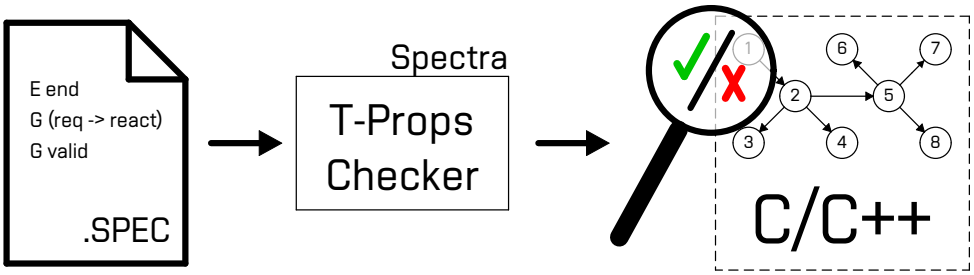
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019
http://excel.fit.vutbr.cz

Automatické ověřování temporálních vlastností programů za běhu

Petra Sečkařová*



Abstrakt
Temporální vlastnosti programů jsou používány ke specifikaci korektního průběhu jejich vykonávání. Jedním z nejčastějších způsobů formálního popisu těchto vlastností je *lineární temporální logika – LTL*. Tato práce se zabývá návrhem a implementací nástroje pro automatizované ověřování temporálních vlastností běhů programů specifikovaných pomocí tzv. past-time LTL. Výsledný program na základě dané specifikace vygeneruje statickou knihovnu, která dokáže spolehlivě ověřit, zda jsou její formule v každém okamžiku běhu kontrolovaného programu splněny, a případné neočekávané nebo nesprávné chování hlásí společně s podrobnou zprávou o okolnostech tohoto chybového stavu, která má napomáhat k nalezení chyby v konkrétním místě kódu.

Klíčová slova: Temporální vlastnosti programů — Monitorování za běhu — Automatická verifikace

Přiložené materiály: Repozitář projektu

*xsecka02@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

Jedním ze základních úkolů testování programu je kontrola, jestli jeho chování odpovídá dané specifikaci. V té však může být často nutné místo očekávaných výstupů programu při určitých vstupech definovat celkově korektní průběh výpočtů, které má program provádět. V takových případech se však úplná automatizace testování ověřujícího, že program specifikaci odpovídá, stává běžnými prostředky značně komplikovanou.

Požadovaný průběh výpočtů se v oblasti verifikace programů často specifikuje pomocí temporálních vlastností jejich běhů. Tato práce si tedy klade za cíl vytvořit nástroj pro automatické ověřování korektnosti jednotlivých běhů programu na základě specifikace jeho temporálních vlastností. Ta k jejich popisu využívá variantu *lineární temporální logiky – LTL* blíže popsanou v sekci 3.

Tato práce vzniká v rámci projektu Testos (Test Tool Set) [2], jehož hlavním cílem je vytvoření sady nástrojů pro podporu automatizovaného testování softwaru. Vytvořený nástroj má být snadno použitelný samostatně i společně s ostatními nástroji této platformy. Aby nebylo nutné zpracovávat testovaný program znovu při každé změně kódu, operuje nástroj pouze základě specifikace s přímým napojením na vykonávání programu. Protože není potřeba konstruovat jeho formální model, může být testovaný systém navíc libovolně komplexní.

V následujících kapitolách budou nejprve vysvětleny základy monitorování temporálních vlastností programů. Dále bude popsán vybraný algoritmus jejich vyhodnocování a konkrétní varianta LTL použitá jako

jazyk specifikace. Nakonec bude představen implementovaný nástroj a jeho použití, shrnuty dosavadní výsledky a popsány vize do budoucna.

2. Ověřování temporálních vlastností

Obrázek 1 ukazuje obecné schéma pro ověřování vlastností běhu systému. Provádí se pomocí monitoru, který vychází z dané specifikace. Ten má za úkol sledovat provádění systému a na základě toho poté určit, zda tento systém definované vlastnosti splňuje.

Napojení monitoru na testovaný systém se nazývá *instrumentace* a udává nejen způsob zpřístupnění potřebných aspektů testovaného systému, ale také vztah mezi vykonáváním systému a monitoru. Ty mohou být vykonávány buď to souběžně, nebo je nejdříve dokončen běh systému, který je vhodně zaznamenáván, a monitor je spouštěn až nad tímto záznamem.

Pro konstrukci co možná nejuniverzálněji použitelného nástroje je lepší volbou souběžné vykonávání monitoru a programu. Ačkoli druhý přístup časově méně zatěžuje původní výpočet systému, na rozdíl od tohoto neumožňuje ověřovat vlastnosti programu průběžně a chování nekonečných běhů už vůbec ne.

Temporální vlastnosti systému udávají časové souvislosti mezi jeho *stavy*. Aby bylo možné automaticky ověřovat, jestli vlastnosti programu odpovídají specifikovaným temporální formulím, je kromě vhodné volby instrumentace nutné (i) identifikovat stavy programu a (ii) implementovat do monitoru spolehlivý algoritmus pro vyhodnocování vlastností jejich vzájemných vztahů.

Rozlišení jednotlivých stavů není možné univerzálně algoritmizovat, protože je spíše součástí specifikace, jejíž splnění má být ověřováno. Aby tedy bylo možné zkoumat temporální vlastnosti nějakého programu, je potřeba v jeho kódu předem vyznačit, kde začíná nový stav. Poté už stačí pouze vybrat a implementovat vhodný algoritmus pro vyhodnocování jejich vztahů.

3. Použitý monitorovací algoritmus

Implementovaný nástroj přebírá hlavní myšlenky monitorovacího algoritmu z práce Havelunda a Roşu popsané v [3]. Ke specifikaci požadovaných vlastností sledovaného běhu je použita lineární temporální logika minulého času (*Past-Time Linear Temporal Logic – ptLTL*), což je varianta LTL jejíž formule popisují, jaké vztahy musí platit mezi stavy dosavadní historie běhu popisovaného systému.

Z hlediska monitorování programů za běhu je zásadní výhodou ptLTL právě to, že mluví o již známých stavech. Klasická LTL, která hovoří o potenciálně nekonečném budoucím průběhu stavů, má sice obecně větší vyjadřovací sílu, ale platnost jejích formulí nelze za běhu sledovaného systému úplně vyhodnotit, protože monitor nemá jistotu, co bude sledovaný systém dělat dál. Jejich průběžné vyhodnocování za běhu programu by tedy vedlo (a) k velkému množství falešných poplachů, pokud by monitor určoval vlastnosti běhu jen podle dosavadní historie, nebo (b) k neúplným průběžným výsledkům, pokud by monitor čekal, co přijde později. To platí i v případě použití varianty LTL omezené na popis n následujících stavů.

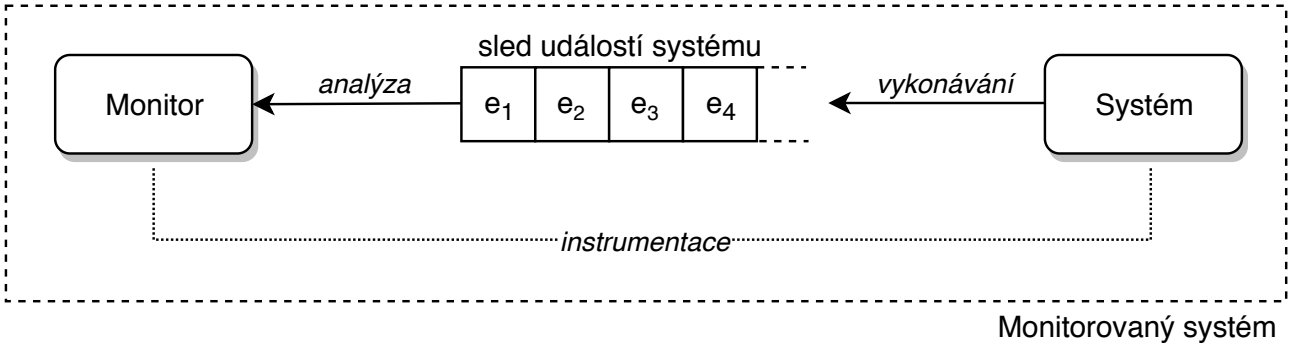
3.1 Jazyk specifikace

Vybraná ptLTL používá výrokové logické operátory doplněné navíc o následující speciální temporální operátory.

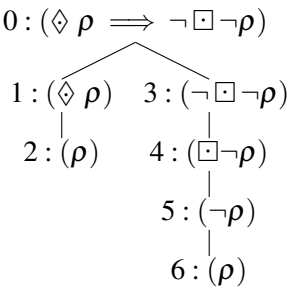
$$\phi \mathcal{S} \rho \; - \; \textit{Since} \qquad \boxdot \rho \; - \; \textit{Globally}$$
$$\odot \rho \; - \; \textit{Last} \qquad \diamondsuit \rho \; - \; \textit{Previously}$$

$\phi \mathcal{S} \rho$ říká, že ϕ platí ve všech stavech od doby, co přestalo platit ρ. Formule $\odot \rho$ je pravdivá, pokud ρ platilo v předcházejícím stavu, na rozdíl od $\diamondsuit \rho$, kterému stačí, aby ρ platilo v libovolném z dosavadních stavů běhu. $\boxdot \rho$ pak platí, pokud ρ platilo ve všech pozorovaných stavech.

Základem vyhodnocování platnosti formulí ptLTL v [3] je vyjádření sémantiky temporálních operátorů rekurzivně. Toto vyjádření je v [4] definováno jako tranzitivní relace mezi dvěma po sobě jdoucími stavy



Obrázek 1. Základní sestava pro monitorování systému podle [1]



Obrázek 2. Příklad rozkladu temporální formule do stromu podformulí a jeho odpovídající reprezentace v poli.

během programu a je zde také dokázáno, že tyto vztahy dokáží obsáhnout sémantiku libovolného operátoru lineární temporální logiky. Pro stavy běhu

$$\sigma = s_0 \dots s_{j-1}, s_j, s_{j+1} \dots$$

lze sémantiku operátorů ptLTL vyjádřit následovně:

$$\begin{aligned}
s_j &\models \odot \rho \quad &\Leftrightarrow\ & s_{j-1} \models \rho \\
s_j &\models \boxdot \rho \quad &\Leftrightarrow\ & s_j \models \rho \wedge s_{j-1} \models \boxdot \rho \\
s_j &\models \diamondsuit \rho \quad &\Leftrightarrow\ & s_j \models \rho \vee s_{j-1} \models \diamondsuit \rho \\
s_j &\models \rho \mathscr{S} \phi \quad &\Leftrightarrow\ & s_j \models \phi \vee (j > 1 \wedge s_j \models \rho \wedge \\
& & & \quad s_{j-1} \models \rho \mathscr{S} \phi)
\end{aligned}$$

V počátečním stavu je platnost formulí podle vzoru [3] odvozována přímo od aktuální hodnoty podformule, tedy $s_0 \models \odot \rho \Leftrightarrow s_0 \models \rho$, to samé pro $\boxdot \rho$ i $\diamondsuit \rho$, a $s_0 \models \rho \mathscr{S} \phi \Leftrightarrow s_0 \models \phi$. Platnost všech formulí ptLTL je pomocí tohoto vyjádření možné v každém stavu vyhodnotit pouze pomocí výrokových operátorů na základě hodnot stavových proměnných a ohodnocení formulí v předchozím stavu.

3.2 Princip použitého algoritmu

Po vzoru zmiňovaného článku je pro účely monitorování každá formule ptLTL uvedená ve specifikaci rozdělena do stromu dílčích podformulí. Každá z nich obsahuje buď to (i) jeden operátor a identifikátory operandů, nebo (ii) čtení hodnoty stavové proměnné zkoumaného systému, které jsou listovými uzly tohoto stromu. Příklad takového stromu je ilustrován na Obrázku 2.

Hodnota každého uzlu stromu je pro účely výpočtu reprezentována dvojicí booleovských hodnot, pro platnost (i) v předchozím a (ii) v aktuálně zkoumaném stavu běhu programu. Platnost podformulí v předchozím stavu je pro počáteční stav vhodně inicializována a před vyhodnocením každého dalšího stavu přepsána původně aktuální hodnotou.

Díky uchovávání výsledků ohodnocení formulí z předchozího kroku stačí pro zhodnocení aktuálního stavu systému pouze booleovské operace a každý krok verifikace je tedy velmi jednoduchý. Jedinou avšak ne příliš zásadní nevýhodou tohoto přístupu je nemožnost zkráceného vyhodnocování formulí, právě kvůli návaznosti jejich hodnot na předchozí stav.

4. Nástroj T-Props Checker

Nástroj, který je výsledkem této práce, je implementován v jazyce C++ pod názvem *T-Props Checker* jako podsystém nástroje Spectra (Specification translation) [5] platformy Testos. Nástroj Spectra slouží jako podpůrný nástroj pro automatizaci testování a jeho účelem je transformovat specifikaci programu do takové formy, aby na jejím základě bylo buď (i) možné generovat automatizované testy, nebo (ii) přímo ověřovat splnění specifikovaných vlastností jako tomu je u T-Props Checkeru.

Jeho vstupem jsou dva názvy souborů, z nichž jeden patří hlavičkovému souboru s deklaracemi všech stavových proměnných. Díky němu může monitor za běhu přistupovat do paměťového prostoru testovaného systému a proto tento soubor musí obsahovat všechny proměnné použité pro specifikaci ověřovaných vlastností. Obsahem druhého vstupního souboru je pak samotná specifikace temporálních vlastností zkoumaného programu. Jsou v ní uvedeny všechny formule, které mají pro daný systém platit, a další informace, jejichž detailní popis i formát je uveden v repozitáři projektu. Výstupem nástroje je poté kód monitoru v podobě statické knihovny, kterou lze snadno napojit do libovolného projektu v jazycích C/C++ způsobem blíže popsaným v sekci 4.2.

Nástroj T-Props Checker se spouští pod projektem Spectra následujícím způsobem:

```
./spectra --tpc -s cesta/ke/spec -d cesta/k/def
```

Argument `--tpc` znamená pro Spectru volbu podsystému T-Props Checker a zbylé dva pak udávají cesty ke vstupním souborům pro tento nástroj.

4.1 Generovaný monitor

Nástroj T-Props Checker nejprve analyzuje specifikaci, přičemž zkonstruuje strom uvedených formulí a podformulí jako na Obrázku 2. Každá formule má přiřazený číselný identifikátor, kde listové uzly mají vždy vyšší identifikátor než jejich rodičovský uzel. Pro každou formuli je kromě sémantického obsahu ukládána také odpovídající textová reprezentace a umístění zápisu formule ve specifikačním souboru.

Na základě tohoto zpracování specifikace je pak vygenerován kód monitoru. Jeho základem je struktura obsahující dvě[1] pole booleovských hodnot uspořádané jako na Obrázku 2. Ohodnocení každé z formulí odpovídá hodnota na indexu shodném s jejím identifikátorem v interní reprezentaci nástroje. Vyhodnocování aktuálního stavu podle předpisů ze sekce 3.1 je prováděno od nejvyšších indexů pole a je tedy vždy zajištěno, že budou všichni potomci vyhodnoceni před svým rodičovským uzlem, než bude jejich hodnoty potřebovat k vlastnímu vyhodnocení.

Pokud je ve specifikaci uvedeno více formulí odděleně, je nad nimi vytvořena kořenová formule provádějící jejich konjunkci. Po každém kroku je zkontrolováno, že kořenová formule platí, a v opačném případě je vytištěna chybová hláška. Ta obsahuje výpis všech změn v posledním kroku, které měly vliv na aktuální hodnotu kořenové formule, od listové úrovně nahoru, vždy s umístěním v původním souboru, jejich aktuální hodnotou a textovým zněním formule.

4.2 Instrumentace generovaného monitoru

Vygenerovaný kód monitoru je nástrojem zkompilován a uložen jako statická knihovna nazvaná *libspectra.a*. Tu je při překladu nutné napojit na ověřovaný program úpravou proměnných prostředí pro kompilátor:

```
LDLIBS += -lspectra
LDFLAGS += -L/cesta/ke/slozce/s/libspectra.a
CFLAGS/CXXFLAGS += -I.
```

Jak bylo zmíněno v sekci 2, ještě před použitím nástroje je potřeba, aby uživatel manuálně označil místa v kódu, kde systém přechází do nového stavu. Tato místa lze značit jednou ze dvou možností. Buď voláním funkce `monitorVerify()` jejíž implementace je zahrnuta v `libspectra.a`, nebo komentářem *T-Props Checker verify*, který bude tímto nástrojem automaticky nahrazen za předchozí variantu, přičemž původní verzi souboru zachová jako `jméno/souboru.old`.

4.3 Evaluace nástroje

Funkcionální testy z repozitáře projektu demonstrují, že nástroj T-Props Checker dokáže zpracovat do spolehlivě fungujícího monitoru formule obsahující libovolné operátory ze sekce 3.1. Část testů pracuje se specifikací jednoduchých temporálních formulí nad programem v jazyce C, počítajícím do desíti.

Testovací sada také obsahuje ukázky zpracování správně i chybně zapsané specifikace nad příkladem reálného kódu pro řízení výtahu v jazyce C++. Skript řídící průběh funkcionálních testů navíc demonstruje způsob překladu testovaného projektu pro obě jazykové varianty a způsob zapojení monitoru je zřejmý z ukázkových programů. V současné době jsou vyvíjeny také další testy demonstrující funkčnost jednotlivých částí implementace zvlášť.

[1] Pro hodnoty aktuálního a předchozího stavu.

5. Závěr

V rámci této práce byl vytvořen nástroj určený jako podpora pro automatizované testování projektů v jazycích C/C++. Specifikaci temporálních vlastností programu zapsanou ve formulích temporální logiky transformuje do kódu monitoru, který je na původní program možné snadno napojit. Monitor je spouštěn souběžně s ověřovaným systémem a vyhodnocuje jeho správnost průběžně.

V případě porušení specifikovaných vlastností je monitorem vytištěna podrobná zpráva o aktuálních změnách vedoucích k selhání verifikace, která má uživateli pomoct lokalizovat chybu v kódu. Správnost implementace nástroje je ověřena sadou automatických testů, poskytujících příklady zápisů specifikace i nutných kroků pro úspěšnou manuální instrumentaci monitoru.

Aby byl implementovaný nástroj co nejsnadněji použitelný a zároveň dobře napojitelný do projektu Testos jako celku, bude dále doplněno webové rozhraní pro jeho spouštění a správu testování s jeho využitím.

Poděkování

Za ochotnou spolupráci a odborný dohled děkuji vedoucímu mé diplomové práce, Ing. Aleši Smrčkovi, Ph.D.

Literatura

[1] Ezio Bartocci and Yliès Falcone. *Lectures on Runtime Verification. Introductory and Advanced Topics*. Springer International Publishing AG, 2018. ze série Lecture Notes in Computer Science, díl 10457.

[2] Skupina Testos. Domovská stránka projektu Testos. FIT VUT v Brně, 2017. [Online; navštíveno 3.1.2019].
URL http://testos.org.

[3] Klaus Havelund and Grigore Roşu. Efficient monitoring of safety properties. *Software Tools for Technology Transfer*, 6(2):158–173, 2004.

[4] Yonit Kesten, Amir Pnueli, and Lion Raviv. *Algorithmic Verification of Linear Temporal Logic Specifications*. Springer, Heidelberg, 1998. ICALP 1998. Lecture Notes in Computer Science, díl 1443.

[5] Skupina Testos. Gitlab repozitář nástroje Spectra. FIT VUT v Brně, 2019. [Online; navštíveno 3.1.2019].
URL https://pajda.fit.vutbr.cz/testos/spectra.