umožňuje prechod do koncového stavu za podmienok uvedených na hrane. Pre túto hranu sa v druhom kroku zmení cieľový stav. V automate pre fragment s najvyšším obmedzením operátora sa nič nemení. V ostatných automatoch má hrana cieľový stav v počiatočnom stave automatu vytvoreného pre fragment s vyšším obmedzením.

3.3.3 Produkt

Na vytvorenie produktu boli použité metódy z knižnice SPOT. Metóda `load` z triedy `ltsmin_model` kompiluje model použitím rozhrania ltlmin a načíta ho pomocou DiVinE modelu zo súboru.

Príklad súboru v jazyku DVE, ktorý načítame metódou `load` sa nachádza na obrázku 3.17. Základnou jednotkou jazyka je systém zložený z procesov. Procesy môžu prechádzať z jedného stavu do druhého prechodmi, ktoré možno strážiť podmienkou. Každý z týchto prechodov môže ovplyvniť lokálnu alebo globálnu premennú [5]. Systém zobrazený na obrázku 3.17 je zložený z procesu **P** s počiatočným a jediným stavom **x**.

```
process P {
 int a = 0, b = 0;
 state x;
 init x;

 trans
   x -> x { guard a < 3 && b < 2; effect a = a + 1; },
   x -> x { guard a < 3 && b < 2; effect b = b + 1; };
}

system sync;
```

Obr. 3.17: Model v jazyku DVE

Produktový automat je možné vytvárať formálnym prístupom pre prienik automatov. Naším riešením je získanie Kripkeho štruktúry pre skúmaný systém pomocou metódy `kripke` z knižnice SPOT, ktorú použijeme na synchronizáciu s naším automatom. Pri synchronizácii sa nevytvára čítačový automat. Stavy Kripkeho štruktúry obsahujú všetky možné hodnoty, ktoré môžu atomické výroky nadobudnúť. Tieto hodnoty závisia od skúmaného systému.

Pri vytváraní produktového automatu používame dodatočnú optimalizáciu korektnú s formulami, ktoré sú negáciou safety formulí formy $\mathbf{F}(\Psi)$, kde Ψ je z gramatiky 3.1.

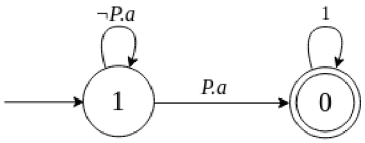
V Ψ nemôže byť použité pravidlo s **G** bez obmedzenia. Vďaka optimalizácii je každý beh v produkte, ktorý dosiahne koncového stavu, prijatý (beh potom nikdy nemôže opustiť prijímajúce stavy). Neprázdnosť jazyka je teda možné rozhodnúť na základe dosiahnuteľnosti koncového stavu. Prechody vedúce z koncových stavov nie je potrebné vytvárať, pretože nemajú vplyv na neprázdnosť jazyka.

Zobrazená Kripkeho štruktúra 3.18 je vytvorená pre model z obrázku 3.17, kde každý zo stavov má označenie s hodnotou atomického výroku.

```
digraph "" {
  rankdir=LR
  label="t\n[all]"
  labelloc="t"
  I [label="", style=invis, width=0]
  I -> 0
  0 [label="P.a=0, P.b=0\n1"]
  0 -> 1 [label=""]
  0 -> 2 [label=""]
  1 [label="P.a=1, P.b=0\n1"]
  1 -> 3 [label=""]
  1 -> 4 [label=""]
  2 [label="P.a=0, P.b=1\n1"]
  2 -> 4 [label=""]
  2 -> 5 [label=""]
  3 [label="P.a=2, P.b=0\n1"]
  3 -> 6 [label=""]
  3 -> 7 [label=""]
  4 [label="P.a=1, P.b=1\n1"]
  4 -> 7 [label=""]
  4 -> 8 [label=""]
  5 [label="P.a=0, P.b=2\n1"]
  5 -> 5 [label=""]
  6 [label="P.a=3, P.b=0\n1"]
  6 -> 6 [label=""]
  7 [label="P.a=2, P.b=1\n1"]
  7 -> 9 [label=""]
  7 -> 10 [label=""]
  8 [label="P.a=1, P.b=2\n1"]
  8 -> 8 [label=""]
  9 [label="P.a=3, P.b=1\n1"]
  9 -> 9 [label=""]
  10 [label="P.a=2, P.b=2\n1"]
  10 -> 10 [label=""]
}
```

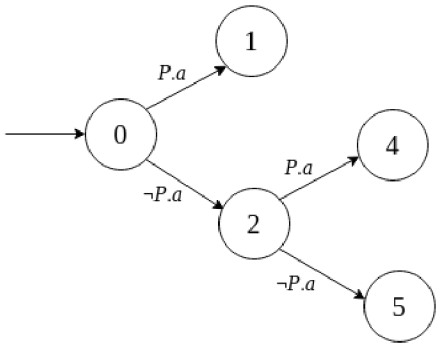
Obr. 3.18: Kripkeho štruktúra v DOT formáte

Príklad 8. Príklad bližšie popisuje vytváranie produktu pre negovanú formulu $\Psi = \mathbf{F}P.a$ a spomenutý systém. Automat pre Ψ je zobrazený na obrázku 3.19.



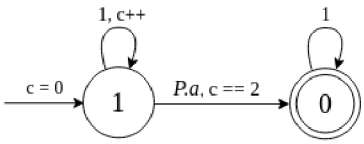
Obr. 3.19: $\Psi = \mathbf{F}P.a$

V produktovom automate sa vytvoria 2 hrany $\neg P.a$ a $P.a$ zo stavu **0** podľa Kripkeho štruktúry a budú viesť do stavov **1** a **2**. V stave **1** je hodnota atomického výroku $P.a$ jedna, čo spôsobí prechod do koncového stavu v našom automate. V tomto prípade je teda formula neplatná a nie je potrebné pokračovať vo vytváraní prechodu zo stavu **1**. Zo stavu **2** vytvoríme prechody ako sú uvedené v Kripkeho štruktúre, to znamená do stavov **4** a **5**. V stave **4** je opäť hodnota $P.a$ jedna a zo stavu **5** už nevedie žiadna hrana, takže pre tento prípad je formula platná.



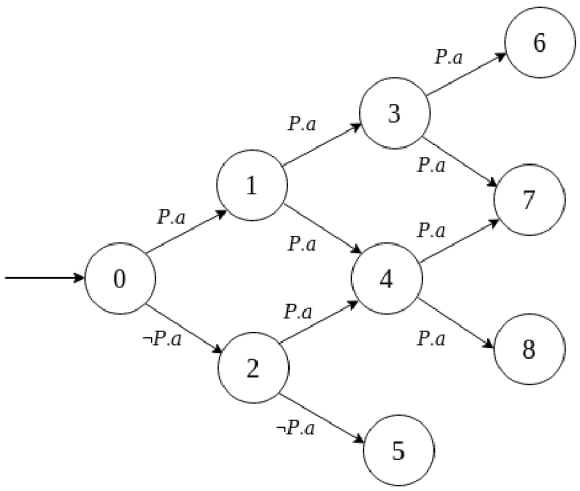
Obr. 3.20: Produktový automat pre $\Psi = \mathbf{F}P.a$ a Kripkeho štruktúru, v Kripkeho štruktúre sú všetky stavy implicitne koncové a má implicitný *sink* stav

Príklad 9. Príklad je zameraný na vytvorenie produktového automatu pre formulu s obmedzeným operátorom a systém 4.1. $\Psi = \mathbf{G}_{\{=2\}}P.a$ je negovaná LTL formula. Automat pre Ψ je zobrazený na obrázku 3.21.



Obr. 3.21: $\Psi = \mathbf{G}_{\{=2\}}P.a$

Pri vytváraní produktové automatu budeme postupovať rovnakým spôsobom ako v príklade 8. Počas 2 krokov je možné prijať ľubovoľné slová, preto je vytvorený prechod aj zo stavu **1** do **3** a **4**. Od stavov **3**, **4** a **5** má čítač hodnotu 2 a je možný prechod do koncového stavu automatu 3.21, ak je prijaté $P.a$. Zo stavu **3** sa vykoná prechod do **6** a **7**, pričom v oboch stavoch platí $P.a$. Došlo teda k prijatiu $P.a$ po 2 krokoch a formula nie je platná. Totožne to bude aj pri prechode zo stavu **4** do **8**. Zo stavu **5** už nevedie žiadna hrana. Automat v tomto stave ostáva a formula je pre túto cestu platná.



Obr. 3.22: Produktový automat pre $\Psi = \mathbf{G}_{\{=2\}}P.a$ a Kripkeho štruktúru

Pre uloženie hodnôt atomických výrokov na hrane a v stave je použitý reťazec. Použitie BDD nie je dostačujúce, pretože formule od firmy Honeywell obsahujú aj aritmetické operácie, kvôli ktorým je nutné poznať presnú hodnotu atomického výroku.

Po transformácii LTL formule na automat si každý zo stavov uchováva informácie o prvej a poslednej odchádzajúcej hrane. Pri vytváraní prechodov v produktovom automate sa začína s prvou hranou, kedy vieme, za akých podmienok sa vykoná prechod. Ak je automat v stave **1** z obrázku 3.21, prvá hrana značí prijatie ľubovoľného slova. Metóda `next` v triede `product` zaistí porovnanie hodnôt z hrany automatu a stavu Kripkeho štruktúry.

Ak je výsledok porovnania hodnôt 1, prechod sa vykoná, v opačnom prípade sa prechádza k ďalšej odchádzajúcej hrane, kedy sa opätovne porovnajú hodnoty.

Kapitola 4

Experimentálne vyhodnotenie

Kapitola popisuje porovnanie automatov vytvorených programom LTL2CBA a nástrojmi SPOT a LTL3BA. Vstupom pre spomenuté nástroje boli negované formule, v prípade programu LTL2BCA sa použili pôvodné formule.

Na experimenty bolo použitých 192 formulí poskytnutých firmou Honeywell. Avšak, každá sa dala zaradiť do jednej z niekoľkých kategórií, v ktorých boli všetky formule veľmi podobné. V jednej kategórii bolo 80 formulí, ostatné kategórie boli tvorené 5–10 formulami. Formulí s obmedzenými operátormi bolo 18.

V sekcii 4.1 porovnávame nástroje na formulách zastupujúcich jednotlivé kategórie. Sekcia 4.2 porovnáva produktové automaty pre vybrané formule.

Pre prehľad uvádzame zoznam parametrov, ktoré sme skúmali:

- počet stavov automatu,
- čas vykonania programu, označený *real*,
- množstvo CPU času stráveného v jadre procesu, označený *sys*,
- pamäť alokovaná programom, označená *mem*.

4.1 Prevod formule na čítačový Büchi automat

Vstupom pre existujúce nástroje boli negované formule uvedené nižšie. Aritmetické a porovnávacie operátory boli vo formuliach nahradené atomickým výrokom, pretože ani jeden z nástrojov ich nepodporuje. V prípade vytvoreného programu LTL2CBA boli zachované. V 1. experimente bola vybraná veľmi jednoduchá formula, ďalej boli testované formule s rôzne veľkým obmedzením operátora.

Experiment 1

ξ_1 = F((Autopilot_vert_state == VERT_SPH && trigger_SPD) && !(Autopilot_vert_state == VERT_PAH))

Pre prevod formule existujúcimi nástrojmi bolo potrebné použiť substitúciu, pretože SPOT a LTL3BA nepodporujú aritmetické operátory:

$$a = \text{Autopilot\_vert\_state} == \text{VERT\_SPH}$$
$$b = \text{Autopilot\_vert\_state} == \text{VERT\_PAH}$$

Vygenerované automaty mali rovnaký počet stavov. V prípade nepoužitia substitúcie by bol automat vygenerovaný iba programom LTL2CBA. Všetky formule bez opakovania temporálneho operátora od firmy Honeywell obsahujú porovnávacie operátory, preto nebola vybraná iná testovacia formula, ktorú by bolo možné previesť do automatu použitím SPOTu alebo LTL3BA.

Rozdiel v časoch na vykonanie programu je minimálny. Nástroj SPOT využil najviac pamäte, aj keď automat pozostával iba z 2 stavov.

SPOT	
počet stavov	2
real	0,048s
sys	0,000s
mem	12,44MB

LTL3BA	
počet stavov	2
real	0,012s
sys	0,003s
mem	3,78MB

LTL2CBA	
počet stavov	2
real	0,002s
sys	0,000s
mem	89,66kB

Tabuľka 4.1: Experiment 1 pre formulu ξ_1

Experiment 2

ξ_2 = F(G=3(!DInput) && !(G=5(DInput)) && G=3(DOutput))

Automat vygenerovaný obomi nástrojmi pozostával zo 7 stavov. Čítačový Büchi automat vytvorený pomocou LTL2CBA bol tvorený 2 stavmi. Použitím čítača bolo možné zredukovať počet stavov v porovnaní s existujúcimi nástrojmi.

Pre túto formulu sú výsledky z hľadiska času a využitej pamäte veľmi podobné predchádzajúcemu experimentu.

SPOT	
počet stavov	7
real	0,009s
sys	0,008s
mem	13,04MB

LTL3BA	
počet stavov	7
real	0,004s
sys	0,000s
mem	3,78MB

LTL2CBA	
počet stavov	2
real	0,002s
sys	0,000s
mem	87,10kB

Tabuľka 4.2: Experiment 2 pre formulu ξ_2

Experiment 3

ξ_3 = F(G=80(art_thr_part == 7) && G=80 !(repo_prot == 7))

Programom LTL2CBA bol vygenerovaný čítačový Büchi automat s 2 stavmi. S použitím nástrojov automaty pozostávali z 82 stavov.

SPOT	
počet stavov	82
real	0,051s
sys	0,000s
mem	40,18MB

LTL3BA	
počet stavov	82
real	0,009s
sys	0,000s
mem	3,91MB

LTL2CBA	
počet stavov	2
real	0,002s
sys	0,002s
mem	87,02kB

Tabuľka 4.3: Experiment 3 pre formulu ξ_3

Experiment 4

ξ_4 = F((!f_ice_sys || !s_ice_sys) && G=120 !desired_art_fun)

Automat bol vygenerovaný iba nástrojom LTL3BA so 122 stavmi. Automat vytvorený programom LTL2CBA aj napriek obmedzeniu 120 pozostával z 3 stavov.

SPOT	
počet stavov	-
real	-
sys	-
mem	-

LTL3BA	
počet stavov	122
real	0,052s
sys	0,000s
mem	4,29MB

LTL2CBA	
počet stavov	3
real	0,011s
sys	0,000s
mem	87,24kB

Tabuľka 4.4: Experiment 4 pre formulu ξ_4

Experiment 5

ξ_5 = F(((G=2(LiftCall)) && F=2((CurrentPosition - LastDestination) == 1 || (LastDestination - CurrentPosition) == 1)) && G=2 G=6 (!DestinationReached))

Pre prevod formule existujúcimi nástrojmi bolo opätovne potrebné použiť substitúciu:

$$a = \text{(CurrentPosition - LastDestination) == 1}$$
$$b = \text{(LastDestination - CurrentPosition) == 1}$$

Büchi automat pre ξ_5 generovaný nástrojom SPOT a LTL3BA pozostával z 10 stavov. Výsledok experimentu sa opäť líšil v množstve alokovanej pamäte. V prípade programu LTL2CBA bol vygenerovaný čítačový Büchi automat s 3 stavmi.

SPOT	
počet stavov	10
real	0,015s
sys	0,000s
mem	14,70MB

LTL3BA	
počet stavov	10
real	0,004s
sys	0,000s
mem	3,80MB

LTL2CBA	
počet stavov	3
real	0,002s
sys	0,000s
mem	98,68kB

Tabuľka 4.5: Experiment 5 pre formulu ξ_5

Experiment 6

ξ_6 = F(((G=2(LiftCall)) && F=2((CurrentPosition - LastDestination) == 1 || (LastDestination - CurrentPosition) == 1)) && G=2 G=8 (!DestinationReached))

$$a = \text{(CurrentPosition - LastDestination) == 1}$$
$$b = \text{(LastDestination - CurrentPosition) == 1}$$

Formula použitá v tomto experimente je veľmi podobná tej z experimentu 5. Rozdiel je v obmedzení operátora **G**. Program LTL2CBA vygeneroval automat s rovnakým počtom stavov aj v prípade navýšeného obmedzenia. Existujúce nástroje skonštruovali automat s 12 stavmi.

SPOT	
počet stavov	12
real	0,024s
sys	0,000s
mem	21,30MB

LTL3BA	
počet stavov	12
real	0,004s
sys	0,003s
mem	3,80MB

LTL2CBA	
počet stavov	3
real	0,002s
sys	0,000s
mem	98,69kB

Tabuľka 4.6: Experiment 6 pre formulu ξ_6

Zhrnutie

Celkovo naša metóda vyprodukovala veľmi kompaktné automaty pre všetky formule, kde ostatné nástroje mali problémy kvôli obmedzeným operátorom. Dôkazom je experiment 4 4.1, kedy nebolo možné vytvoriť automat pomocou nástroja SPOT.

4.2 Produkt so systémom

Produktový automat bol vytváraný nástrojom SPOT a programom LTL2CBA. LTL2CBA pracoval s Kripkeho štruktúrou pre určitý model systému rovnako ako SPOT a pri tvorbe produktu bola použitá optimalizácia popísaná v sekcii 3.3.3, ktorá umožňuje ukončiť konštrukciu pri nájdení koncového stavu. Pre experimenty boli použité negované formule uvedené nižšie poskytnuté firmou Honeywell. Vstupom pre program LTL2CBA boli pôvodné formule.

V 1. experimente bola vybraná veľmi jednoduchá formula. V ďalších experimentoch boli vybrané formule, ktorých obmedzenie nebolo veľké a automat bolo možné zostrojiť aj nástrojom SPOT. Ako bolo uvedené v experimente 4 4.1, pre formulu s vysokým obmedzením nebolo možné vytvoriť automat s použitím SPOTu.

V nasledujúcich experimentoch bol použitý model systému z obrázku 4.1. Kripkeho štruktúra pre tento model pozostávala z 23 stavov.

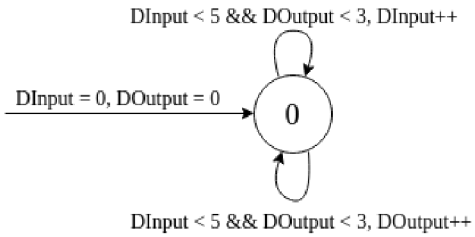
```
process P {
 int DInput = 0, DOutput = 0;
 state x;
 init x;

 trans
   x -> x { guard DInput < 5 && DOutput < 3; effect DInput = DInput + 1; },
   x -> x { guard DInput < 5 && DOutput < 3; effect DOutput = DOutput + 1; };
}

system sync;
```

Obr. 4.1: Model systému v jazyku DVE pre ξ_1

Konštrukcia Kripkeho štruktúry, ktorú vytvorí SPOT, nepoužíva čítače, ale expanduje ich. Alternatívou by mohlo byť vytvorenie tejto štruktúry s čítačmi zobrazenej na obrázku 4.2, ktorá by mala 1 stav. Následne vzniknutý produktový automat by nemal viacej stavov ako by mal automat pre formulu.



Obr. 4.2: Kripkeho štruktúra s použitím čítačov

Experiment 1

ξ_1 = F((DInput) && !(DOutput))

Na tento experiment bola použitá jednoduchá formula, pre ktorú bol vytvorený Büchi automat s 2 stavmi. V prípade programu LTL2CBA bol na začiatku vytvárania produktu dosiahnutý koncový stav vytvoreného Büchi automat pre jednu cestu. Z tohto dôvodu má produktový automat menej stavov ako Kripkeho štruktúra. SPOT vygeneroval produkt s 37 stavmi.

SPOT	
počet stavov	37
real	0,007s
sys	0,000s
mem	14,37MB

LTL2CBA	
počet stavov	19
real	0,007s
sys	0,000s
mem	13,56MB

Tabuľka 4.7: Experiment 1 pre formulu ξ_1

Experiment 2

ξ_2 = F(G=5 DInput && G=5 !DOutput)

SPOT v tomto experimente opäť vytvoril produktový automat s viacej stavmi ako LTL2CBA. Čítačový Büchi automat mal 2 stavy.

SPOT	
počet stavov	27
real	0,008s
sys	0,008s
mem	14,74MB

LTL2CBA	
počet stavov	23
real	0,007s
sys	0,000s
mem	13,26MB

Tabuľka 4.8: Experiment 2 pre formulu ξ_2

Experiment 3

ξ_2 = F(G=5 DInput && G=5 !DOutput)

Experiment bol vykonaný na rovnakej formuli ako experiment 2 v sekcii 4.2 a čítačový Büchi automat mal 2 stavy. Upravený bol skúmaný model systému 4.1, v ktorom hodnota **DOutput** nebola ovplyvnená a mala hodnotu 0. Hodnota **DInput** bola porovnávaná s hodnotou 10. Kripkeho štruktúra následne pozostávala z 11 stavov. Cieľom týchto úprav

bolo poukázať na skutočnosť, že negácia umožňuje rýchlejšie zistenie, kedy sa platnosť formule poruší. V tomto prípade sa zo stavu 6 v Kripkeho štruktúre nevytvárali ďalšie stavy v produktovom automate, pretože došlo k prechodu do koncového stavu čítačového Büchi automatu. Ako bolo spomenuté v sekcii 3.2, v čítačovom automate nedochádza k prechodu z koncového stavu do nekoncového. SPOT vygeneroval produktový automat s 11 stavmi, zatiaľ čo program LTL2CBA vytvoril produktový automat so 7 stavmi.

SPOT	
počet stavov	11
real	0,008s
sys	0,000s
mem	14,73MB

LTL2CBA	
počet stavov	7
real	0,007s
sys	0,007s
mem	13,21MB

Tabuľka 4.9: Experiment 3 pre formulu ξ_3

Experiment 4

ξ_4 = F((G=3(!DInput) && F=5(!DInput)) && G=3 DOutput)

V tomto experimente bol vygenerovaný produktový automat nástrojom LTL2CBA veľmi podobný tomu v experiment 2 v sekcii 4.2. Počet stavov bol 23. V prípade nástroja SPOT bol počet stavov 26. Čítačový Büchi automat pozostával z 2 stavov.

SPOT	
počet stavov	26
real	0,009s
sys	0,009s
mem	15,02MB

LTL2CBA	
počet stavov	23
real	0,007s
sys	0,000s
mem	13,28MB

Tabuľka 4.10: Experiment 4 pre formulu ξ_4

Zhrnutie

Optimalizácia na základe toho, že automat neprechádza z koncového stavu do nekoncového ako bolo spomenuté v sekcii 3.3.3, spôsobila čiastočnú redukciu stavového priestoru produktového automatu. Najväčšia redukcia počtu stavov bola preukázaná v experimente 1 4.2, keď na začiatku vytvárania produktu bol dosiahnutý koncový stav Büchi automatu pre jednu sekvenciu a bolo vytvorených o polovicu menej stavov oproti nástroju SPOT.

V ostatných prípadoch nedošlo k výraznému zníženiu počtu stavov, pretože Kripkeho štruktúra nepoužívala čítače.

Kapitola 5

Záver

Pri vývoji veľkých systémov je potrebné garantovať, že systém funguje správne. Overovanie korektnosti systému človekom je časovo náročné a náchylné na chyby. Preto sú nástroje overujúce správnosť dôležité. Jedným z nástrojov je technika LTL overovanie modelov, ktorá to dokáže urobiť automaticky, spoľahlivo a bez chýb. Systém je popísaný LTL vlastnosťami, ktoré sú prevedené do automatu na overenie.

Cieľom tejto práce bolo vytvoriť automat s čítačmi pre LTL formulu s obmedzenými operátormi, čo zahŕňalo naštudovanie lineárnej temporálnej logiky a prevod formulí do automatov. Následne bolo navrhnuté riešenie a algoritmus, akým by sa takéto automaty vytvárali.

Výsledkom je nástroj, ktorý dokáže previesť formule do čítačového automatu. Vďaka čítačom sa podarilo zmenšiť veľkosť automatu niekoľkonásobne, pretože nedochádzalo k vytváraniu podobných stavov a veľkosť nebola závislá na počte krokov, počas ktorých mali temporálne operátory obmedzenú platnosť. Pri LTL formulách s vysokým obmedzením operátorov nebol automat zostrojený ani existujúcimi nástrojmi. Vo väčšine prípadov dochádzalo k problémom s nedostatkom pamäti, čomu sme predišli.

Ďalej došlo k redukcii počtu stavov v produktovom automate. Testovali sme splniteľnosť negácie a pri prechode do koncového stavu vytvoreného Büchi automatu bola preukázaná neplatnosť formule. Nebolo potrebné prijímať ďalšie znaky, pretože podľa sekcie 3.2 nie je možné prejsť z koncového stavu do nekoncového. Maximálny počet stavov odpovedal počtu stavov v Kripkeho štruktúre a k jeho navyšovaniu nedochádzalo.

V práci by som chcela pokračovať na možnom rozšírení, ktoré by sa týkalo prevodu zložitejších LTL formulí s plnou logikou, ktorá je netriviálnym problémom, do automatu. Rozšírenie by ďalej zahŕňalo vytváranie Kripkeho štruktúry a produktového automatu s čítačmi.

Literatúra

[1] Babiak, T., Křetínský, M., Řehák, V. a Strejček, J. LTL to Büchi Automata Translation: Fast and More Deterministic. In: Flanagan, C. a König, B., ed. *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 95–109. ISBN 978-3-642-28756-5.

[2] Baier, C. a Katoen, J.-P. Linear Temporal Logic. In: *Principles of Model Checking*. Cambridge, Massachusetts: The MIT Press, 2008, s. 229–312. ISBN 978-0-262-02649-9.

[3] Biere, A., Cimatti, A., Clarke, E. a Zhu., Y. Symbolic Model Checking without BDDs. In: Cleaveland, W. R., ed. *Tools and Algorithms for the Construction and Analysis of Systems*. Springer-Verlag, 1999, s. 193–207. ISBN 3-540-65703-7.

[4] Biere, A., Cimatti, A., Clarke, E. a Zhu., Y. *Bounded Model Checking* [online]. 2003 [cit. 2020-02-19]. Dostupné z: http://www.cs.cmu.edu/~emc/papers/Books%20and%20Edited%20Volumes/Bounded%20Model%20Checking.pdf.

[5] Brim, L., Barnat, J., Cerna, I. a spol. *Quick Guide Through the DVE Specification Language* [online]. 2008 [cit. 2020-05-08]. Dostupné z: https://divine.fi.muni.cz/darcs/branch-3.0/gui/help/divine/language.html.

[6] Browne, M., Clarke, E. a Grümberg, O. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*. 1988, zv. 59, č. 1, s. 115 – 131. DOI: https://doi.org/10.1016/0304-3975(88)90098-9. ISSN 0304-3975. Dostupné z: http://www.sciencedirect.com/science/article/pii/0304397588900989.

[7] Cavanagh, J. J. F. *Sequential logic: analysis and synthesis*. 1st. Boca Raton: [b.n.], 2007. ISBN 978-0-8493-7564-4.

[8] Clarke, E., Biere, A., Raimi, R. a spol. Bounded Model Checking Using Satisfiability Solving. In: *Formal Methods in System Design*. Kluwer Academic Publishers., 2001, s. 7–34. ISSN 1572-8102.

[9] Couvreur, J.-M. On-the-fly Verification of Linear Temporal Logic. In: Wing, J. M., Woodcock, J. a Davies, J., ed. *FM'99 — Formal Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, s. 253–271. ISBN 978-3-540-48119-5.

[10] Couvreur, J.-M. *Contribution à l'algorithmique de la vérification* [online]. 2004 [cit. 2020-02-22]. Dostupné z: https://www.univ-orleans.fr/lifo/Membres/couvreur/jmc-habile.pdf.

[11] Dijkstra, E. W. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. 1975, zv. 18, č. 8, s. 453–457. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/360933.360975.

[12] Duret-Lutz, A. a Poitrenaud, D. SPOT: an extensible model checking library using transition-based generalized Büchi automata. In: *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings.* 2004, s. 76–83. ISSN 1526-7539.

[13] Gastin, P. a Oddoux, D. Fast LTL to Büchi Automata Translation. In: *Proceedings of the 13th International Conference on Computer Aided Verification*. Berlin, Heidelberg: Springer-Verlag, 2001, s. 53–65. CAV '01. ISBN 3540423451.

[14] Giannakopoulou, D. a Lerda, F. From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata. In: Peled, D. A. a Vardi, M. Y., ed. *Formal Techniques for Networked and Distributed Sytems — FORTE 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, s. 308–326. ISBN 978-3-540-36135-0.

[15] Gunasekaran, A. Communication Protocols and Applications for Virtual Enterprises. In: *Agile Manufacturing - The 21st Century Competitive Strategy*. Elsevier, 2001, kap. 21, s. 405. ISBN 978-0-08-043567-1.

[16] Holzmann, G. *The SPIN Model Checker: Primer and Reference Manual*. 1st. Addison-Wesley Professional, 2011. ISBN 0321773713.

[17] Křetínský, J. a Esparza, J. *Deterministic Automata for the (F,G)-fragment of LTL* [online]. 2012 [cit. 2019-12-15]. Dostupné z: https://arxiv.org/pdf/1204.5057.pdf.

[18] Latvala, T., Biere, A., Heljanko, K. a Junttila, T. *Simple Bounded LTL Model Checking* [online]. 2004 [cit. 2020-02-19]. Dostupné z: http://fmv.jku.at/papers/LatvalaBiereHeljankoJunttila-HUT-TCS-TR-A92-2004.pdf.

[19] Li, W., Kan, S. a Huang, Z. A Better Translation From LTL to Transition-Based Generalized Büchi Automata. *IEEE Access*. 2017, zv. 5, s. 27081–27090.

[20] O'Regan, G. Model Checking. In: Mackie, I., ed. *Mathematics in Computing*. London: Springer-Verlag, 2013, kap. 24, s. 383–392. ISBN 978-3-030-34208-1.

[21] Pnueli, A. The temporal logic of programs. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 1977, s. 46–57.

[22] Rabin, M. a Scott, D. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*. Apríl 1959, zv. 3, s. 114–125. DOI: 10.1147/rd.32.0114.

[23] Sebastiani, R. a Tonetta, S. "More Deterministic"vs. "Smaller"Büchi Automata for Efficient LTL Model Checking. In: *CHARME*. 2003.

[24] Sharygina, N. a Peled, D. A Combined Testing and Verification Approach for Software Reliability. In: Oliveira, J. N. a Zave, P., ed. *FME 2001: Formal Methods for Increasing Software Productivity*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 611–628. ISBN 978-3-540-45251-5.

[25] Tauriainen, H. a Heljanko, K. Testing Spin's LTL Formula Conversion into Büchi Automata with Randomly Generated Input. In: Havelund, K., Penix, J. a Visser, W., ed. *SPIN Model Checking and Software Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, s. 54–72. ISBN 978-3-540-45297-3.

Príloha A

Obsah priloženého pamäťového média

Adresárová štruktúra je nasledujúca:

- `doc/` - dokumentácia vygenerovaná nástrojom Doxyfile
- `examples/`
  - `honeywell/` - formule poskytnuté firmou Honeywell
  - `product/` - model systému použitý v experimentoch
- `lib/` - knižnice potrebné pre preklad
- `src/` - zdrojové kódy
  - `include/` - hlavičkové súbory
  - `spot/` - časť knižnice SPOT na vytvorenie Kripkeho štruktúry z modelu systému
- `tests/` - experimenty
- `tex/` - zdrojový kód správy LaTeX
- README.md - základný prehľad s manuálom na spustenie a inštaláciu
- Makefile
- PDF s technickou správou

Príloha B

Formule poskytnuté firmou Honeywell

G ((G=5 (DInput)) → F=5 (DOutput))

G ((G=3 (! DInput) && ! (G=5 (DInput))) → F=3 (! DOutput))

X G ((! G=3 (! DInput) && ! (G=5 (DInput))) → F=3 (DOutput == Previous(DOutput)))

G ((G=80 (art_thr_part == 7)) → F=80 (repo_prot == 7))

G ((! f_ice_sys || ! s_ice_sys) → F=120 (desired_art_fun))

G (main_a_s_t >= -108 && main_a_s_t <= 108)

G (((Pitch >= -10) && (Pitch <= 20) && (Speed >= 100) && (Speed <= 300) && (Altitude >= 100) && (Altitude <= 40000)) → (C1))

G ((fabs(Vertical_speed) < 500) → (C5))

G ((Autopilot_altsel_state == ALTSEL_ARM && trigger_ALTCAP) → (Autopilot_altsel_state == ALTSEL_CAP))

G ((Autopilot_altsel_state == ALTSEL_ARM && trigger_APFAIL && C2) → (Autopilot_altsel_state == ALTSEL_OFF))

G ((Autopilot_altsel_state == ALTSEL_CAP && trigger_AP && C2) → (Autopilot_altsel_state == ALTSEL_OFF))

G (((G=2 (LiftCall)) && F=2 ((CurrentPosition - LastDestination) == 2 || (LastDestination - CurrentPosition) == 2)) → F=2 F=7 (DestinationReached))

G (((G=2 (LiftCall)) && F=2 ((CurrentPosition - LastDestination) == 3 || (LastDestination - CurrentPosition) == 3)) → F=2 F=8 (DestinationReached))

G ((G=2 (LiftCall)) → F=2 F=40 (DestinationReached))

G ((Moving) → (BrakesOffOut))

G (((G=2 (LiftCall)) && F=2 ((CurrentPosition - LastDestination) > 1 || (LastDestination - CurrentPosition) > 1)) → F=2 (Moving))

G ((numberOfValids == 2 && ! valid2) → (outputData == (signal1 + signal3) / 2))

G ((numberOfValids == 2 && ! valid3) → (outputData == (signal1 + signal2) / 2))

G ((numberOfValids == 1 && valid1) → (outputData == signal1))

G ((! valid1 && valid2 && ! valid3) → (numberOfValids == 1))

G ((! valid1 && ! valid2 && valid3) → (numberOfValids == 1))