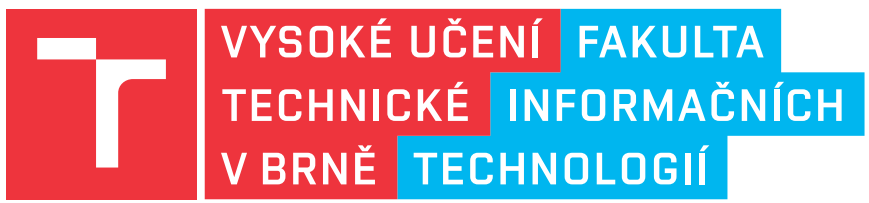
Analýza založená na modelech
Model checking

Adam Rogalewicz



únor 2025

MBA - Základní informace

Kontrolované aktivity:

- 4 projekty po 10 bodech.
- Závěrečná zkouška za 60b (minimum 25b).

Další aktivity:

- Přednáška - úterý 10:00. Délka 2 hodiny.
- Cvičení - 5x za semestr (kombinovaně numerické/počítačové)

Vyučující:

- doc. Rogalewicz, doc. Češka, dr. Fiedor, Ing. Andriushchenko

MBA - Předběžný plán semestru

Přednášky:

- 1. týden: Úvod do Model checking (doc. Rogalewicz)
- 2.-4. týden: Petriho sítě (doc. Rogalewicz/doc. Holík)
- 5.-7. týden: Časované automaty (doc. Rogalewicz)
- 8.-10. týden: Markovovy řetězce (doc. Češka/Ing. Andriushchenko)
- 11.-13. týden: UML/SysML (dr. Fiedor)

Cvičení a zadání projektů:

1. Petriho sítě (kombinované + zadání projektu)
2. Časované automaty (kombinované + zadání projektu)
3. Markovovy řetězce (numerické)
4. Markovovy řetězce (počítačové + zadání projektu)
5. Mathlab-Simulink (počítačové + zadání projektu)

Model-based design a analýza

- Klasický postup:
  Kód v programovacím jazyku $\rightarrow$ analýza a testování
- Model-based design:
  Model systému $\rightarrow$ Simulace
  $\rightarrow$ Testování (Model-Based testing)
  $\rightarrow$ Analýza (Model checking)
  $\rightarrow$ Generování kódu
  $\rightarrow$ ...

Možné modelovací jazyky a formalismy:

- Konečné automaty/přechodové systémy, vývojové diagramy (flow-charts), Petriho sítě, časované automaty, Markovovy řetězce, UML/SysML, Mathlab/Simulink, lossy chanel systems, procesní algebry, DEVS, ...

Model checking

Model checking je automatizovaná technika, která na základě konečně stavového modelu systému a formálně definované vlastnosti systematicky ověří, zdali tato vlastnost platí pro všechny dostupné stavy (resp. vybraný stav) tohoto modelu. (C. Baier and J.-P. Katoen: Principles of Model Checking)

- Vlastnosti popsány různými formalismy (množiny špatných/dobrých stavů, logiky, automaty, ...)
- Obvykle založeno na systematickém prohledávání stavového prostoru.
- 2007: E.M.Clarke, E.A.Emerson a J. Sifakis – Turingova cena za Model checking.
  ... for their role in developing Model-Checking into a highly effective verification technology that is widely adopted in the hardware and software industries.

Fáze model checkingu

1. Modelování:
   - Vytvoření modelu ve vybraném modelovacím jazyce
   - První testy s pomocí simulace
   - Formalizace požadované vlastnosti
2. Běh: Spuštění model checkeru pro ověření vlastnosti na modelu
3. Analýza výsledku:
   - Vlastnost splněna $\rightarrow$ pokračujeme analýzou další vlastnosti (je-li třeba)
   - Vlastnost porušena $\rightarrow$ analýza protipříkladu pomocí simulace $\rightarrow$ oprava modelu nebo definice vlastnosti.
   - Out of memory $\rightarrow$ snaha o zmenšení modelu.

Model checking lze zobecnit na nekonečně stavové modely

- Obecně nerozhodnutelné (např. Halting problem pro TS)
- Rozhodnutelné pro některé modelovací jazyky a některé typy vlastností
- Využití řady heuristik

Typy model checkingu:

- Explicitní model checking
  - Přímá práce se stavy modelu
  - Škáluje do cca 10^9 stavů
- Symbolický model checking[1]
  - Pracuje se s množinami stavů reprezentovanými vhodnými formalismy (logiky, automaty, intervaly, upward-closed množiny, ...)
  - Krok výpočtu se provádí symbolicky pro všechny stavy z (nekonečné) množiny
  - Akcelerace výpočtu (provedení nekonečného množství stejných kroků v rámci jednoho kroku výpočtu)
- Omezený model checking
  - Prohledání všech stavů do omezené hloubky od počáteční konfigurace
  - Motivace: velké množství chyb se projeví poměrně brzy od počátku výpočtu
  - Negarantuje korektnost modelu

[1] Poznámka: Strom pokrytí P/T Petriho sítí je technikou symbolického model checkingu.

Typy vlastností:

- Bezpečnost (safety)
  - Něco se nesmí nikdy stát
  - Deadlock, segmentation fault, dva procesy zároveň v kritické sekci
  - Porušení vlastnosti $\rightarrow$ konečný protipříklad
  - Často lze popsat množinou špatných stavů, nebo pomocí temporálních logik.
- Živost (liveness)
  - Něco se musí vždy stát
  - Program vždy zastaví, nedochází ke stárnutí, ...
  - Porušení vlastnosti $\rightarrow$ nekonečný protipříklad
  - Často popisováno pomocí temporálních logik (LTL, CTL, CTL*, TCTL)
  - Model checking těchto vlastností často mnohem obtížnější

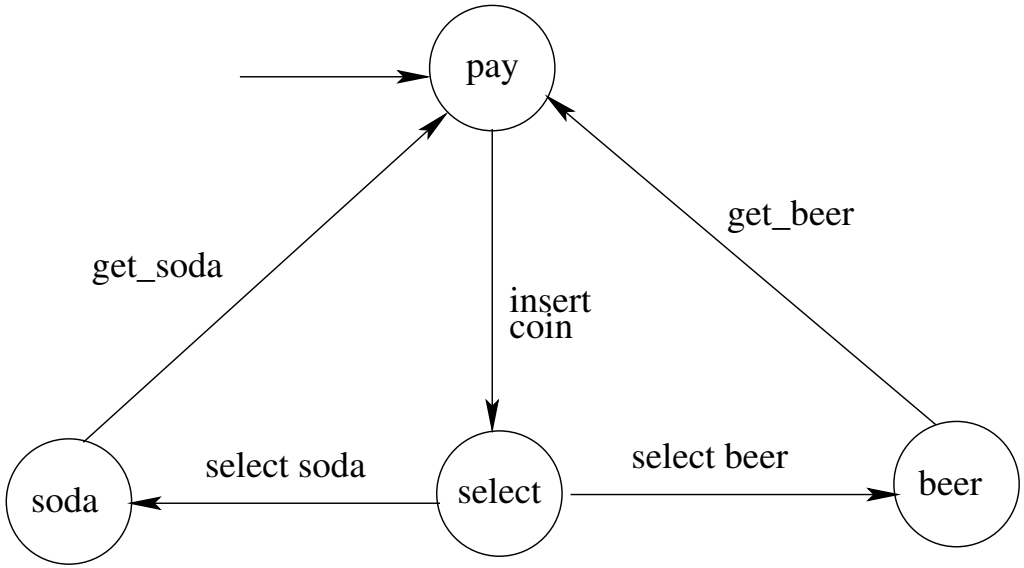
Přechodové systémy

Definition 1 (Přechodový systém)

Přechodový systém je n-tice $P = (S, Act, \rightarrow, I, AP, L)$, kde

- S je množina stavů
- Act je množina akcí
- $\rightarrow \subseteq S \times Act \times S$ je přechodová relace
- $I \subseteq S$ je množina počátečních stavů
- AP je množina atomických výroků
- $L : S \rightarrow 2^{AP}$ je ohodnocení stavů

- P je konečný, pokud množiny S, Act a AP jsou konečné.
- Konečný běh systému je sekvence $s_0 \alpha_1 s_1 \alpha_2 \ldots \alpha_n s_n$ taková, že $s_i \overset{\alpha_{i+1}}{\rightarrow} s_{i+1}$ pro každé $0 \leq i < n$.
- Nekonečný běh je sekvence $s_0 \alpha_1 s_1 \alpha_2 s_2 \alpha_3 \ldots$ taková, že $s_i \overset{\alpha_{i+1}}{\rightarrow} s_{i+1}$ pro každé $0 \leq i$.
- Běh je počáteční, pokud $s_0 \in I$.



$S = \{pay, select, soda, beer\}$, $I = \{pay\}$,
$Act = \{inser\_coin, get\_soda, get\_beer, select\_beer, select\_soda\}$

- Podmínka 1: Automat vydá nápoj pouze po vložení mince.
- K ověření podmínky 1 poslouží predikáty $AP = \{paid, drink\}$ a funkce L: $L(pay) = \emptyset, L(select) = \{paid\}, L(soda) = L(beer) = \{paid, drink\}$
- Množina špatných stavů je $Bad = \{s \in S | drink \in L(s) \wedge paid \notin L(s)\}$[2]
- Množinu dostupných stavů $Reach = \{s \in S \mid pay\alpha_1 \ldots \alpha_k s$ je běh$\}$.
- $Bad \cap Reach = \emptyset \rightarrow$ podmínka platí.

[2]Množinu chybných běhů lze také formálně popsat LTL formulí $\phi_1 \equiv \Diamond(drink \wedge \neg paid)$
ϕ_1 říká, že v budoucnu budeme ve stavu, kde platí predikát *drink* a neplatí *paid*.
Více o logice LTL v předmětu SAV

Definition 2 (Přímý následník a předchůdce)

Nechť $P = (S, Act, \rightarrow, I, AP, L)$ je přechodový systém.

- $Post(s, \alpha) = \{s' \in S \mid s \xrightarrow{\alpha} s'\}$
- $Pre(s, \alpha) = \{s' \in S \mid s' \xrightarrow{\alpha} s\}$

Stav s nazýváme koncovým, pokud $\bigcup_{\alpha \in Act} Post(s, \alpha) = \emptyset$

Definition 3 (Deterministický přechodový systém)

Nechť $P = (S, Act, \rightarrow, I, AP, L)$ je přechodový systém.
P je deterministický vzhledem k akcím (action-deterministic), pokud

- $|I| \leq 1$
- Pro každé $s \in S$ a $\alpha \in Act$: $Post(s, \alpha) \leq 1$

P je AP-deterministický pokud

- $|I| \leq 1$
- Pro každé $s \in S$ a $A \in 2^{AP}$: $(Post(s, \alpha) \cap \{s' \mid L(s') = A\}) \leq 1$

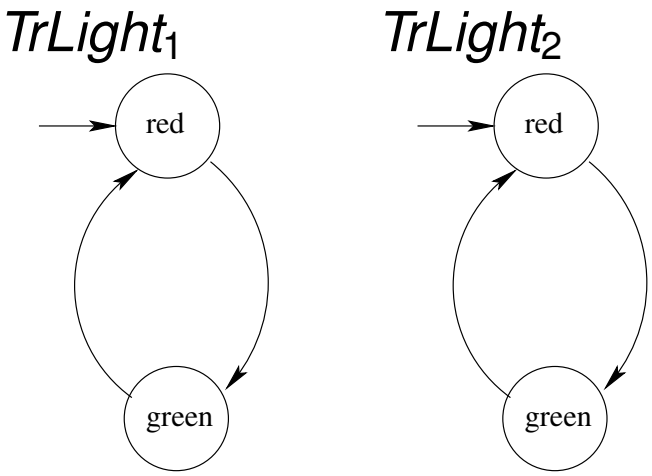
Proložení přechodových systémů

- Pro několik přechodových systémů chceme definovat jejich paralelní kompozici $P_1||P_2||\ldots||P_n$.
- Nejjednodušší způsob je tzv. proložení systémů, kdy systémy fungují vedle sebe nezávisle na sobě.

Definition 4 (Proložení přechodových systémů)

Nechť $P_1 = (S_1, Act_1, \rightarrow_1, I_1, AP_1, L_1)$ a $P_2 = (S_2, Act_2, \rightarrow_2, I_2, AP_2, L_2)$ jsou dva přechodové systémy. Jejich proložením $P_1|||P_2$ je přechodový systém definovaný následovně:
$P_1|||P_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$, kde

- $< s_1, s_2 > \overset{\alpha}{\rightarrow} < s_1', s_2' >$ pokud $s_1 \overset{\alpha}{\rightarrow} s_1'$ a $s_2 = s_2'$ nebo $s_1 = s_1'$ a $s_2 \overset{\alpha}{\rightarrow} s_2'$
- $L(< s_1, s_2 >) = L_1(s_1) \cup L_2(s_2)$

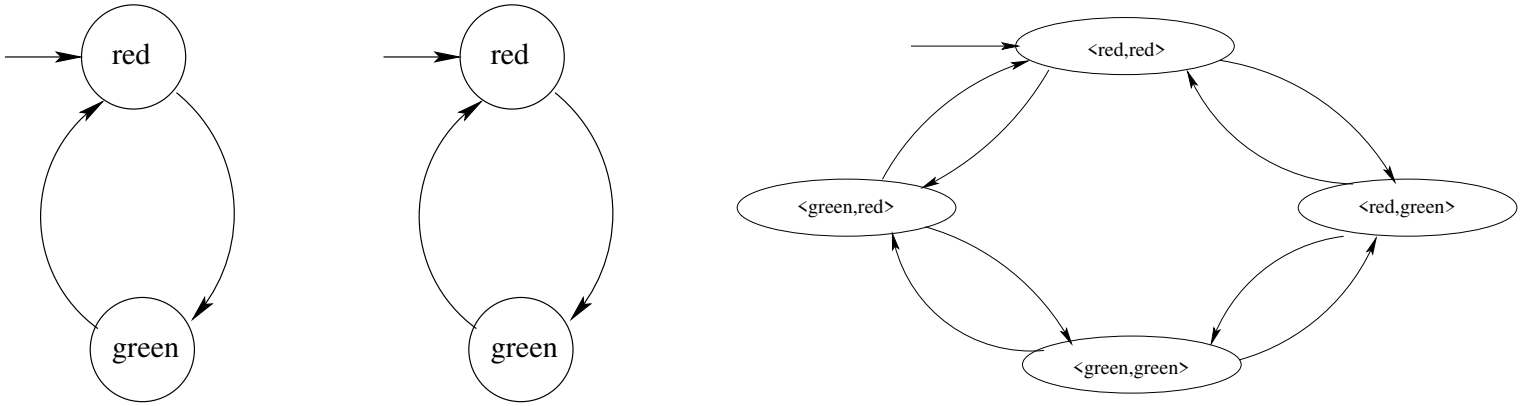


Podmínka 1: semafory nemohou být oba zároveň na volno

- Sestrojíme proložení $TrLight_1 ||| TrLight_2$
- Lze využít predikáty AP=$\{$bad$\}$ a $L(< green, green >) = \{Bad\}$ a pro $s \neq < green, green >$: $L(s) = \emptyset$
- Ověříme, že není dostupný stav označený predikátem *bad*[3]

[3] Chybné běhy popisuje LTL formule $\phi_1 \equiv \Diamond bad$

$TrLight_1$ $TrLight_2$ $TrLight_1 ||| TrLight_2$



Podmínka 1: semafory nemohou být oba zároveň na volno

- Sestrojíme proložení $TrLight_1 ||| TrLight_2$
- Lze využít predikáty AP={bad} a $L(< green, green >) = \{Bad\}$ a pro $s \neq < green, green >$: $L(s) = \emptyset$
- Ověříme, že není dostupný stav označený predikátem *bad*[3]

  $\rightarrow$ podmínka neplatí, nutno dodat synchronizaci mezi semafory.

[3] Chybné běhy popisuje LTL formule $\phi_1 \equiv \Diamond bad$

Handshake

Jednou z možností komunikace mezi paralelními komponentami je tzv. handshake, neboli synchronizace na společných akcích.

Definition 5 (Handshake)

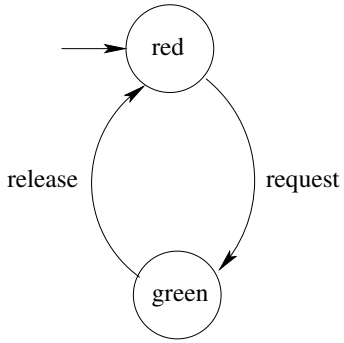
Nechť $P_1 = (S_1, Act_1, \rightarrow_1, I_1, AP_1, L_1)$ a $P_2 = (S_2, Act_2, \rightarrow_2, I_2, AP_2, L_2)$ jsou dva přechodové systémy a $H \subseteq Act_1 \cap Act_2$ je množina synchronizačních akcí. Přechodový systém $P_1||_H P_2$ je definovaný následovně: $P_1||_H P_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$, kde

- $\alpha \in H : < s_1, s_2 > \xrightarrow{\alpha} < s_1', s_2' >$ pokud $s_1 \xrightarrow{\alpha} s_1'$ a $s_2 \xrightarrow{\alpha} s_2'$
- $\alpha \notin H : < s_1, s_2 > \xrightarrow{\alpha} < s_1', s_2' >$ pokud $s_1 \xrightarrow{\alpha} s_1'$ a $s_2 = s_2'$ nebo $s_1 = s_1'$ a $s_2 \xrightarrow{\alpha} s_2'$
- $L(< s_1, s_2 >) = L_1(s_1) \cup L_2(s_2)$

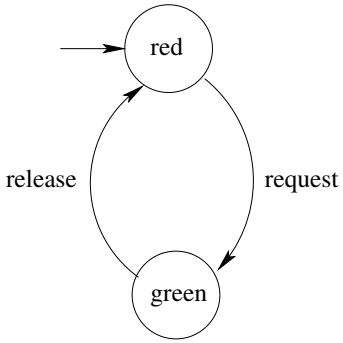
Poznámka: $P_1||_\emptyset P_2 = P_1|||P_2$

$P_1||P_2 = P_1||_{Act_1 \cap Act_2} P_2$

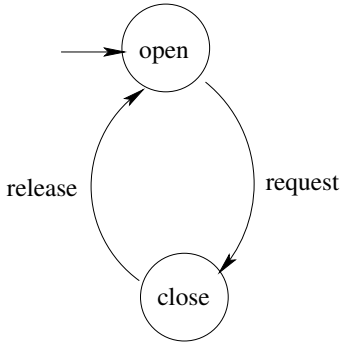
TrLight$_1$



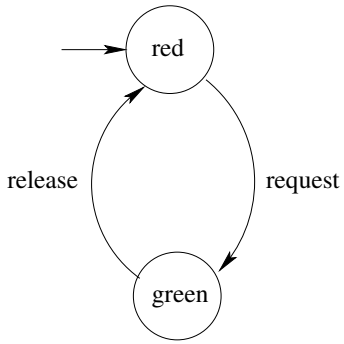
TrLight$_2$



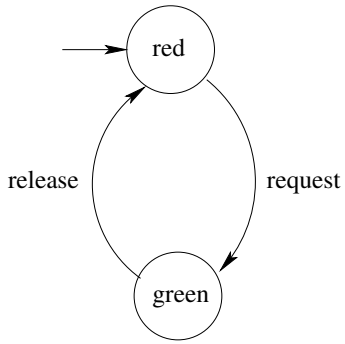
Arbiter



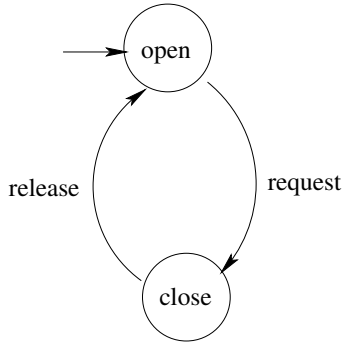
$TrLight_1$



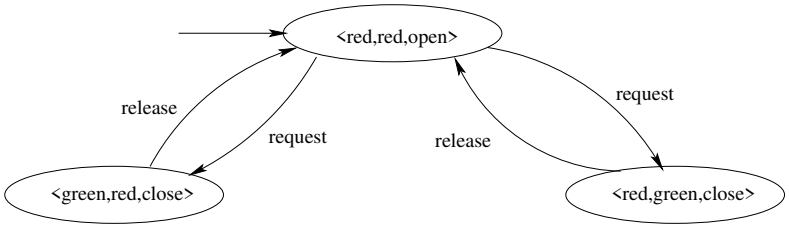
$TrLight_2$



Arbiter



$(TrLight_1 ||| TrLight_2) || Arbiter$



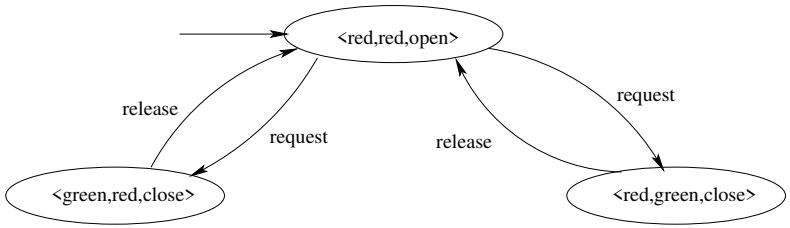
Podmínka 1: semafory nemohou být společně na volno:

- Ověříme, že stav $< green, green, \_ >$ není dostupný
  $\rightarrow$ podmínka platí

Podmínka 2: pokud jsou oba semafory na stůj, tak arbiter musí být ve stavu ”open”:

- Ověříme, že stav $< red, red, close >$ není dostupný
  $\rightarrow$ podmínka platí

$(TrLight_1 ||| TrLight_2)||Arbiter$

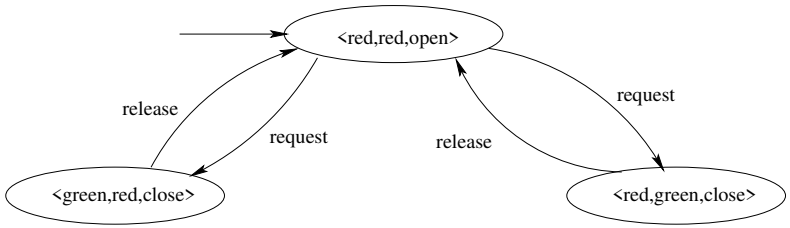


Podmínka 3: (fairness) V jakémkoliv stavu platí, že v budoucnu se na každém ze semaforů objeví ”volno”

- Nelze popsat množinou špatných stavů $\rightarrow$ nutno použít temporální logiku.
- Využijeme predikáty $AP = \{volno_1, volno_2\}$ a označení stavů:
  $L(< green, red, \_ >) = \{volno_1\}, L(< red, green, \_ >) = \{volno_2\}$,
  $L(green, green, \_) = \{volno_1, volno_2\}$, pro ostatní stavy $L(s) = \emptyset$
- Ověříme platnost LTL formule $\phi_3 \equiv (\Box\Diamond volno_1) \wedge (\Box\Diamond volno_2)$

[4] LTL model checking, viz předmět SAV.

$(TrLight_1 ||| TrLight_2) || Arbiter$



Podmínka 3: (fairness) V jakémkoliv stavu platí, že v budoucnu se na každém ze semaforů objeví ”volno”

- Nelze popsat množinou špatných stavů $\rightarrow$ nutno použít temporální logiku.
- Využijeme predikáty $AP = \{volno_1, volno_2\}$ a označení stavů:
  $L(< green, red, \_ >) = \{volno_1\}, L(< red, green, \_ >) = \{volno_2\}$,
  $L(green, green, \_) = \{volno_1, volno_2\}$, pro ostatní stavy $L(s) = \emptyset$
- Ověříme platnost LTL formule $\phi_3 \equiv (\Box\Diamond volno_1) \wedge (\Box\Diamond volno_2)$
  $\rightarrow$ podmínka neplatí, například existuje nekonečný běh,
  $(< red, red, open > \rightarrow < green, red, close > \rightarrow)^\omega$.
- Ověření platnosti LTL formule lze provést pomocí jejich automatizovaného převodu na automaty nad nekonečnými slovy, průnikem s modelem systému a testem na prázdnost. [4]

[4] LTL model checking, viz předmět SAV.

Exploze stavového prostoru

- Při vytváření modelu $P_1||P_2$ je počet stavů $|S| = |S_1|.|S_2|$
- Pro $P_1||\ldots||P_{10}$ kde každý z procesů má 10 stavů je stavový prostor 10^{10}
- Problém pro explicitní model checking konečně stavových systémů
- Obecně problém pro jakoukoliv analýzu a testování paralelních programů.
- Problém stavové exploze vzniká též při determinizaci (nejen) konečných automatů—$|S_{det}| = 2^{|S|}$.
- Petriho sítě: model který umožňuje efektivnější reprezentaci stavového prostoru a méně trpí na explozi stavového prostoru.

Jakákoliv verifikační metoda s využitím technik založených na modelech je pouze tak dobrá jak dobrý je vlastní model systému.

Poznámka: Toto platí pro jakoukoliv model-based techniku (simulace, testování, ...)