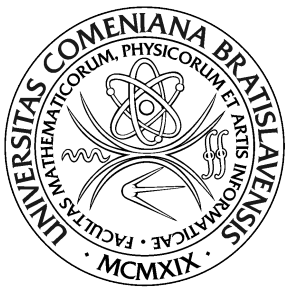
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



SOFTVÉROVÝ NÁSTROJ PRE UNITY

Diplomová práca

2018 Bc. Róbert Ruska

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

SOFTVÉROVÝ NÁSTROJ PRE UNITY

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	doc. RNDr. Damas Gruska, PhD.

Bratislava, 2018 Bc. Róbert Ruska

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Róbert Ruska
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Softvérový nástroj pre UNITY
Software tool for UNITY

Anotácia: Cieľom práce je vytvoriť softvérový nástroj podporujúci zápis, simuláciu a verifikáciu programov zapísaných v jazyku UNITY.

Cieľ: Cieľom práce je vytvoriť softvárový nástroj podporujúci zápis, simuláciu a verifikáciu programov zapísaných v jazyku UNITY.

Vedúci: doc. RNDr. Damas Gruska, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 09.10.2017

Dátum schválenia: 09.10.2017 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a za pomoci konzultácií u môjho školiteľa.

. . . . . .

Bratislava, 2018 Bc. Róbert Ruska

Poďakovanie

//TBD

Abstrakt

//TBD

Kľúčové slová: //TBD

Abstract

//TBD

Keywords: //TBD

Obsah

Kapitola 1

Úvod

//TBD

Kapitola 2

Motivácia

//TBD

Kapitola 3

Teória

3.1 UNITY

UNITY (Unbounded Nondeterministic Iterative Transformations) je výpočtový model, dôkazový systém a zároveň aj spôsob, ktorým môžeme zobrazovať programy. Fráza "*UNITY program*" znamená "*program zapísaný v neviazanej ne-deterministickej iteratívnej transformačnej notácii*". Nebol navrhnutý ako programovací jazyk, ale ako notácia na ilustrovanie programov a myšlienok. UNITY program sa skladá z deklarácie a inicializácie premenných a z množiny priradení. Vykonávanie programu začína z počiatočného stavu, ktorý spĺňa inicializované podmienky a vykonáva sa do nekonečna. V každom kroku programu sa vykoná jedno priradenie, ktoré sa vyberá nedeterministicky. Predpoklad je, že program je "*spravodlivý*". To znamená, že: každé priradenie je vybraté, teda vykonané nekonečne veľakrát. UNITY program nešpecifikuje kedy sa má daný príkaz vykonať, len popisuje, čo sa má vykonať a aký je počiatočný stav, pritom jediné obmedzenie je podmienka, že každé priradenie musí byť vykonané nekonečno veľakrát. V nasledujúcej kapitole si ukážeme syntax a sémantiku notácie, logiku a spôsob dokazovania

vlastností UNITY programov.

3.1.1 Syntax a sémantika

program ⟶ **Program** *program-name*
declare *declare-section*
always *always-section*
initially *initially-section*
assign *assign-section*
end

Unity program sa skladá zo sekcií: "*declare-section*", "*always-section*", "*initially-section*", "*assign-section*".
Meno programu ("*program name*") je definované na začiatku programu.

"*Declare-section*" slúži na pomenovanie a definovanie typu premenných používaných v programe. Syntax je podobná syntaxu PASCAL-u. Základné typy sú integer a boolean, ale existujú aj typy ako array, set a sequence.

"*Always-section*" slúži na zapisovanie nových premenných ako funkcie iných premenných a na zapisovanie podmienok, ktoré musia vždy platiť počas behu programu. Táto sekcia nie je nevyhnutná pri písaní programu.

"*Initially-section*" slúži na inicializovanie deklarovaných premenných. Premenné, ktoré nie sú inicializované, majú náhodné hodnoty.

"*Assign-section*" obsahuje množinu priradení.

Vykonávanie začína z inicializačného stavu, ktorý je definovaný v "*Initially-section*". V každom kroku programu sa náhodne vyberie jedno priradenie na vykonanie z množiny v sekcií "*Assign-section*". Priradenie je vybraté náhodne. Keďže program sa vykonáva do nekonečna, každé priradenie bude

vykonané nekonečne veľakrát, tým pádom podmienka "*spravodlivosti*" bude splnená.

Stav programu sa volá "*fixed point*" (fixný bod) práve vtedy keď, vykonanie hocijakého priradenia programu nemení aktuálny stav programu. Unity programy nemajú vstupné/výstupné hodnoty.

3.1.1.1 Declare-section

Syntax definovania deklarácie premenných typu integer a boolean je nasledovný:

$$N \text{ - počet premenných}$$
$$variable_0,\ variable_1,...,variable_N : typ$$

Syntax definovania deklarácie premenných typu array je nasledovný:

$$N \text{ - počet premenných}$$
$$size \text{ - veľkosť pola}$$
$$variable_0,\ variable_1,...,variable_N : \text{array}[1..veľkosť] \text{ of } typ$$

Na príklad:

declare

a : ***integer***

$b,\ c,\ d$: ***boolean***

$A,\ M$: array[1..*10*] of ***integer***

K : array[1..*3*] of ***boolean***

3.1.1.2 Assignment statement

$$\begin{aligned} \textit{assignment-statement} &\longrightarrow \textit{assignment-component} \\ &\quad \{\ //\ \textit{assignment-component}\} \\ \textit{assignment-component} &\longrightarrow \textit{enumerated-assignment} \\ &\quad /\ \textit{quantified-assignment} \end{aligned}$$

Assignment-statement sa skladá z jedného alebo z viacerých *assignment-component*-ov rozdelených znakom "||". *Assignment-component* môže byť typu *enumerated-assignment* a *quantified-assignment*. Premenné sa môžu vyskytnúť aj viac ráz na ľavej strane, je však zodpovednosťou programátora, aby všetky hodnoty, ktoré sú im priradené, boli identické. Na priraďovanie sa používa znak ":=". Každy *Assignment-component* v jednom *Assignment-statement*-te je vykonané samostatne, nezávislo a simultánne.

Na príklad:

$x, y, z := 0, 1, 2$

Podobné viacnásobné priradenie môžeme zapísať ako množinu *Assignment-component*-ov rozdelených znakom "||" :

$x, y := 0, 1 \ \ || \ \ z := 2$

alebo

$x := 0 \ \ || \ \ y := 1 \ \ || \ \ z := 2$

Môže sa použiť aj kvantifikačná notácia na priradenie hodnôt do premenných:

$< \ || \ \ i : 0 \leq i \leq N :: A[i] := B[i] >$

je ekvivalentné s :

$A[0] := B[0] \ \ || \ \ A[1] := B[1] \ \ || \ \ ... \ \ || \ \ A[N] := B[N]$

Nasledujúca notácia je používaná v matematike na definovanie hodnôt premenných na základe podmienok:

$$x = \begin{cases} -1 & \text{if } \ y < 0 \\ 0 & \text{if } \ y = 0 \\ 1 & \text{if } \ y > 0 \end{cases} \tag{3.1}$$

Podobná notácia sa používa aj v UNITY, ale jednotlivé možnosti sú rozdelené znakom "$\sim$":

$$\begin{aligned} x := -1 \ \ if \ \ y < 0 \sim \\ 0 \ \ if \ \ y = 0 \sim \\ 1 \ \ if \ \ y > 0 \end{aligned}$$

Enumerated-assignment

enumerated-assignment $\longrightarrow$ *variable-list* := *expr-list*
variable-list $\longrightarrow$ *variable*{,*variable*}
expr-list $\longrightarrow$ *simple-expr-list* | *conditional-expr-list*
simple-expr-list $\longrightarrow$ *expr*{,*expr*}
conditional-expr-list $\longrightarrow$ *simple-expr-list* *if* *boolean-expr*
{,*simple-expr-list* *if* *boolean-expr*}

Enumerated-assignment sa skladá zo zoznamu premenných na ľavej strane a zo zoznamu výrazov na pravej strane. Zoznam môže byť bez podmienky alebo s podmienkou. Priraďuje hodnoty výrazov na pravej strane znaku ":=" príslušným premenným na ľavej strane znaku ":=". Priradenie je podobné ako klasické viacnásobné priradenie. Najprv sa vyhodnotia všetky výrazy na pravej strane a potom sa priradia premenným na ľavej strane. Priradenie je úspešné iba vtedy ak počet výrazov na pravej strane a počet premenných na ľavej strane je rovnaký.

Priradenie s podmienkou (*conditional-expr-list*) priradí všetky *simple-expr-list* s podmienkou kde podmienka je platná (*true*). Ak ani jedna podmienka

nie je platná všetky premenné na ľavej strane ostávajú bez zmeny. Ak ich je platná viac ako jeden, tak zodpovedajúce *simple-expr-list* musia mať rovnakú hodnotu. Toto garantuje že každé priradenie je *determinsitické*.

Príklady:

1.Vymeň hodnotu x, y.

$x, y := y, x$

2.Nastav x na absolútnu hodnotu y.

$x := y \quad if \quad y \geq 0 \quad \sim \quad -y \quad if \quad y \leq 0$

keď $y = 0$ na priradenie môže byť použité aj y aj $-y$

3.Pridaj $A[i]$ k sum a zvýš i kým $i < N$.

$sum, i := A[i], i + 1 \quad if \quad i < N$

Quantified-assignment

quantified-assignment $\longrightarrow <$ || *quantification assignment-statement*$>$

quantification $\longrightarrow$ *variable-list : boolean-expr ::*

Premenné z *variable-list* sa nazývajú "viazané" alebo "kvantifikované". Rozsah kvantifikácie je definovaná zátvorkami "<" a ">". Podmienka (*boolean-expr*) by mala obsahovať viazané premenné v definovanom rozsahu, konštanty alebo premenné z programu. "Prípad vyhovujúci kvantifikátor" je množina hodnôt viazaných premenných pre ktoré platí *boolean-expr*.

Príklad:

1.Majme array $A[0..N]$ a $B[0..N]$, priradme $max(A[i], B[i])$ do $A[i]$, pre každé i, $0 \leq i \leq N$.

$< \; || \, i : 0 \leq i \leq N :: A[i] := B[i] \; if \; A[i] < B[i] >$

3.1.1.3 Assign-section

assign-section $\longrightarrow$ *statement-list*

statement-list $\longrightarrow$ *statement* { $\Box$ *statement* }

statement $\longrightarrow$ *assignmen-statement* / *quantified-statement-list*

quantified-statement-list $\longrightarrow$ $<$ $\Box$ *quantification statement-list*$>$

V tejto sekcii sa definujú všetky *assignment-statementy*. Je dôležité aby počet tvrdení bolo konečné. Toto zaručuje podmienka, že *bool-expr* v kvantifikácii (*quantification*) nesmie obsahovať premenné, ktorých hodnota sa môže zmeniť počas behu programu.

Príklady:

1.Priradenie jednotkovej matice do $U[0..N, 0..N]$.

2 *priradenia*

assign

$< \ || \ j,k : 0 \leq j \leq N \wedge 0 \leq k \leq N \wedge j \neq k :: U[j,k] := 0 >$

$\Box < \ || \ j : 0 \leq j \leq N :: U[j,j] := 1 >$

$(N+1)^2$ *priradení*

assign

$< \ \Box \ j,k : 0 \leq j \leq N \wedge 0 \leq k \leq N :: U[j,k] := 0$

$if \ j \neq k \ \sim \ 1 \ if \ j = k >$

3.1.1.4 Initially-section

V tejto sekcií sa definujú hodnoty premenných. Syntax je tá istá, ako v *assign-section*, len na priradenie hodnoty premennej sa používa znak "=". Hodnoty môžu byť konštanty alebo funkcie inicializačných hodnôt iných premenných. Na správne definovanie inicializačnej sekcie musia platiť tri podmienky: (1)

premenné sa môžu vyskytovať na ľavej strane maximálne iba raz, (2) existuje usporiadanie rovností také, že každá premenná v kvantifikácii je buď viazaná, alebo sa nachádza na ľavej strane nejakej predchádzajúcej rovnosti (táto podmienka zaručuje, že program je kompilovateľný), (3) existuje usporiadanie rovností také, že každá premenná na pravej strane, alebo v indexe sa nachádza na ľavej strane nejakej predchádzajúcej rovnosti.
Príklady:

initially

$$N = 3$$
$$\square < ||k : 0 < k \leq N :: A[N-k] = k >$$

v takomto prípade nie je možné vymeniť znak "$\square$" na "$||$", vtedy by neexistovalo usporiadanie príkazov také, že N je inicializované pred jeho použitím

initially

$$B[0] = 0||N = 2$$
$$\square < \square i : 0 < i \leq N :: A[i] = B[i-1]$$
$$\square B[i] = A[i] >$$

takýto zápis môžeme chápať ako:
$A[1] = B[0], B[1] = A[1], A[2] = B[1], B[2] = A[2]$

3.1.1.5 Always-section

Slúži na zapisovanie nových premenných ako funkcie iných premenných a na zapisovanie podmienok, ktoré musia vždy platiť počas behu programu. Syntax je podobná k syntaxu definovanej v *initially-section*. Premenná na ľavej strane sa nazýva transparentná, ak je funkciou netransparentných a nie je na ľavej strane inicializácií alebo priradení. Platia tiež rovnaké podmienky

ako pri *initially-section*.

Na príklad:

ne - počet zamestnancov

nf - počet mužov

nm - počet žien

Vzťah

$$ne = nf + nm$$

Platnosť vzťahu $ne = nf + nm$ musí byť počas celého vykonávania zachovaný, preto môžeme vždy modifikovať ne podľa toho ako sa zmenilo nf alebo nm. Ďalšou možnosťou je vytvorenie funkcie $ne(x, y)$, ktorá by bola neustále volaná s premennými nm, nf a vracala by novú hodnotu ne. Novou možnosťou, ktorú poskytuje *always-section* je definovanie vzťahu $ne = nf + nm$ v sekcii *always-section*. Premenná ne nesmie byť inicializovaná, ani do nej nesmie byť nič priradené v *assign-section*. Avšak ne sa vyskytuje v podmienkach a na pravých stranách v priradeniach, pričom každý výskyt ne by mal byť nahradený jeho definíciou $nf + nm$, bez žiadnej zmeny sémantiky programu.

Sekcia *always* nie je nevyhnutná. Každá transparentná premenná môže byť prepísaná na netransparentnú. Má to rôzne výhody. Umožňuje ľahšie dokazovanie pomocou množiny invariantov (3.1.2.4). Transparentné premenné môžeme chápať ako "*makro-instrukcie*", ktorých definícia môže byť nahradená na hocijakom mieste v programe.

3.1.1.6 Dodatočná syntax

Quantified Expression

$$expr \longrightarrow < op\ quantification\ expr >$$

$$op \longrightarrow \min \mid \max \mid + \mid \times \mid \wedge \mid \vee \mid = \mid ...$$

Označenie *expr* znamená výraz a označenie *op* znamená operátor. Výsledok výrazu je aplikovanie operátora na množinu výrazov, ktoré získame dosadením uzemnených premenných do vnoreného výrazu. Ak neexistuje instancia kvantifikácie tak hodnota výrazu má hodnotu neutrálneho prvku operátora *op*.

Neutrálne hodnoty:

min	max	+	×	∧	∨	=
∞	−∞	0	1	*true*	*false*	*true*

Príklady:

V nasledujúcich prípadoch platí, $N \geq 0$.

$< \vee \ i : 0 \leq i \leq N :: b[i] >$.

{platí, ak nejake $b[j$ je *true*}

$< min \ i : 0 \leq i \leq N :: A[i] >$.

{najmenší prvok poľa $A[0..N]$}

$< + \ i : 0 \leq i \leq N \wedge A[i] < A[j] :: 1 >$.

{počet prvkov menších ako $A[k]$, ak $A[k]$ je v $A[0..N]$}

3.1.2 Logika

Definujme symboly $p, p', q, q', r, r', b, b'$ ako predikáty a s, t ako priradenia ("statment-y") programu. Predpokladajme, že každý program má aspoň jedno priradenie. Toto vieme zaručiť pridaním priradenia ktorý nemá žiadny vplyv na beh programu ako napr. $x := x$.
Výrok $\{p\}\ s\ \{q\}$ označuje, že po vykonaní "statment-u" s v hocijakom stave, ktorý spĺňa predikát p, prejdeme do stavu ktorý bude spĺňať predikát q. Predikát p nazývame predpoklad ("precondition"), predikát q nazývame záver ("postcondition"). Predpoklad je, že každé priradenie raz skončí. Toto vždy platí ak vyhodnotenie každého výrazu v priradení, raz skončí. Tvrdenie $\{true\}\ s\ \{p\}$ hovorí, že niekedy raz ("eventually") p bude platiť.

Definície faktov o tvrdeniach vo formáte $\frac{h}{c}$, kde h označuje hypotézu a c konklúziu, záver.

$$\frac{}{\{p\}\ s\ \{true\}}$$

$$\frac{}{\{false\}\ s\ \{q\}}$$

$$\frac{\{p\}\ s\ \{true\}}{\neg p}$$

$$\frac{\{p\}\ s\ \{q\}\ ,\ \{p'\}\ s\ \{q'\}}{\{p \vee p'\}\ s\ \{q \vee q'\}}$$

$$\frac{p' \Rightarrow p\ ,\ \{p\}\ s\ \{q\}\ ,\ q \Rightarrow q'}{\{p \vee p'\}\ s\ \{q \vee q'\}}$$

3.1.2.1 Kvantifikované tvrdenia

Pre program F majme nasledovné kvantifikované tvrdenia:

$$< \forall \ s : s \ in \ F :: \{p\} \ s \ \{q\} >$$

a

$$< \exists \ s : s \ in \ F :: \{p\} \ s \ \{q\} >$$

označujú, že tvrdenie $\{p\} \ s \ \{q\}$ platí pre všetky priradenia v programe F a existuje aspoň jedno priradenie pre ktoré platí. Ak program obsahuje kvantifikované priradenia, kvantifikácia platí pre každé jedno priradenia zo "statement-list"-u.

$$< \forall \ s \ : \ s \ in \ < \Box \ i \ : \ b(i) \ :: \ t(i) > :: \{p\} \ s \ \{q\} >$$

je dokázané pomocou tvrdenia $\{p \ \wedge \ b(i)\} \ t(i) \ \{q\}$. Podobne,

$$< \exists \ s \ : \ s \ in \ < \Box \ i \ : \ b(i) \ :: \ t(i) > :: \{p\} \ s \ \{q\} >$$

je dokázané ukázaním, že existuje j také, že platí $b(j)$ a súčasne platí aj $\{p \ \wedge \ b(i)\} \ t(i) \ \{q\}$.

3.1.2.2 Výpočtový model vykonávania programu

Pre každý program máme množinu vykonávacích sekvencií, ktoré sú nekonečné. Každá je jedným z možných sekvencií vykonávania programu. Nech R je jedna sekvencia z tejto množiny a $R_i \geq 0$ označuje i-ty element sekvencie. Každý R_i je n-tica ktorá obsahuje $R_i.state$ - stav a $R_i.label$ - označenie, kde label označuje ktoré priradenie sa vykonalo v i-tom stave. Stav R_0 je inicializačný stav. Inicializačný stav nemusí byť rovnaký pre každú vykonávaciu sekvenciu programu. Stav R_{j+1} je jednoznačne určený stavom R_j a label-om R_j. Teda stav R_0 a $\{R_k.label \mid 0 \leq k < j\}$ určujú R_j stav. Kvôli pravidla "spravodlivosti" máme nasledujúcu podmienku:

Pre každé R a priradenie s, $R_i.label = s$, pre nekonečné i:

$p[R_j]$ - znamená, že p platí v stave $R_j.state$.
Tvrdenie $\{p\}\ \ s\ \ \{q\}$ znamená, že pre každé R a i,
$(p[R_i]\ \ \wedge\ \ R_i.label = s) \Rightarrow q[R_{i+1}]$

3.1.2.3 Základné pojmy

V tejto sekcii sú definované tri základné logické relácie: *unless* (a špeciálne formy ako *stable* a *invariant*), *ensures* a $leads{-}to$. Výraz fixný-bod ($fixed\ \ point$) je tiež predstavený.

Vlastnosti programu majú nasledovnú formu:

$p\ \ unless\ \ q$
$p\ \ je\ \ stable$ (môžeme zapísať aj ako $stable\ \ p$)
$p\ \ je\ \ invariant$ (môžeme zapísať aj ako $invariant\ \ p$)
$p\ \ ensures\ \ q$
$p \mapsto q$

kde p a q sú predikáty programu.
Vlastnosti programu *unless* , *stable* a *invariant* sú nazývané ako $safety$ vlastnosti. *Ensures* a $leads - to$ sú zase nazývané ako *progress* vlastnosti. "Safety" vlasnosť znamená, že vlasnosť ktorá je $safety$ zaručuje, že žiadný zlý krok sa nestane v programe. Vlasnosť "Progress" zase zaručuje, že určite sa stane nejaký správný, dobrý pokrok. Používa sa univerzálna kvantifikácia,

$x = k\ \ unless\ \ x > k$

kde x je premenná programu a k je ľubovoľná premenná znamená:

$< \forall\ \ k\ \ ::\ \ x = k\ \ unless\ \ x > k >.$

3.1.2.4 Unless

Pre daný program F, $p\ unless\ q$ je definované nasledovne:

$$p\ unless\ q\ \equiv <\forall\ s\ :\ s\ in\ F\ ::\ \{p\ \wedge\ \neg q\}\ s\ \{p\ \vee\ q\}>.$$

Formula znamená, ak p je $true$ teda platí a q je $false$ teda neplatí v nejakom stave programu počas výpočtu, tak v ďalšom korku p ostáva $true$ teda bude dalej platit, alebo q bude platit teda zmení sa jej hodnota na $true$.
Preto ak p platí v nejakom stave počas výpočtu programu F, môžeme povedať:

- q nebude nikdy platiť a p bude stále platiť, alebo
- q bude určite raz (eventually) platiť ak p platí aspoň pokiaľ q začne platiť

Na príklad:

1. x neklesne

   $$x = k\ \ unless\ \ x > k$$

   alebo

   $$x \geq k\ \ unless\ \ x > k$$

   alebo

   $$x \geq k\ \ unless\ \ false$$

2. Správa ostáva v kanáli, až kým nie je prijatá a potom je vymazaná z kanálu:

   $$inch\ \wedge\ \neg rcvd\ \ unless\ \neg inch\ \wedge\ rcvd$$

Speciálne prípady unless: stable a invariant

$$p \text{ is } stable \equiv p\ \ unless\ \ false$$

$$p \text{ is } invariant \equiv (initial\ \ condition\ \ \Rightarrow q)\ \wedge\ q\ \ is\ \ stable$$

Ak predikát p je *stable*, tak ak raz bude platit teda bude *true*, potom ostane vždy *true*, teda bude vždy platit. *Invariant* je vždy true. Všetky stavy ktoré sa vyskytnú počas vykonávania programu musia spľňajú všetky *invarianty*. Tieto dva koncepty sú mimoriadne, dôležité k dokazovaniu rôznych vlastností programu.

Ak I, J sú *stable* predikáty programu F, aj $I \ \wedge \ J, I \ \vee \ J$ sú *stable*. Toto tvrdenie platí aj pre *invarianty*.

Substitučná axióma hovorí, ak $x = y$ je *invariant* programu, tak x môžeme nahradiť za y vo všetkých vlastnostiach programu. Toto je jednoduché zovšeobecnenie Leibniz-ovo pravidla o substitúciách vo výrazoch. Ak I je *invariant* je zameniteľný s *true* a opačne. Toto umožnuje lachšie dokazovanie.
Príklad:

Chceme dokazať $p \ is \ stable$ a vieme, že platí $p \ unless \ q$
a $\neg q \ is \ invariant$.

$p \ unless \ q$ - predpoklad

$p \ unless \ false$ - z predchádzajúcich

$p \ is \ stable$ - z definície

Ak I je *invariant*, tak p môže byť zameniteľné s $I \ \wedge \ p$ alebo $\neg I \ \vee \ p$ a podobne. Dôkaz, $p \ is \ stable$ je takto oveľa ľahšie, keďže stačí ukázať, že $I \ \wedge \ p \ is \ stable$.

Predikáty definované v $always-section$ môžu byť považované za *invariantov*. Tieto *invarianty* sú získané takým istým spôsobom ako inicializačný stav z $initailly-section$. Pomocou týchto *invariantov* získaných z $always-section$ je oveľa lahšie dokazovanie iných vlastností.

3.1.2.5 Ensures

Ensures slúži na zadefinovanie najzákladnejšej *progress* vlastnosti UNITY programov. Pre daný program F, $p\ ensures\ q$ je zadefinované nasledovne:

$$p\ ensures\ q\ \equiv\ p\ unless\ q\ \wedge\ <\exists s:\ s\ in\ F\ ::\ \{p\ \wedge\ \neg q\}\ s\ \{q\}>$$

Formula znamená, ak p platí, teda je *true* v nejakom stave výpočtu, tak p ostane platné, teda *true* pokiaľ q neplatí, teda je $false$ ($p\ unless\ q$) a určite raz (eventually) sa stane q *true*, teda bude platiť po vykonaní nejakého príkazu s.

Na príklad:

1. Dôkaz, že x je nesklesajúca a raz stúpne

   $x = k\ \ ensures\ \ x > k$

   znamená

   $<\forall k\ \ ::\ \ x = k\ \ unless\ \ x > k>$ a

   $<\forall k\ \ ::\ <\exists\ s\ \ ::\ \ \{x = k\}\ \ s\ \ \{x > k\}>>$

   Preto, pre každú rôznu hodnotu k, rôzne priradenie programu musí zvýšiť x.

3.1.2.6 Leads-to

Skoro všetky *progress* vlasnosti programov sú definované pomocou $leads-to$ ($\mapsto$). Program má vlasnosť $p\ \mapsto\ q$ len vtedy, ak táto vlastnosť môže byť odvodená, konečným počtom aplikácií nasledujúcich pravidielt:

- $$\frac{p\ ensures\ q}{p\ \mapsto\ q}$$

- $$\frac{p\ \mapsto\ q\ ,\ q\ \mapsto\ r}{p\ \mapsto\ r}$$

pre lubovoľnú množinu W

- $$\frac{<\forall\ m{:}\ m\ \in\ W{::}\ p(m)\ \mapsto\ q>}{<\exists\ m{:}\ m\ \in\ W{::}\ p(m)\ \mapsto\ q>}$$

Z vlastnosti $p\ \mapsto\ q$ môžeme odvodiť, že ak p sa bude platiť teda stane sa *true*, tak q je alebo bude platiť, teda stane sa *true*. Nemožno však odvodiť, že p ostane platné, až kým q nie je *platn*. Toto je rozdiel medzi *ensures* a $leads-to$. Z $p\ \mapsto\ q$ môžeme odvodit:

$$p[R_i]\ \Rightarrow\ <\exists\ j\ :\ j \geq i\ ::\ q[R_j]>.$$

Používa sa notácia $p\ \mapsto\ q \mapsto\ r$ na zápis faktu, že platí $p\ \mapsto\ q$ a $q\ \mapsto\ r$. Príklad:

Dôkaz, že program má vlasnosť $x \neq 0\ \mapsto\ x = 0$

$x \neq 0$ *ensures* $x \geq 0$, z programu
$x \neq 0\ \mapsto\ x \geq 0$, z definície $leads-to$
$x \geq 0$ *ensures* $x = 0$, z programu
$x \geq 0\ \mapsto\ x = 0$, z definície $leads-to$
$x \neq 0\ \mapsto\ x = 0$, tranzizitivita, z definície $leads-to$
□

3.1.2.7 Fixed Point

3.1.3 Časové Automaty

3.1.4 Model Checking

3.1.5 UPPAL

Kapitola 4

Prehľad problematiky

//TBD

Kapitola 5

Návrh

//TBD

Kapitola 6

Implementácia

//TBD

Kapitola 7

Výsledky

//TBD

Kapitola 8

Záver

//TBD

Literatúra

[Dav09] Alexandre David. A theory of timed automata. *Computer Science Department, Stanford University, Satnford, CA 94305-2095, US*, 11, 2009.

[FA93] Kimmi S. Pettersson Flemming Andersen, Kim dam Petersen. Program verification using hol-unity. *Tele Danmark Research, Lyngso Allé 2, DK-2970 Horsholm*, 15, 1993.

[HCC89] Gruia-Catalin Roman H. Conrad Cunningham. A unity-style programming logic for a shared dataspace language. *Washington University Open Scholarship*, 17, 1989.

[JB95] Frederik Larsson Paul Pettersson Wang Yi Johan Bengtsson, Kim Larsen. Uppaal - a tool suite for automatic verification of real-time systems. *Department of Computer Systems, Uppsala University, SWEDEN*, 12, 1995.

[KC89] Jayadev Misra K.Mani Chandy. *Paralel Program Design*, volume 516. Addison-Wesley Publishing Company, Inc., 1989.

[Mis94a] Jayadev Misra. A logic for concurrent programming. *Department of Computer Sciences The University of Texas at Austin*, 48, 1994.

[Mis94b] Jayadev Misra. A logic for concurrent programming - safety. *Department of Computer Sciences The University of Texas at Austin*, 48, 1994.

[Pau94] Lawrence C. Paulson. Mechanizing unity in isabelle. *Cambridge Computer Laboratory*, 27, 1994.

[Pfi95] Peter Pfippinghaus. Fundamental study on the logic of unity. *SIEMENS AG, Corporate Research and Development, Basic Technologies, Software and Engineering, Otto-Hahn-Ring 6, D-81739 München, Germany*, 41, 1995.

[RA92] David L. Dill*** Rajeev Alur**. A theory of timed automata. 53, 1992.

[Rao95] Josyula Ramachandra Rao. *Extensions of the UNITY Methodology*, volume 181. Springer-Verlag Berlin Heidelberg, 1995.

Zoznam obrázkov