MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Rozšírenie a refaktorizácia nástroja BioDiVinE

DIPLOMOVÁ PRÁCA

Martin Demko

Brno, jar 2014

Vyhlásenie

Vyhlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: RNDr. David Šafránek, Ph.D.

Poďakovanie

Rád by som poďakoval RNDr. Davidovi Šafránkovi, Ph.D. za jeho čas a trpezlivosť pri osvetľovaní problematiky a za jeho cenné rady a podnety. Zároveň by som chcel poďakovať aj rodine a svojej priateľke za ich morálnu podporu a pomoc s nejednou jazykovou dilemou.

Zhrnutie

Práca sa venuje rozšíreniu nástroja BioDiVinE 1.0 o nový obecnejší popis vstupného modelu podporujúceho aj nelineárne funkcie prevažne sigmoidálneho charakteru. Zároveň s tým musí nová verzia tohto programu poskytovať aj prechod od nelineárneho vstupného modelu k multi-afinnému, aby sa mohol nový nástroj využívať k pôvodnému účelu, a síce k overovaniu vlastností modelov.

Keďže sa v skutočnosti jedná o sadu nástrojov a nie o ucelený program, dali sme si za úlohu zmeniť túto skutočnosť a vytvoriť pre užívateľa jednoduchú, ale použiteľnú konzolovú aplikáciu, ktorá bude zahŕňať pôvodnú funkcionalitu.

Kľúčové slová

ordinárne diferenciálne rovnice, po častiach multi-afinný ODE model, nelineárne regulačné funkcie, overovanie modelov, LTL, Michaelis-Mentenovej kinetika, Hillova kinetika

Obsah

Úvod

Biologické systémy sú obvykle zložité komplexné siete obrovských rozmerov. Práca bioinformatikov a systémových biológov však dokazuje, že sa tieto siete skladajú z veľkého počtu drobnejších častí, tzv. motívov. Tie sú oveľa jednoduchšie na pozorovanie, aj keď sa často ukazuje, že môžu aj pri svojej jednoduchosti vykazovať zaujímavé emergentné vlastnosti.

Pre praktické skúmanie dynamiky takýchto motívov alebo stredne veľkých systémov je vhodnou technikou overovanie modelov (viď kapitolu 1.5). Existuje množstvo nástrojov a aplikácií slúžiacich k tomuto účelu (viď kapitolu 3). Jedným z nich je aj nástroj BioDiVinE 1.0 vyvinutý v Laboratóriu systémovej biológie na Fakulte informatiky Masarykovej univerzity v Brne. Bohužiaľ je už mierne zastaraný a zároveň nie je vhodný pre veľkú škálu modelov.

V rámci neustáleho technického pokroku vznikol návrh tieto chyby odstrániť a nový nástroj otestovať a poskytnúť širšej verejnosti. O teoretických podkladoch, postupe práce, samotnej implementácii a praktických príkladoch hovoria nasledujúce kapitoly. Hotový nástroj bude voľne k dispozícii na `https://github.com/martindemko/BioDiVinE-1.1`.

Kapitola 1

Základné pojmy

Než začneme zachádzať hlbšie do problematiky tejto práce a opisovať postupy a nástroje v nej použité, je potrebné vysvetliť na počiatku niekoľko pojmov. Tieto sa v práci mnohokrát opakujú a ich včasným uvedením predídeme nepochopiteľnosti textu. Je tiež dôležité poznamenať, že táto kapitola je z veľkej miery doslovne citovaná zo zdrojov uvedených vždy na konci každej podkapitoly.

1.1 LTL - Lineárna Temporálna Logika

Temporálna logika obecne je špeciálna vetva logiky zaoberajúca sa logickou štruktúrou výrokov v čase. Je to formalizmus vhodný pre overovanie vlastností formálnych dynamických systémov resp. matematických modelov.

Lineárna temporálna logika (ďalej len LTL) je najjednoduchšia verzia temporálnej logiky, ktorá neumožňuje vetvenie času ani kvantifikátory. Môžeme ju považovať tiež za konkrétny výpočtový kalkulus pracujúci s tzv. formulami, definovanými nasledujúcou syntaxou:

Atomické propozície (ďalej len AP)

$$A > 0$$
$$B \leq 5.834$$
$$C \neq \text{"}nie\text{"}$$
$$\textit{atď}\ldots$$

Logické operátory

$\neg,\ \vee$	– *základné logické operátory*
$\wedge,\ \rightarrow,\ \leftrightarrow,\ \mathbf{true},\ \mathbf{false}$	– *odvodené logické operátory*

Temporálne operátory

- $\mathbf{X}\phi$ - ne**X**t, *vyjadruje platnosť* ϕ *v ďalšom stave*

- $\mathbf{G}\phi$ - **G**lobal, *vyjadruje trvalú platnosť* ϕ
- $\mathbf{F}\phi$ - **F**uture, *vyjadruje platnosť* ϕ *v niektorom z budúcich stavov*
- $\psi\mathbf{U}\phi$ - **U**ntil, *vyjadruje platnosť* ψ*, až kým nezačne platiť* ϕ
- $\psi\mathbf{R}\phi$ - **R**elease, *vyjadruje platnosť* ϕ*, až kým nezačne platiť* ψ*, a to vrátane tohto bodu. Ak* ψ *nikdy nezačne platiť, musí* ϕ *platiť do nekonečna,*

  kde ϕ *a* ψ *sú atomické propozície.*

Potom platí nasledujúce:

- Ak $p \in AP$, tak p je formula.
- Ak ϕ a ψ sú formuly, tak $\neg\psi$, $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$, $\mathbf{X}\psi$, $\mathbf{F}\psi$, $\mathbf{G}\psi$, $\phi\mathbf{U}\psi$ a $\psi\mathbf{R}\phi$ sú formuly.

Takto vytvorená LTL formula môže byť splniteľná nekonečnou postupnosťou pravdivých vyhodnotení jednotlivých $p \in AP$. Túto postupnosť si možno predstaviť ako nekonečné slovo w, pre ktoré platí $w = a_0, a_1, a_2, \ldots$ a kde a_i je pravdivostná hodnota nejakej $p \in AP$. Nech $w(i) = a_i$ a $w^i = a_i, a_{i+1}, \ldots$ je podpostupnosť alebo sufix slova w. Potom formálna definícia relácie splniteľnosti $\vDash$ medzi slovom w a LTL formulou vyzerá:

- $w \vDash p$, ak $p \in AP \bigwedge p = w(0)$
- $w \vDash \neg\phi$, ak ϕ a ψ sú LTL formuly $\bigwedge w \nvDash \phi$
- $w \vDash \phi \vee \psi$, ak $w \vDash \phi \bigvee w \vDash \psi$
- $w \vDash \mathbf{X}\phi$, ak $w^1 \vDash \phi$
- $w \vDash \phi\mathbf{U}\psi$, ak $\exists i,\ i \geq 0 \wedge w^i \vDash \psi \bigwedge \forall k,\ 0 \leq k < i \wedge w^k \vDash \phi$

Predchádzajúce platí pre základné logické a temporálne operátory, ktoré majú ale dostatočne expresívnu silu, aby s ich pomocou mohli byť zadefinované ľubovolné LTL formuly. Ovšem pre uľahčenie zápisu aj čítania si môžeme dodefinovať rozšírenú paletu operátorov za predpokladu platnosti predchádzajúcich pravidiel:

- $\phi \wedge \psi \ \equiv\ \neg(\neg\phi \vee \neg\psi)$, ak ϕ, ψ sú LTL formuly
- $\phi \rightarrow \psi \ \equiv\ \neg\psi \vee \psi$,
- $\phi \leftrightarrow \psi \ \equiv\ (\phi \rightarrow \psi) \vee (\psi \rightarrow \phi)$,

- $\mathbf{true} \equiv p \vee \neg p$, ak $p \in AP$
- $\mathbf{false} \equiv \neg\mathbf{true}$,
- $\phi\mathbf{R}\psi \equiv \neg(\neg\phi\mathbf{U}\neg\psi)$,
- $\mathbf{F}\phi \equiv \mathbf{true}\mathbf{U}\phi$,
- $\mathbf{G}\phi \equiv \neg\mathbf{F}\neg\phi$,

Napriek tomu, že je LTL tou najprimitívnejšou temporálnou logikou, jej prevod do Büchiho automatu je v najhoršom prípade exponenciálne zložitý. Dôvod tohto prevodu bude vysvetlený v kapitole 1.5. [14]

1.2 Büchiho automat

Automat obecne je matematický model stroja s konečným množstvom pamäte spracovávajúci vstup o neznámej veľkosti. Kvôli obmedzeniu pamäte ho nazývame konečným automatom. Vstup sa nazýva slovo a môže byť konečný aj nekonečný. Büchiho automat je potom najjednoduchším konečným automatom nad nekonečným slovom, a preto patrí do skupiny ω-automatov.

Formálne je konečný automat $\mathcal{A}$ pätica $(\Sigma, Q, \Delta, Q_0, F)$, pre ktorú platí:

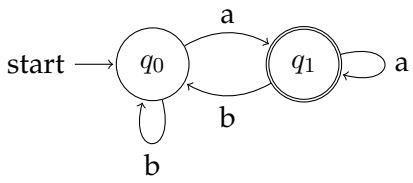
- Σ je konečná abeceda
- Q je konečná množina stavov
- $\Delta \subseteq Q \times \Sigma \times Q$ je relácia, nazývaná prechodová funkcia
- $Q_0 \subseteq Q$ je podmnožina množiny stavov, nazývaná iniciálne stavy
- $F \subseteq Q$ je podmnožina množiny stavov, nazývaná akceptujúce stavy.

Príklad jednoduchého automatu je daný na Obr. 1.1.

Automat nad konečným slovom akceptuje toto slovo, ak po prejdení posledného znaku slova zodpovedajúceho prechodu medzi dvoma stavmi, je tento posledný stav v množine F. Avšak automat nad nekonečným slovom nemôže nikdy prejsť cez posledný znak. Preto takýto automat akceptuje nekonečné slovo len v prípade, že počas prechádzania slova je aspoň jeden stav navštívený nekonečne často a zároveň tento stav patrí aj do množiny F.

Automaty môžeme ešte rozlíšiť na deterministické a nedeterministické. Deterministický automat má jednoznačne určené prechody medzi stavmi.

Tým sa myslí, že zo stavu $q \in Q$ sa pod znakom $s \in \Sigma$ dá prejsť maximálne do jednoho stavu $q^{'} \in Q$. Naproti tomu nedeterministické automaty umožňujú prechod zo stavu q pod slovom s do stavov $Q^{'} \subseteq Q$. Našťastie existuje algoritmus prevodu nedeterministického konečného automatu na deterministický, ale iba nad konečným slovom. Nevýhodou je ale veľký nárast počtu stavov. [15]



Obr. 1.1: Jednoduchý deterministický automat $\mathcal{A} : \Sigma = \{a, b\}$, $Q = \{q_0, q_1\}$, $\Delta = \{(q_0, a, q_1), (q_0, b, q_0), (q_1, a, q_1), (q_1, b, q_0)\}$, $Q_0 = \{q_0\}$, $F = \{q_1\}$

1.3 Zákon o mass action kinetike

Tento zákon vyjadruje základné pravidlo fungovania chemických reakcií. Konkrétne rýchlosť, s akou chemické substancie, či už veľké makromolekuly alebo malé ióny, do seba narážajú a interagujú za tvorby nových chemických látok. Predpokladajme, že substráty A a B spolu reagujú za vzniku novej látky, produktu C:

$$A + B \xrightarrow{k} C. \tag{1.1}$$

Rýchlosť tejto reakcie predstavuje rýchlosť tvorby produktu C, a síce $\frac{d[C]}{dt}$, ktorá stelesňuje počet kolízií za jednotku času medzi reaktantami A a B a zároveň pravdepodobnosť, že tieto kolízie majú dostatočnú energiu na prekonanie aktivačnej energie reakcie. To samozrejme záleží po prvé na koncentrácii reaktantov, ale aj na ich tvare a veľkosti a tiež na teplote a pH roztoku. Kombináciou týchto a aj ďalších faktorov dostávame nelineárnu ordinárnu diferenciálnu rovnicu (z angl. *ordinary differential equation*, skrátene ODE) [25]:

$$\frac{d[C]}{dt} = k[A][B] \tag{1.2}$$

Identifikácia vzťahu 1.1 s rovnicou 1.2 sa nazýva zákon mass action kinetiky (z angl. *law of mass action*) a konštanta k je potom rýchlostná konštanta (z angl. *rate constant*) tejto reakcie.

V skutočnosti nejde o zákon ako taký. Nie je to neporušiteľné pravidlo ako Newtonov gravitačný zákon, ale skôr veľmi užitočný model, ktorý však nemusí byť vo všetkých prípadoch validný. [18]

1.4 Michaelis-Mentenovej a Hillova kinetika

Je nutné začať od enzýmovej kinetiky, pretože práve tá bola hlavným katalyzátorom pre objavenie nových spôsobov modelovania chemických reakcií. Tiež je dobré si uvedomiť, prečo tomu tak bolo. Enzýmová kinetika totiž patrí medzi niekoľko prípadov, v ktorých použitie mass action kinetiky nie je validné. Podľa nej sa so zvyšujúcou koncentráciou substrátu S zvyšuje rýchlosť reakcie zlučovania s enzýmom E lineárne. Zatiaľ čo v *in-vivo* prípade táto rýchlosť postupne konverguje k určitému maximu, cez ktoré sa nedá dostať ani dodatočným zvýšením koncentrácie substrátu S.

Model, ktorý vysvetľoval túto odchýlku od zákona o mass action kinetike, bol prvýkrát prezentovaný v roku 1913 nemeckým biochemikom Leonardom Michaelisom a kanadskou fyzičkou Maud Mentenovou (odtiaľ *kinetika Michaelis-Mentenovej*). V ich reakcii premieňal enzým E subtrát S na produkt P v dvoch fázach. Prvá bola reverzibilná reakcia zlučovania S s E za vzniku enzým-substrátového komplexu C a druhá predstavovala rozpad komplexu C za vzniku produktu P a uvoľnenia nezmeneného enzýmu E (viď rovnicu 1.3).

$$S + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} C \xrightarrow{k_3} E + P \tag{1.3}$$

Je dôležité si všimnúť, že druhá reakcia nie je reverzibilná.

Existujú dva spôsoby ako analyzovať túto rovnicu a obe sú si veľmi podobné. Ide o rovnovážnu aproximáciu a aproximáciu kvázistacionárneho stavu. Začneme aplikáciou zákona o mass action kinetike na tieto reakcie, čo nám vo výsledku dá nasledujúce diferenciálne rovnice, vyjadrujúce rýchlosti zmien jednotlivých chemických látok:

$$\frac{d[S]}{dt} = k_2[C] - k_1[S][E], \tag{1.4}$$

$$\frac{d[E]}{dt} = (k_2 + k_3)[C] - k_1[S][E], \tag{1.5}$$

$$\frac{d[C]}{dt} = k_1[S][E] - (k_2 + k_3)[C], \tag{1.6}$$

$$\frac{d[P]}{dt} = k_3[C]. \tag{1.7}$$

Všimnite si, že platí $\frac{d[C]}{dt} + \frac{d[E]}{dt} = 0$, a preto

$$[E] + [C] = [E_0], \tag{1.8}$$

kde $[E_0]$ je absolútna koncentrácia enzýmu v reakcii. [19]

A. Rovnovážna aproximácia

V pôvodnej analýze Michaelis a Mentenová predpokladali, že substrát je v neustálej rovnováhe s enzým-substrátovým komplexom, a teda, že platí

$$k_1[S][E] = k_2[C]. \tag{1.9}$$

Potom na základe platnosti vzťahov 1.8 a 1.9, môžeme zadefinovať nasledujúci vzťah:

$$[C] = \frac{[E_0][S]}{K_s + [S]}, \tag{1.10}$$

kde $K_s = \frac{k_2}{k_1}$. Z toho zase vyplýva, že rýchlosť reakcie V, respektíve rýchlosť tvorby produktu P, môže byť zadaná takto:

$$V = \frac{d[P]}{dt} = k_3[C] = \frac{k_3[E_0][S]}{K_s + [S]} = \frac{V_{max}[S]}{K_s + [S]}, \tag{1.11}$$

kde $V_{max} = k_3[E_0]$ je maximálna reakčná rýchlosť a predstavuje prípad, keď všetky molekuly enzýmu sú naviazané na substrát.

Pri malej koncentrácii substrátu je rýchlosť reakcie lineárna, ak táto koncentrácia nepresiahne celkové množstvo enzýmu. Avšak pri väčších koncentráciách substrátu je rýchlosť reakcie limitovaná množstvom enzýmu a disociačnou konštantou (v našom prípade k_3). Ak sa koncentrácia substrátu približuje hodnote K_s, znamená to, že reakčná rýchlosť je rovná polovici svojho maxima.

Poznamenajme však, že vzťah 1.9 v reálnych podmienkach takmer nikdy neplatí a rýchlosť reakcie je tak ovplyvnená aj disociačnou konštantou enzým-substrátového komplexu v smere spätného rozpadu (v našom prípade k_2). Práve z tohto dôvodu tu hovoríme o aproximácii. [19]

B: Aproximácia kvázistacionárneho stavu

Inou alternatívou analýzi tejto enzymatickej reakcie je práve aproximácia kvázistacionárneho stavu (z angl. *Quasi-steady state aproximation*), ktorá je v dnešnej dobe aj najpoužívanejšia. Jej tvorcovia, George Briggs a J.B.S.

Haldane predpokladali, že rýchlosť tvorby enzým-substrátového komplexu aj jeho disociácia sú si od počiatku rovné. Takže platí $\frac{d[C]}{dt} = 0$.

Aby sme dali tejto aproximácii správny matematický základ, je vhodné zadefinovať nasledujúce bezrozmerné [1] premenné:

$$\sigma = \frac{[S]}{[S_0]}, \qquad \chi = \frac{[C]}{[E_0]}, \qquad \tau = k_1[E_0]t,$$
$$\kappa = \frac{k_2 + k_3}{k_1[S_0]}, \qquad \epsilon = \frac{[E_0]}{[S_0]}, \qquad \alpha = \frac{k_2}{k_1[S_0]}, \tag{1.12}$$

s pomocou ktorých dostaneme systém iba dvoch diferenciálnych rovníc:

$$\frac{d\sigma}{d\tau} = -\sigma + \chi(\sigma + \alpha), \tag{1.13}$$
$$\epsilon\frac{d\chi}{d\tau} = \sigma - \chi(\sigma + \kappa). \tag{1.14}$$

V porovnaní s koncentráciou substrátu nejakej reakcie je koncentrácia enzýmu vo väčšine prípadov oveľa menšia. Táto skutočnosť pekne odzrkadľuje efektívnosť enzýmov ako katalyzátorov chemických reakcií. Preto je ϵ veľmi malé, typicky v rozmedzí od 10^{-2} do 10^{-7}. Napriek tomu je reakcia 1.14 rýchla a tiež rýchlo spadá do rovnováhy, v ktorej zostáva, aj keď sa hodnota premennej σ mení. Preto uvažujeme túto aproximáciu ako $\epsilon\frac{d\chi}{d\tau} = 0$. Toto tvrdenie je ekvivalentné tomu úvodnému, že $\frac{d[C]}{dt} = 0$.

Potom z tejto aproximácie vyplýva:

$$\chi = \frac{\sigma}{\sigma + \kappa}, \tag{1.15}$$
$$\frac{d\sigma}{d\tau} = -\frac{q\sigma}{\sigma + \kappa}, \tag{1.16}$$

kde $q = \kappa - \alpha = \frac{k_3}{k_1[S_0]}$. Rovnica 1.16 popisuje rýchlosť prírastku substrátu a je pomenovaná zákon Michaelis-Mentenovej (z angl. *Michaelis-Menten law*). V znení pôvodných premenných tento zákon vyzerá takto:

$$V = \frac{d[P]}{dt} = -\frac{d[S]}{dt} = \frac{k_3[E_0][S]}{[S] + K_m} = \frac{V_{max}[S]}{[S] + K_m}, \tag{1.17}$$

kde $K_m = \frac{k_2+k_3}{k_1}$. Vzťahy 1.11 a 1.17 sú si už na prvý pohľad veľmi podobné. Jediným rozdielom sú konštanty K_s a K_m. Ide o dva podobné výsledky na základe rôznych predpokladov.

---

1. Existuje viacero spôsobov, ako systém diferenciálnych rovníc previesť na bezrozmerný. O tom však táto práca nepojednáva. Ďalšie informácie ohľadom tejto problematiky je možné nájsť v [20].

Tak ako zákon o mass action kinetike aj Michaelis-Mentenovej zákon (viď rovnicu 1.17) nie je platný univerzálne. Je však veľmi užitočný a používaný, pretože koeficient K_m je experimentálne dobre pozorovateľný a teda aj ľahko merateľný na rozdiel od individuálnych rýchlostných konštánt jednotlivých chemických substancií v reakcii. [19]

C: Enzýmová spolupráca

Reakčná rýchlosť mnohých enzýmov nemá klasickú hyperbolickú krivku, tak ako je predpokladané pri použití Michaelis-Mentenovej kinetiky, ale miesto toho má sigmoidálny charakter. To je spôsobené vlastnosťou týchto enzýmov, ktorá im umožňuje viazať na seba viacero substrátov naraz a zároveň tým ovplyvniť obtiažnosť tohto viazania. Táto vlastnosť sa nazýva kooperácia alebo súčinnosť či spolupráca.

Tieto enzýmy majú viac ako len jedno viazacie miesto (z angl. *binding site*) a pri naviazaní prvej molekuly dochádza k zmene konformácie vzniknutého komplexu, čo môže ovplyvniť naviazanie ďalšej molekuly pozitívne, ale aj negatívne. Pre každú novú molekulu sa tento proces stupňuje.

Predpokladajme, že enzým E dokáže viazať až dve molekuly substrátu S, takže sa môže nachádzať v troch rôznych stavoch:

1. voľná molekula enzýmu (E),
2. komplex s jednou naviazanou molekulou substrátu (C_1),
3. komplex s dvoma naviazanými molekulami substrátu (C_2).

Potom samotné reakcie vyzerajú nasledovne:

$$S + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} C_1 \xrightarrow{k_3} E + P, \tag{1.18}$$

$$S + C_1 \underset{k_5}{\overset{k_4}{\rightleftarrows}} C_2 \xrightarrow{k_6} C_1 + P. \tag{1.19}$$

Použitím zákona o mass action kinetike dostaneme najskôr päť diferenciálnych rovníc a po úprave len tri. Následným uplatnením aproximácie kvázistacionárneho stavu dostaneme:

$$[C_1] = \frac{K_2[E_0][S]}{K_1K_2 + K_2[S] + [S]^2}, \tag{1.20}$$

$$[C_2] = \frac{[E_0][S]^2}{K_1K_2 + K_2[S] + [S]^2}, \tag{1.21}$$

kde $K_1 = \frac{k_2+k_3}{k_1}$, $K_2 = \frac{k_5+k_6}{k_4}$ a $[E_0] = [E] + [C_1] + [C_2]$. Reakčná rýchlosť potom vyzerá takto:

$$V = k_3[C_1] + k_6[C_2] = \frac{(k_3K_2 + k_6[S])[E_0][S]}{K_1K_2 + K_2[S] + [S]^2}. \tag{1.22}$$

Ak budeme teraz pre ukážku uvažovať prípad pozitívnej spolupráce, tak naviazanie prvej molekuly S bude relatívne pomalé, ale naviazanie druhej molekuly S už bude rýchlejšie. Tento jav môžeme vyjadriť ako $k_4 \to \infty$ a zároveň $k_1 \to 0$, zatiaľ čo k_1k_4 je konštantná hodnota. V tomto prípade ale naopak platí, že $K_1 \to \infty$ a $K_2 \to 0$, zatiaľ čo K_1K_2 je tiež konštantné. Po aplikácii týchto nových obmedzení na vzťah 1.22 dostávame:

$$V = \frac{k_6[E_0][S]^2}{K_m^2 + [S]^2} = \frac{V_{max}[S]^2}{K_m^2 + [S]^2}, \tag{1.23}$$

kde $K_m^2 = K_1K_2$ a $V_{max} = k_6[E_0]$.

Obecne sa dá povedať, že ak enzým dokáže viazať n molekúl substrátu, existuje tiež n rovnovážnych konštánt $K_1, \ldots, K_n$, pre ktoré bude platiť $K_n \to 0$ a $K_1 \to \infty$, zatiaľ čo K_1K_n bude stále konštanta a obecná rovnica reakčnej rýchlosti potom vyzerá:

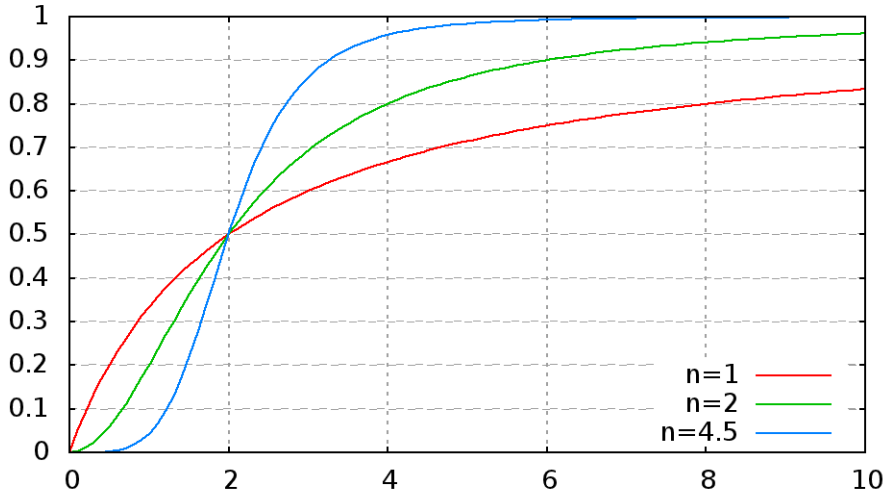
$$V = \frac{V_{max}[S]^n}{K_m^n + [S]^n}, \tag{1.24}$$

kde $K_m^n = \prod_{i=1}^{n} K_i$. Táto rovnica je známa ako Hillova rovnica alebo Hillova kinetika a konštanta K_m^n vyjadruje koncentráciu, pri ktorej je rýchlosť reakcie v polovici svojho maxima, tj. $\frac{V_{max}}{2}$. Typicky faktor n, vyjadrujúci strmosť reakčnej krivky (viď Obr. 1.2) býva menší ako skutočný počet viazacích miest na enzýme, často to dokonca nie je ani celé číslo. Stojí za zmienku, že v prípade, keď $n = 1$, zodpovedá Hillova kinetika Michaelis-Mentenovej kinetike, preto náš nástroj ponúka iba možnosť zadania Hillovej rovnice (viď kapitolu 2.B).

Táto metóda aproximácie je okrem enzýmovej kinetiky veľmi vhodná pre simulovanie regulácie génovej expresie pomocou represorov alebo aktivátorov. [19]

1.5 Model checking

Model checking alebo overovanie modelov je automatizovaná technika formálnej verifikácie špecifikovaných vlastností konečného stavového systému. Hlavnou výzvou tejto problematiky je úspešne si poradiť s problémom explózie stavového priestoru (z angl. *state space explosion*).



Obr. 1.2: Faktor n Hillovej kinetiky pre veľkosti $1, 2$ a 4.5 ($K_m = 2$)

Proces overovania modelov pozostáva z nasledujúcich podúloh:

Modelovanie Prvou úlohou je prevedenie skúmaného systému do formálneho matematického modelu, ktorý bude akceptovaný vybraným overovacím nástrojom. V niektorých prípadoch je to ľahká úloha, avšak v iných je potreba použiť vhodné abstrakcie za účelom odstránenia irelevantných detailov alebo naopak zvýraznenia istých črtov skúmaného systému.

Špecifikácia Ešte pred samotným overovaním, je nevyhnutné špecifikovať vlastnosti, ktoré má skúmaný systém spĺňať, pretože práve tie budeme ďalej overovať. Špecifikáciou sa myslí použitie nejakého vhodného formalismu. Typicky napríklad temporálnej logiky, ktorá umožňuje skúmať správanie systému v čase. My budeme používať LTL (viď kapitolu 1.1).

Overovanie Model checking dokáže overiť, či model vyhovuje danej špecifikácii, ale nedokáže rozhodnúť, či daná špecifikácia pokrýva všetky vlastnosti, ktorým skúmaný systém vyhovuje. Toto je veľmi významný problém formálneho overovania modelov.

Výsledkom overovania modelu je buď tvrdenie áno (v zmysle model spĺňa vlastnosť) alebo nie (model nespĺňa vlastnosť) a v tomto prípade by mal použitý nástroj poskytnúť možnosť trasovania chyby

(z angl. *error trace*). Táto chybová trasa grafom sa obvykle používa ako protipríklad k overovanej vlastnosti. S jej pomocou môžeme lepšie pochopiť dôvod a miesto vzniku chyby a upraviť systém podľa toho. [16]

Prvé algoritmy pre overovanie modelov používali ako formalizmus matematického modelu daného systému Kripkeho štruktúru. Je to obdoba nedeterministického konečného automatu (viď kapitolu 1.2), ktorého stavy sú označené výrazmi z množiny 2^{AP}, kde AP sú atomické propozície formuly f (viď kapitolu 1.1). Všetky tieto stavy sú akceptujúce.

Neskôr sa používal μ-kalkulus, ale v súčasnosti sú najrozšírenejším formalizmom automaty. Konkrétne budeme pojednávať o použití Büchiho automatu (viď kapitolu 1.2). Tento je veľmi vhodný, pretože sa s jeho pomocou dá vyjadriť rovnako model systému ako aj špecifikácia vlastnosti po vynaložení určitého výpočtového úsilia.

Prvou fázou tvorby modelu systému je vytvorenie Kripkeho štruktúry, ktorá sa ale veľmi jednoducho prevedie na automat $\mathcal{A}$. Špecifikáciu vlastnosti môžeme vyjadriť ako automat $\mathcal{S}$. Potom sú oba tieto automaty vytvorené nad rovnakou abecedou $\Sigma = 2^{AP}$. Overenie modelu teraz znamená jednoducho zistiť, či platí $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S})$, kde $\mathcal{L}(\mathcal{A})$ je jazyk zodpovedajúci automatu $\mathcal{A}$ a $\mathcal{L}(\mathcal{S})$ je jazyk zodpovedajúci automatu $\mathcal{S}$ (Jazyk je množina všetkých slov, ktoré je možno vygenerovať príslušným automatom.). To znamená, že každé chovanie modelovaného systému, určeného jazykom $\mathcal{L}(\mathcal{A})$ sa nachádza medzi povolenými chovaniami, určenými jazykom špecifikácie $\mathcal{L}(\mathcal{S})$.

Toto tvrdenie môžeme preformulovať. Nech $\overline{\mathcal{L}(\mathcal{S})}$ je komplement k $\mathcal{L}(\mathcal{S})$, potom pri overovaní modelu dokazujeme, či platí $\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{S})} = \emptyset$. Ak je tento prienik neprázdny, predstavuje to chovanie, ktoré je protipríkladom k overovanej vlastnosti. Použiť predchádzajúcu formuláciu nám umožňuje vedomosť, že Büchiho automaty sú uzavreté na prienik a doplnok (komplement). Vďaka tomu môžeme urobiť nasledovné:

$$\begin{aligned} \emptyset = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{S}^{'}) = \mathcal{L}(\mathcal{A} \cap \mathcal{S}^{'}) = \mathcal{L}(\mathcal{M}), \\ \text{kde } \mathcal{M} = \mathcal{A} \cap \mathcal{S}^{'} \text{ a } \mathcal{S}^{'} = \mathcal{S}(\neg\phi), \\ \text{kde } \phi \text{ je LTL formula a } \neg\phi \text{ jej negácia,} \end{aligned}$$

potom $\mathcal{S}^{'}$ je komplement $\mathcal{S}(\phi)$ a v skutočnosti dokazujeme prázdnosť jazyka $\mathcal{L}(\mathcal{M})$ (Pre úplnosť treba dodať, že $\mathcal{S}(\phi)$ je automat $\mathcal{S}$ zkonštruovaný zo špecifikácie vlastnosti ϕ.).

Výhodou použitia automatov pre vyjadrenie modelu skúmaného systému aj špecifikácie vlastnosti je, že stačí skonštruovať automat pre vlast-

nosť (viď kapitolu 1.5.1) a automat modelu sa konštruuje za behu samotného overovania pomocou algoritmu pre prienik automatov. To znamená, že v mnohých prípadoch môžeme vyvrátiť splniteľnosť oveľa skôr, než sa vytvorí celý stavový priestor automatu modelu nájdením prvého protipríkladu. Čo značne šetrí čas a priestor. Táto technika sa nazýva overovanie modelu za behu (z angl. *on-the-fly model checking*). [15]

1.5.1 Prevod LTL do BA

Ako sme už predtým niekoľko krát spomenuli, prevod LTL formuly na Büchiho automat nie je primitívny algoritmus. V skutočnosti je príliš rozsiahly pre naše potreby zoznamovania sa s problematikou. Z tohto dôvodu ho tu nebudeme rozoberať, avšak uvedieme niekoľko dôležitých informácii, ktoré sa tohto problému týkajú.

Ešte pred začiatkom sa musí samotná LTL formula ϕ previesť do negatívnej normálnej formy (z angl. *negation normal form*). Najskôr sa preformulujú niektoré temporálne operátory:

- $\mathbf{F}\ \phi\ \Rightarrow\ \mathbf{true}\ \mathbf{U}\ \phi$
- $\mathbf{G}\ \phi\ \Rightarrow\ \mathbf{false}\ \mathbf{R}\ \phi$

a tiež logické operátory tak, aby zostali iba $\wedge$, $\vee$ a $\neg$. V poslednom kroku prípravy sú všetky negácie presunuté dovnútra:

- $\neg(\psi\ \mathbf{U}\ \phi)\ \Rightarrow\ (\neg\psi)\ \mathbf{R}\ (\neg\phi)$
- $\neg(\psi\ \mathbf{R}\ \phi)\ \Rightarrow\ (\neg\psi)\ \mathbf{U}\ (\neg\phi)$
- $\neg(\mathbf{X}\ \phi)\ \Rightarrow\ \mathbf{X}\ (\neg\phi)$.

Ďalej pokračuje dlhý algoritmus [17], ktorého výsledkom je Büchiho automat $\mathcal{S}$. Jeho konštrukcia má exponenciálnu časovú aj priestorovú zložitosť závislú od veľkosti formuly ϕ. Avšak v praxi býva takto skonštruovaný automat ovykle menší.

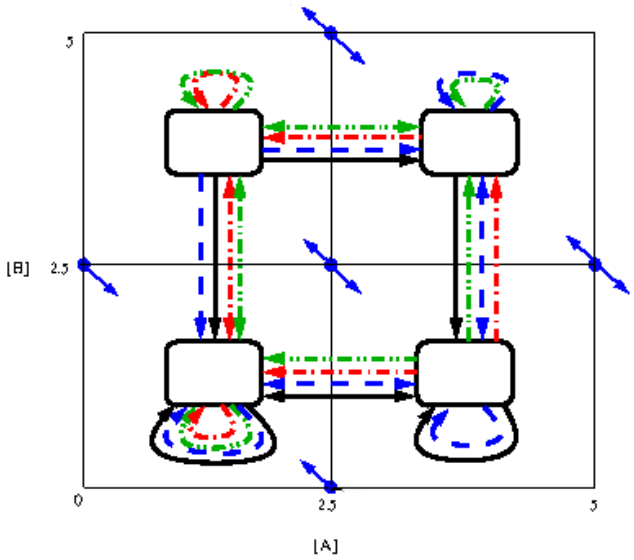
Je dôležité ešte raz pripomenúť, že pri overovaní modelu vlastne nezisťujeme, či model spĺňa vlastnosť, ale naopak zisťujeme, či model nespĺňa opak vlastnosti. Dôvodom k tomu je, že je výpočtovo ľahšie vytvoriť automat z negácie vlastnosti, ako vytvoriť komplement automatu z pôvodnej vlastnosti, ktorý by mohol mať až dvojnásobne exponenciálnu priestorovú zložitosť. [17]

1.5.2 Farebný model checking

Farebný model checking sa snaží vysporiadať s parametrizáciou formálneho modelu, ktorá ho rozširuje o nový rozmer. Ak sa budeme na jednotlivé parametre $p_i \in P$, kde P je množina všetkých neznámych parametrov, dívať ako na intervaly alebo skôr ako na množiny hodnôt, vďaka diskretizácii. Potom kombinácia evaluacií všetkých neznámych parametrov tvorí parametrický priestor $\mathcal{P} = \prod_{i=1}^{n}[min(p_i), max(p_i)]$, kde $n = |P|$. Tento rozširuje model systému o parametrizované multi-afinné funkcie $f_i(x, \pi_i)$, kde $\pi_i \in \mathcal{P}$ namiesto bežných multi-afinných funkcií $f_i(x)$, kde $x = (x_1, \ldots, x_n)$ je vektor premenných a $f = (f_i, \ldots, f_n) : \mathbb{R}^n \to \mathbb{R}^n$ je vektor multi-afinných funkcií (viď kapitolu 2). Hodnota π_i vyjadruje konkrétnu evaluáciu parametrov v celom modeli a označujeme ju ako farba.

Aby sa zabránilo opakovanému vytváraniu automatu pre každú evaluáciu π_i za účelom overenia modelu, bola vytvorená nová pomocná štruktúra pomenovaná parametrizovaná Kripkeho štruktúra (ďalej len PKS). Tá v sebe tradične uchováva celý stavový priestor modelu, ale navyše s novou informáciou, ktorá rozhoduje pod ktorou farbou, resp. konkrétnou evaluáciou parametrov sa dá prejsť z prechodu s do prechodu $s^{'}$. Každý prechod musí byť umožnený aspoň pod jednou farbou, ale zároveň môže byť aj pod všetkými. Celý stavový priestor je tak zjednotenie všetkých jednofarebných stavových priestorov (viď Obr. 1.3).

Vďaka tomu prebieha overovanie všetkých parametrizácií modelu naraz. Výsledkom je najväčšia množina parametrizácií, v ktorých model spĺňa danú vlastnosť. [1]



Obr. 1.3: Parametrizovaná Kripkeho štruktúra. Každá farba predstavuje inú parametrizáciu a tvori odlišný stavový priestor. [1]

Kapitola 2

Biochemický dynamický vstupný model

Táto kapitola je z veľkej miery doslovne citovaná z nižšie uvedených zdrojov.

A. Multi-afinný ODE model

Vstupným modelom sa u nás myslí model biochemických reakcií, ktorý je v našom poňatí braný ako po častiach multi-afinný systém diferenciálnych rovníc. Ale začnime od počiatku a postupne sa dopracujeme k tomuto výsledku.

Na základe pravidla o mass action kinetike (viď kapitolu 1.3) je možné modelovať ľubovolnú biochemickú reakciu alebo dokonca sústavu takýchto reakcií pomocou sústavy nelineárnych ODE [25].

Uvažujme o multi-afinnom systéme, ktorý má formu $\dot{x} = f(x)$, kde $x = (x_1, \ldots, x_n)$ je vektor premenných a $f = (f_1, \ldots, f_n) : \mathbb{R}^n \to \mathbb{R}^n$ je vektor multi-afinných funkcií. Tieto funkcie sú vlastne polynómy, v ktorých je stupeň premenných $x_1, \ldots, x_n$ obmedzený na hodnotu 1. Každá premenná x_i, kde $i \in \{1, \ldots, n\}$ predstavuje koncentráciu špecifickej chemickej látky a je interpretovaná ako $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$.

Z dôvodu, že premenné môzeme vyjadriť len ako nezáporné reálne čisla, je možné tiež spojitý stavový priestor nášeho matematického systému obmedziť iba na prvý, resp. kladný kvadrant $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n \mid x \geq 0\}$.

Ak uvažujeme o premenných ako o nestabilných chemických látkach, ktoré sami od seba degradujú v čase, môžeme s kľudom obmedziť náš spojitý stavový priestor $\mathcal{D}$ ešte viac. A síce na n-dimenzionálny obdĺžnik $\mathcal{D} = \prod_{i=1}^{n} [0, max_i] \subset \mathbb{R}^n$, kde max_i je horná hranica koncentrácie premennej x_i. [5][6]

B. Nelineárny ODE model

Multi-afinný systém diferenciálnych rovníc dokáže pokryť skoro celú mass action kinetiku s jedinou výnimkou. A tou sú homodiméry a reak-

cie s nimi spojené. Dôvodom je predchádzajúce obmedzenie multi-afinných funkcií $f_1, \ldots, f_n$ s ohľadom na stupeň premenných $x_1, \ldots, x_n$.

Teoreticky sme schopní popísať akýkoľvek biochemický model pomocou pravidiel tejto kinetiky. V skutočnosti, ak sa pokúsime formulovať tieto pravidlá pre rozsiahly model, zistíme, že s narastajúcou veľkosťou rastie komplexita týchto pravidiel a navyše k úplnosti modelu je potrebné poznať veľké množstvo čo najpresnejšie vyčíslených parametrov. Práve tento druhý problém môže byť v niektorých prípadoch experimentálne neriešiteľný. Či už ide o veľké enzymatické komplexy alebo o látky s veľmi krátkou existenciou.

Práve preto sa ponúkajú možnosti aproximácie, ktoré nielen zmenšujú systém, a tým aj dimenzionalitu matematického modelu, ale zjednodušujú aj výpočtovú zložitosť. Takouto možnosťou je aj aproximácia kvázistacionárneho stavu (viď kapitolu 1.4.B). Napríklad Michaelis-Mentenovej kinetika (viď kapitolu 1.4) či obecnejšia Hillova kinetika (viď kapitolu 1.4.C) a tiež sigmoidálne prepínače publikované v [12]. Ďalej sem možno rátať aj Heavisideove alebo schodové funkcie [26].

Všetky vyššie zmienené abstrakcie náš nástroj ponúka a budeme ich jednotne označovať ako regulačné funkcie $\rho(x_i)$, kde $i \in \{1, \ldots, n\}$, ktoré môžu nadobúdať nasledujúce formy:

$$\begin{aligned} hill^+(x_i, \theta_i, d, a, b) &= a + (b-a)\frac{[x_i]^d}{[\theta_i]^d + [x_i]^d}; \\ hill^+(x_i, \theta_i, d, a, b) &= 1 - hill^+(x_i, \theta_i, d, a, b); \end{aligned}$$

$$\begin{aligned} s^+(x_i, e, \theta_i, a, b) &= a + (b-a)\frac{1 + tanh(e(x_i - \theta_i))}{2}; \\ s^-(x_i, e, \theta_i, a, b) &= 1 - s^+(x_i, e, \theta_i, a, b); \end{aligned}$$

$$\begin{aligned} h^+(x_i, \theta_i, a, b) &= a, \text{ ak } x_i < \theta_i;\ b \text{ inak}; \\ h^-(x_i, \theta_i, a, b) &= 1 - h^+(x_i, \theta_i, a, b); \end{aligned}$$

kde $hill^+, hill^-$ sú funkcie Hillovej kinetiky;
s^+, s^- sú sigmoidálne prepínače;
h^+, h^- sú Heavisideove (schodové) funkcie;
$\theta_i \in \mathbb{R}^+;\ \theta_i \leq max_i;\ i \in \{1, \ldots, n\}$;
$a, b \in \mathbb{R}_0^+;\ e, d \in \mathbb{R}^+$.

Špeciálnym prípadom je recipročná hodnota sigmoidálnej funkcie:

$$s^+(x_i, e, \theta_i, a, b)^{-1} \;=\; s^-(x_i, e, \theta_i + \frac{ln(\frac{a}{b})}{2e}, b^{-1}, a^{-1}),$$

ktorú označujeme ako $s^+inv(x_i, e, \theta_i, a, b)$. Potom klesajúcu recipročnú funkciu označíme obdobne ako doplnok rastúcej:

$$s^-inv(x_i, e, \theta_i, a, b) \;=\; 1 - s^+(x_i, e, \theta_i, a, b).$$

Dôkaz možno nájsť v článku [12] na strane 6.

Takto redukované diferenciálne rovnice teraz majú formu racionálnych polynómov, získaných ako lineárna kombinácia týchto regulačných funkcií. V skutočnosti výsledný matematický model nie je multi-afinný, ale na druhú stranu je ho možné aproximovať v zmysle po častiach multi-afinného systému.

Nevyhnutnou súčasťou modelu je množina prahových hodnôt (z angl. *threshold*) $\theta^i_m \in \mathbb{R}^+$ spĺňajúca $min_i = \theta^i_1 < \theta^i_2 < \ldots < \theta^i_{\eta_i} = max_i$, kde $i \in \{1, \ldots, n\}$, $m \in \{1, \ldots, \eta_i\}$ a platí, že $\eta_i \geq 2$.

Teraz už môžeme zadefinovať úplný formát nášho vstupného nelineárneho ODE modelu. Tento model $\mathcal{M}$ je daný ako $\dot{x} = f(x)$, kde x je stále vektor premenných $(x_1, \ldots, x_n)$, ale $f = (f_1, \ldots, f_n) : \mathbb{R}^n \to \mathbb{R}^n$ je tentokrát vektor nelineárnych polynomiálnych funkcií, ktoré majú tvar:

$$f_i(x) = \sum (s \prod_{j \in I} \rho^j_i(x)), \tag{2.1}$$

kde $i \in \{1, \ldots, n\}$, $s \in \{-1, 1\}$, I je neprázdna podmnožina konečnej množiny indexov všetkých použitých regulačných funkcií a ρ^j_i je príslušná regulačná funkcia, ktorá môže nadobúdať tieto hodnoty:

$$\rho(x_t) = \begin{cases} c, & c \in \mathbb{R} \\ x_k, & k \in \{1, \ldots, n\} \\ h^*(x_k, \theta^k_m, a, b), & m \in \{1, \ldots, \eta_k - 1\};\ a, b \in \mathbb{R}^+_0 \\ s^*(x_k, e, \theta^k_m, a, b), & e \in \mathbb{R}^+ \\ s^*inv(x_k, e, \theta^k_m, a, b), & \\ hill^*(x_k, \theta^k_m, d, a, b), & d \in \mathbb{R}^+ \end{cases}, \tag{2.2}$$

kde $* \in \{+, -\}$. Ak $\rho^{j_1}_i, \rho^{j_2}_i$ sú regulačné funkcie, ktoré patria do jednoho produktu $s \prod_{j \in I} \rho^j_i(x)$ tak, že $j_1, j_2 \in I$, musí platiť $dep(\rho^{j_1}_i) \cap dep(\rho^{j_2}_i) = \emptyset$, kde $dep(\rho)$ je množina premenných x, na ktorých je ρ závislá.

Do modelu je priamo zavedená aj možnosť zadania konštantnej funkcie (v prípade, že $\rho(x_t) = c$) a lineárnej funkcie (v prípade $\rho(x_t) = x_k$). Ďalšou nevyhnutnou súčasťou modelu sú iniciálne podmienky, vyjadrujúce počiatočné koncentrácie jednotlivých látok. Tie nie sú vyjadrené presne, ale namiesto toho sú určené intervalom, ktorého hranice musia byť z množiny $\{\theta_1^i, \ldots, \theta_{\eta_i}^i\}$.

V prípade, že sa použijú v modeli iba konštantné a lineárne funckie, dostávame automatický obyčajný multi-afinný model tak, ako je uvedený v kapitole 2.A. V opačnom prípade musí byť na tento model aplikovaná optimálna lineárna abstrakcia (viď kapitolu 2.1.A), aby sme dostali potrebný multi-afinný model. Tentokrát však po častiach multi-afinný. Z týchto dôvodov je takto definovanný model prakticky použiteľný pre ľubovolný biologický systém. [5][6]

C. Po častiach multi-afinný ODE model

Nahradením všetkých regulačných funkcií $\rho(x_i)$ z predchádzajúcej časti sústavou vhodných po častiach lineárnych rampových funkcií, ktoré sú definované nasledovne:

$$r^+(x_i, \theta_i, \theta_i^{'}, y, y^{'}) = \begin{cases} y, & \text{pre } x_i < \theta_i, \\ y + (y^{'} - y)\frac{x_i - \theta_i}{\theta_i^{'} - \theta_i}, & \text{pre } \theta_i < x_i < \theta_i^{'}, \\ y^{'}, & \text{pre } x_i > \theta_i^{'}. \end{cases}$$

$$\begin{aligned} &\text{kde } i \in \{1, \ldots, n\}, \\ &\quad y = x_j, y^{'} = x_j^{'}; j \in \{1, \ldots, n\} \wedge j \neq i, \\ &\quad \theta_i, \theta_i^{'} \in \mathbb{R}^+,\ \theta_i < \theta_i^{'} \leq max_i \end{aligned}$$

a potom klesajúce rampové funkcie sú definované ako kvantitatívny doplnok rastúcich:

$$r^-(x_i, \theta_i, \theta_i^{'}, y, y^{'}) = 1 - r^+(x_i, \theta_i, \theta_i^{'}, y, y^{'}),$$

dostávame po častiach multi-afinný ODE model (ďalej len PMA model z angl. *piecewise multi-affine ODE model*). PMA model $\mathcal{P}$ je daný ako $\dot{x} = f(x)$, kde x je stále vektor premenných $(x_1, \ldots, x_n)$, ale $f = (f_1, \ldots, f_n) : \mathbb{R}^n \to \mathbb{R}^n$ je tentokrát vektor po častiach multi-afinných funkcií.

Stále platí množina prahových hodnôt $\theta_m^i \in \mathbb{R}^+$ spĺňajúca $min_i = \theta_1^i < \theta_2^i < \ldots < \theta_{\eta_i}^i = max_i$, kde $i \in \{1, \ldots, n\}$, $m \in \{1, \ldots, \eta_i\}$ a platí, že $\eta_i \geq 2$,

ktorá však môže byť rozšírená o nové hodnoty – výsledky predchádzajúcej lineárnej abstrakcie.

Uvažujme Ω ako časť modelu $\mathcal{P}$ tak, že $\Omega = \prod_{i=1}^{n}\{1,\ldots,\eta_i - 1\}$. Funkcia $g : \mathbb{R}^n \to \mathbb{R}^+$ je vtedy po častiach multi-afinná, ak je multi-afinná na každom n-dimenzionálnom intervale $(\theta^1_{j_1}, \theta^1_{j_1+1}) \times \ldots \times (\theta^n_{j_n}, \theta^n_{j_n+1})$, kde $(j_1,\ldots,j_n) \in \Omega$ a zároveň $\forall i, 1 \leq i \leq n, j_i < max_i$. Potom dostávame n-dimenzionálny PMA model pozostávajúci z funkcií f v nasledujúcom tvare:

$$f_i(x) = \sum (s \prod_{j \in I} \sigma_i^j(x)), \tag{2.3}$$

kde $i \in \{1,\ldots,n\}$, $s \in \{-1, 1\}$, I je neprázdna podmnožina konečnej množiny indexov všetkých použitých lineárnych (v zmysle lineárnych alebo po častiach lineárnych) funkcií a σ_i^j je príslušná lineárna funkcia, ktorá môže nadobúdať tieto hodnoty:

$$\sigma(x_t) = \begin{cases} c, & c \in \mathbb{R} \\ x_k, & k \in \{1,\ldots,n\} \\ r^*(x_k, \theta^k_m, \theta^k_{m+1}, x_t, x_t^{'}), & m \in \{1,\ldots,\eta_k - 1\}; \\ & t \in \{1,\ldots,n\} \wedge t \neq i \end{cases}, \tag{2.4}$$

kde $* \in \{+,-\}$. Ak $\sigma_i^{j_1}, \sigma_i^{j_2}$ sú lineárne funkcie, ktoré patria do jednoho produktu $s\prod_{j\in I}\sigma_i^j(x)$, tak že $j_1, j_2 \in I$, musí platiť $dep(\sigma_i^{j_1}) \cap dep(\sigma_i^{j_2}) = \emptyset$, kde $dep(\sigma)$ je množina premenných x, na ktorých je σ závislá.

Prípad $\sigma(x_t) = c$ predstavuje konštantnú funkciu a $\sigma(x_t) = x_k$ zase lineárnu funkciu. Rovnako ako v časti 2.B. Samozrejmou nevyhnutnosťou sú iniciálne podmienky, vyjadrujúce počiatočné koncentrácie jednotlivých látok. Tieto sú zhodné s hodnotami z predchádzajúceho nelineárneho modelu.

Takto definovaný PMA systém je možné použiť priamo ako vstupný model. Nemusí byť len výsledkom lineárnej abstrakcie predchádzajúceho nelineárneho ODE systému. [5][6]

2.1 Abstrakcia

A. Optimálna lineárna abstrakcia nelineárnych funkcií

V minulej kapitole sme objasnili, prečo je potrebné prevádzať regulačné funkcie $\rho(x_i)$, kde x_i je premenná na po častiach lineárne rampové funkcie

$r_j^*(x_i, \theta_j^i, \theta_{j+1}^i, y, y')$, kde $j \in \{1, \ldots, m\}$ a m je počet požadovaných rámp, θ^i je prahová hodnota premennej x_i, y a y' sú funkčné hodnoty v týchto prahoch a $* \in \{+, -\}$. V tejto časti sa pokúsime objasniť princíp tohto prevodu. Vynára sa hneď niekoľko problémov:

1. Ako vhodne rozdeliť ľubovolnú krivku na zadaný počet rámp.

2. Ako vhodne rozdeliť niekoľko kriviek rovnakej dimenzie na zadaný počet rámp.

3. Ako vybrať správny počet rámp, tak aby stavový systém nebol príliš veľký a zároveň tak, aby abstrakcia nebola príliš hrubá.

Na prvú otázku dáva odpoveď algoritmus dynamického programovania vyvinutý pre počítačovú grafiku, konkrétne pre aproximáciu digitalizovaných polygonálnych kriviek s minimálnou chybou [22]. Tento bol upravený a rozšírený pre viacero kriviek naraz v článku [12] a to tak, že zmienená chyba sa počíta pre všetky krivky naraz. Aby to bolo možné, musia byť všetky krivky diskretizované v rovnakých x-ových bodoch. V našom prípade sa krivkami myslia Hillove funkcie (viď kapitolu 1.4.C) a sigmoidálne funckie popísané tiež v článku [12].

Účelom tejto abstrakcie je nájsť také body x_i, kde $i \in \{1, \ldots, n-1\}$ a n je počet požadovaných lineárnych segmentov, ktoré budú sekať krivky na tieto segmenty s minimálnou odchýlkou (chybou). Nesmieme však zabudnúť, že počiatočný x_0 a koncový bod x_n krivky je vopred zadaný.

Chyba sa počíta ako súčet druhých mocnín vzdialeností dvoch bodov na y-ovej ose. Týmito bodmi sú funkčná hodnota krivky a funkčná hodnota rampy pre konkrétny bod x_i na úseku ohraničenom bodmi x_a a x_b, kde $x_i \in \langle x_a, x_b \rangle$. Rampa je v tomto zmysle braná ako lineárna spojnica medzi funkčnými hodnotami týchto hraničných bodov. Ako už bolo zmienené vyššie, chyba na úseku $\langle x_a, x_b \rangle$ sa počíta pre všetky krivky naraz a v tomto prípade sa hľadá hodnota najvyššia. Prakticky to znamená, že sa počas procesu aproximácie hľadajú minimálne odchýlky chýb z maximálnych rozdielov kriviek. Ak nájdená hodnota chyby je menšia ako predchádzajúca, tak body x_a a x_b možno pre ďalšiu iteráciu brať ako hraničné body nového segmentu. Ak sa menšia hodnota chyby už nenájde je hraničný bod x_b definitívne uložený. Bod x_a je uložený už z predchádzajúcej iterácie. So všetkými takto získanými bodmi už nie je problém previesť každú funkciu na jej po častiach lineárnu formu. Zároveň treba tieto body uložiť ako nové prahové hodnoty modelu (viď kapitolu 2.B). Príklad abstrakcie znázorňuje Obr. 2.2.

Odpoveď na tretiu otázku ohľadom správneho počtu rámp je nasledujúca. Užívateľ musí nastaviť požadovaný počet sám, to mu však umožňuje, aby skúšal rôzne úrovne abstrakcie osobne. [12]

B. Obdĺžniková abstrakcia PMA modelu

Výhodou multi-afinného systému je, že ho možno abstrahovať do podoby obdĺžnikového prechodového systému (z angl. *Rectangular Abstraction Transition System*, skrátene RATS) [2]. K tomu je potreba pôvodný n-dimenzionálny systém, kde n je počet premenných, najskôr rozdeliť na menšie n-dimenzionálne oddiely. Každá premenná má priradenú množinu prahových hodnôt alebo thresholdov, ktoré vyjadrujú významné, či zaujímavé koncentračné hladiny. Každej premennej sú priradené aspoň dve takéto hodnoty, a to minimálna a maximálna. Ďalšie môžu byť zadefinované vo vstupnom modeli alebo môžu pochádzať z predchádzajúcej lineárnej abstrakcie regulačných funkcií.

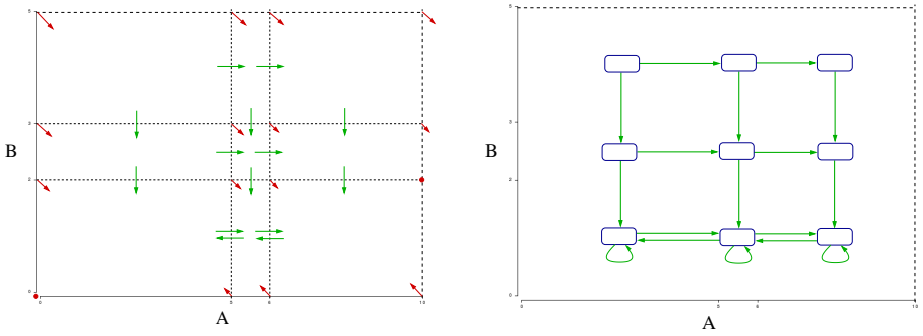
Teraz už nie je problém rozdeliť multi-afinný systém podľa týchto prahov na menšie n-rozmerné ohraničené časti nazvané obdĺžniky. Tieto si môžeme predstaviť ako uzly grafu. Ak sú dva obdĺžniky susedmi, môže medzi nimi vzniknúť prechod v zmysle hrany medzi dvoma uzlami. Takto dostávame diskrétny prechodový systém (RATS) z pôvodne dynamického systému.

Prechody sú samozrejme orientované. Hodnotu orientácie dostaneme zo smerových vektorov vypočítaných vo všetkých rohoch n-rozmerného obdĺžnika. A to tak, že ak aspoň jeden vektor v spoločných rohoch so susediacim obdĺžnikom má smer k tomuto susedovi, pridáme odchádzajúcu hranu z tohto uzla do susedného. Ak aspoň jeden vektor v spoločných rohoch so susediacim obdĺžnikom má smer od tohto suseda, pridáme prichádzajúcu hranu zo susediaceho uzla k tomuto uzlu. To znamená, že hrany môžu byť aj obojsmerné. Podmienkou je, že ak nastane situácia, kedy uzol nebude mať žiadnu odchádzajúcu hranu, musíme mu pridať slučku. Tento jav symbolizuje rovnovážny stav v daných podmienkach (viď Obr. 2.1).

Je známy fakt, že obdĺžniková abstrakcia je nadaproximáciou vzhľadom k trajektóriam pôvodného dynamického modelu. [4]

2.2 Vlastnosti modelu

Nepriamou súčasťou modelu je aj vlastnosť, ktorej splniteľnosť budeme testovať. Táto je popísaná v tvare LTL formuly (viď kapitolu 1.1), ktorá

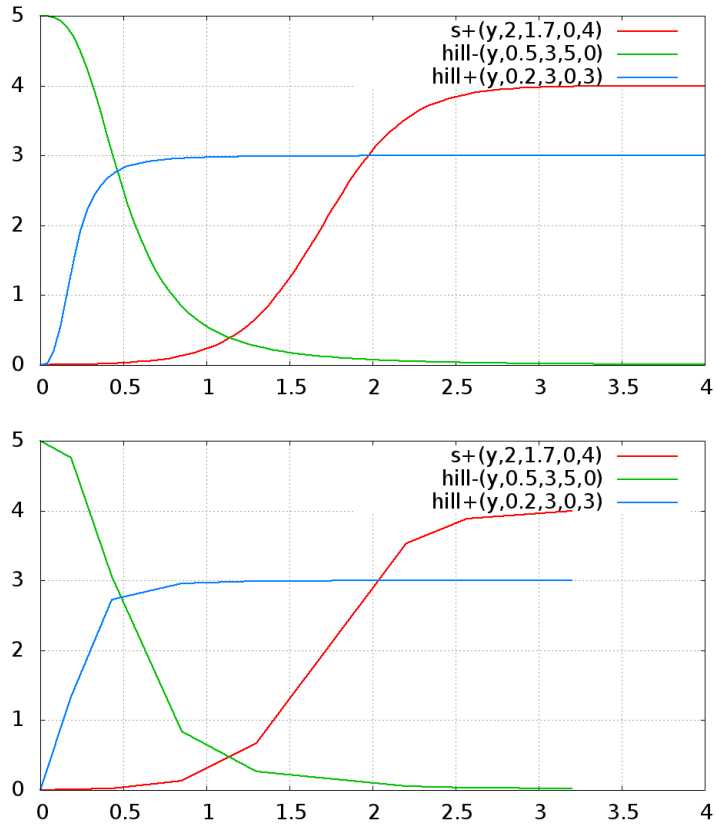


Obr. 2.1: Príklad obdĺžnikovej abstrakcie 2-rozmerného modelu a výsledný RATS.

obsahuje atomické propozície (d'alej iba AP). To sú výrazy s pravdivostnou hodnotou (*true* alebo *false*) skladajúce sa z porovnávaní hodnôt premenných x_i s ich prahovými hodnotami θ_j^i, kde $i \in \{1, \ldots, n\}$ a n je počet premenných a $j \in \{1, \ldots, m\}$ a m je počet prahových hodnôt premennej x_i (viď kapitolu 2), pomocou relačných operátorov ($<$, $>$, $<=$ alebo $>=$). Inými slovami AP vyzerá:

$$x_i \ op \ \theta_j^i, \ \text{ kde } \ op \in \{<, >, <=, >=\}.$$

Viacero AP možno spájať logickými spojkami.



Obr. 2.2: Tri krivky s popisom. V prvom prípade bez použitia abstrakcie a v druhom prevedené na sústavu rámp po použití abstrakcie s parametrami: diskretizačných bodov 500, požadovaný počet segmentov 7.

Kapitola 3

Východiskový stav a podobné nástroje

3.1 BioDiVinE 1.0

Pôvodný nástroj BioDiVinE bol vyvinutý v Laboratóriu systémovej biológie (SYBILA)[1] v spolupráci s Laboratóriom paralelných a distribuovaných systémov (ParaDiSe)[2]. Obe sídlom na Fakulte informatiky Masarykovej univerzity.

BioDiVinE 1.0 je nástroj vytvorený pre verifikáciu vlastností biochemických modelov zadaných ako systém ODE (viď kapitolu 2). Je to nadstavba nástroja DiVinE [3], určeného na model checking (viď kapitolu 1.5). K tomuto účelu ponúka niekoľko odlišných algoritmov.

Vstupom je, ako už bolo povedané biochemický model v tzv. .bio formáte (podrobnosti v [7]), ktorý môže predstavovať genovú regulačnú sieť alebo vzájomnú interakciu proteínov či iné. Dôležitou súčasťou je aj súbor s jednou alebo viacerými vlastnosťami zapísanými ako LTL formuly (viď kapitolu 1.1). Tieto dva súbory sa musia skombinovať do jednoho použitím príslušného nástroja, pričom vlastnosť sa zároveň prevedie z LTL formuly na Büchiho automat (viď kapitolu 1.2). Takto hotový vstup sa môže predať jednomu z overovacích algoritmov.

Je vidno, že postup nie je veľmi užívateľsky prívetivý. Užívateľ musí najskôr použiť jeden nástroj pre vytvorenie vstupného súboru, potom ho ručne premenovať, aby ho ďalší nástroj, tentokrát overovací, správne rozpoznal. Výsledky v podobe protipríkladu alebo podrobný výpis priebehu procesu sú vygenerované do nových súborov .trail alebo .report po zadaní príslušného vstupného prepínača vybraného algoritmu. Jedinou výhodou je, že väčšina prepínačov je unifikovaná, pretože nástroje boli vytvorené jedným vývojárskym tímom.

Celý nástroj BioDiVinE pritom obsahuje zbytočne veľa ďalších rozšírení pre iné typy modelov, pretože pôvodný nástroj DiVinE bol prevzatý ako celok aj s týmito predchádzajúcimi rozšíreniami.

---

1. http://sybila.fi.muni.cz/home
2. http://paradise.fi.muni.cz/index.html

Kvôli vyššie spomenutým dôvodom sme si dali za cieľ okrem rozšírenia vstupného modelu a abstrakcie aj refaktorizáciu starého nástroja. [4]

3.2 PEPMC

Nástroj vyvinutý v Laboratóriu systémovej biológie (SYBILA)[3] pre overovanie modelov s ohľadom na odhad parametrov (z angl. *Parameter Estimation by Parallel Model Checking*, skrátene PEPMC). Funguje na princípe paralelizovaného farebného overovania modelov (viď kapitolu 1.5.2). Modelom je systém ODE podobne ako v nástroji BioDiVinE 1.0, ale rozšírený o možnosť zadávania po častiach lineárnych rampových funkcií (viď kapitolu 2). Dôležitým rozdielom oproti BioDiVinE-u a súčasne hlavnou podstatou nástroja PEPMC však je parametrizácia modelu a s tým súvisiace prehľadávanie parametrického priestoru.

Existuje myšlienka zlúčiť tieto dva nástroje, bráni tomu však mierne odlišný vstup, datový model a samozrejme rozdielne jadrá programov. Nebudeme tu však tento problém ďalej rozoberať, pretože nie je súčasťou našej témy. [6]

3.3 RoVerGeNe

RoVerGeNe je podobne ako PEPMC nástroj pre analýzu genetických regulačných sietí s neurčitými parametrami. Model siete je zapísaný ako po častiach multi-afinný systém diferenciálnych rovníc. Vlastnosti, proti ktorým sa model overuje, majú formu LTL formuly a parametre sú zadané ako intervaly. Nástroj sa snaží overiť, či model spĺňa danú vlastnosť pre všetky parametre.

Na rozdiel od PEPMC však RoVerGeNe dokáže pracovať aj s lineárnymi kombináciami parametrov. To v praxi znamená, že obdĺžniková abstrakcia (viď kapitolu 2.1.B) u neho nie je použiteľná, pretože vyžaduje, aby n-dimenzionálne štvoruholníky, s ktorými pracuje, boli ortogonálne čiže obdĺžniky. Namiesto toho používa tento nástroj štandardné polyhedrálne operácie balíka MPT [21] a celý je naprogramovaný v Matlabe. [9] [8]

3. http://sybila.fi.muni.cz/home

Kapitola 4

BioDiVinE 1.1

V kapitole 3.1 boli predstavené niektoré nedostatky pôvodného nástroja BioDiVinE 1.0, ktoré sme si dali za cieľ odstrániť. Súčasne bol záujem o rozšírenie jeho funkcionality a odstránenie nadbytočných častí kódu.

Prvotný impulz spočíval v zovšeobecnení vstupného modelu, aby poskytoval širší záber možností. Aby nebol obmedzený iba na mass action kinetiku (viď kapitolu 1.3) a s tým súvisiace nelineárne diferenciálne rovnice. Do úvahy prichádzali samozrejme najbežnejšie používané Hillova kinetika a Michaelis-Mentenovej kinetika (viď kapitolu 1.4), ale aj Heavisidove funkcie [26], sigmoidálne funkcie spomínané v [12] a obecné rampové funkcie (viď kapitolu 2.B), ktoré slúžia aj ako produkt optimálnej lineárnej abstrakcie (viď kapitolu 2.1.A).

Tým sa dostávame k ďalšej novej funkcionalite. Zlúčenie nového rozšíreného vstupného modelu (viď kapitolu 2) a pôvodne používanej obdĺžnikovej abstrakcie (viď kapitolu 2.1.B). Prechodom medzi tým je práve vyššie spomenutá optimálna lineárna abstrakcia, ktorá prevedie novozavedené nemulti-afinné funkcie na po častiach multi-afinné, presne ako je potrebné pre obdĺžnikovú abstrakciu.

Pri zápise jednotlivých rovníc modelu sme sa chceli vyhnúť operácii delenia, preto sme sa rozhodli všetky pridávané funkcie zapisovať v podstate unifikovaným spôsobom, ktorý jasne určuje, o aký typ funkcie ide, či je rastúca alebo klesajúca a ďalej parametre nevyhnutné pre jej vyčíslenie (viď kapitolu 4.1). Takisto sme pridali pár noviniek ako definovanie pomenovaných konštánt a voliteľné, užívateľom zadávané parametre pre lineárnu abstrakciu. Konkrétne ide o počet bodov, v ktorých majú byť funkcie evaluované a o počet segmentov, na ktoré majú byť funkcie rozdelené a linearizované.

Z týchto dôvodov sme sa rozhodli vytvoriť úplne nový parser (viď kapitolu 5.1) a súčasne s ním aj nový datový model pre uchovanie nových dát. Niektoré prvky boli síce zhodné so starým BioDiVinE-om, ale spôsob, akým boli naimplementované v starej verzii, bol veľmi zle rozšíriteľný. Začať v týchto bodoch odznova nám prišlo jednoduchšie a prehľadnejšie.

Avšak kód starého nástroja bol hodne rozsiahly a skladal sa z veľkého množstva rôznych tried, z ktorých mnohé už pre naše účely neboli potrebné. Preto bolo nutné najskôr celý kód postupne prechádzať a debugovať, aby sme mu lepšie porozumeli. Na základe toho sme boli schopní eliminovať neužitočné a nepoužívané časti kódu. Odstránili sme tak počas fázy refactoringu asi polovicu tried.

Samotné pripojenie našich nových tried pre parser a datový model, ale nebolo triviálne. K tomu všetkému sme potrebovali zachovať časť starého parseru, ktorá mala za úlohu načítať Büchiho automat (viď kapitolu 1.2) so skúmanou vlastnosťou.

Overovacie algoritmy boli naštastie napísané dostatočne obecne, takže po úspešnom zavedení nových tried do starých sa nevyskytlo veľa problémov. Najdôležitejšie pre ucelenie celého nástroja však bolo vytvorenie konzolového užívateľského prostredia (ďalej len CLI), ktoré do jednoho príkazu ukryje to, čo by inak bolo nutné spraviť v troch krokoch (viď kapitolu 3.1). Okrem toho automaticky overí všetky vlastnosti zadané vo vstupnom súbore s týmito vlastnosťami a následne zmaže dočasné súbory.

Vstupom pre CLI je názov vybraného overovacieho algoritmu, voliteľné prepínače a dva súbory. Jedným je cesta k súboru s modelom s príponou .bio a druhým je cesta k súboru s jednou alebo viacerými vlastnosťami zapísanými ako LTL formuly (viď kapitolu 1.1) s príponou .ltl.

Bohužial sme boli donútení odstrániť GUI z pôvodného nástroja, ktoré slúžilo na vytváranie a ukladanie modelov a vizualizáciu výsledkov, pretože sa ukázalo nezlučiteľné s novými rozšíreniami a bolo by nutné vytvoriť úplne novú verziu. Táto činnosť je ponechaná pre budúce možnosti rozšírenia.

Pôvodný BioDiVinE disponoval aj možnosťou paralelne distribuovaného spracovávania prostredníctvom protokolu MPI (z angl. *Message Passing Interface*) [23]. Táto možnosť zostala zachovaná v rámci všetkých pôvodných častí zdrojového kódu.

Kvôli lepšiemu a celistvejšiemu prehľadu všetkého vyššie zmieneného ponúkame pohľad na architektúru nástroja BioDiVinE 1.1 v diagrame (Obr. 4.1), kde je červenou farbou zvýraznená oblasť veľkej časti našej práce.

4.1 Vstupný súbor s modelom

Hlavnou náplňou tejto podkapitoly je zoznámiť bližšie užívateľa so syntaxou vstupného modelu. Tým sa myslí súbor s príponou .bio a vyzerá nasledovne:

- VARS : *premenná1*, *premenná2*, ...
- CONSTS : *konštanta1*, *hodnota1*; *konštanta2*, *hodnota2* ; ...
- EQ : *premenná1* = *rovnica1*
- EQ : *premenná2* = *rovnica2*
- THRES : *premenná1*: *prah1*, *prah2*, *prah3*, ...
- THRES : *premenná2*: *prah1*, *prah2*, ...
- VAR_POINTS : *premenná1*: *počet_bodov*, *počet_segmentov*; *premenná2*: *počet_bodov*, *počet_segmentov*
- INIT : *premenná1*: *od*, *do*; *premenná2*: *od*, *do*
- system async;

Niekoľko vysvetliviek a obmedzení:

1. Všetko, čo je písané neskoseným písmom, musí zostať zachované. Naopak to, čo je napísané skoseným písmom, treba nahradiť vlastnými hodnotami.
2. Biele znaky sú kompletne ignorované, dôležité sú iba oddeľovače vyznačené hrubým písmom (: ; , =). Na koncoch riadkov sa nikdy oddeľovač nesmie nachádzať s výnimkou posledného riadku (system async;).
3. Názvy premenných a konštánt musia ako prvý znak obsahovať písmeno nasledované ľubovolnou kombináciou písmen, čísel a znakov _ ~ { }.
4. Čísla majú tvar celých alebo desatinných čísel s desatinnou bodkou.
5. Riadok (CONSTS) nie je povinný. Je tiež možné používať iba nepomenované konštanty (čiže čísla).
6. Riadky (VARS) a (CONSTS) sa musia nachádzať pred ostatnými riadkami – sú úvodné.
7. Riadok (VAR_POINTS) nie je povinný a *počet_bodov* znamená počet bodov, v ktorých budú evaluované regulačné funkcie pre danú premennú, *počet_segmentov* určuje počet rámp, na ktoré sa tieto funkcie prevedú.

8. Hodnoty *od* a *do* v riadku (INIT) musia byť niektoré z hodnôt v príslušnom riadku (THRES) pre danú premennú, pre ktoré musí platiť $od < do$.

9. Každá premenná má povinný vlastný riadok (EQ) a (THRES) a musí sa nachádzať aj v riadku (INIT) s príslušnými hodnotami.

10. Riadok (system async;) je povinný a musí sa nachádzať na konci súboru z historických dôvodov.

11. Rovnica v riadku (EQ) je matematicky popísaná v kapitole 2 a v bežnom jazyku nižšie:

    - Parser podporuje unárnu operáciu $-$, binárne operácie $+$, $-$ a $*$ a guľaté zátvorky, ktoré môžu byť vnorené ľubovolne veľakrát.
    - Rovnica môže okrem operácií a zátvoriek obsahovať pomenované aj nepomenované konštanty, premenné a nasledujúce funkcie:
      - rpcoor(premenná, prah1, prah2, hodnota1, hodnota2), rmcoor(premenná, prah1, prah2, hodnota1, hodnota2),
      - rp(premenná, prah1, prah2, parameter1, parameter2), rm(premenná, prah1, prah2, parameter1, parameter2),
      - hp(premenná, prah, parameter1, parameter2), hm(premenná, prah, parameter1, parameter2),
      - sp(premenná, faktor, prah, parameter1, parameter2), sm(premenná, faktor, prah, parameter1, parameter2),
      - spinv(premenná, faktor, prah, parameter1, parameter2), sminv(premenná, faktor, prah, parameter1, parameter2),
      - hillp(premenná, prah, faktor, parameter1, parameter2), hillm(premenná, prah, faktor, parameter1, parameter2),
    - Úvodné znaky (r, h, s) môžu byť zadané aj ako veľké písmená.
    - Znak (p) znamená (+) v zmysle rastúcej funkcie a znak (m) znamená (–) v zmysle klesajúcej funkcie.
    - Premenná musí byť jedna zo skôr uvedených premenných v riadku (VARS).
    - Hodnoty (prah, faktor, hodnota, parameter) môžu byť zadané ako pomenované alebo nepomenované konštanty.
    - Prah musí byť jednou z prahových hodnôt premennej.

- **Nie je možné násobiť spolu rovnaké premenné ani premennú s regulačnými funkciami závislými od tejto premennej, ani samotné regulačné funkcie závislé na rovnakej premennej.** Bolo by to porušenie pravidla o stupni premennej z kapitoly 2.A.

4.2 Vstupný súbor s vlastnosťou

Súbor s vlastnosťou alebo vlastnosťami je detailne popísaný v diplomovej práci M. Jaroša [13]. Tu si iba v krátkosti zhrnieme syntax. Súbor s príponou .ltl môže obsahovať nasledujúce riadky:

- #define *názov_premennej hodnota_premennej*
- #property *vlastnosť_vo_forme_ltl_formuly*

a platia pre ne nasledujúce pravidlá:

1. Súbor môže obsahovať ľubovolný počet definícií (riadok #define), mali by sa však odlišovať v názve premennej. Ten môže obsahovať malé písmená, čísla a znak _. Začínať ale musia buď písmenom alebo znakom _.

2. Hodnota premennej je v podstate výraz s logickou hodnotou (čiže atomická propozícia, AP (viď kapitolu 1.1)). Tá môže mať buď priamo hodnotu *true* alebo *false*, alebo môže byť tvorená reťazcom obsahujúcim meno nejakej premennej z modelu, relačný operátor (<, >, <=, >=) a číselnú hodnotu (celú alebo reálnu), ktorá sa musí nachádzať medzi prahovými hodnotami premennej z modelu.

3. Počet riadkov (#property) tiež nie je obmedzený, ale užívateľ by mal vedieť, že každý takýto riadok znamená novú vlastnosť, ktorá sa bude testovať na modeli, a to znamená výpočtovú réžiu.

4. Samotná formula *vlastnosť_vo_forme_ltl_formuly* môže byť tvorená nasledujúcimi hodnotami:

   - Unárnymi temporálnymi operátormi (**X**ϕ, **F**ϕ, **G**ϕ) a binárnymi temporálnymi operátormi (ϕ**R**ψ, ϕ**U**ψ)(viď kapitolu 1.1), kde ϕ a ψ sú výrazy s logickou hodnotou.
   - Výrazmi s logickou hodnotou, ktoré sú ďalej tvorené logickými operátormi (**!** (negácia), **&&** (konjunkcia), **||** (disjunkcia), **->** (implikácia), **<->** (ekvivalencia)) a premennými definovanými na začiatku súboru.

- Syntax podporuje aj ľubovolne vnorené guľaté zátvorky.

5. Pre úplnosť treba dodať, že táto syntax podporuje aj pár ďalsích, rozšírených operátorov a takmer ku každému ešte druhý možný zápis, ale náš výpočet je dostatočný pre zápis ľubovolnej LTL formuly.

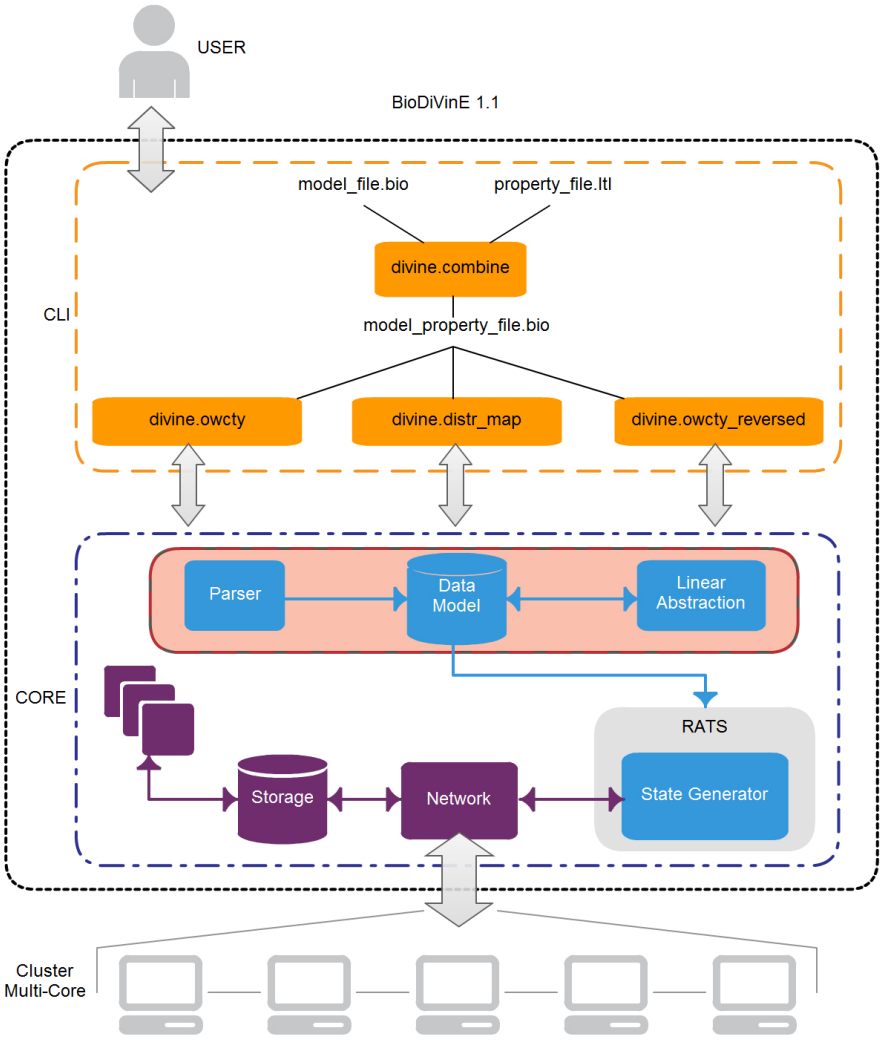
4.3 Priebeh procesu

Po zavolaní CLI spolu so zadanými vstupnými parametrami sa najskôr skontroluje ich počet. Ak je tento menší, program vypíše *help* ponuku (viď tabuľku v kapitole 6). V opačnom prípade skontroluje existenciu vstupných súborov a predá ich pomocnému nástroju *divine.combine*. Ten zistí počet vlastností a pre každú z nich vytvorí nový súbor a zároveň prevedie vlastnosť z LTL formuly na Büchiho automat. Tento výstupný súbor má tiež koncovku .bio, ale je iba pomocný a pred ukončením CLI bude zmazaný. Ďalej sa zistí požadovaný overovací algoritmus. Môže ísť o algoritmy *owcty*, *distr_map* alebo *owcty_reversed*. Žiadne iné nie sú prípustné.

Vybraný algoritmus sa spustí pre každý novovytvorený súbor .bio ako nástroj *divine.názov_algoritmu* a predajú sa mu prípadné parametre. Potom sa vstup predá triede s parserom a vytvorí sa datový model, ktorý prejde lineárnou aproximáciou (viď kapitolu 2.1.A), ak je to potrebné. Tým sa myslí prípad, keď rovnice modelu obsahujú aspoň jednu Hillovu funkciu alebo aspoň jednu sigmoidálnu funkciu. Heavisidove funkcie sa upravujú zvlášť hneď po načítaní, pretože ich možno previesť na jedinú rampu, a teda lineárna aproximácia z kapitoly 2.1.A pre ne nemá zmysel.

Prahové hodnoty vygenerované aproximáciou aj prevodom Heavisidových funkcií sa uložia medzi ostatné. To na jednu stranu zvyšuje výpočtovú réžiu, pretože sa zväčšuje stavový priestor generovaný nasledujúcou obdĺžnikovou aproximáciou, a to lineárne k počtu nových prahových hodnôt. No na strane druhej sa tým zlepšuje presnosť tejto aproximácie a celého výpočtu, pretože sa abstrahovaný model približuje skutočnému spojitému modelu.

Už spomenutá obdĺžniková aproximácia vygeneruje RATS (viď kapitolu 2.1.B), čiže diskrétny systém pôvodne spojitého modelu, ktorý sa overuje proti vlastnosti v podobe automatu. Ak ju RATS spĺňa, potom možno s istotou tvrdiť, že aj pôvodný model túto vlastnosť spĺňa. Naopak, ak je vygenerovaný protipríklad, nemusí vôbec zodpovedať protipríkladu v skutočnom spojitom modeli. Avšak stále platí, že model, či už abstrahovaný alebo skutočný, danú vlastnosť nespĺňa. [4]



Obr. 4.1: Diagram architektúry nástroja BioDiVinE 1.1. V červenej oblasti je zhrnutý náš prínos.

Kapitola 5

Implementácia

Na implementáciu boli použité programy *flexc++*, *bisonc++* a programovací jazyk c++ vo verzii c++11. Preto je pre správne nainštalovanie aplikácie dôležité mať prekladač *gcc* verzie 4.7.3 alebo novší. Z historických dôvodov je pre správne preloženie nástroja potrebné mať aj nasledujúce nástroje či programy:

- MPI - verzia 1.2 alebo novšia
- Automake - verzia 1.10.1 alebo novšia
- Autoconf - verzia 2.61

[7]

5.1 Parser

Na úvod je treba povedať, že za výrazom parser sa ukrýva aj druhý pomocný nástroj tzv. scanner. Samotný parser je nástroj na syntaktickú analýzu vstupu a v niektorých jednoduchých príkladoch môže byť dostatočný, ale nie v našom. Aby bolo parsovanie úspešné musí mu predchádzať ešte lexikálna analýza a tú má na starosti práve scanner, ktorý zjednodušene povedané načítava "slová". Tie potom posiela parseru, ktorý z nich skladá "vety", respektíve vzory. Len čo je nejaký vzor rozpoznaný, parser vykoná príslušné inštrukcie nad spracovávanými dátami.

Pre vytvorenie parseru a scanneru sú k dispozícii voľne dostupné sesterské programy. Najstaršie z nich sú *flex* a *bison*. Tie sa však ukázali byť až príliš zastarané a neschopné vytvoriť jednoduchú štruktúru typu trieda, ktorá by sa dala ľahko začleniť do už existujúceho kódu. Ďalšou možnosťou, ktorú sme zvažovali, boli programy *flex++* a *bison++*. Tie údajne podporujú triedy, ale nájsť k nim použiteľný užívateľský manuál sa ukázalo byť nemožné. Naopak programy *flexc++* a *bisonc++* sú prehľadne zdokumentované a veľmi dobre začleniteľné do zdrojového kódu v jazyku c++, preto sa ukázali byť tou najlepšou voľbou.

Obidva programy generujú niekoľko súborov pre vlastnú potrebu, ktoré nemá cenu meniť, pretože po každom spustení programu sa prepíšu. Sú to súbory *ParserBase.h*, *parse.cc* pre program *bisonc++* a súbory *ScannerBase.h* a *lex.cc* pre program *flexc++*. Hlavné súbory určené pre programátora sú bez prípony a môžu byť nazvané ľubovoľne. Tie naše sa volajú *parser2* a *scanner2*. Ďalšie užitočné a modifikovateľné súbory sú *Parser.h*, *Parser.ih*, *Scanner.h* a *Scanner.ih*, ktoré dodávajú potrebnú modularitu. To umožňuje narábať s parserom a scannerom ako s klasickými triedami, čo bolo v našom prípade veľmi žiadané.

Už na prvý pohľad vidno podľa názvov, že ide o kompatibilné programy vyvíjané rovnakým tímom, ktorý navyše ponúka veľmi dobrý užívateľský návod (viď [11] [10]).

5.2 Dátový model

Nový dátový model sa skladá z troch tried. Sú to *Entite*, *Summember* a *Model*. Počas procesu parsovania vstupného modelu sa jednotlivé dáta ukladajú do príslušných štruktúr inštancie triedy *Model*. Nazvime ju *dataModel*. Dátami rozumieme užívateľom špecifikované údaje na príslušných riadkoch podľa popisu v kapitole 4.1. Jedinou výnimkou sú rovnice čiže riadky začínajúce nápisom "EQ:". Keďže nechceme užívateľa obmedzovať striktne zadanou definíciou pre písanie rovníc, ale naopak chceme mu umožniť, aby mohol písať obecné rovnice a ľubovoľne pritom zátvorkovať, musíme nutne rovnice pred uložením upraviť, aby dostali podobu podľa definície v kapitole 2.B. K tomu slúži trieda *Entite* s pomocou triedy *Summember*.

Parsovanie rovníc prebieha slovo za slovom, kde pod pojmom slovo rozumieme skupinu znakov rozpoznaných scannerom. Takýmto slovom v rovnici môže byť číslo, premenná, konštanta, regulačné funkcie (viď kapitolu 2), ale aj guľaté zátvorky a znamienka $+$, $-$ a $*$ (čítaj krát). Prioritou je odstránenie zátvoriek a správne vyhodnotenie znamienok, aby všetky členy rovnice boli na jednej úrovni. Tento člen predstavuje v podstate inštanciu triedy *Summember*. Induktívne, jeden člen môže byť zložený z násobku viacerých členov:

$$clen1 \;=\; clen1.1 * clen1.2 * clen1.3 \ldots \tag{5.1}$$

Inými slovami, každé číslo, reťazec znakov (názov premennej alebo konštanty, nie však funkcia) alebo regulačnú funkciu považujeme za člen. Každý násobok týchto členov považujeme tiež za člen. Cieľom preto je dostať

rovnicu do tvaru sčitovania alebo odčitovania jednotlivých členov:

$$rovnica \;=\; clen1 + clen2 - clen3 + clen4 \ldots \tag{5.2}$$

Dokonca veľmi jednoducho sa možno zbaviť aj odčitovania jednoduchým prevodom:

$$clen2 - clen3 \;=\; clen2 + (-1)clen3, \tag{5.3}$$

takže vo výsledku dostávame:

$$rovnica \;=\; clen1 + clen2 + (-1)clen3 + clen4 \ldots \tag{5.4}$$

Pre potrebu úpravy pri parsovaní ukladáme každý jednotlivý člen ako inštanciu triedy *Entite* do zásobníka. Nazvime ju entita. Obsahuje vektor členov čiže inštancií triedy *Summember*. Potom pri načítaní znamienka krát ($*$) sa rekurzívne z dvoch posledných načítaných entít v zásobníku (každá obsahujúca jeden člen) stane entita jedna (obsahujúca jeden člen tvorený násobkom členov z pôvodných entít). Dve posledné inštancie zahodíme a navrch zásobníku uložíme túto novú.

Pri načítaní znamienka plus ($+$) sa rekurzívne z dvoch posledných entít v zásobníku (každá obsahujúca jeden člen) stane jedna nová entita, tentokrát však obsahujúca členy dva. Staré entity sa zahodia, nová sa opäť uloží. Obecne možno povedať, že pri sčitovaní má výsledná entita toľko členov, koľko mali členov dohromady pôvodné entity. Pri načítaní znamienka mínus ($-$) sa najskôr členy druhej entity prevedú na záporné a potom sa pokračuje rovnako ako pri sčitovaní.

Špeciálne treba spomenúť prípad, keď entity pri násobení obsahujú každá niekoľko členov. Výsledkom bude entita jedna obsahujúca toľko členov, koľko tvorí násobok počtu členov pôvodných entít. Toto všetko je regulérne v zmysle rozšírenia bežnej operácie roznásobenia aj pre nečíselné členy.

Vďaka tomuto post-parsovaciemu procesu konečne dostávame rovnicu v požadovanom tvare a môžeme ju uložiť ako vektor členov do dataModelu.

5.3 Lineárna abstrakcia nelineárnych funkcií

Keď sú všetky dáta v nami požadovanom formáte, je možné vykonať lineárnu aproximáciu všetkých nelineárnych funkcií (viď kapitolu 2.1.A) dátového modelu (viď kapitolu 5.2).

Ešte predtým ale spomeňme Heavisidove alebo schodové funkcie [26], ktoré sú samostatnou kategóriou. Keďže sa jedná o nespojité funkcie, je potrebné ich niečím nahradiť. Rampové funkcie (viď kapitolu 2.C) sa zdajú byť vhodnou náhradou. Počas parsovania ešte pred uložením do dátového modelu sú schodové funkcie prevedené priamo na rampy, a to v pomere jedna ku jednej. Práve z tohto dôvodu, že stačí nahradiť každú Heavisidovu funkciu jednou rampovou funkciou, nie je nutné zahŕňať ich do celého procesu lineárnej aproximácie, kde sa každá krivka nahradí niekoľkými (minimálne dvoma) rampami. Napriek tomu sa aj tu jedná o lineárnu abstrakciu, avšak odlišného typu.

Schodové funkcie sú definované tak, že prechod medzi funkčnými hodnotami funkcie je v zadanom bode, prahu. Problém ale nastáva s funkčnou hodnotou v tomto konkrétnom bode. Aby sme z nespojitej funkcie urobili spojitú, vytvoríme dva nové prahy. Jeden napravo a druhý naľavo od pôvodného prahu, oba v nepatrnej vzdialenosti. V závislosti od toho, či je to funkcia rastúca alebo klesajúca, sa spoja správne prahové hodnoty so správnymi funkčnými hodnotami a prechod medzi nimi je spojitý a zároveň lineárny. Nové prahové hodnoty sa samozrejme uložia medzi ostatné, aby nechýbali pri obdĺžnikovej abstrakcii (viď kapitolu 2.1.B).

Lineárnu abstrakciu vykonávame vždy pre všetky nelineárne funkcie (krivky) závislé od rovnakej premennej. To znamená, že sa vykoná toľkokrát, koľko máme v modeli premenných. Ak na nejakej premennej nie sú závislé žiadne nelineárne funkcie, tak sa preskočí.

Následne je na vybrané krivky aplikovaná abstrakcia podľa postupu v kapitole 2.1.A. Takto dostávame z pôvodnej množiny kriviek množinu množín rámp, kde každá množina rámp nahrádza v rovnici krivku. Tento proces je iteratívny a čiastočne rekurzívny. Pretože krivka sa musí nahradiť sumou nových rampových funkcií. To ale znamená, že z pôvodného člena rovnice (viď kapitolu 5.2), obsahujúceho nahrádzanú krivku, sa stane viacero nových duplicitných členov, každý však s inou nahrádzajúcou rampou. Nesmieme zabudnúť pôvodný člen zmazať. A rekurzívny preto, lebo je možné, že v pôvodnom člene rovnice bol násobok hneď niekoľkých kriviek. V takom prípade treba nahradiť všetky duplicitné výskyty ďalšej nahrádzanej krivky v nových, rampu obsahujúcich, duplicitných členoch. Príklad pre zjednodušenie, ak máme člen rovnice, v ktorom je násobok troch rôznych kriviek a každá bude nahradená piatimi rampami, tak po skončení lineárnej abstrakcie bude rovnica miesto jednoho obsahovať členov hneď stodvadsaťpäť.

Vidíme, že tento proces podstatne zväčšuje rovnice, a teda aj stažuje ich evaluáciu. V skutočnosti je toto z hľadiska výpočtového zanedbateľné.

Podstatnejšie je, že touto aproximáciou vzniknú nové prahové hodnoty, ktoré zväčšujú počet stavov produkovaných následnou obdĺžnikovou abstrakciou. Tú tu podrobne popisovať nebudeme, pretože nie je produktom našej implementácie.

5.4 CLI

CLI alebo konzolové užívateľské prostredie (z angl. *Command-line interface*) slúži ako obal pre nástroj BioDiVinE 1.1. Pretože tak, ako aj pôvodný BioDiVinE 1.0, je aj nová verzia vlastne sada nástrojov a my sme chceli užívateľovi poskytnúť jeden schopný nástroj. Nikomu nebránime používať samostatne tieto nástroje, ale chceme dať aj možnosť vyhnúť sa im.

CLI volá nástroje *divine.combine*, *divine.owcty*, *divine.owcty_reversed* a *divine.distr_map* pomocou systémového volania exec(2) a čaká na ich dokončenie. Samotný nástroj *divine.combine* je vlastne shellový skript volajúci ďalší nástroj pre prevod LTL formuly (viď kapitolu 1.1) do Büchiho automatu (viď kapitolu 1.2) *divine.ltl2ba*.

Kapitola 6

Použitie programu

CLI sa volá z príkazového riadku spolu s minimálne tromi vstupnými parametrami. Ak sa nezadá správny počet parametrov alebo je prvým parametrom `-h`, resp. `--help`, vyvolá sa help ponuka čiže návod na použitie CLI.

Prvým parametrom je názov algoritmu, ktorý sa použije na overenie modelu. Na výber sú *owcty*, *distr_map* a *owcty_reversed*. Tieto zodpovedajú nástrojom rovnakých mien s prefixom *divine.*, ako bolo povedané v kapitole 5.4.

Ďalšie parametre môžu byť voliteľné prepínače určené vybranému overovaciemu algoritmu. CLI ich jednoducho prevezme a následne predá na vstup volanému overovaciemu nástroju. Môže ísť o nasledujúce prepínače:

- Prepínače spoločné pre všetky algoritmy:

-v,	--version	zobrazí verziu vybraného algoritmu
-h,	--help	zobrazí help ponuku vybraného algoritmu
-q,	--quiet	zapína tichý mód
-c,	--statelist	vytvorí .ce_states súbor s protipríkladom
-t,	--trail	vytvorí .trail súbor s protipríkladom
-r,	--report	vytvorí .report súbor s údajmi o behu procesu
-L,	--log	zapne zaznamenávanie protokolu
-H x,	--htsize x	nastaví veľkosť hash tabuľky takto $x < 33$? tak 2^x inak x
-X w,		nastaví meno vytváraných súborov na w (w.trail, w.report, w.ce_states, ...)
-V,		zobrazí dodatočné informácie

- Prepínače navyše pre algoritmus *owcty*:

-C x,	--compress x	nastaví metódu kompresie stavov (0 = žiadna, 1 = Huffman)
-s,	--simple	prepne na testovanie dosiahnuteľnosti akceptujúceho stavu

- Prepínače navyše pre algoritmus *owcty_reversed*:

-R,	--remove	odstraňuje prechody (veľmi pomalé, šetrí pamäť)
-s,	--simple	prepne na testovanie dosiahnuteľnosti akceptujúceho stavu

Predposledným parametrom je cesta k súboru obsahujúcemu dynamický biochemický model. Popis k nemu možno nájsť v kapitole 4.1. Povinnosťou súboru je prípona *.bio*.

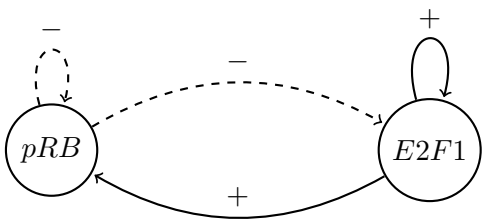
Posledným vstupným parametrom je cesta k súboru obsahujúcemu testovanú vlastnosť vo forme LTL formuly (viď kapitolu 1.1). V skutočnosti môže týchto vlastností obsahovať niekoľko. Všetky sa postupne overujú voči modelu. Súbor musí mať koncovku *.ltl* a detailnejšie je popísaný v kapitole 4.2.

V prípade, že by užívateľ chcel alebo potreboval použiť niektorý z overovacích nástrojov samostatne, dokladáme tieto dodatočné informácie. Potom, čo si užívateľ zvolí model a vlastnosť, čiže dva separátne súbory, musí zavolať nástroj *divine.combine* a tieto súbory mu predať ako prvý (model) a druhý (vlastnosť) parameter. V závislosti od počtu vlastností v súbore s príponou *.ltl* sa vytvorí rovnaký počet nových súborov *názov_modelu.propX.bio*, kde *X* je poradové číslo vlastnosti. Následne už len stačí vybranému overovaciemu nástroju predať jeden z vygenerovaných súborov ako vstupný parameter. Prepínače v tabuľke umiestnenej vyššie sú rovnako použiteľné.

Kapitola 7

Prípadová štúdia

Pre krátku ukážku nástroja BioDiVinE 1.1 sme si zvolili preskúmať prechod medzi fázami bunkového cyklu. Konkrétne medzi fázami G_1 (zodpovedá za rast bunky do pôvodnej veľkosti a produkciu proteínov k tomu určených) a S (zodpovedá za syntézu DNA a je prvou fázou budúceho delenia) a označujeme ho ako fázový prechod G_1/S. U mnohobunkových organizmov a u cicavcov špeciálne je prechod medzi týmito fázami riadený zložitou sieťou navzájom sa regulujúcich proteínov. Bližšie informácie k tejto sieti možno nájsť v článku [24]. My sa zameriame na jeden konkrétny motív tvoriaci jadro tejto siete. Ide o vzájomnú reguláciu nádorového supresora pRB a transkripčného faktora $E2F1$ (viď Obr. 7.1). Napriek tomu, že je tento model pomerne malý, ukazuje sa, že má kľúčovú úlohu.



Obr. 7.1: Jadro regulačnej siete fázového prechodu G_1/S bunkového delenia cicavcov

Prepis tohto modelu pomocou Hillovej kinetiky vyzerá:

$$\frac{d[pRB]}{dt} = k_1 \frac{[E2F1]}{K_{m_1} + [E2F1]} \frac{K_{m_2}}{K_{m_2} + [pRB]} - \gamma_{pRB}[pRB], \tag{7.1}$$

$$\frac{d[E2F1]}{dt} = k_p + k_2 \frac{a^2 + [E2F1]^2}{K_{m_3}^2 + [E2F1]^2} \frac{K_{m_4}}{K_{m_4} + [pRB]} - \gamma_{E2F1}[E2F1]. \tag{7.2}$$

Hodnoty všetkých konštánt sú zvolené tak, aby odrážali experimentálne

chovanie. Pochádzajú z článku [24] a vyzerajú nasledovne:

$$K_{m_1} = 0.5, \quad K_{m_2} = 0.5, \quad K_{m_3} = 4, \quad K_{m_4} = 5, \tag{7.3}$$

$$k_1 = 1, \quad k_2 = 1.6, \quad k_p = 0.05, \quad a = 0.04, \tag{7.4}$$

$$\gamma_{pRB} = 0.005, \quad \gamma_{E2F1} = 0.1. \tag{7.5}$$

Tento model je zaujímavý tým, že vykazuje známky bistability, k čomu mu pomáha fakt, že faktor $E2F1$ je pozitívne autoregulovaný. Pokúsime sa toto tvrdenie otestovať na našom nástroji. Vlastnosti, ktoré by mali zodpovedať tomuto chovaniu, sú prebrané z článku [1] a nachádzajú sa v prílohách B, C a D. Samotný model je popísaný v prílohe A. Vo všetkých prípadoch budeme volať algoritmus *distr_map* s parametrami *-Srt*.

- Vlastnosť číslo jedna $(G(E2F1 < 2))$ (viď príloha B), ktorá mala dokazovať, že ak počiatočná koncentrácia faktora $E2F1$ bude nízka, tak nízka aj zostane, sa nám podarilo vyvrátiť. Protipríklad (viď prílohu B.3) nám ukazuje, že počiatočná nízka koncentrácia pRB nestačí dostatočne potláčať produkciu $E2F1$, ktorá rastie k maximu. Môže to byť spôsobené nevhodnou kombináciou parametrov, pretože ich hodnoty sme prebrali z rozsiahlejšieho modelu v [24] a vlastnosti zase z modelu v [1] určeného pre testovanie parametrizácie v programe PEPMC (viď kapitolu 3.2). Výpočet trval vyše tridsať minút a podrobnú správu z priebehu procesu vidno v prílohe B.1. Okrem toho prikladáme aj časť výstupu programu s užitočnými informáciami (viď príloha B.2.

- Vlastnosť číslo dva $(G(E2F1 >= 2))$ (viď príloha C) je spĺňaná ukážkovým modelom. Z toho vyplýva, že keď má raz transkripčný faktor $E2F1$ vysokú koncentráciu, tak už do nízkych hodnôt neklesne. Pravdepodobne to vychádza zo silnej pozitívnej autoregulácie, ktorú nedokáže potlačiť ani inhibícia supresorom pRB. Iniciálna koncentrácia $E2F1$ bola nastavená na interval $\langle 4, 6\rangle$. Výpočet trval len necelých desať minút, z čoho šesť a pol minúty trvala optimálna lineárna aproximácia (viď kapitolu 2.1.A). Podrobnejšiu správu možno nájsť v prílohe C.1 a časť výpisu programu v prílohe C.2.

- Tretia vlastnosť $((E2F1 < 2) \rightarrow FG(E2F1 >= 2))$ (viď prílohu D) mala dokázať, že aj keď bude mať transkripčný faktor $E2F1$ na počiatku nízku koncentráciu, dokáže prekonať hranicu (v tomto prípade 2) a natrvalo udržať svoju koncentráciu na vyššej hladine, než

je táto hranica. Ukázalo sa, že model túto vlastnosť spĺňa a dôvodom bude opäť pravdepodobne silná pozitívna autoregulácia a naopak slabá inhibícia. Výpočet trval vyše dvadsaťpäť minút, z čoho opäť šesť a štvrť minúty trvala lineárna aproximácia. Správa z priebehu procesu sa nachádza v prílohe D.1 a časť výstupu programu zase v prílohe D.2.

Na základe získaných údajov sa domnievame, že koncentrácia transkripčného faktora $E2F1$ rastie nad všetky medze. Aby sme si to mohli potvrdiť, rozhodli sme sa otestovať ďalšiu vlastnosť (viď prílohu E). Tá mala dokázať, že aj keď bude mať faktor $E2F1$ na počiatku nízku koncentráciu, rovnakú ako pri testovaní tretej vlastnosti, tak nakoniec dosiahne naozaj vysokej hodnoty (v tomto prípade až 8).

Po štyridsiatich minútach výpočtu sa ukázalo, že model naozaj túto vlastnosť spĺňa. A teda naša domnienka o bezmedznom raste vyzerá byť správna. Detailnejšia správa sa nachádza v prílohe E.1 a časť výpisu programu v prílohe E.2.

Kapitola 8

Záver

Cieľom práce bolo najskôr zjednodušiť pôvodný nástroj BioDiVinE 1.0 a následne ho rozšíriť o novú funkcionalitu. Zjednodušením máme na mysli refaktorizáciu, počas ktorej sa podarilo odstrániť nepotrebné triedy s prefixom *dve* (dvadsaťšesť), *bymoc* (sedem) spolu s jeho VM a testami (vyše ďalších tridsať zdrojových súborov), nejaké obecné triedy (tri) a odstránené bolo tiež GUI (dvadsaťtri súborov), zbytky nepoužívanej Java aplikácie a vyše dvadsať ďalších nepotrebných súborov.

Novou funkcionalitou mal byť nový obecnejší popis vstupného modelu, aby bolo možné overovať širšiu škálu modelov voči ich vlastnostiam. Celé to malo prehľadne spojiť konzolové užívateľské prostredie.

Všetko vyššie vymenované sa nám podarilo a zostáva iba dúfať, že aj úspešne. Na túto otázku by mohol nadviazať niekto ďalší tým, že by náš nástroj patrične otestoval na dostatočnom počte reálnych modelov, ktoré by pochádzali buď z experimentálne doložených štúdií alebo z podobne zameraných programov.

Literatúra

[1] J. Barnat, L. Brim, A. Krejčí, D. Šafránek, A. Streck, M. Vejnár, and T. Vejpustek. On parameter synthesis by parallel model checking. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 9(3):693–705. IEEE Computer Society Press, 2012.

[2] J. Barnat, L. Brim, I. Černá, S. Dražan, J. Fabriková, and D. Šafránek. On algorithmic analysis of transcriptional regulation by LTL model checking. *Theoretical Computer Science*, 410(33-34):3128–3148, 2009.

[3] J. Barnat, L. Brim, I. Černá, and P. Šimeček. DiVinE – The Distributed Verification Environment. In *Proceedings of 4th International Workshop on Parallel and Distributed Methods in verifiCation*, pages 89–94, 2005.

[4] J. Barnat, L. Brim, I. Černá, S. Dražan, J. Fabriková, J. Láník, H. Ma, and D. Šafránek. Biodivine: A framework for parallel analysis of biological models. *In Proceedings of 2nd International Workshop on Computational Models for Cell Processes (COMPMOD 2009)*, pages 31–45. EPTCS 6, 2009.

[5] J. Barnat, L. Brim, I. Černá, S. Dražan, J. Fabriková, and D. Šafránek. Computational analysis of large-scale multi-affine ode models. *In Proceedings of High performance computational systems Biology (HiBi 2009)*, pages 81–99. IEEE Computer Society Press, 2009.

[6] J. Barnat, L. Brim, D. Šafránek, and M. Vejnár. Parameter scanning by parallel model checking with applications in systems biology. *In Proceedings of High performance computational systems Biology (HiBi 2010)*, pages 95–104. IEEE Computer Society Press, 2010.

[7] J. Barnat, L. Brim, S. Dražan, J. Fabriková, and D. Šafránek. BioDiVinE 1.0, Tool Page [online]. Faculty of informatics Masaryk university, Brno, Czech republic, [cit. 2014-05-17]. Available at <`http://sybila.fi.muni.cz/tools/biodivine/v1/index`>.

[8] G. Batt and C. Belta. RoVerGeNe: A tool for the Robust Verification of Gene Networks, Tool Page [online], [cit. 2014-05-17]. Available at <http://iasi.bu.edu/~batt/rovergene/rovergene.htm>.

[9] G. Batt, C. Belta, and R. Weiss. Model checking liveness properties of genetic regulatory networks. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2007.

[10] F.B. Brokken. Bisonc++ (Version 4.09.01), User Guide [online]. Center for Information Technology, University of Groningen, The Netherlands, [cit. 2014-05-21]. Available at <http://bisoncpp.sourceforge.net/bisonc++.html>.

[11] F.B. Brokken, J.P. van Oosten, and R. Berendsen (until 0.5.3). Flexc++ (Version 2.01.00), User Guide [online]. University of Groningen, The Netherlands, [cit. 2014-05-21]. Available at <http://flexcpp.sourceforge.net/flexc++.html>.

[12] R. Grosu, G. Batt, F.H. Fenton, J. Glimm, C. Le Guernic, S.A. Smolka, and E. Bartocci. From cardiac cells to genetic regulatory networks. *In Proceedings of CAV'11, the 23rd International Conference on Computer Aided Verification, Lecture Notes in Computer Science*, 6806:396–411. Springer, 2011.

[13] M. Jaroš. Převod formulí ltl na automaty [online]. Master thesis, Faculty of informatics Masaryk university, Brno, Czech republic, 2004 [cit. 2014-05-19]. Available at <http://is.muni.cz/th/4017/fi_m/>.

[14] E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*, chapter 3. Temporal Logics, pages 27–33. MIT Press, 1999.

[15] E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*, chapter 9. Model Checking and Automata Theory, pages 121–140. MIT Press, 1999.

[16] E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*, chapter 1.3. The Process of Model Checking, pages 4–4. MIT Press, 1999.

[17] E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*, chapter 9.4. Translating LTL into Automata, pages 132–138. MIT Press, 1999.

[18] J.P. Keener and J. Sneyd. *Mathematical Physiology*, volume 8 of *Interdisciplinary Applied Mathematics*, chapter 1.1. The Law of Mass Action, pages 3–4. Springer-Verlag, 1998.

[19] J.P. Keener and J. Sneyd. *Mathematical Physiology*, volume 8 of *Interdisciplinary Applied Mathematics*, chapter 1.2. Enzyme kinetics, pages 5–16. Springer-Verlag, 1998.

[20] J.P. Keener and J. Sneyd. *Mathematical Physiology*, volume 8 of *Interdisciplinary Applied Mathematics*, chapter 1.4. Appendix: Math Background, pages 24–30. Springer-Verlag, 1998.

[21] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT) [online], [cit. 2014-05-17]. Available at <http://control.ee.ethz.ch/~mpt/>.

[22] J.C. Perez and E. Vidal. Optimum polygonal approximation of digitized curves. *Pattern Recognition Letters*, 15(8):743–750. Elsevier Science Inc., 1994.

[23] M. Snir, S.W. Otto, D.W. Walker, J. Dongarra, and S. Huss-Lederman. *MPI: The Complete Reference*. MIT Press, 1995.

[24] M. Swat, A. Kel, and H. Herzel. Bifurcation Analysis of the Regulatory Modules of the Mammalian G1/S Transition. *Bioinformatics*, 20(10):1506–1511, 2004.

[25] G. Teschl. *Ordinary Differential Equations and Dynamical Systems*, volume 140 of *Graduate Studies in Mathematics*, chapter 1. Introduction, pages 3–31. American Mathematical Society, 2012.

[26] E.W. Weisstein. Heaviside Step Function, From MathWorld–A Wolfram Web Resource [online], [cit. 2014-05-17]. Available at <http://mathworld.wolfram.com/HeavisideStepFunction.html>.

Dodatok A

Model

V skutočnom súbore musí byť jeden zápis na jednom riadku. Tu sme však pre úsporu miesta donútený toto pravidlo porušiť.

Obsah súboru `tcbb1.bio`:

```
VARS: pRB, E2F1

CONSTS: k1,1; y_pRB,0.005; kp,0.05; k2,1.6; y_E2F1,0.1;
a,0.04

VAR_POINTS: pRB: 1500, 5; E2F1: 1500, 5

EQ: pRB = k1*Hillp(E2F1,0.5,1,0,1)*Hillm(pRB,0.5,1,1,0)-
y_pRB*pRB

EQ: E2F1 = kp + k2*Hillm(pRB,5,1,1,0)*(a*a*0.0625*
Hillm(E2F1,4,2,1,0) + Hillp(E2F1,4,2,0,1)) - y_E2F1*
E2F1

THRES: pRB: 0, 1, 2, 6, 7, 8, 10, 3, 4, 5
THRES: E2F1: 0, 1, 2, 4, 6, 8, 10, 12, 15

INIT: pRB: 0, 1; E2F1: 1, 2

system async;
```

Dodatok B

Vlastnosť č. 1

Pre úplnosť treba dodať, že sme vykonali zmenu v súbore s modelom. Treba prepísať na riadku začínajúcom `INIT:` hodnoty pre E2F1 na hodnoty 0, 1.

Obsah súboru `prop1.ltl`:

```
#define a E2F1 < 2

#property G (a)
```

B.1 Správa

Obsah súboru `tcbb1.prop1.bio.report`:

```
File:examples/tcbb2010/tcbb1.bio.prop1.bio
Algorithm:distr_map -S -r -t
Problem:LTL MC
Nodes:1
Time:sum:1858.759542
Time:avg:1858.759542
Time:min:1858.759542
Time:max:1858.759542
Time:values:1858.759542
VMSize:sum:95.984375
VMSize:avg:95.984375
VMSize:min:95.984375
VMSize:max:95.984375
VMSize:values:95.984375
MsgSent:sum:53
```

```
MsgSent:avg:53
MsgSent:min:53
MsgSent:max:53
MsgSent:values:53
MsgSentUser:sum:0
MsgSentUser:avg:0
MsgSentUser:min:0
MsgSentUser:max:0
MsgSentUser:values:0
StatesStored:sum:309
StatesStored:avg:309
StatesStored:min:309
StatesStored:max:309
StatesStored:values:309
SuccsCalls:sum:489
SuccsCalls:avg:489
SuccsCalls:min:489
SuccsCalls:max:489
SuccsCalls:values:489
CrossTrans:sum:0
CrossTrans:avg:0
CrossTrans:min:0
CrossTrans:max:0
CrossTrans:values:0
InitTime:sum:406.196788
InitTime:avg:406.196788
InitTime:min:406.196788
InitTime:max:406.196788
InitTime:values:406.196788
States:sum:309
States:avg:309
States:min:309
States:max:309
States:values:309
Trans:sum:821
Trans:avg:821
Trans:min:821
Trans:max:821
Trans:values:821
CEGenerated:Yes
```

```
IsValid:No
ReporterCreated:Sun May 25 01:59:12 2014
Workstations:demon-1215B
```

B.2 Časť výstupu

Obsah súboru `tcbb1.prop1.bio.output`:

```
0: Computation init: 406.197 s
0: iteration nr. 1: shrinkA.size: 0
0: iteration nr. 2: shrinkA.size: 11
======================
Accepting cycle found!
======================
0: state size: 24
0: appendix size: 56
0: states generated: 309
0: hashtable size: 65536
0: get_succs called: 489
0: trans. relaxed: 1146
0: all memory used: 95.9844 MB
0: Computation done: 1858.76 s
0: -------------
0: Accepting cycle: YES
0: Number of iterations: 2
```

B.3 Protipríklad

Prvý stĺpec vyjadruje koncentráciu transkripčného faktora $E2F1$ a druhý koncentráciu nádorového supresora pRB. V zátvorkách sú udané čísla stavov. Tie však užívateľa zaujímať nemusia.

Obsah súboru `tcbb1.prop1.bio.trail`:

```
pre-initial
0.607071(1),0.516344(1)-PP:0
1(2),0.516344(1)-PP:0
```

```
2(3),0.516344(1)-PP:0
2.1014(4),0.516344(1)-PP:0
3(5),0.516344(1)-PP:0
4(6),0.516344(1)-PP:0
5(7),0.516344(1)-PP:0
6(8),0.516344(1)-PP:0
6.86458(9),0.516344(1)-PP:0
7(10),0.516344(1)-PP:0
8(11),0.516344(1)-PP:0
10(12),0.516344(1)-PP:0
16.0173(13),0.516344(1)-PP:0
16.0173(13),1(2)-PP:0
16.0173(13),1.62108(3)-PP:0
16.0173(13),2(4)-PP:1
16.0173(13),4(5)-PP:1
16.0173(13),5.23549(6)-PP:1
16.0173(13),6(7)-PP:1
16.0173(13),8(8)-PP:1
16.0173(13),9.42628(9)-PP:1
16.0173(13),10(10)-PP:1
16.0173(13),12(11)-PP:1
================
16.0173(13),15(12)-PP:1
```

Dodatok C

Vlastnosť č. 2

Pre úplnosť treba dodať, že druhá vlastnosť si vyžaduje zmenu súboru s modelom. Treba prepísať na riadku začínajúcom `INIT:` hodnoty pre E2F1 tak, aby menšia z nich bola väčšia ako 2.

Obsah súboru `prop2.ltl`:

```
#define b E2F1 >= 2

#property G (b)
```

C.1 Správa

Obsah súboru `tcbb1.prop2.bio.report`:

```
File:examples/tcbb2010/tcbb1.bio.prop2.bio
Algorithm:distr_map -S -r -t
Problem:LTL MC
Nodes:1
Time:sum:585.322444
Time:avg:585.322444
Time:min:585.322444
Time:max:585.322444
Time:values:585.322444
VMSize:sum:95.542969
VMSize:avg:95.542969
VMSize:min:95.542969
VMSize:max:95.542969
VMSize:values:95.542969
MsgSent:sum:6
```

```
MsgSent:avg:6
MsgSent:min:6
MsgSent:max:6
MsgSent:values:6
MsgSentUser:sum:0
MsgSentUser:avg:0
MsgSentUser:min:0
MsgSentUser:max:0
MsgSentUser:values:0
StatesStored:sum:113
StatesStored:avg:113
StatesStored:min:113
StatesStored:max:113
StatesStored:values:113
SuccsCalls:sum:113
SuccsCalls:avg:113
SuccsCalls:min:113
SuccsCalls:max:113
SuccsCalls:values:113
CrossTrans:sum:0
CrossTrans:avg:0
CrossTrans:min:0
CrossTrans:max:0
CrossTrans:values:0
InitTime:sum:378.099570
InitTime:avg:378.099570
InitTime:min:378.099570
InitTime:max:378.099570
InitTime:values:378.099570
States:sum:113
States:avg:113
States:min:113
States:max:113
States:values:113
Trans:sum:208
Trans:avg:208
Trans:min:208
Trans:max:208
Trans:values:208
CEGenerated:No
```

```
IsValid:Yes
ReporterCreated:Sun May 25 02:57:19 2014
Workstations:demon-1215B
```

C.2 Časť výstupu

Obsah súboru `tcbb1.prop2.bio.output`:

```
0: Computation init: 390.049 s
0: iteration nr. 1: shrinkA.size: 0
0: state size: 24
0: appendix size: 56
0: states generated: 113
0: hashtable size: 65536
0: get_succs called: 113
0: trans. relaxed: 208
0: all memory used: 95.543 MB
0: Computation done: 595.286 s
0: -------------
0: Accepting cycle: NO
0: Number of iterations: 1
```

Dodatok D

Vlastnosť č. 3

Obsah súboru `prop3.ltl`:

```
#define a E2F1 < 2
#define b E2F1 >= 2

#property ((a) -> F G(b))
```

D.1 Správa

Obsah súboru `tcbb1.prop3.bio.report`:

```
File:examples/tcbb2010/tcbb1.bio.prop1.bio
Algorithm:distr_map -S -r -t
Problem:LTL MC
Nodes:1
Time:sum:1557.317334
Time:avg:1557.317334
Time:min:1557.317334
Time:max:1557.317334
Time:values:1557.317334
VMSize:sum:95.527344
VMSize:avg:95.527344
VMSize:min:95.527344
VMSize:max:95.527344
VMSize:values:95.527344
MsgSent:sum:21
MsgSent:avg:21
MsgSent:min:21
MsgSent:max:21
```

```
MsgSent:values:21
MsgSentUser:sum:0
MsgSentUser:avg:0
MsgSentUser:min:0
MsgSentUser:max:0
MsgSentUser:values:0
StatesStored:sum:185
StatesStored:avg:185
StatesStored:min:185
StatesStored:max:185
StatesStored:values:185
SuccsCalls:sum:633
SuccsCalls:avg:633
SuccsCalls:min:633
SuccsCalls:max:633
SuccsCalls:values:633
CrossTrans:sum:0
CrossTrans:avg:0
CrossTrans:min:0
CrossTrans:max:0
CrossTrans:values:0
InitTime:sum:377.244623
InitTime:avg:377.244623
InitTime:min:377.244623
InitTime:max:377.244623
InitTime:values:377.244623
States:sum:185
States:avg:185
States:min:185
States:max:185
States:values:185
Trans:sum:427
Trans:avg:427
Trans:min:427
Trans:max:427
Trans:values:427
CEGenerated:No
IsValid:Yes
ReporterCreated:Sun May 25 03:48:27 2014
Workstations:demon-1215B
```

D.2 Časť výstupu

Obsah súboru `tcbb1.prop3.bio.output`:

```
0: Computation init: 377.245 s
0: iteration nr. 1: shrinkA.size: 0
0: iteration nr. 2: shrinkA.size: 8
0: iteration nr. 3: shrinkA.size: 8
0: iteration nr. 4: shrinkA.size: 6
0: iteration nr. 5: shrinkA.size: 4
0: iteration nr. 6: shrinkA.size: 1
0: state size: 24
0: appendix size: 56
0: states generated: 185
0: hashtable size: 65536
0: get_succs called: 633
0: trans. relaxed: 1392
0: all memory used: 95.5273 MB
0: Computation done: 1557.32 s
0: -------------
0: Accepting cycle: NO
0: Number of iterations: 6
```

Dodatok E

Vlastnosť č. 4

Obsah súboru `prop4.ltl`:

```
#define a E2F1 > 8

#property FG (a)
```

E.1 Správa

Obsah súboru `tcbb1.prop4.bio.report`:

```
File:examples/tcbb2010/tcbb1.bio.prop1.bio
Algorithm:distr_map -S -r -t
Problem:LTL MC
Nodes:1
Time:sum:2559.146457
Time:avg:2559.146457
Time:min:2559.146457
Time:max:2559.146457
Time:values:2559.146457
VMSize:sum:95.527344
VMSize:avg:95.527344
VMSize:min:95.527344
VMSize:max:95.527344
VMSize:values:95.527344
MsgSent:sum:27
MsgSent:avg:27
MsgSent:min:27
MsgSent:max:27
MsgSent:values:27
```

```
MsgSentUser:sum:0
MsgSentUser:avg:0
MsgSentUser:min:0
MsgSentUser:max:0
MsgSentUser:values:0
StatesStored:sum:238
StatesStored:avg:238
StatesStored:min:238
StatesStored:max:238
StatesStored:values:238
SuccsCalls:sum:994
SuccsCalls:avg:994
SuccsCalls:min:994
SuccsCalls:max:994
SuccsCalls:values:994
CrossTrans:sum:0
CrossTrans:avg:0
CrossTrans:min:0
CrossTrans:max:0
CrossTrans:values:0
InitTime:sum:473.229497
InitTime:avg:473.229497
InitTime:min:473.229497
InitTime:max:473.229497
InitTime:values:473.229497
States:sum:238
States:avg:238
States:min:238
States:max:238
States:values:238
Trans:sum:743
Trans:avg:743
Trans:min:743
Trans:max:743
Trans:values:743
CEGenerated:No
IsValid:Yes
ReporterCreated:Fri May 23 14:19:57 2014
Workstations:demon-1215B
```

E.2 Časť výstupu

Obsah súboru `tcbb1.prop4.bio.output`:

```
0: Computation init: 473.23 s
0: iteration nr. 1: shrinkA.size: 0
0: iteration nr. 2: shrinkA.size: 12
0: iteration nr. 3: shrinkA.size: 25
0: iteration nr. 4: shrinkA.size: 20
0: iteration nr. 5: shrinkA.size: 17
0: iteration nr. 6: shrinkA.size: 7
0: iteration nr. 7: shrinkA.size: 1
0: iteration nr. 8: shrinkA.size: 1
0: state size: 24
0: appendix size: 56
0: states generated: 238
0: hashtable size: 65536
0: get_succs called: 994
0: trans. relaxed: 2998
0: all memory used: 95.5273 MB
0: Computation done: 2559.15 s
0: -------------
0: Accepting cycle: NO
0: Number of iterations: 8
```