MASARYKOVA UNIVERZITA

FAKULTA INFORMATIKY

Verifikácia hybridných systémov

Bakalárska práca

Brno 2010 Mária Svoreňová

Prehlasujem, že táto práca je mojím pôvodným autorským dielom, ktoré som vypracovala samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používala alebo z nich čerpala, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

V Brne 4. 1. 2010 Mária Svoreňová

Poďakovanie:
Na tomto mieste by som rada poďakovala RNDr. Jiřímu Barnatovi, Ph.D., za cenné rady, pripomienky a čas, ktorý mi venoval.

Kľúčové slová

Hybridný automat, hybridný systém, verifikácia, verifikačný nástroj, dosažiteľnosť, bezpečnosť, syntéza ovládača, HyTech, PHAVer, Checkmate, d/dt, HSolver, MultiParametric Toolbox, MATISSE, KeYmaera.

Obsah

Úvod

V dnešnej dobe sú mnohé úkony riadené strojmi, či už ide o riadenie výroby, dopravy alebo zveľaďovanie ľudského komfortu. Chyby takýchto systémov môžu spôsobiť veľké finančné straty, ba čo viac, ohroziť ľudský život. Preto je pri ich navrhovaní potrebná kontrola, ktorá určí, či systém spĺňa naše požiadavky a či pracuje tak, ako má. Na to slúži verifikácia modelu systému. Úlohou mojej bakalárskej práce je spracovať prehľad nástrojov, ktoré sa zaoberajú verifikáciou rozličných vlastností hybridných systémov.

Na úvod si predstavíme pojem hybridného automatu, popíšeme formálne jeho syntax a predovšetkým sémantiku, ukážeme si pár základných príkladov. V druhej, ústrednej kapitole, bude spracovaný prehľad 8 nástrojov, ktoré sú zamerané na formálnu verifikáciu hybridných systémov. Popíšeme si predovšetkým ich vstupné obmedzenia, možnosti a v neposlednej rade si načrtneme samotný proces overovania vlastností. Bližšie si predstavíme nástroje HyTech a KeYmaera, ako zástupcov dvoch rozličných prístupov k danej problematike. Tretia kapitola bude porovnávať jednotlivé vlastnosti všetkých nástrojov, a to ako softwarové požiadavky a charakteristiky, tak aj hlavné črty a silu vstupného jazyka, obsahovať bude aj porovnanie nástrojov na spoločnom modele. Na záver si rozoberieme problém návrhu takéhoto programu.

U čitateľa predpokladám znalosť základov teórie automatov, temporálnych logík a logík ako takých, jemnú predstavu matematickej analýzy a úskalí integrovania, a taktiež znalosť základnej myšlienky klasického model checkingu.

Kapitola 1

Hybridné automaty

V praxi sa často stretávame so systémami správajúcimi sa podľa určitých pekných (spojitých) modelov, ktoré sa ale môžu meniť napríklad na základe hodnôt jednotlivých premenných či ubehnutého času. Ako príklady môžeme uviesť skákajúcu loptu, ovládanie rámp na železničnom prechode či riadenie uhýbacích manévrov vo vzdušnom priestore. Formálnou reprezentáciou takýchto systémov sú práve hybridné automaty a ich kompozície.

1.1 Definícia hybridného automatu

Hybridný automat je matematický model dynamického systému, ktorého chovanie sa skladá z diskrétnych, ale aj spojitých zmien. Samozrejme existuje viacero možností pre formálnu definíciu, my budeme od tejto chvíle pod hybridným automatom rozumieť nasledovnú štruktúru.

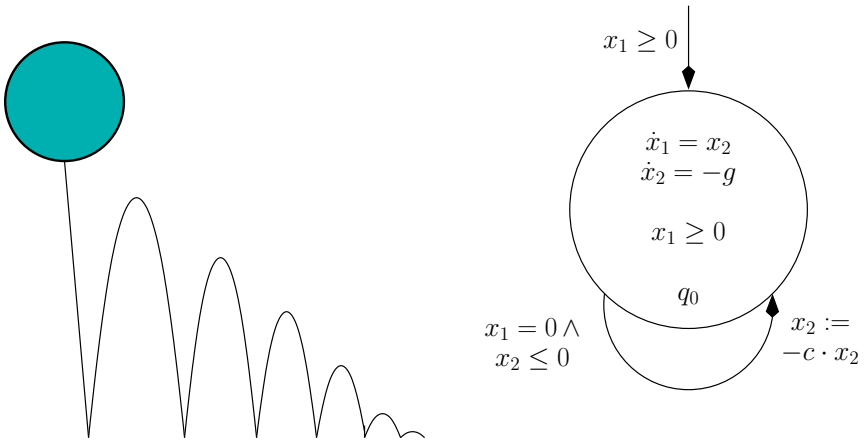
Definícia 1.1. *Hybridný automat* H je šestica $H = (Q, X, Init, f, Dom, R)$, kde

- Q je konečná množina diskrétnych stavov,
- $X \subseteq \mathbb{R}^n$ množina spojitých premenných, $n \in \mathbb{N}$,
- $Init \subseteq Q \times X$ množina iniciálnych stavov,
- $f \colon Q \times X \to X$ je vektorové pole, zložky (pre spojité premenné) nazývame flow funkcie, určujú zmenu jednotlivých premenných za jednotku času, teda deriváciu,
- $Dom \colon Q \to 2^X$ sú domény jednotlivých diskrétnych stavov,
- $R \colon Q \times X \to Q \times X$ je prechodová funkcia.

Pre grafickú reprezentáciu sa používa prechodový graf. Jeho vrcholy sú tvorené diskrétnymi stavmi, každý z nich má svoju doménu a flow funkcie.

Medzi dvoma vrcholmi $p, q \in Q$ existuje hrana, ak existujú $x, y \in X$ také, že $((p, x), (q, y)) \in R$, je naviac označená príslušným guardom – množinou všetkých takýchto prvkov x. Do iniciálneho diskrétneho stavu q, podobne ako v klasických automatoch, smeruje vstupná hrana nadpísaná podmnožinou $Y \subseteq X$ takou, že $(q, Y) \in Init$ sú všetky iniciálne stavy s diskrétnym stavom q.

Príklad 1.1. Jedným z najznámejších príkladov hybridného automatu je "bouncing ball" – skákajúca lopta.



Obr. 1.1: Bouncing ball – skákajúca lopta

Premenná x_1 určuje výšku lopty, x_2 jej rýchlosť. Konštanta $c \in \langle 0, 1 \rangle$ je koeficient pri odraze, g zrýchlenie. Automat je v stave q_0 počas spojitého skákania, pri dopade na zem, nastáva diskrétny prechod.
Zápis automatu podľa predchádzajúcej definície je nasledovný:

- $Q = \{q_0\}$,
- $X = \mathbb{R}^2$,
- $Init = Q \times \{(x_1, x_2) \in \mathbb{R}^2 | x_1 \geq 0\}$,
- $f(q_0, x_1) = x_2$, $f(q_0, x_2) = -g$,
- $Dom(q_0) = \{(x_1, x_2) \in \mathbb{R}^2 | x_1 \geq 0\}$,
- $R(q_0, \{(x_1, x_2) \in \mathbb{R}^2 | x_1 = 0 \wedge x_2 \leq 0\}) = (q_0, (x_1, -cx_2))$.

1.2 Výpočet - sémantika

Výpočet hybridného automatu je charakterizovaný tzv. hybridnou časovou množinou.

Definícia 1.2. *Hybridná časová množina* je konečná alebo nekonečná postupnosť intervalov $\tau = \{I_i\}_{i=0}^{N}$, pre ktorú platí

- $I_i = \langle \tau_i, \tau_i' \rangle$ pre všetky $i < N$,
- pre $N < \infty$, $I_N = \langle \tau_N, \tau_N')$ alebo $\langle \tau_N, \tau_N' \rangle$,
- $\tau_i \leq \tau_i' = \tau_{i+1}$ pre všetky i.

Jednotlivé intervaly budú reprezentovať časový interval, počas ktorého sa automat nachádza v jednom konkrétnom diskrétnom stave. Rovnosť krajných bodov susedných intervalov naznačuje, že zmena diskrétneho stavu je instantná, trvá 0 časových jednotiek. Množinu všetkých hybridných časových množín budeme značiť T.

Na množine $\tau \in T$ zavedieme *reláciu predchádzania* $\prec$. Pre $t_1 \in \langle \tau_i, \tau_i' \rangle \in \tau$, $t_2 \in \langle \tau_j, \tau_j' \rangle \in \tau$ píšeme $t_1 \prec t_2$ práve vtedy, keď $t_1 < t_2$ alebo $i < j$. Táto relácia udáva lineárne usporiadanie na hybridnej časovej množine.
Druhou užitočnou reláciou je *prefixová relácia* na množine T. Hovoríme, že $\tau = \{I_i\}_{i=0}^{N}$ je prefixom $\hat{\tau} = \{\hat{I}_i\}_{i=0}^{M}$, píšeme $\tau \sqsubseteq \hat{\tau}$, ak sú identické alebo τ je konečná, $N \leq M$, $I_i = \hat{I}_i$ pre všetky $i = 0, 1, \ldots, N-1$ a $I_N \subseteq \hat{I}_N$. Táto relácia udáva čiastočné usporiadanie na T.

Na klasifikáciu hybridných časových množín používame pojem dĺžky. Existujú však dve možnosti ako dĺžku chápať. *Diskrétnou dĺžkou* nazývame funkciu $\langle \cdot \rangle \colon T \to \mathbb{N} \cup \{\infty\}$, ktorá pre $\tau = \{I_i\}_{i=0}^{N} \in T$ vracia N v prípade, že τ je konečná postupnosť, ∞ inak (teda počet intervalov v postupnosti). *Spojitá dĺžka* je funkcia $\| \cdot \| \colon T \to \mathbb{R}_+$, ktorá pre $\tau = \{I_i\}_{i=0}^{N} \in T$ vracia hodnotu $\sum_{i=0}^{N} (\tau_i' - \tau_i)$ (teda množstvo času definované množinou).
Je zjavné, že ak $\tau \sqsubseteq \hat{\tau}$, potom $\langle \tau \rangle \leq \langle \hat{\tau} \rangle$ a $\|\tau\| \leq \|\hat{\tau}\|$.

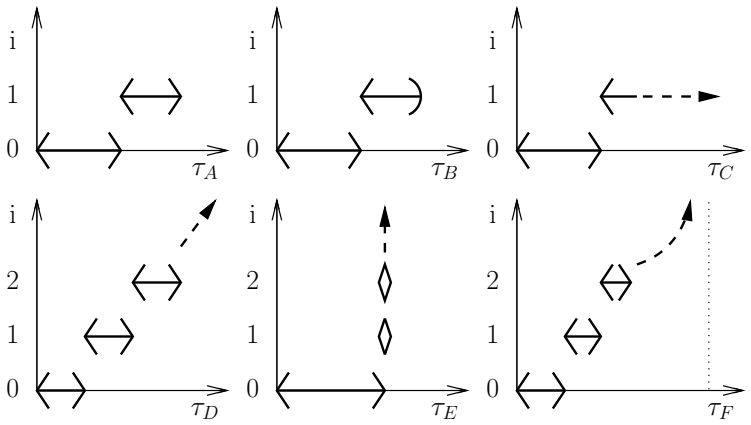
Definícia 1.3. Hybridná časová množina $\tau = \{I_i\}_{i=0}^{N} \in T$ sa nazýva

- *konečná*, ak $\langle \tau \rangle$ je konečná a posledný interval je uzavretý,
- *konečná – otvorená*, ak je konečná a posledný interval je ohraničený a sprava otvorený,
- *nekonečná*, ak $\langle \tau \rangle = \infty$ alebo $\|\tau\| = \infty$,
- *Zeno*, ak je nekonečná ale $\|\tau\| < \infty$.

Príklady jednotlivých množín sú na obrázku 1.2. Jedným z najznámejších systémov so Zeno chovaním je práve skákajúca lopta.

Definícia 1.4. *Hybridná trajektória* je trojica (τ, q, x), kde

- $\tau = \{I_i\}_{i=0}^{N} \in T$ je hybridná časová množina,



Obr. 1.2: τ_A konečná, τ_B konečná – otvorená, τ_C a τ_D nekonečná, τ_E a τ_F Zeno.

- $q = \{q_i\}_{i=0}^{N}$ je postupnosť funkcií $q_i \colon I_i \to Q$,
- $x = \{x_i\}_{i=0}^{N}$ je postupnosť funkcií $x_i \colon I_i \to \mathbb{R}^n$.

Definícia 1.5. *Výpočet* hybridného automatu $H = (Q, X, Init, f, Dom, R)$ je hybridná trajektória (τ, q, x), spĺňajúca podmienky

- počiatočná podmienka: $(q_0, x_0) \in Init$,
- diskrétny vývoj: $(q_{i+1}(\tau_{i+1}), x_{i+1}(\tau_{i+1})) \in R(q_i(\tau_i'), x_i(\tau_i'))$,
- spojitý vývoj: pre všetky i
  1. $q_i \colon I_i \to Q$ je konštantná, teda $q_i(t) = q_i(\tau_i)$ pre všetky $t \in I_i$,
  2. $x_i \colon I_i \to X$ je riešením diferenciálnej rovnice $\dot{x}_i = f(q_i(t), x_i(t))$ nad I_i s počiatkom v $x_i(\tau_i)$,
  3. pre všetky $t \in \langle \tau_i, \tau_i' \rangle$ platí $x_i(t) \in Dom(q_i(t))$.

1.3 Kompozícia hybridných automatov

Zložitejší hybridný systém je väčšinou jednoduchšie popísať niekoľkými hybridnými automatmi, ktoré reprezentujú jeho menšie časti, a tie následne spojiť kompozíciu do jedného veľkého celku.

Definícia 1.6. Nech $H_i = (Q_i, X_i, Init_i, f_i, Dom_i, R_i)$, $i = 1, \ldots, n$, sú hybridné automaty. Hybridný automat

$$H = (Q, X, Init, f, Dom, R) = H_1 \| H_2 \| \ldots \| H_n$$

nazveme kompozíciou (súčinom) týchto automatov, ak platí:

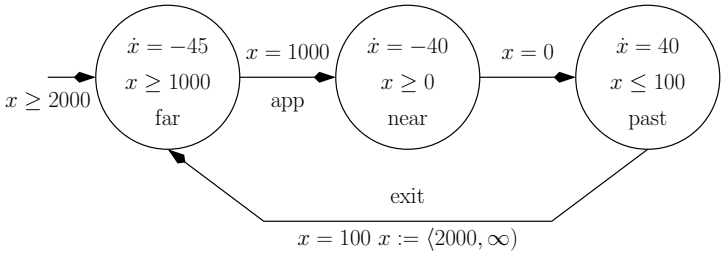
- $Q = Q_1 \times Q_2 \times \ldots \times Q_n$,
- $X = \bigcup_{i=1}^{n} X_i$,
- $Init \subseteq Q \times X$: ak $(q_1, q_2, \ldots, q_n, \vec{x}) \in Init$, potom pre všetky $i = 1, \ldots, n$ platí $(q_i, \vec{x}_i) \in Init_i$, kde $\vec{x}_i$ je vektor hodnôt z $\vec{x}$ patriacich premenným z X_i,
- $f\colon Q \times X \to X$: ak $f(q_1, q_2, \ldots, q_n, \vec{x}) = \vec{y}$, potom pre všetky $i = 1, \ldots, n$ platí $f(q_i, \vec{x}_i) = \vec{y}_i$, pričom predpokladáme, že nedochádza ku kolízii – stavy rôznych automatov neurčujú flow funkcie rovnakých premenných,
- $Dom\colon Q \to 2^X$: $Dom(q_1, \ldots, q_n) = \bigcap_{i=1}^{n}(X_1 \cup \ldots \cup X_{i-1} \cup Dom_i(q_i) \cup X_{i+1} \cup \ldots \cup X_n)$,
- $R\colon Q \times X \to Q \times X$: $R(q_1, \ldots, q_n, \vec{x}) = (s_1, \ldots, s_n, \vec{y})$ ak platí, že existuje práve jedno $1 \leq i \leq n$, pre ktoré $R_i(q_i, \vec{x}_i) = (s_i, \vec{y}_i)$, pre ostatné $1 \leq j \neq i \leq n$ platí $q_j = s_j$, $\vec{x}_j = \vec{y}_j$

Obmedzenie pre prechodovú reláciu kompozície, a to že v jednom momente mení diskrétny stav len jeden z pôvodných automatov, nie je v skutočnosti obmedzením. Ak sú v jednom momente splnené podmienky pre prechod vo viacerých automatoch, v kompozícii sa to prejaví ako postupný prechod, pričom medzi nimi neuplynie žiadny čas.

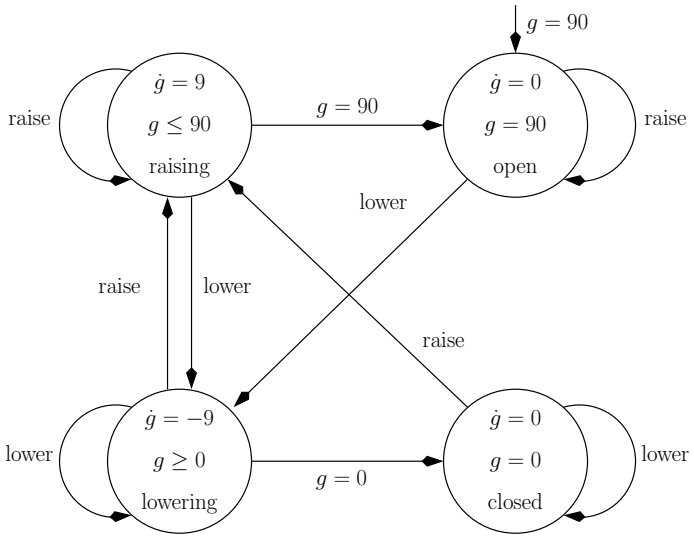
Automaty teda môžu medzi sebou komunikovať pomocou spoločných premenných. Často sa však využívajú aj synchronizačné značky, ktorými sú označované prechody. Ak je takýto prechod prevádzaný, značka slúži v iných automatoch ako guard.

Príklad 1.2. Ďalším príkladom hybridného systému je železničný prechod. Systém sa skladá z troch častí – vlak, rampy a ich ovládacia jednotka. Je oveľa intuitívnejšie popísať správanie týchto zložiek samostatne a potom ich skomponovať pomocou súčinu, ako analyzovať správanie tohto systému ako celku (minimálne z hľadiska počtu stavov).
Na obrázkoch 1.3, 1.4, 1.5 sú postupne znázornené automaty jednotlivých častí. Premenná x určuje vzdialenosť vlaku od železničného prechodu. Keďže $\dot{x}$ je derivácia x, udáva rýchlosť vlaku. Na začiatku je vlak ďaleko (*far*) od prechodu a pohybuje sa rýchlosťou 45 metrov za sekundu. Ako sa vlak približuje, senzor umiestnený 1000 metrov od prechodu vlak zachytí a pošle signál *app* ovládaču. Vlak spomalí na 40 metrov za sekundu. Ak je ovládač nečinný (*idle*) v čase obdržania signálu *app*, počká α sekúnd, kým vydá pokyn na uzavretie rámp. Toto oneskorenie je modelované hodinami t. V prípade, že sú rampy otvorené, začnú sa sklápať z 90 na úroveň 0 rýchlosťou 9 stupňov za sekundu. Pozícia rámp v stupňoch je reprezentovaná premennou

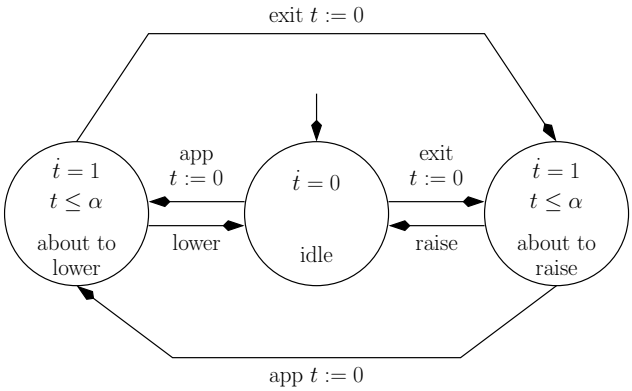
y. Druhý senzor, umiestnený 100 metrov za prechodom, rozpozná vzďaľujúci sa vlak, a pošle ovládaču signál *exit*, ktorý po ďalších α sekundách pošle rampám príkaz na zdvíhanie (*raise*). Pre jednoduchosť predpokladáme, že vzdialenosť medzi vlakmi je aspoň 1500 metrov.
Ovládač musí vždy akceptovať signály *app* a *exit*, rampy musia vždy akceptovať signály od ovládača.



Obr. 1.3: Vlak



Obr. 1.4: Rampy



Obr. 1.5: Ovládacia jednotka

Kapitola 2

Prehľad nástrojov

Existuje množstvo programov zaoberajúcich sa hybridnými automatmi, predovšetkým z akademického prostredia. Patria medzi ne modelovacie jazyky, vizuálne modelovacie nástroje, programy pre simuláciu, analýzu a verifikáciu. My sa zameriame na verifikačné nástroje.

Verifikácia slúži na dokazovanie, že analyzovaný systém spĺňa nami požadované vlastnosti, charakterizované nejakou špecifickou, obyčajne temporálnou, logikou. Formálna verifikácia je postavená na skúmaní stavového priestoru, čo môže byť, a vo väčšine reálnych prípadov aj je, veľmi vyčerpávajúce, stavový priestor hybridného systému je predsa nespočetný vďaka spojitému vývoju. Nájdenie všetkých stavov dosažiteľných z množiny iniciálnych stavov nazývame problémom dosažiteľnosti (reachability problem). Pre obecný dynamický systém je však nerozhodnuteľným problémom. V praxi sa hľadaná množina aproximuje zhora, v prípade potreby sa aproximácia zjemňuje, avšak terminácia takéhoto algoritmu nie je zaručená.

Často používaným prístupom je tiež obmedzenie sa na určitú podtriedu hybridných automatov, pre ktoré vieme množinu dosažiteľných stavov určiť presne. Model hybridného automatu je veľmi silný, no prílišná obecnosť nám bráni v automatickej analýze. Aby sme boli schopní exaktne počítať, systém musí byť dostatočne jednoduchý. Táto jednoduchosť sa týka hlavne flow funkcií, z ktorých v priebehu analýzy potrebujeme počítať hodnoty premenných v čase, invariantov diskrétnych stavov a guardov a efektov jednotlivých prechodov.

Ďalšou častou otázkou je bezpečnosť (safety) – majme určenú množinu "zlých" stavov, môže sa náš systém dostať do niektorého z týchto stavov? Problém bezpečnosti sa väčšinou rieši pomocou dosažiteľnosti, nájdeme všetky dosažiteľné stavy a pozrieme sa, či je prienik s množinou zlých stavov neprázdny. Ak áno, môžeme systém prehlásiť za bezpečný. V prípade, že sa

v prieniku nachádza nejaký stav, vieme pomocou spätného sledovania nájsť protipríklad, cestu z iniciálneho stavu do stavu nebezpečného, čo je užitočné pri odstraňovaní chýb v systéme. Niektoré nástroje ponúkajú možnosť generovania ovládača (controller), ktorý zabraňuje prechodu systému do nežiaducich stavov.

U jednotlivých nástrojov si uvedieme ich vstupné formalizmy, porovnáme ich výhody a nevýhody a popíšeme princípy verifikačného algoritmu. U nástrojov HyTech a KeYmaera si tento proces predstavíme bližšie.

2.1 HyTech

HyTech je jedným z prvých nástrojov pre prácu s hybridnými automatmi, prvá verzia vznikla v roku 1995 na Kalifornskej univerzite v Berkeley, autormi sú Thomas A. Henzinger, Pei-Hsin Ho a Howard Wong-Toi. Jeho hlavnou funkciou je symbolický model checking lineárnych hybridných automatov, podtriedy hybridných automatov, u ktorých je možná analýza pomocou počítania s mnohostenmi. Ďalšou veľmi užitočnou schopnosťou je prevádzanie parametrickej analýzy, teda určenie hodnôt parametrov tak, aby systém spĺňal požiadavky.

Prvá verzia programu bola postavená na komerčnom nástroji Mathematica a s predikátmi sa narábalo ako so symbolickými formulami. Na základe pozorovania, že lineárne predikáty nad n premennými definujú zjednotenie mnohouholníkov v $\mathbb{R}^n$, druhá geenrácia HyTechu kombinuje Mathematicu s knižnicou pre výpočty nad mnohouholníkmi. Najnovšia tretia verzia, z roku 1997, je celá písaná v C++ a je až trikrát rýchlejšia ako predchádzajúca.

2.1.1 Lineárne hybridné automaty

Na definíciu lineárnych hybridných automatov potrebujeme niekoľko pojmov:

- *atomický lineárny predikát* je nerovnosť medzi racionálnymi konštantami a lineárnou kombináciou spojitých premenných z X s racionálnymi koeficientami, napr. $3x_1 - x_2 \leq \frac{3}{4}$,
- *konvexný lineárny predikát* je konečná konjunkcia atomických lineárnych predikátov,
- *lineárny predikát* je konečná disjunkcia konvexných lineárnych predikátov.

Hybridný automat nazveme *lineárnym*, ak platí:

- linearita: pre každý diskrétny stav $q \in Q$ sú flow funkcie $f(q,x)$ pre všetky $x \in X$, $Dom(q)$ aj $Init$ zúžené na q konvexné lineárne predikáty, guardy všetkých prechodov sú taktiež konvexné lineárne predikáty,
- nezávislosť spojitého vývoja: pre každé $q \in Q$, $x \in X$ je $flow(q,x)$ predikát iba nad deriváciami premenných z X (neobsahuje samotné premenné).

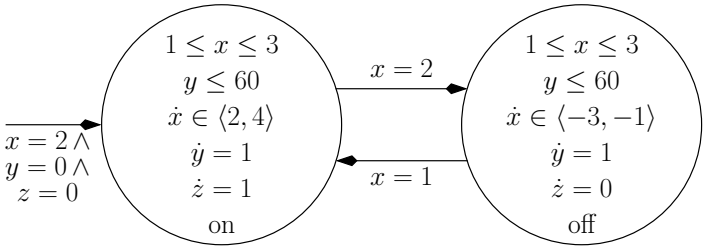
Druhá požiadavka zabezpečuje, že spojitý vývoj je nezávislý na konkrétnych hodnotách premenných, závisí iba od diskrétneho stavu. Na jednej strane je to dosť obmedzujúce, nemôžeme použiť napríklad $\dot{x} = x$, ale máme povolené premenné s typickým rastom ako hodiny ($\dot{x} = 1$), stopky ($\dot{x} = 1$ alebo $\dot{x} = 0$), premenné s obmedzeným rastom ($\dot{x} \in \langle 1, 3 \rangle$). Naviac môžeme vyjadriť závislosť premenných, napríklad $\dot{x} \geq \dot{y}$ - premenná x rastie vždy aspoň tak rýchlo ako premenná y.

Často sa stáva, že systém, ktorý chceme verifikovať, nie je v lineárnej forme. Vývojári HyTechu uvádzajú dve techniky pre linearizáciu hybridného automatu. Prvou je prevod premenných na hodiny. Premennú, ktorej hodnotu vieme určiť z uplynutého času od chvíle, kedy jej bola naposledy priradená nejaká konštantná hodnota, alebo z tejto konštantnej hodnoty, vieme previesť na hodiny. Rovnako aj jej výskyty v invariantoch, guardoch a podobne vieme nahradiť podmienkami na tieto hodiny. Druhá technika, z pôvodného názvu linear phase-portrait approximation, nahrádza nelineárne predikáty jednoduchšími lineárnymi predikátmi. Nelineárny predikát p' nahradíme slabším predikátom p tak, aby p implikoval p'. Teda ak aproximovaný systém spĺňa špecifikáciu, spĺňa ju aj pôvodný systém.

2.1.2 Analýza a verifikácia

Vstup pre HyTech pozostáva z dvoch častí. Prvá časť obsahuje textový popis skupiny lineárnych hybridných automatov vo vstupnom jazyku, ktoré v kompozícii tvoria verifikovaný systém. Druhú časť vstupu tvorí postupnosť príkazov pre analýzu. Jazyk pre analýzu je jednoduchý, využíva množstvo operácií vrátane booleovských operátorov, kvantifikátorov a funkcie Post. Funkcia Post počíta región (množinu stavov) dosažiteľný z aktuálneho regiónu pomocou spojitého vývoja bez zmeny stavu alebo pomocou jedného prechodu. Pre triedu lineárnych hybridných automatov platí, že pokiaľ počiatočný región je popísaný lineárnymi predikátmi, nasledujúci región určený funkciou Post vieme vypočítať a je tiež popísaný lineárnymi predikátmi. Pre zjednodušenie obsahuje HyTech zabudované makrá pre analýzu dosažiteľnosti, parametrickú analýzu, generovanie protipríkladu a aproximáciu výrazov.

Proces verifikácie si popíšeme na konkrétnom príklade. Budeme analyzovať termostat regulujúci teplotu, ktorý udržiava teplotu v miestnosti medzi 1 až 3°C, v prvej hodine. Na obrázku 2.1 je jeho lineárny hybridný automat, premenná x udáva teplotu, y sú globálne hodiny a z sú hodiny merajúce čas, počas ktorého je termostat zapnutý.



Obr. 2.1: Termostat

Najprv, ako príklad jednoduchého problému, budeme overovať, že termostat hreje menej ako $\frac{2}{3}$ času v prvých dvoch minútach. Vstup pre HyTech s príkazmi pre analýzu vyzerá nasledovne:

```
1 unsafe := y=2 \& z >= 2/3 y; \\
2 reachable := \\
3   reach forward from init\_states endreach;
       \\
4 if not empty(reachable \& unsafe)\\
5   then print trace to unsafe using reachable
       ;\\
6   else prints "Safety property satisfied
       .";\\
7 endif;
```

Overovanie prevedieme na overovanie bezpečnosti klasickým model checkingom, určíme, ktoré stavy považujeme za nebezpečné. V 2. a 3. riadku dáme príkaz na počítanie všetkých dosažiteľných stavov, tieto HyTech určí iterovaním funkcie Post na iniciálne stavy, pričom si uchováva pointery na predchodcov. V prípade, že prienik týchto dvoch regiónov je neprázdny, vygeneruje sa protipríklad pomocou spätných pointerov. V opačnom prípade môžeme systém označiť za bezpečný. Ako výsledok HyTech vráti výpočet automatu, kde stav a hodnoty premenných v krajných bodoch intervalov časovej množiny sú nasledujúce:

$$(\text{on}, 2, 0, 0), (\text{on}, 3, \frac{1}{2}, \frac{1}{2}), (\text{off}, 3, \frac{1}{2}, \frac{1}{2}), (\text{off}, 1, \frac{7}{6}, \frac{1}{2}), (\text{on}, 1, \frac{7}{6}, \frac{1}{2}), (\text{on}, \frac{8}{3}, 2, \frac{4}{3})$$

Na ďalšom príklade si ukážeme parametrickú analýzu. Nasledujúca postupnosť príkazov slúži na určenie času α, počas ktorého je termostat aktívny v prvej hodine:

```
1 unsafe := y=60 \& z>= alpha; \\
2 reachable := \\
3   reach forward from init\_states endreach;
      \\
4 bad_alpha_values := omit all locations \\
5   hide non_parameters in reachable & unsafe
      endhide; \\
6 prints "Spec. violated for parameter values
    :"; \\
7 print omit all location bad_alpha_values; \\
8 good_alpha_values := ~bad_alpha_values; \\
9 prints "Spec. satisfied for parameter values
    :" \\
10 print omit all locations good_alpha_values;
```

Tak ako predtým, určíme nebezpečné stavy a dosažiteľné stavy. Príkazy v 4. a 5. riadku prevedú existenčnú kvantifikáciu, aby sme dostali predikát popisujúci nebezpečné dosažiteľné stavy ako obmedzenie na parameter. Symbol ~ na 8. riadku značí negáciu. Ako výsledok dostaneme:

```
Spec. violated for parameter values:
alpha <= 36
Spec. satisfied for parameter values:
alpha > 36
```

2.1.3 Hodnotenie

HyTech ponúka efektívnu analýzu systémov reprezentovaných lineárnymi hybridnými automatmi, a to buď priamo alebo pomocou aproximácie. Používa presnú aritmetiku a dáva aj presnú odpoveď. Je však stavaný pre systémy s jednoduchšou dynamikou, väčšie množstvo parametrov, či na sebe závislé parametre mu taktiež robia problémy. Samotní tvorcovia priznávajú, že nájdenie dostatočne jednoduchého a zároveň dostatočne popisného stupňa abstrakcie systému je v podstate umením. To však platí aj pre ostatné verifikačné nástroje.

2.2 PHAVer

PHAVer – polyhedral hybrid automata verifier, vznikol vo francúzskom Verimagu, jeho autorom je Goran Frehse. Slúži na overovanie bezpečnosti hybridných I/O systémov s afinnou dynamikou. Taktiež je schopný počítať

simulačné relácie a rozhodovať ekvivalenciu medzi hybridnými automatmi. Posledná verzia PHAVeru vyšla vo februári 2007.

Ako vstup pre PHAVer môžeme použiť ľubovoľný systém, ktorý vieme popísať kompozíciou afinných hybridných I/O automatov. Oproti lineárnym hybridným automatom, hybridný I/O automat rozlišuje medzi premennými, ktoré zadáva užívateľ ako vstupné a tými, ktoré sú kontrolované deriváciami. Toto je dôležité pre určovanie ekvivalencie. Namiesto definovania samotnej derivácie kontrolovaných premenných, v hybridnom I/O automate sa definujú takzvané aktivity diskrétnych stavov. Náš záujem bude v afinných automatoch, tu sú aktivity určené ako konjunkcie lineárnych výrazov s premennými a ich deriváciami.

Pre takto definované systémy však dosažiteľnosť nie je rozhodnuteľná, preto ich zhora aproximujeme lineárnymi hybridnými automatmi (definícia v časti venovanej HyTechu). Myšlienka je nasledujúca: pre každý diskrétny stav q, s doménou $Dom(q)$ budeme aktivitu

$$\bigwedge_{i=1}^{m} a_i\dot{x}_i + b_i x_i \bowtie_i c_i \qquad \bowtie_i \in \{<, \leq\}$$

aproximovať lineárnymi výrazmi podľa pravidla

$$\forall i = 1, \ldots, m \qquad a_i\dot{x} \bowtie_i c_i - d_i \qquad d_i = \inf_{x \in Dom(q)} b_i x.$$

V prípade, že je aproximácia príliš hrubá, diskrétny stav sa rozdelí. Nadrovinu, podľa ktorej sa tak stane, môže definovať sám užívateľ na základe hlbších znalostí systému. Overovanie bezpečnosti ďalej pokračuje určením množiny dosažiteľných stavov a jej prienikom s množinou zlých stavov, pričom sa používa presná aritmetika podobne ako v HyTechu.

Hodnotenie

PHAVer je veľmi sľubný nástroj, aj keď v súčastnosti sa zdá, že jeho vývoj nepokračuje. Ponúka množstvo jedinečných možností ako počítanie simulačných relácií či určovanie ekvivalencie. Jeho algoritmus je efektívny a relatívne rýchly. Je síce aplikovateľný len na afinné hybridné I/O automaty, no podporuje kompozíciu. Pri rozdeľovaní diskrétnych stavov necháva voľnú ruku užívateľovi, čo umožňuje získať užitočné odpovede v krátkom čase.

2.3 Checkmate

Checkmate je verifikačný toolbox pre Matlab, vyvinutý v roku 2000 tímom výskumníkov na Univerzite Carnegie Mellon. Slúži na modelovanie, simuláciu a verifikáciu špeciálnej podtriedy hybridných automatov, z originálneho

názvu threshold event-driven hybridných systémov (TEDHS).

V TEDHS sa diskrétny stav mení po prekročení istej hranice – threshold. Táto je modelovaná nadrovinou. V reči našej definície z predchádzajúcej kapitoly to znamená, že guardy a invarianty sú lineárne nerovnosti a navzájom sa dopĺňajú, teda akonáhle nie je splnený invariant, je splnený nejaký guard, správanie je neblokujúce.

Checkmate modely sú konštruované pomocou pre Matlab klasických a dobre známych Simulink a Stateflow blokov. Spojité rovnice pre stavy, parametre a špecifikácia, ktorú budeme overovať, sú definované pomocou Simulink GUI a Matlabovských m-súborov, ktoré si vytvára sám užívateľ. TEDHS vznikne kombináciou troch blokov – prvý charakterizuje spojitý vývoj diferenciálnou rovnicou, druhý popisuje mnohosten, hranicu, a tretí, Stateflow blok, udáva diskrétnu dynamiku ako konečne stavový prechodový systém. Checkmate konvertuje takto určený TEDHS na polyhedral-invariant hybridný automat (PIHA). PIHA je podtrieda hybridných automatov s týmito obmedzeniami:

- spojitá dynamika je pre každý diskrétny stav charakterizovaná obyčajnou diferenciálnou rovnicou (ODE),
- každý guard je lineárna nerovnosť (nadrovina),
- efekt každého prechodu je identita,
- kým automat zotrváva v nejakom diskrétnom stave, nie je splnený žiadny guard. Z tohto obmedzenia plynie, že invarianty stavov sú konvexné mnohouholníky, definované ako konjunkcia doplnkov jednotlivých guardov, z toho pochádza aj názov polyhedral-invariant.

Tieto obmedzenia sú potrebné pre zjednodušenie verifikácie a umožnenie simulácie, aj keď značne redukujú možnosť aplikácie.

V Checkmate, formálna verifikácia je opäť prevádzaná počítaním množiny dosažiteľných stavov. Pretože takýto výpočet je pre systémy s dynamikou zadanou pomocou ODE veľmi náročný, využíva sa aproximácia. Pre systém vytvoríme faktorový prechodový systém, pre ktorý platí, že prechod medzi P_1 a P_2 existuje práve vtedy, ak existujú v pôvodnom systéme $p_1 \in P_1$, $p_2 \in P_2$ a medzi nimi prechod. Faktorový prechodový systém teda obsahuje všetky prechody originálneho systému, prípadne viac. Ak faktorový systém spĺňa špecifikáciu, spĺňa ju aj originálny. V opačnom prípade nám Checkmate ponúkne možnosť zjemnenia aproximácie. Takto postupujeme až kým nedostaneme kladnú odpoveď alebo sa nerozhodneme skončiť.

Hodnotenie

Checkmate je postavený na veľmi populárnom nástroji. Pretože môžeme používať obecné ODE, dokážeme v ňom modelovať oveľa komplexnejšie systémy ako v HyTechu (avšak stále len špeciálnu podtriedu hybridných systémov), no nepodporuje kompozíciu. Aproximácia, využívaná pri verifikácii, síce znižuje náklady na výpočet, no pri analýze dosažiteľnosti sú tieto náklady ešte stále príliš vysoké už aj pri použití piatich premenných.

2.4 d/dt

Nástroj d/dt vznikol v roku 2002 vo francúzskom výskumnom centre Verimag. Pomocou analýzy dosažiteľnosti rieši problémy ako bezpečnosť a syntéza ovládača pre bezpečnosť.

Ako vstup akceptuje d/dt hybridný automat s týmito obmedzeniami:

- spojitý vývoj je afinné zobrazenie tvaru $\dot{x} = Ax + Bu$, kde u nadobúda hodnotu z nejakého ohraničeného konvexného mnohouholníka,
- všetky invariaty a guardy prechodov sú definované ako konvexné mnohouholníky (pomocou konjunkcie lineárnych nerovníc),
- efekty prechodov sú tiež afinné zobrazenia.

Taktiež je možné určiť aproximačné parametre ako krok v čase alebo granularita aproximácie.

Množina dosažiteľných stavov je aproximovaná zhora pomocou ortogonálneho mnohouholníka, výpočet je postavený na numerickom integrovaní. Overovanie bezpečnosti určí prienik množiny dosažiteľných a zlých stavov. V prípade, že je neprázdny, na výstupe sa objaví obsah tohto prieniku. d/dt sa používa aj na syntézu ovládača, ktorý udržiava beh systému mimo množiny zlých stavov. Tento proces je založený na odvodení maximálnej invariantnej množiny, tj. množiny stavov, v ktorej keď bude ovládač systém udržiavať, je schopný predísť vstupu do zlých stavov. Táto množina sa aproximuje zdola, pomocou operátora predchodcu. d/dt následne odvodí pravidlá pre ovládač, ktoré obmedzujú invarianty a guardy pôvodného systému tak, aby výsledný automat sĺňal požadované bezpečnostné podmienky.

Hodnotenie

Funkcie nástroja d/dt sú zaujímavé, predovšetkým schopnosť využitia poznatkov z analýzy dosažiteľnosti pre odvodenie ovládača je pozoruhodná. Avšak podobne ako u iných nástrojov, je obmedzená expresivita, chýba komplexnosť pri špecifikovaní dynamiky systému a náklady sú príliš vysoké.

2.5 HSolver

HSolver je ďalší nástroj pre overovanie bezpečnosti. Používa myšlienku prevodu nekonečného stavového priestoru hybridného systému na konečný stavový priestor, a to tak, že spojitý priestor sa rozdelí do boxov – viacrozmerných pravouholníkov. Jeho "otcom" je rakúšan Stefan Ratschan, momentálne pôsobiaci v Prahe. Posledná verzia vyšla začiatkom tohto roka.

Vstup je definovaný pomocou tzv. obmedzení (constraint) pre jednotlivé diskrétne stavy – Booleovských kombinácií rovností či nerovností výrazov obsahujúcich spojité premenné, spolu s klasickými aritmetickými operáciami, no využiť môžeme napríklad aj funkcie exp, sin, cos. Takýmito obmedzeniami sú popísané iniciálne stavy, flow funkcie, domény i prechody. Flow funkcie však musia byť určené tak, aby boli diferencovateľné.

Po rozdelení stavového priestoru na boxy, HSolver použije RSolver, program, ktorý z obmedzení určí, ktoré body z boxov nie sú na žiadnej ceste z iniciálneho do zlého stavu (to môžu byť aj body stavového priestoru) a odstráni ich. V prípade, že po takomto odstraňovaní ostanú nejaké body, bezpečnosť systému nie je zaručená a program zjemní abstrakciu (nie vždy musí viesť k rozdeľovaniu boxov, pokiaľ je počet bodov v nich dostatočne nízky). Program končí, ak je schopný dokázať bezpečnosť alebo sú dosiahnuté hranice pre maximálny počet boxov či minimálny priemer. HSolver taktiež prevádza jednoduchú analýzu stavového priestoru, v tomto prípade sa z boxov odstraňujú body, ktoré nie sú na žiadnej ceste z iniciálneho stavu. Na záver máme možnosť grafického výstupu, kde sa na grafe zobrazí výsledok našej abstrakcie.

Hodnotenie

HSolver používa klasickú intervalovú metódu pre overovanie hybridných systémov, naviac ju rozširuje o algoritmus pre odstraňovanie nezaujímavých častí stavového priestoru, takže zjemnenie abstrakcie nemusí výlučne znamenať zjemnenie mriežky, čo je častou príčinou exponenciálneho nárastu v takýchto algoritmoch. Vstupný jazyk je jednoduchý, obmedzenia pre automat sú relatívne slabé, no nepodporuje kompozície. Podobne ako v predchádzajúcich prípadoch, cena veľmi výrazne stúpa s narastajúcim počtom premenných.

2.6 MultiParametric Toolbox

MultiParametric Toolbox (MPT) je rozsiahly toolbox pre Matlab. Autormi sú Michal Kvasnica (bývalý študent Slovenskej technickej univerzity v Bratislave), Pascal Grieder a Mato Baotic. Posledná verzia pochádza z konca roku

2006.

MPT pracuje s diskrétnymi hybridnými automatmi – čas neplynie spojito ale "po jednotkách" (čo vylučuje Zeno chovanie). Konkrétne s lineárnymi diskrétnymi automatmi, kde derivácia je lineárnou kombináciou premenných, a po častiach afinnými automatmi, tu môže derivácia naviac obsahovať pričítanie nejakej konštanty a derivácie sa môžu líšiť v rôznych diskrétnych stavoch. Vstup sa definuje priamo v toolboxe alebo pomocou jazyka HYSDEL, ktorý je určený pre modelovanie hybridných systémov špeciálne pre MPT.

MPT ponúka veľké množstvo operácií, ktoré sa dajú rozdeliť do štyroch základných kategórií – modelovanie, optimálne riadenie, analýza a výpočtová geometria (výpočty nad elipsami, počítanie konvexného obalu, konvexné zjednotenia atď.). Optimálne riadenie rieši otázku ovládača a generovanie jeho kódu, analýza skúma stavový priestor, hľadá invariantné množiny, rieši bezpečnosť. MPT zvláda tiež simuláciu a grafické výstupy.

Hodnotenie

MPT je na veľmi vysokom stupni vývoja a bežne sa používa v priemyselných aplikáciách, najmä na generovanie kódov pre ovládače. Algoritmus pre syntézu je samozrejme komplikovaný, no ponúka silnú infraštruktúru pre odstraňovanie chýb a ďalšie úpravy systémov. Naviac, ako Matlabovský toolbox, je vsadený do dobre známeho prostredia. Jazyk HYSDEL je však relatívne slabý, nepodporuje kompozície a jeho modely sú značne zjednodušené.

2.7 MATISSE

MATISSE je ďalší z rady Matlabovských toolboxov, rieši otázku bezpečnosti pre diskrétne lineárne hybridné systémy. Vznikol na Pensylvánskej univerzite v roku 2005. Pri svojich výpočtoch používa aj MultiParametric toolbox, samozrejme v starej, jednoduchšej verzii.

Ako vstup akceptuje MATISSE diskrétny lineárny hybridný automat, definovaný podobne ako v sekcii venovanej PHAVeru. Je stavaný pre overovanie systémov vyšších dimenzií, ktoré najprv zjednoduší a potom verifikuje.

MATISSE je založený na myšlienke abstrakcie lineárneho systému pomocou aproximovanej bisimulačnej relácie. Oproti klasickej bisimulácii, nevyžadujeme presnú zhodu správania originálneho systému a jeho abstrakcie. Aproximovaná bisimulačná relácia sa snaží zachytiť najdôležitejšie črty správania a prehliadať tie nepodstatné. Výsledkom je opäť lineárny systém. Stupeň aproximácie je určený precíznosťou aproximovanej bisimulácie. Táto určuje

aj hranicu vzdialenosti medzi výpočtami originálneho systému a jeho abstrakcie. Pri overovaní bezpečnosti sa okrem prieniku dosažiteľných a zlých stavov kontroluje aj vzdialenosť týchto množín. V prípade, že je väčšia ako presnosť aproximovanej bisimulácie, prehlásime systém za bezpečný.

Hodnotenie

MATISSE pracuje dobre so systémami viacerých premenných, no dynamika musí byť lineárna a navyše v diskrétnom čase. Algoritmus je však výpočtovo efektívny, vyhodnocuje viac menej iba množinu nerovností matíc a kvadratické výpočty.

2.8 KeYmaera

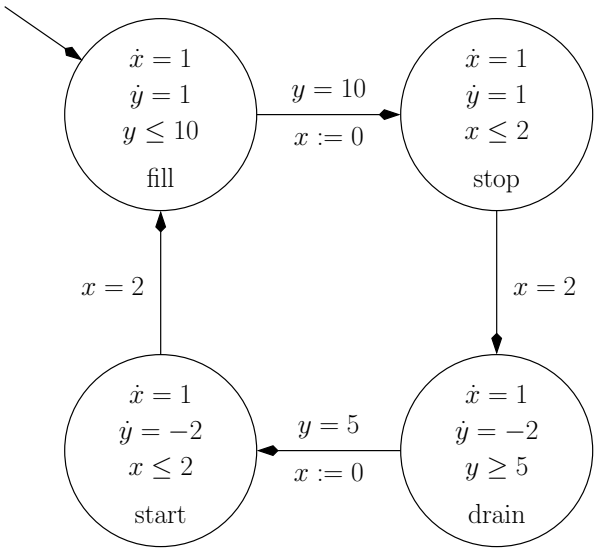
KeYmaera je nástroj pre verifikáciu hybridných systémov, ktorý je, na rozdiel od predchádzajúcich, založený na technikách formálneho dokazovania, v angličtine sa takýto nástroj označuje ako theorem prover. Kombinuje deduktívne, reálne algebraické a počítačové algebraické metódy dokazovania. Ponúka automatizované dokazovanie vlastností charakterizované diferenciálnou dynamickou logikou, užívateľ však môže podľa vlastného uváženia v jednotlivých krokoch nástroj usmerňovať.

Autorom KeYmaery je nemec André Platzer, posledná verzia, KeYmaera 1.5, vyšla v lete minulého roka. Tento nástroj je postavený na systéme KeY (interaktívny theorem prover pre dokazovanie korektnosti Java programov) a nástroji Mathematica. Obsahuje viac ako 206000 riadkov Java kódu, 4800 Java tried a 141 pravidiel pre dokazovanie. KeYmaera rieši dosažiteľnosť, bezpečnosť, životnosť (existuje beh systému, ktorý spĺňa požadovanú vlastnosť?), problém ovládača, parametrov a iné.

2.8.1 Hybridné programy a diferenciálna dynamická logika

Vstupný súbor pre KeYmaeru obsahuje ako popis správania systému, tak aj vlastnosť, ktorú chceme overiť. Je písaný v diferenciálnej dynamickej logike ($d\mathcal{L}$) a používa hybridné programy. Najprv si ukážeme konkrétny príklad, potom ich presnú syntax. Budeme skúmať bezpečnosť nádrže s vodou, ktorá reguluje hladinu vody medzi 1 až 12 pomocou vypúšťania a napúšťania. Na obrázku 2.2 je znázornený príslušný hybridný automat, kde x sú hodiny a premenná y určuje hladinu vody. Príslušný vstupný súbor vyzerá takto:

```
1 \problem{
2   \*state variable declarations*\
3   \[ R x,y,st; x:=0; y:=1; st:=0 \]
4   (
```



Obr. 2.2: Nádrž s vodou

```
5     st=0  \*initial state characterization*\
6     ->
7     \[  \*system dynamics*\
8       (  \*repeat the following discrete or
           continuous transitions*/
9         (?(st=0);
10          (?(y=10); x:=0; st:=1)
11          ++ {x'=1,y'=1,y<=10}
12        )
13        ++ (?(st=1);
14          (?(x=2); st:=2)
15          ++ {x'=1,y'=1,x<=2}
16        )
17        ++ (?(st=2);
18          (?(y=5); x:=0; st:=3)
19          ++ {x'=1,y'=-2,y>=5}
20        )
21        ++ (?(st=3);
22          (?(x=2); st:=0)
23          ++ {x'=1,y'=-2,x<=2}
24        )
25      )*
26    \] (1<=y & y<=12)  \*safety/postcondition*\
27  )
```

```
28 }
```

Verifikačný problém je blok \problem{...}. Tento blok obsahuje jedinú formulu diferenciálnej dynamickej logiky, v tomto prípade je tvaru φ -> \[α\]ψ. To znamená, že pokiaľ je φ iniciálne pravdivá $d\mathcal{L}$ formula (iniciálne stavy), potom po všetkých prevedeniach α (hybridný program popisujúci správanie systému) je $d\mathcal{L}$ formula ψ pravdivá (podmienka bezpečnosti).

Diferenciálna dynamická logika je logika pre špecifikáciu a následnú verifikáciu hybridných systémov. Syntax týchto formulí vyzerá nasledovne:

φ ::=	\forall R x; φ	pre všetky reálne hodnoty x platí formula φ
	\exists R x; φ	existuje reálne x, pre ktoré φ platí
	!φ	negácia
	φ & ψ	konjunkcia
	φ | ψ	disjunkcia
	φ -> ψ	implikácia
	φ <-> ψ	ekvivalencia
	\[α\]φ	po všetkých behoch α platí φ
	\<α\>φ	existuje aspoň jeden beh α, po ktorom φ platí
	pred	predikát z reálnej aritmetiky

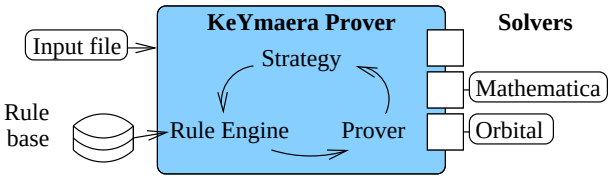
Tu φ a ψ sú $d\mathcal{L}$ formule, α je hybridný program. Syntax hybridného programu je obdobne intuitívna:

α ::=	α; β	sekvenčná kompozícia β po α
	$x := t$	priradenie hodnoty t premennej x
	x :=*	náhodné, nedeterministické priradenie reálnej hodnoty premennej x
	if(F) then α fi	ak F platí, preveď α, inak preskoč
	if(F) then α else β fi	ak F platí, preveď α, inak β
	?F	otestuj či F platí v momentálnom stave, ak nie, preskoč
	while(F) α end	opakuj α pokiaľ F platí
	α*	opakuj α ľubovoľne veľakrát, vrátane 0-krát
	{x'=t,y'=s, F}	spojitý vývoj určený diferenciálnymi rovnicami s invariantom F
	{x'=t,y'<=s,y'>=r, F }	spojitý vývoj určený rovnicami či nerovnicami s invariantom F
	{\exists R u; (x'=t & y'=s & F)}	spojitý vývoj s rušením u (môže sa vyskytovať v t, s, F) s invariantom F
	R x,y,z	deklarácia reálnych premenných

Kde α, β sú hybridné programy, F formula reálnej aritmetiky (nie nutne lineárna, môže obsahovať kvantifikátory), t, s sú výrazy nad reálnymi číslami s operáciami $+$, $-$, $*$, $/$, mocnina. Je zjavné, že vstupné obmedzenia na hybridný systém sú menšie ako u predchádzajúcich nástrojov. KeYmaera je vhodná predovšetkým pre hybridné systémy s parametrami.

2.8.2 Verifikácia deduktívnym dokazovaním

KeYmaera pri svojom procese verifikácie kombinuje automatické dokazovanie formulí, elimináciu univerzálneho kvantifikátoru a symbolické výpočty pomocou systémov pre počítačovú algebru ako Mathematica či Orbital (matematická knižnica vytvorená autorom KeYmaery). Na obrázku 2.3 je znázornená architektúra nástroja, ktorá využíva zásuvné moduly, čo umožňuje rôzne kombinovať stratégie podľa aktuálneho problému.



Obr. 2.3: Architektúra nástroja KeYmaera

Pred spustením verifikácie má užívateľ možnosť vybrať si stratégiu pre dokazovanie, zvoliť nakoľko interaktívne bude verifikácia prebiehať (či v prípade viacerých možností sa chce rozhodnúť sám alebo nechá voľbu na nástroji), nastaviť hľadanie protipríkladu alebo grafický výstup (znázornenie hybridného systému). Ďalej nasleduje samotná verifikácia. KeYmaera rozpisuje a rozkladá $d\mathcal{L}$ formulu, samotný verifikačný problém, podľa preddefinovaných pravidiel $d\mathcal{L}$ kalkulu. Takže vlastnosť sa nedokazuje na celom systéme naraz, ale na jednotlivých častiach podľa sekvenčnej následnosti. Toto dokazovanie je prerušované elimináciou kvantifikátorov či aritmetickými výpočtami s diferenciálnymi rovnicami. Tento problém rieši počítačová algebra, ak nenájde polynomiálne riešenie, pracuje s diferenciálnou rovnicou podľa jej lokálnej dynamiky alebo všetky výskyty premennej nahradí jej riešením v obecnom čase τ. Táto nová premenná je kvantifikovaná univerzálne pre prípad $\backslash[\alpha\backslash]\varphi$ a existenčne pre $\backslash\langle\alpha\backslash\rangle\varphi$, naviac pre každý bod z intervalu $\langle 0, \tau\rangle$ musí byť splnený príslušný invariant. Po skončení procesu dokazovania má užívateľ možnosť prehliadnuť si jednotlivé kroky, ktoré nástroj vykonal.

2.8.3 Hodnotenie

KeYmaera ponúka úplne odlišný prístup oproti iným nástrojom, s verifikačným problémom narába ako s formulou logiky $d\mathcal{L}$, ktorej pravdivosť sa snaží dokázať. Vďaka obecnosti logiky a popisu systému je schopná pracovať takmer s akýmkoľvek hybridným systémom a rozhodovať mnoho druhov vlastností, zvlášť nápomocná je pri hľadaní obmedzení pre parametre. Užívateľ má možnosť sledovať všetky kroky procesu a aktívne sa do nich zapájať. KeYmaera bola úspešne použitá aj na komerčné účely, napríklad bolo pomocou nej dokázaných niekoľko vlastností ETCS (European Train Control System) či systému vzdušných manévrov.

Kapitola 3

Zhrnutie a porovnanie nástrojov

V predchádzajúcej kapitole sme si bližšie predstavili, ako pracujú jednotlivé nástroje, teraz si navzájom porovnáme nielen ich základné črty, ale aj možnosti a silu vstupných jazykov.

V tabuľke 3.1 sú uvedené základné charakteristiky nástrojov. Ako prvý sa uvádza popis hlavnej funkcie v rámci hybridných automatov, teda aký druh verifikácie je možné prevádzať. Potom sú to rozširujúce vlastnosti, je možné vidieť, že nástroje HSolver, Checkmate či MATISSE sú zamerané len na jednoduché overovanie bezpečnosti či dosažiteľnosť, ostatné disponujú ďalšími funkciami, z ktorých najčastejšie sú parametrická analýza alebo syntéza ovládača. Z tohto rozdelenia sa výrazne vymyká MPT, ktorý vznikol ako komplexný toolbox pre Matlab a preto ponúka obrovské množstvo ďalších funkcií, najmä v oblasti výpočtovej geometrie. Polovica nástrojov je písaná jazykom C++, zvyšné sú toolboxy pre Matlab, až na KeYmaeru, ktorá nielenže prináša ako theorem prover úplne iný prístup k problematike, ale je to aj jediný program písaný v Jave. KeYmaera spolu s MPT a HSolverom sú aj jediné projekty, na ktorých vývoji sa stále pracuje. Čo sa týka grafického výstupu, môže byť veľmi nápomocný pri úprave systému, no chýba skoro u polovice nástrojov.

Nároky jednotlivých softwarov a ich dostupnosť sú porovnané v tabuľke 3.2. Všetky toolboxy pre Matlab sú samozrejme multiplatformné, sú závislé iba na Matlabe, podobne ako KeYmaera, ktorá sa viaže na Javu. HyTech a PHAVer, ako C++ programy, boli skompilované pod rôznymi operačnými systémami, d/dt a HSolver sú spustiteľné iba v operačnom systéme Linux. Čo sa týka požiadaviek týchto programov, KeYmaera okrem Javy (verzia 1.5 a novšia) využíva aj nástroj Mathematica, ktorý podobne ako Matlab slúži na prevádzanie rôznych matematických operácií. V Checkmate sa

	Základná charakteristika	**Doplnkové funkcie**	**Povaha**	**Stav projektu**	**Grafický výstup**
HyTech	symbolický model checking lineárnych hybridných automatov	parametrická analýza	C++ program	neaktívny	nie
PHAVer	overovanie bezpečnosti hybridných I/O systémov s afinnou dynamikou	počítanie simulačných relácií, rozhodovanie ekvivalencie	C++ program	neaktívny	áno
Checkmate	modelovanie, simulácia a verifikácia threshold event-driven hybridných systémov		Matlab toolbox	neaktívny	nie
d/dt	analýza dosažiteľnosti pre spojité a hybridné systémy	syntéza ovládača	C++ program	neaktívny	áno
HSolver	overovanie bezpečnosti hybridných systémov		C++ program	aktívny	áno
MPT	dizajn a analýza optimálnych ovládačov pre hybridné systémy, verifikácia	výpočtová geometria	Matlab toolbox	aktívny	áno
MATISSE	dosažiteľnosť a overovanie bezpečnosti pre lineárne hybridné systémy		Matlab toolbox	neaktívny	áno
KeYmaera	theorem prover pre hybridné systémy	syntéza ovládača, parametrická analýza	Java program	aktívny	nie

Tabuľka 3.1: Charakteristika nástrojov I

systémy definujú pomocou Simulink a Stateflow blokov, tieto musia byť teda súčasťou Matlabu. MATISSE zas pre svoje správne fungovanie potrebuje toolbox MPT. Všetky nástroje sú voľne dostupné, PHAVer má naviac zverejnený aj zdrojový kód. Inštalačné súbory môžete nájsť na stránkach jednotlivých projektov, ktorých zoznam je napríklad aj v použitej literatúre, akurát u nástroja d/dt je potrebné napísať email tvorcovi Thaovi Dangovi, ktorý vám pošle odkaz na stránku s týmito súbormi. Na internete sú tiež dostupné rôzne manuály, príručky pre prácu s nástrojmi, či už lepšej alebo horšej kvality, a niekoľko príkladov s ich vstupnými súbormi.

Spomínané nástroje sa líšia predovšetkým vo svojich vstupných formalizmoch, v type a popise vstupného modelu a podmienok overovania či syntézy. V HyTechu sa automat definuje v samostatnom súbore, explicitne sa popisuje každý stav a s ním spojené prechody. Invarianty, guardy aj flow funkcie sú určené konvexnými lineárnymi predikátmi, ktoré sme si zadefinovali v predchádzajúcej kapitole. Flow funkcie však môžu obsahovať okrem konštánt len derivované premenné. PHAVer, ktorý vznikol ako následník HyTechu má vstupné formalizmy takmer identické, až na to, že vo flow funkciách už môžu vystupovať aj nederivované premenné. Ako jediné dva nástroje podporujú kompozície automatov. Checkmate využíva na popis automatu matlabovský Stateflow blok, dynamika systému je v tomto prípade neobmedzená. Nepodporuje ale invarianty, guardy zostávajú lineárne, reset funkcia je popísaná ako afinné zobrazenie. U d/dt sa taktiež popisuje každý stav osobitne, flow funkcie sú popísané ako lineárne rovnice nad nederivovanými premennými. Invarianty aj guardy sú konvexné mnohouholníky. HSolver je o niečo obecnejší, ako derivácie premenných môžu vystupovať obecné rovnice či nerovnice nad premennými aj ich deriváciami. Obecnými rovnicami či nerovnicami charakterizujeme aj guardy, invarianty sú konvexné mnohouholníky, reset funkcia využíva priraďujúce výrazy. MPT definuje automaty pomocou špeciálneho jazyka Hysdel, dynamika jednotlivých premenných je diskrétna, popísaná obecnými rovnicami nad premennými. V invariantoch a guardoch okrem rovníc či nerovníc využíva aj booleovské výrazy, reset funkcia nie je podporovaná. MATISSE pracuje s klasickými matlabovskými m-súbormi, vstupný systém je veľmi jednoduchý, premenné sa iba lineárne vyvíjajú v čase, dokonca diskrétnom. KeYmaera popisuje celý systém ako formulu diferenciálnej dynamickej logiky, všetky súčasti ako invarianty, guardy, flow funkcie či reset funkcie využívajú obecné rovnice, poprípade nerovnice, nad premennými.

	Operačný systém	**Požiadavky**	**Distribúcia**	**Dostupnosť**	**Dostupnosť výukových materiálov**
HyTech	Linux, Digital Unix, DEC Ultrix, HP-UX, Solaris, Sun OS		freeware	na stránkach	dobrá
PHAVer	Linux, Windows XP, Mac OS X		freeware, open source	na stránkach	dobrá, príklady
Checkmate	multiplatformný	Matlab 6.5, Simulink 4.1 a vyššie, Stateflow 4.1 a vyššie, Optimization Toolbox, Control System Toolbox	freeware	na stránkach	dobrá
d/dt	Linux		freeware	na emailovú žiadosť tvorcovi	mauál je iba pre staršiu verziu, príklady
HSolver	Linux		freeware	na stránkach	dobrá, relatívne veľká databáza príkladov
MPT	multiplatformný	Matlab	freeware	na stránkach	dobrá, príklady
MATISSE	multiplatformný	Matlab , MPT, Yalmip, Sedumi	freeware	na stránkach	nedostatočná, iba funkcia help po nainštalovaní nástroja
KeYmaera	multiplatformný	Java 1.5 a vyššie, Mathematica	freeware	na stránkach	dobrá, príklady

Tabuľka 3.2: Charakteristika nástrojov II

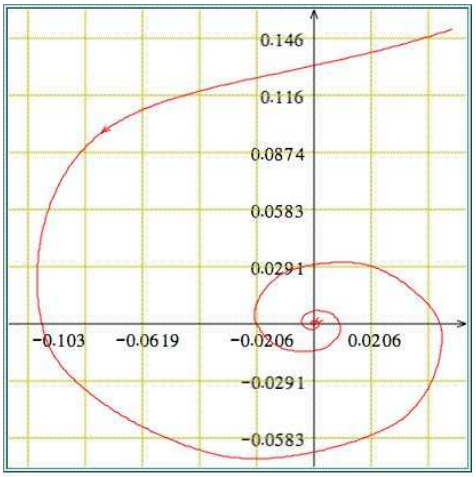
	Definícia automatu	**Invarianty**	**Flow funkcie**	**Guardy**	**Reset funkcia**	**Kompozície**
HyTech	explicitný popis stavov a prechodovej funkcie	konvexné lineárne predikáty	konvexné lineárne predikáty nad deriváciami	konvexné lineárne predikáty	priraďujúce výrazy	áno
PHAVer	explicitný popis stavov a prechodovej funkcie	konvexné lineárne predikáty	konvexné lineárne predikáty nad premennými a ich deriváciami	konvexné lineárne predikáty	priraďujúce výrazy	áno
Checkmate	Stateflow	nepodporované	obecné diferenciálne rovnice	lineárne nerovnosti	afinné zobrazenia	nie
d/dt	explicitný popis stavov a prechodovej funkcie	konvexné mnohouholníky	lineárne rovnice nad premennými	konvexné mnohouholníky	afinné zobrazenia	nie
HSolver	popis stavov a obmedzení pre prechody	konvexné mnohouholníky	konjunkcia obecných rovníc a nerovníc nad premennými a ich deriváciami	obecné rovnice a nerovnice	priraďujúce výrazy	nie
MPT	pomocou jazyka Hysdel, popis premnných a ich spojitého vývoja	konjunkcie rovníc, nerovníc a booleovských výrazov	diskrétne, obecné rovnice nad premennými	konjunkcie rovníc, nerovníc a booleovských výrazov	nepodporovaná	nie
MATISSE	matlabovské m-súbory, definícia matíc lineárneho systému	nepodporované	diskrétne, obecné rovnice diskretizované aj v priestore	nepodporované	nepodporovaná	nie
KeYmaera	$d\mathcal{L}$ formula	obecné rovnice a nerovnice nad premennými	obecné rovnice a nerovnice nad premennými	obecné rovnice a nerovnice nad premennými	obecné rovnice nad premennými	nie

Tabuľka 3.3: Charakteristika vstupného jazyka

3.1 Porovnanie na spoločnom modele

V rámci objektívneho porovnania bolo mojou úlohou spomínané programy nainštalovať a následne spustiť na nejakom spoločnom modele. Pri týchto pokusoch som však narazila na niekoľko závažných problémov. U KeYmaery to bol chýbajúci prístup k nástroju Mathematica, na ktorý nie je na fakulte obstaraná licencia. Matlab je na jednom školskom serveri nainštalovaný, verzia 6.1, no toolbox Checkmate vyžaduje aspoň verziu 6.5. Taktiež sa mi nepodarilo nainštalovať ani toolbox MPT, u ktorého chýbali licencie na určité tzv. solvery, ktoré prevádzajú pod Matlabom rôzne špecifické výpočty. Na MPT sa viaže toolbox MATISSE, ktorý tým pádom nebolo možné spojazdniť. Čo sa týka C++ programov, HyTech a PHAVer už nie sú spustiteľné ani pod rôznymi operačnými systémami. Jedinými nástrojmi, na ktorých uvediem porovnanie na spoločnom modele, teda zostávajú C++ programy d/dt a HSolver.

Porovnanie je prevádzané na dvoch modeloch, obidva majú len lineárnu dynamiku kvôli d/dt. Prvým je skákajúca lopta (bouncing ball) uvedená v prvej kapitole, s obmedzenou výškou od 0 do 100 a rýchlosťou od -49 do 49. Zrýchlenie je nastavené na 9.8 a koeficient pri odraze 0.7, ani jeden z nástrojov totiž nepodporuje parametrickú analýzu. Na začiatku začne lopta padať z ľubovoľnej výšky rýchlosťou 0. U nástroja d/dt je však potrebné množinu iniciálnych stavov resp. guard upraviť (rýchlosť -0.5 až 0.0, resp. výška od 0.0 do 0.5), pretože pôvodne boli tieto moc "ploché" a d/dt prevádza floating-point výpočty nad mnohostenmi. Overované sú tri vlastnosti, a to či rýchlosť dosiahne hodnotu -49, -48 a -30. Prvá ani druhá vlastnosť nie sú splniteľné, čo vyplýva z aritmetických výpočtov, avšak rýchlosť -30 je dosažiteľná. Príslušné vstupné súbory sú nazvané bb1, bb2 a bb3. Druhým modelom je jednoduchá špirála, vyvíjajúca sa v dvoch premenných, tretia určuje jej hĺbku, ktorá sa vyvíja nezávisle od prvých dvoch premenných. Graf v jednom z počiatočných bodov, určený prvými dvoma premennými, spolu s diferenciálnymi rovnicami, je na obrázku 3.1. Model je bez prechodov, hodnoty všetkých premenných sú obmedzené hodnotami -1 a 1. Iniciálna množina je $0.025 \leq x \leq 0.05$, $0.1 \leq y \leq 0.15$, $0.05 \leq z \leq 0.1$. Prvá overovaná vlastnosť je či sa hodnoty všetkých premenných môžu dostať do intervalu $(-1.0, -0.9)$. Z grafu je jasné, že špirála sa vyvíja smerom dovnútra, a teda vzhľadom k iniciálnej množine je takéto ohodnotenie premenných nemožné. Druhou overujeme, či hodnoty premenných môžu spĺňať nerovnosti $0.025 \leq x \leq 0.05$, $0.1 \leq y \leq 0.15$, $-1.0 \leq z \leq 1.0$. Túto zrejme spĺňajú už aj iniciálne stavy, preto vlastnosť platí. Príslušné vstupné súbory majú názvy spiral1, spiral2. Overenie platnosti týchto vlastností je prevádzané ako overovanie bezpečnosti, množina zlých stavov je určená vyššie uvedenými obmedzeniami. Samotné testovanie prebiehalo na stroji s procesorom Intel(R) Pentium(R) 4 CPU 2.80GHz (1MB L2 cache) a RAM 4x



x` = -x -4*y
y` = 4*x -y

Obr. 3.1: Graf jednej zo špirál určených druhým modelom

512 MB DDR 266MHz. Všetky vstupné aj výstupné súbory sa nachádzajú v prílohe.

Nástroj d/dt má vlastné jednoduché grafické rozhranie. Pomocou neho sa načítavajú vstupné súbory, prvý, s koncovkou .hyb popisujúci model a druhý, s koncovkou .par, charakterizujúci parametre výpočtu ako krok v čase (time step), tolerancia pre chybu ale taktiež vlastnosti grafického výstupu. Následne sa z ponuky vyberá medzi analýzou dosažiteľnosti a verifikáciou, pri verifikácii musí súbor .hyb obsahovať aj popis množiny nežiaducich stavov. Každý výpočet je sprevádzaný vykreslením množiny dosažiteľných stavov podľa udaných parametrov.

HSolver je program spustiteľný iba z príkazového riadku, súbor sa načítava pomocou rúry. Vstupný súbor je v tomto prípade iba jeden, popisujúci model spolu s množinou zlých stavov. Samostatná analýza množiny dosažiteľných stavov nie je možná. Podobne ako v d/dt existuje možnosť grafického výstupu, no pri žiadnom z uvedených vstupných súborov tento nefungoval. Výpočet sa ukončí v prípade, že je systém bezpečný alebo počet abstraktných stavov dosiahne hornú hranicu, ktorá je iniciálne nastavená na počet 1000.

		Dĺžka vstupných súborov	**Čas výpočtu**	**Dôvod ukončenia**	**Výsledok**
d/dt	bb1	468, 296	12:11	chyba (index out of range)	
	bb2	468, 296	10:42	chyba (index out of range)	
	bb3	468, 296	4:43		protipríklad
	spiral1	353, 374	0:10		systém bezpečný
	spiral2	349, 374	0:01		protipríklad
HSolver	bb1	167	6:34		systém bezpečný
	bb2	167	5:46	dosiahnutá horná hranica (1000) pre počet stavov	protipríklad
	bb3	167	5:45	dosiahnutá horná hranica (1000) pre počet stavov	protipríklad
	spiral1	265	19:59	dosiahnutá horná hranica (1000) pre počet stavov	protipríklad
	spiral2	261	16:38	dosiahnutá horná hranica (1000) pre počet stavov	protipríklad

Tabuľka 3.4: Porovnanie na spoločnom modele

Výsledky porovnania sú zhrnuté v tabuľke 3.4, porovnávanými aspektami sú dĺžka vstupných súborov a výpočtu, dôvod ukončenia výpočtu (v prípade, že bol ukončený neštandardne) a samozrejme výsledok. Vstupný jazyk pre HSolver je značne jednoduchší oproti tomu použitému v d/dt, čo sa odráža aj na dĺžke vstupných súborov. Nástroj d/dt zjavne horšie pracuje s komplexnejším Zeno systémom, no na špirále naopak funguje veľmi efektívne a rýchlo. HSolver jemné delenie na špirále nezvládol, výpočet vždy skončil až po dosiahnutí hornej hranice pre počet stavov, a teda protipríkladom, aj v prípade, keď už iniciálne stavy patrili medzi nežiaduce. Na skákajúcej lopte to však dopadlo lepšie ako u d/dt, vďaka použitému algoritmu s postupným delením boxov, aj keď druhú vlastnosť opäť vyhodnotil ako neplatnú.

Kapitola 4

Návrh riešenia

Po predstavení a porovnaní existujúcich nástrojov pre verifikáciu hybridných systémov je zjavné, že ani jeden z nich nepokrýva všetky požiadavky potenciálneho užívateľa, ktorý používa tieto systémy ako model svojho reálneho systému. Vyvstáva teda otázka, ako by mal vyzerať ideálny nástroj tohto druhu a či je možné v skutočnosti dosiahnuť tohto ideálu.

Ideálny verifikačný nástroj by mal byť multiplatformný, nemal by mať veľké softwarové nároky, a keď už by prevádzal matematické operácie v nejakom osobitnom nástroji, tento by mal byť voľne dostupný. Mal by byť voľne stiahnuteľný z internetu, kde by mala byť taktiež bohatá databáza súvisiacich článkov a manuálov, a ukážkových vstupných súborov. Prostredie by malo byť jednoduché, prehľadné, s takým ovládaním, aby bolo ľahko pochopiteľné pre nového užívateľa, ideálne nie cez príkazový riadok. Čo sa týka funkcionality, je rozumné vyžadovať ako samostatnú analýzu dosažiteľnosti, tak aj overovanie bezpečnosti, syntézu ovládača a parametrickú analýzu. Ako vhodné doplňujúce vlastnosti sa javia simulácia alebo určovanie ekvivalencie systémov. Pri každom výpočte by mala byť možnosť grafického výstupu s nastaviteľnými vlastnosťami. Vstupný jazyk, pomocou ktorého sa charakterizuje model, by mal byť jednoduchý a prehľadný, no zároveň dostatočne silný. Flow funkcie by mali byť obecné diferenciálne rovnice, poprípade nerovnice, nad premennými aj ich deriváciami. Takisto invarianty a guardy by mali byť obecné rovnice či nerovnice, reset funkcie priraďujúce výrazy. Automaty by sa mali dať skladať pomocou kompozície. Prevádzané výpočty by mali byť rýchle, čo najjemnejšie aproximované, pričom túto hranicu by si mal byť užívateľ schopný nastaviť.

Prečo teda neexistuje nástroj, ktorý by bol silný, rýchly a obecný? Odpoveď sa skrýva predovšetkým v matematických operáciách, ktorým sa pri verifikácii tohto druhu nevyhneme. Pri určovaní množiny dosažiteľných stavov je to hlavne riešenie diferenciálnych rovníc a nerovníc a s tým spojená

integrácia. Problém integrovania je ale obecne neriešiteľný. Ponúkajú sa nám dve možnosti ako sa týmto problémom vyhnúť, a to aproximácia spojitého vývoja premenných nejakými jednoduchšími (napríklad lineárnymi či afinnými) rovnicami alebo zameranie sa len na určitú podtriedu hybridných automatov, u ktorých je integrácia prevádzaná jednoduchým automatizovateľným spôsobom. Pri vyhodnocovaní guardov riešime rovnice relatívne zložitého typu, hodnotu premennej tam totiž nahrádzame jej hodnotou v čase, určenou z dynamiky tejto premennej. Toto je tiež problém, ktorý obecne nemusíme vedieť vyriešiť. Situácia sa ale zjednodušuje obmedzením dynamiky, ako bolo uvedené vyššie, no musíme tiež zjednodušiť tvar guardov, napríklad vylúčiť výskyt premenných v niektorých funkciách alebo vzájomné násobenie premenných (aby nám po dosadení nevznikali polynómy vysokých stupňov). Stále musíme dbať aj na invarianty jednotlivých stavov, tie však riešime podobne ako guardy. Tieto matematické operácie si samozrejme vyžadujú určité množstvo času, ktoré je závislé aj od zložitosti samotného systému, napríklad koľko diskrétnych stavov obsahuje, koľko rôznych prechodov z nich vieme urobiť alebo či má Zeno chovanie. Výpočet značne komplikuje aj vyšší počet parametrov. To, že výpočet prebehne v krátkom čase teda zaručiť tiež nevieme. Ostatné uvedené požiadavky na ideálny verifikačný nástroj sú v podstate splniteľné, závisia od zvoleného spôsobu implementácie či syntaxe vstupného jazyka.

Pri návrhu algoritmu je teda potrebné nájsť prijateľný pomer medzi hĺbkou analýzy a komplexnosťou systému, pričom sa riadime tým, na čo presne by sme chceli nástroj využívať. Ak sa chceme zamerať na overovanie čo najväčšieho počtu vlastností, je vhodné pracovať napríklad len s takzvanými timed automatmi. Premenné tejto podtriedy hybridných automatov sú všetky rovnakého typu a to hodiny, ich derivácia je v každom stave rovna 1. Pre ne existuje celá rada efektívnych algoritmov, všetky spomínané ťažkosti s výpočtami sú redukované na minimum a používa sa presná aritmetika. Naopak, ak chceme byť schopní pracovať so systémami s obecnou dynamikou, je potrebná aproximácia a tak sa aj analýza dosažiteľnosti stáva postačujúcim cieľom.

Zaujímavou otázkou, ktorou sa nezaoberal ani jeden z nástrojov, je automatická úprava modelu po vygenerovaní protipríkladu. V súčastnosti už existuje niekoľko pokusov o riešenie tohto problému, no žiaden nie je úplne automatizovaný. Protipríklad totiž popisuje iba jednu cestu, jednu možnosť nevyhovujúceho správania, preto odstránenie celého problému vyžaduje iteratívne úpravy a interakciu s užívateľom. Spôsob úpravy totiž nie je protipríkladom jednoznačne určený a on jediný tiež pozná všetky požiadavky na výsledný systém, ktoré nemusíme byť schopní zakomponovať do algoritmu.

Na záver je potrebné povedať, že analýza, a prípadne syntéza, síce majú kľúčovú úlohu pri odstraňovaní chýb systému, no takéto nástroje sú závislé od úrovne abstrakcie reálneho systému. Jej voľba nie je nikdy jednoznačne daná a teda je takmer nemožné vytvoriť správny a dostatočne silný model na prvýkrát. Preto je rovnako nevyhnutné vyvíjať aj interaktívne prostredia, kde by napríklad simulácia napomáhala vhodnému výberu abstrakcie a dekompozície.

Záver

Práca spracováva tému hybridných automatov a ich verifikácie. Na úvod definuje túto štruktúru a jej chovanie, spolu s niekoľkými príkladmi, ktoré napomáhajú čitateľovi pochopiť hlavnú ideu. Nasledujúci prehľad nástrojov pre verifikáciu popisuje základné charakteristiky 8 dostupných programov, predovšetkým čo sa týka vstupných formalizmov a použitého algoritmu. Prehľadné porovnanie všetkých relevantných aspektov týkajúcich sa či už softwarových požiadaviek alebo obmedzení pre vstupné modely sa nachádza v tretej kapitole. Taktiež obsahuje zrovnanie na spoločných modeloch, avšak nie všetky nástroje sú stále fungujúce resp. nie všetky som bola schopná spustiť za poskytnutých podmienok. Na záver je uvedený rozbor problému návrhu podobného nástroja, ktorý so sebou prináša množstvo komplikácií.

Hybridné systémy modelujú obrovské množstvo či už automatizovaných alebo prírodných javov nášho každodenného života. Pre správne fungovanie systémov je nevyhnutná ich verifikácia, no výber príslušného nástroja nie je vždy jednoduchý. Práca slúži ako prehľadný sprievodca pre nových užívateľov v oblasti hybridných systémov, jednak v teoretickej rovine, no predovšetkým môže dobre poslúžiť pri výbere vhodného nástroja na overenie konkrétnej vlastnosti. Pre prípadného vývojára, ktorý má záujem o ďalší rozvoj v tejto oblasti, ponúka náčrt potrebných krokov a úskalí, s ktorými treba pri tejto problematike rátať.

Čo sa týka pokračovania tejto práce, vidím viacero zaujímavých možností. Samozrejme je to návrh nového verifikačného nástroja, čo však vyžaduje relatívne hlboké znalosti v oblasti matematiky. Ako bolo načrtnuté v poslednej kapitole, v súčasnosti sú pokusy o automatizáciu úpravy modelu na základe vygenerovaného protipríkladu, táto oblasť ešte nie je dostatočne preskúmaná. Perspektívnou sa javí aj možnosť návrhu interaktívneho prostredia, ktoré by napomáhalo vhodnému výberu abstrakcie systému, tu by bolo vhodné neobmedziť sa len na hybridné automaty, ale uvažovať obecnejšie systémy.

Prílohy

Vstupné súbory pre d/dt

bb1.hyb

```
dimension: 2;

constants:
 g = -9.8,
 c = -0.7;

initloc: 0;
initset:
type rectangle
 0.0    100.0,
 -0.5   0.0;

badset:
type rectangle
 0.0   100.0,
 -49.0  -48.5;

location: 0;
matrixA:
 0.0  1.0,
 0.0  0.0;
scalB: 1.0;
inputset:
type convex_vert
 0.0  [g];
stayset:
type rectangle
 0.0    100.0,
 -49.0  49.0;

```

```
31 transition:
32 label go00:
33 if in guard:
34 type rectangle
35  0.0    0.5,
36  -49.0  0.0;
37 reset:
38 matrixP:
39  1.0  0.0,
40  0.0  [c];
41 goto 0;
42 ;
43
44 limits:
45 x[0] <= 100.0 and
46 x[0] >= 0.0 and
47 x[1] <= 49.0 and
48 x[1] >= -49.0
```

bb2.hyb

```
 1 dimension: 2;
 2
 3 constants:
 4  g = -9.8,
 5  c = -0.7;
 6
 7 initloc: 0;
 8 initset:
 9 type rectangle
10  0.0    100.0,
11  -0.5   0.0;
12
13 badset:
14 type rectangle
15  0.0   100.0,
16  -48.0  -47.5;
17
18 location: 0;
19 matrixA:
20  0.0  1.0,
21  0.0  0.0;
22 scalB: 1.0;
23 inputset:
```

```
24 type convex_vert
25  0.0  [g];
26 stayset:
27 type rectangle
28  0.0    100.0,
29  -49.0  49.0;
30
31 transition:
32 label go00:
33 if in guard:
34 type rectangle
35  0.0    0.5,
36  -49.0  0.0;
37 reset:
38 matrixP:
39  1.0  0.0,
40  0.0  [c];
41 goto 0;
42 ;
43
44 limits:
45 x[0] <= 100.0 and
46 x[0] >= 0.0 and
47 x[1] <= 49.0 and
48 x[1] >= -49.0
```

bb3.hyb

```
 1 dimension: 2;
 2
 3 constants:
 4  g = -9.8,
 5  c = -0.7;
 6
 7 initloc: 0;
 8 initset:
 9 type rectangle
10  0.0    100.0,
11  -0.5   0.0;
12
13 badset:
14 type rectangle
15  0.0   100.0,
16  -30.0  -29.5;
```

```
17
18 location: 0;
19 matrixA:
20  0.0  1.0,
21  0.0  0.0;
22 scalB: 1.0;
23 inputset:
24 type convex_vert
25  0.0  [g];
26 stayset:
27 type rectangle
28  0.0    100.0,
29  -49.0  49.0;
30
31 transition:
32 label go00:
33 if in guard:
34 type rectangle
35  0.0    0.5,
36  -49.0  0.0;
37 reset:
38 matrixP:
39  1.0  0.0,
40  0.0  [c];
41 goto 0;
42 ;
43
44 limits:
45 x[0] <= 100.0 and
46 x[0] >= 0.0 and
47 x[1] <= 49.0 and
48 x[1] >= -49.0
```

bb1.par, bb2.par, bb3.par

```
1 dimension: 2;
2
3 location: 0,
4   time_step 0.2
5   abs_tol 0.5  0.5
6   rel_tol 0.1
7   hull 0
8   tmax 10.0
9 ;
```

```
10
11 mesh_size 0.1 0.1
12 ;
13
14 dblmin 1E-20
15 polylib_priority qhull
16 cdd_zero 1E-12
17 ;
18
19 file_out ddt_format
20 display ddt_viewer
21 verbose s
22 angle 60
23 colour 5
24 zoom 1 1 0
25 projection 1 1 0
26 viewing_mode 2
27 refresh 0
28 width 600
29 height 600
```

spiral1.hyb

```
1 dimension:3;
2
3 initloc: 0;
4 initset:
5 type rectangle
6   0.025 0.05,
7   0.1   0.15,
8   0.05  0.1;
9
10 badset:
11 type rectangle
12   -1.0    -0.90,
13   -1.0    -0.90,
14   -1.0    -0.90;
15
16 location: 0;
17 matrixA:
18  -1.0  -4.0   0.0,
19   4.0  -1.0   0.0,
20   0.0   0.0   0.5;
21 scalB: 0.0;
```

```
22 transition:
23 ;
24
25 limits:
26 x[0] <= 1.0 and
27 x[0] >= -1.0 and
28 x[1] <= 1.0 and
29 x[1] >= -1.0 and
30 x[2] <= 1.0 and
31 x[2] >= -1.0
```

spiral2.hyb

```
 1 dimension:3;
 2
 3 initloc: 0;
 4 initset:
 5 type rectangle
 6   0.025 0.05,
 7   0.1   0.15,
 8   0.05  0.1;
 9
10 badset:
11 type rectangle
12   0.025    0.05,
13   0.1    0.15,
14   -1.0    1.0;
15
16 location: 0;
17 matrixA:
18  -1.0  -4.0   0.0,
19   4.0  -1.0   0.0,
20   0.0   0.0   0.5;
21 scalB: 0.0;
22 transition:
23 ;
24
25 limits:
26 x[0] <= 1.0 and
27 x[0] >= -1.0 and
28 x[1] <= 1.0 and
29 x[1] >= -1.0 and
30 x[2] <= 1.0 and
31 x[2] >= -1.0
```

spiral1.par, spiral2.par

```
dimension: 3;

location: 0,
 time_step 0.2
 abs_tol 12 12 12
 rel_tol 10
 tmax 2.5
 method enu
;

mesh_size
 0.01 0.005 0.005
;

dblmin 1E-16
polylib_priority qhull
cdd_zero 1E-12
;

display ddt_viewer
verbose i
file_out ddt_format
iterreach 1

angle 60
zoom 1 1 1
projection 0 2 1
rotation 0 210 0
viewing_mode 3
refresh 0
zmin -0.05
zmax 0.8
xmin -0.25
xmax 0.25
ymin -0.15
ymax 0.25
width 600
height 600
```

Vstupné súbory pre HSolver

bb1.hs

```
1 VARIABLES [x1,x2]
2 MODES [m1]
3 STATESPACE
4   m1[[0,100],[-49,49]]
5 INITIAL
6   m1{x1>=0/\x2=0}
7 FLOW
8   m1{x1_d=x2}{x2_d=-9.8}
9 JUMP
10  m1->m1{x1=0/\x2<=0/\x2'=-0.7*x2}
11 UNSAFE
12  m1{x2=-49}
```

bb2.hs

```
1 VARIABLES [x1,x2]
2 MODES [m1]
3 STATESPACE
4   m1[[0,100],[-49,49]]
5 INITIAL
6   m1{x1>=0/\x2=0}
7 FLOW
8   m1{x1_d=x2}{x2_d=-9.8}
9 JUMP
10  m1->m1{x1=0/\x2<=0/\x2'=-0.7*x2}
11 UNSAFE
12  m1{x2=-48}
```

bb3.hs

```
1 VARIABLES [x1,x2]
2 MODES [m1]
3 STATESPACE
4   m1[[0,100],[-49,49]]
5 INITIAL
6   m1{x1>=0/\x2=0}
7 FLOW
8   m1{x1_d=x2}{x2_d=-9.8}
9 JUMP
10  m1->m1{x1=0/\x2<=0/\x2'=-0.7*x2}
```

```
11 UNSAFE
12   m1{x2=-30}
```

spiral1.hs

```
 1 VARIABLES [x1,x2,x3]
 2 MODES [m1]
 3 STATESPACE
 4   m1[[-1,1],[-1,1],[-1,1]]
 5 INITIAL
 6   m1{x1>=0.025/\x1<=0.05/\x2>=0.1/\x2
       <=0.15/\x3>=0.05/\x3<=0.1}
 7 FLOW
 8   m1{x1_d=-x1-4*x2}{x2_d=4*x1-x2}{x3_d=0.5*
       x3}
 9 JUMP
10 UNSAFE
11   m1{x1>=-1.0/\x1<=-0.90/\x2>=-1.0/\x2
       <=-0.90/\x3>=-1.0/\x3<=-0.90}
```

spiral2.hs

```
 1 VARIABLES [x1,x2,x3]
 2 MODES [m1]
 3 STATESPACE
 4   m1[[-1,1],[-1,1],[-1,1]]
 5 INITIAL
 6   m1{x1>=0.025/\x1<=0.05/\x2>=0.1/\x2
       <=0.15/\x3>=0.05/\x3<=0.1}
 7 FLOW
 8   m1{x1_d=-x1-4*x2}{x2_d=4*x1-x2}{x3_d=0.5*
       x3}
 9 JUMP
10 UNSAFE
11   m1{x1>=0.025/\x1<=0.05/\x2>=0.1/\x2
       <=0.15/\x3>=-1.0/\x3<=1.0}
```

Literatúra

[1] Tomlin, Claire J.: Lectures notes from *AA278A: Hybrid Systems: Modeling, Analysis, and Control.* Stanford, California USA, 2005.

[2] Carloni, Luca P., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, Alberto L.: *Languages and Tools for Hybrid Systems Design.* Massachusetts USA, now Publishers Inc., 2006.

[3] *HyTech: The HYbrid TECHnology Tool.* http://embedded.eecs.berkeley.edu/research/hytech/.

[4] Henzinger, Thomas A., Pei-Hsin, H., Wong-Toi, H.: *HyTech: A Model Checker for Hybrid Systems.* In Software Tools for Technology Transfer, 1, 1997, s. 110-122.

[5] *PHAVer – Polyhedral Hybrid Automaton Verifier.* http://www-verimag.imag.fr/~frehse/phaver_web/index.html.

[6] *CheckMate – Hybrid System Verification Toolbox for MATLAB.* http://www.ece.cmu.edu/~webk/checkmate/.

[7] *d/dt – Reachability Analysis of Continuous and Hybrid Systems.* http://www-verimag.imag.fr/~tdang/Tool-ddt/ddt.html.

[8] *HSolver.* http://hsolver.sourceforge.net/.

[9] Ratschan, S., Dzetkulič, T., She, Z.: *HSolver User Manual.* Prague, Czech republic, 2009.

[10] *MPT – The Multi-Parametric Toolbox for Matlab.* http://control.ee.ethz.ch/~mpt/.

[11] Kvasnica, M., Grieder, P., Baotić, M., Christophersen, F. J.: *Multi-Parametric Toolbox.* Zürich, Switzerland, 2006.

[12] *MATISSE.* http://www.seas.upenn.edu/~agirard/Software/MATISSE/.

[13] Girard, A., Pappas, George J.: *Approximate bisimulation relations for constrained linear systems.* Philadelphia, Pennsylvania USA, 2005.

[14] *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems.* http://symbolaris.com/info/KeYmaera.html.

[15] Platzer, A., Quesel, J.: *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems.* In Automated Reasoning, Third International Joint Conference, IJCAR 2008, Sydney, Australia, Proceedings, 2008, s. 171-178.

[16] Platzer, A.: *Combining Deduction and Algebraic Constraints for Hybrid System Analysis.* In 4th International Verification Workshop, VERIFY'07, Workshop at Conference on Automated Deduction (CADE), Bremen, Germany, CEUR Workshop Proceedings, 2007, s. 164-178.