

# IV113 Validace a verifikace

## Formální verifikace

doc. RNDr. Jiří Barnat, Ph.D.

## Validace a Verifikace

- Jeden z obecných cílů V&V je prokázat správné chování produktu.

## Připomenutí

- Proces testování je neúplný.
- **Testováním dokážeme odhalit chybu, ale nedokážeme prokázat bezchybnost.**

## Závěr

- Je zapotřebí jiného způsobu verifikace systémů.

## **Cíl formální verifikace**

- Cílem je prokázat, že systém pracuje správně, takovým způsobem, aby míra důvěry ve výsledek procesu verifikace byla stejná, jako míra důvěry v matematický důkaz.

## **Metody formální verifikace v IV113**

- Dokazování (Theorem proving)
- Ověřování modelu (Model Checking)

## **Nutné podmínky pro realizaci**

- Přesně definovaná sémantika chování systému.
- Přesně formulované požadavky na zkoumaný systém.

## Verifikace sekvenčních programů

## Program je korektní pokud

- Pokud pro platný vstup **skončí** a vrátí **korektní** výsledek.
- Dokazují se dvě tvrzení: že je program parciálně korektní, a že výpočet programu vždy skončí.

## Parciální korektnost (Korektnost, Soundness)

- Pokud výpočet programu nad vstupními hodnotami, pro které je program definován, skončí, je výsledek výpočtu správný.

## Terminace (Úplnost, Konvergence, Completeness)

- Pro vstupní hodnoty, pro které je program definován, výpočet programu skončí.

## Sekvenční programy

- Vstup-výstupně uzavřené konečné programy.
  - Hodnoty vstupu jsou známy před vykonáním programu.
  - Výstup po skončení programu uložen ve výstupní proměnné.
- Quick sort, největší společný dělitel, . . .

## Princip verifikace

- Na programy a jednotlivé instrukce se nahlíží jako na transformátory stavů.
- Cílem je prokázat, že vstupní a výstupní hodnoty se k sobě mají v odpovídajícím vztahu.
- Tj. verifikovat korektnost postupu aplikovaného za účelem transformace vstupních proměnných na odpovídající výstupní proměnné.

## Stav výpočtu

- Stav výpočtu programu je jednoznačně určen lokací (hodnotou čítače instrukcí) a valuací proměnných.

## Atomické predikáty

- Základní tvrzení o jednotlivých stavech, jejichž pravdivost, lze určit ze stavu samotného, tj. dle konkrétní valuace.
- Příklady atomických propozic:  $(x == 0)$ ,  $(x1 \geq y3)$ , ...

## Popis množiny stavů

- Specifikovány boolovskou kombinací atomických predikátů
- Příklad:  $(x == m) \wedge (y > 0)$

## **Assertion**

- Množina povolených stavů svázaná s konkrétním místem (lokací) programu.
- Invariant lokace programu.

## **Assertions – dynamická kontrola**

- Ověřování invariantů v době běhu programu.
- Přesnější detekce místa chyby – konkrétní lokace.

## **Assertions – důkazy korektnosti**

- Přiřazení vlastností jednotlivým místům grafu toku řízení.
- Robert Floyd: Assigning Meanings to Programs (1967)

## Princip

- Programy = transformátory stavů.
- Specifikace = vztah mezi vstupním a výstupním stavem.

## Hoarova logika

- Navržena za účelem ukazování parc. korektnosti programů.
- Necht'  $P$  a  $Q$  jsou predikáty a  $S$  program, pak

$$\{P\} S \{Q\}$$

je tzv. *Hoarova trojice*.

## Zamýšlený význam trojice $\{P\} S \{Q\}$

- $S$  je program, který transformuje stav splňující *vstupní podmínku*  $P$  na stav, který splňuje *výstupní podmínku*  $Q$ .

## Příklad

- $\{z = 5\} x = z * 2 \{x > 0\}$
- Platná trojice, výstupní podmínka by mohla být přesnější.
- Silnější výstupní podmínka:  $\{x > 5 \wedge x < 20\}$ , zjevně platí, že  $\{x > 5 \wedge x < 20\} \implies \{x > 0\}$ .

## Nejslabší vstupní podmínka (weakest precondition)

- $P$  je **nejslabší vstupní podmínka**, pokud
- platí  $\{P\}S\{Q\}$  a zároveň
- $\forall P'$ , pro které platí  $\{P'\}S\{Q\}$ , platí  $P' \implies P$ .

## Postup dokazování $\{P\} S \{Q\}$

- Zvolíme vhodné podmínky  $P'$  a  $Q'$
- Dokazujeme  $\{P'\} S \{Q'\}$ ,  $P \implies P'$  a  $Q' \implies Q$ .
- V Hoarově důkazovém systému jsou axiomy a odvozovací pravidla, ze kterých se vytvářejí platné trojice pro strukturálně složitější programy.
- Pokud se podaří vyskládat transformaci  $P'$  na  $Q'$ , tak  $\{P'\} S \{Q'\}$  je platná trojice.
- $P \implies P'$  a  $Q' \implies Q$  **se dokazují běžným způsobem.**

## Axiom

- Axiom pro přiřazení:  $\{\phi[x \text{ nahrazeno } k]\} x := k \{\phi\}$

## Význam

- Trojice  $\{P\}x := y\{Q\}$  je axiomem v Hoarově dokazovacím systému, pokud platí, že  $P$  je shodné s  $Q$ , ve kterém byly všechny výskyty  $x$  nahrazeny výrazem  $y$ .

## Příklad

- $\{y+7>42\} x:=y+7 \{x>42\}$  je axiom
- $\{r=2\} r:=r+1 \{r=3\}$  není axiom
- $\{r+1=3\} r:=r+1 \{r=3\}$  je axiom

## Příklad

- Dokažte, že následující program vrátí hodnotu větší než 0, pokud je spuštěn pro hodnotu 5.
- Program:  $out := in * 2$

## Důkaz

1) Formulujeme Hoarovu trojici:

$$\{in = 5\} out := in * 2 \{out > 0\}$$

2) Odvodíme vhodnou vstupní podmínku programu:

$$\{in * 2 > 0\}$$

3) Dokážeme Hoarovu trojici:

$$\{in * 2 > 0\} out := in * 2 \{out > 0\} \quad (\text{axiom})$$

4) Dokážeme pomocné tvrzení:

$$(in = 5) \implies (in * 2 > 0)$$

## Pravidlo

- Sekvenční kompozice: 
$$\frac{\{\phi\}S_1\{\chi\} \wedge \{\chi\}S_2\{\psi\}}{\{\phi\}S_1;S_2\{\psi\}}$$

## Význam

- Jestliže  $S_1$  transformuje stav splňující  $\phi$  na stav splňující  $\chi$  a  $S_2$  transformuje stav splňující  $\chi$  na stav splňující  $\psi$ , pak sekvence  $S_1; S_2$  transformuje stav splňující  $\phi$  na stav splňující  $\psi$ .

## Dokazování

- Pro účely důkazu  $\{\phi\}S_1; S_2\{\psi\}$  je nutné nalézt  $\chi$  a ukázat  $\{\phi\}S_1\{\chi\}$  a  $\{\chi\}S_2\{\psi\}$ .

Axiom pro **skip**:  $\{\phi\} \text{ skip } \{\phi\}$

Axiom pro **:=**:  $\{\phi[x := k]\} x := k \{\phi\}$

Sekvenční kompozice: 
$$\frac{\{\phi\} S_1 \{\chi\} \wedge \{\chi\} S_2 \{\psi\}}{\{\phi\} S_1; S_2 \{\psi\}}$$

Alternativa: 
$$\frac{\{\phi \wedge B\} S_1 \{\psi\} \wedge \{\phi \wedge \neg B\} S_2 \{\psi\}}{\{\phi\} \text{if } B \text{ then } S \text{ else } S \text{ fi} \{\psi\}}$$

Iterace: 
$$\frac{\{\phi \wedge B\} S \{\phi\}}{\{\phi\} \text{while } B \text{ do } S \text{ od } \{\phi \wedge \neg B\}}$$

Důsledek: 
$$\frac{\phi \implies \phi', \{\phi'\} S \{\psi'\}, \psi' \implies \psi}{\{\phi\} S \{\psi\}}$$

**Dokažte, že pro  $n \geq 0$  počítá  $n!$ .**



```
r = 1;
```

```
while (n  $\neq$  0) {  
    r = r * n;
```

```
    n = n - 1;  
}
```

**Poznámky k důkazu:**

**Dokažte, že pro  $n \geq 0$  počítá  $n!$ .**

- $\{ n \geq 0 \wedge t=n \}$  {P}  
   $r = 1;$

```
while (n  $\neq$  0) {  
  r = r * n;
```

```
  n = n - 1;  
}  
{ r=t! }
```

{Q}

**Poznámky k důkazu:**

- Formulace dokazovaného jako Hoareho trojice.
- Všimněme si použití pomocné proměnné  $t$ .

**Dokažte, že pro  $n \geq 0$  počítá  $n!$ .**

- $\{ n \geq 0 \wedge t=n \}$  {P}  
   $r = 1;$   
   $\{ n \geq 0 \wedge t=n \wedge r = 1 \}$  {I<sub>1</sub>}  
  while ( $n \neq 0$ ) {  
     $r = r * n;$   
  
     $n = n - 1;$   
  }  
   $\{ r=t! \}$  {Q}

**Poznámky k důkazu:**

- $(n \geq 0 \wedge t=n) \implies (n \geq 0 \wedge t=n \wedge 1=1)$
- $\{n \geq 0 \wedge t=n \wedge 1=1\} r=1 \{ n \geq 0 \wedge t=n \wedge r=1 \}$

**Dokažte, že pro  $n \geq 0$  počítá  $n!$ .**

- $\{ n \geq 0 \wedge t=n \}$   $\{P\}$   
   $r = 1;$   
   $\{ n \geq 0 \wedge t=n \wedge r = 1 \}$   $\{I_1\}$   
  while  $(n \neq 0)$   $\{ r=t!/n! \wedge t \geq n \geq 0 \}$   $\{I_2\}$   
     $r = r * n;$   
  
     $n = n - 1;$   
  }  
   $\{ r=t! \}$   $\{Q\}$

**Poznámky k důkazu:**

- Invariant cyklu:  $\{I_2\} \equiv \{ r=t!/n! \wedge t \geq n \geq 0 \}$
- $I_1 \implies I_2$        $( I_2 \wedge \neg(n \neq 0) ) \implies Q$

**Dokažte, že pro  $n \geq 0$  počítá  $n!$ .**

- $\{ n \geq 0 \wedge t=n \}$   $\{P\}$   
   $r = 1;$   
   $\{ n \geq 0 \wedge t=n \wedge r = 1 \}$   $\{I_1\}$   
  while ( $n \neq 0$ )  $\{ r=t!/n! \wedge t \geq n \geq 0 \}$  {  $\{I_2\}$   
     $r = r * n;$   
     $\{ r=t!/(n-1)! \wedge t \geq n > 0 \}$   $\{I_3\}$   
     $n = n - 1;$   
  }  
   $\{ r=t! \}$   $\{Q\}$

**Poznámky k důkazu:**

- $\{ r*n = t!/(n-1)! \wedge t \geq n > 0 \} r=r*n \{I_3\}$
- $I_2 \wedge (n \neq 0) \implies ( r*n = t!/(n-1)! \wedge t \geq n > 0 )$

**Dokažte, že pro  $n \geq 0$  počítá  $n!$ .**

- $\{ n \geq 0 \wedge t=n \}$  {P}  
   $r = 1;$   
   $\{ n \geq 0 \wedge t=n \wedge r = 1 \}$  {I<sub>1</sub>}  
  while ( $n \neq 0$ )  $\{ r=t!/n! \wedge t \geq n \geq 0 \}$  { {I<sub>2</sub>}  
     $r = r * n;$   
     $\{ r=t!/(n-1)! \wedge t \geq n > 0 \}$  {I<sub>3</sub>}  
     $n = n - 1;$   
  }  
   $\{ r=t! \}$  {Q}

**Poznámky k důkazu:**

- $\{ r = t!/(n-1)! \wedge t \geq (n-1) \geq 0 \} n=n-1 \{I_2\}$
- $I_3 \implies ( r = t!/(n-1)! \wedge t \geq (n-1) \geq 0 )$

## Pozorování

- Díky Hoareho logice jsme převedli důkaz korektnosti programu na sadu matematických tvrzení, typicky využívající Peanovu aritmetiku.

## Poznámka o korektnosti a (ne)úplnosti

- Hoarova logika je korektní, tj. pokud je možné dokázat  $\{P\}S\{Q\}$ , pak vykonání programu  $S$  ze stavu splňujícím  $P$  může skončit pouze ve stavu splňujícím  $Q$ .
- Je-li důkazový systém dostatečně silný na popis aritmetiky celých čísel, je nutně neúplný, tj. existují tvrzení, které nelze v systému dokázat a nelze dokázat ani jejich negaci.

## Potíže s tvorbou důkazů

- Pro potřeby důkazu je často nutné vhodně zesílit vstupní podmínku nebo oslabit výstupní podmínku.
- Je velmi obtížné hledat invarianty cyklů.

## Důkaz v praxi – částečná korektnost

- Často se zjednodušuje na konstatování invariantů a prokázání, že se skutečně jedná o invarianty cyklu (typicky indukci).

## Více o Hoarově logice na FI

- IV022 Návrh a verifikace algoritmů
- IA159 Formal Verification Methods

## Výstup procesu dokazování

- a) Důkaz nalezen a ověřen.
- b) Důkaz nenalezen.
  - Tvrzení platí, lze dokázat, ale důkaz se nám nepodařilo nalézt.
  - Tvrzení platí, ale nelze jej v daném systému dokázat.
  - Tvrzení neplatí.

## Pozorování

- V případě nenalezení důkazu metoda nedává žádnou nápořvedu k tomu, proč se tvrzení nepodařilo dokázat.

## Automatizace hledání důkazů

## Důkaz

- Konečná posloupnost transformací předpokladů  $\psi$  na požadovaný závěr  $\varphi$  s využitím axiomů daného dokazovacího systému a již dokázaných tvrzení.

## Pozorování

- Pro systémy s konečným počtem axiomů a odvozovacích pravidel lze důkazy dané délky systematicky generovat, tj. **pro platná tvrzení** lze v konečném čase nalézt důkaz.
- Všechny rozumné dokazovací systémy mají nekonečně mnoho axiomů. Uvažme např. axiom  $x = x$ , ve skutečnosti je to pouze zkratka za axiomy  $1=1, 2=2, 3=3, \dots$ ).

## Hledání důkazu platného tvrzení

- Potenciálních konečných posloupností k ověření je příliš (nekonečně) mnoho.
- **Obecně pro nalezení důkazu daného tvrzení v daném dokazovacím systému neexistuje algoritmus**, je např. nerozhodnutelné, zda program zastaví pro každý vstup.
- Bez rozumné strategie, nelze očekávat nalezení důkazu platného tvrzení v rozumně krátké době.
- Strategie hledání důkazu je udávána uživatelem s odpovídající kvalifikací a matematickým cítěním.

## Nástroje pro podporu dokazování (Theorem Prover)

- Cílem je pro danou množinu axiomů a důkazový systém nalézt důkaz daného tvrzení.
- Důkaz se hledá střídavě dvěma způsoby:
  - Algoritmický mód – aplikace odvozených pravidel a axiomů
    - Řízen uživatelem nástroje
    - Dedukce, resoluce, unifikace, přepisování, ...
  - Hledací mód – hledá nová platná tvrzení
    - Využití hrubé výpočetní síly.

## Existující nástroje

- Popis dokazovacího systému, programu i požadovaného tvrzení ve vstupním jazyce dokazovacího nástroje.
- Theorem prover: PVS, MONA, TVLA, ...

## Verifikace paralelních programů

## Paralelní kompozice

- Komponenty souběžně přispívají k transformaci výchozího stavu na cílový.
- Významová funkce pro paralelně běžící programy vznikne libovolným proložením atomických akcí jednotlivých komponent. (**Interleaving.**)

## Nekompozicionalita významových funkcí

- Významovou funkci paralelního programu nelze získat jako složení významových funkcí jednotlivých komponent.
- Výsledek může být závislý na proložení akcí.

## Příklad

- Systém:  $(y=x; y++; x=y) \parallel (y=x; y++; x=y)$
- Vstup-výstupní proměnná  $x$
- Významová funkce obou procesů je  $\lambda x \rightarrow x+1$ .
- Složení významových funkcí:  $(\lambda x \rightarrow x+1) \cdot (\lambda x \rightarrow x+1)$ .
- $(\lambda x \rightarrow x+1) \cdot (\lambda x \rightarrow x+1) 0 = 2$

## 2 konkrétní běhy

- Stav =  $(x, y_1, y_2)$
- $(0, -, -) \xrightarrow{y_1=x} (0, 0, -) \xrightarrow{y_2=x} (0, 0, 0) \xrightarrow{y_1++} \xrightarrow{x=y_1} (1, 1, 0) \xrightarrow{y_2++} \xrightarrow{x=y_2} (\mathbf{1}, 1, 1)$
- $(0, -, -) \xrightarrow{y_1=x} (0, 0, -) \xrightarrow{y_1++} \xrightarrow{x=y_1} (1, 1, -) \xrightarrow{y_2=x} (1, 1, 1) \xrightarrow{y_2++} \xrightarrow{x=y_2} (\mathbf{2}, 1, 2)$

## Pozorování

- Konkrétní časování událostí souvisejících s interakcí programů je forma vstupu.
- Paralelní programy jsou svým způsobem reaktivní systémy, neboť nejsou dopředu známa všechna vstupní data.

## Důsledek

- U paralelních programů často požadujeme (specifikujeme) chování, která se nedají vyjádřit s použitím vstupních a výstupních podmínek.

- Pokud program P pošle zprávu programu Q, tak potom nepošle druhou zprávu, dokud neobdrží od programu Q potvrzení.
- Program P nezmění podruhé hodnotu proměnné  $x$ , dokud program Q hodnotu  $x$  nepřečte.

## Formalizace slovního popisu

- Lze vypořádat, že v popisu vlastností paralelních programů se typicky vyskytují jisté formulace, například:
  - Určitě nastane nějaká konkrétní událost.
  - Může nastat nějaká konkrétní událost.
  - Trvale platí jistý predikát či vlastnost.
  - Platí něco, dokud se nestane něco jiného.
- Lze formalizovat pomocí temporálních logik.
- Pnueli, 1977

## Pozorování

- Pro modální logiky je možné vystavět podobné důkazové systémy, jako pro Hoarovu logiku.
- Tyto důkazové techniky vykazují stejné/podobné nevýhody jako verifikace paralelních programů s využitím vstupních a výstupních podmínek.

## Model checking

- Jiná metoda ověření platnosti formule temporální logiky.

# Ověřování modelu (Model Checking)

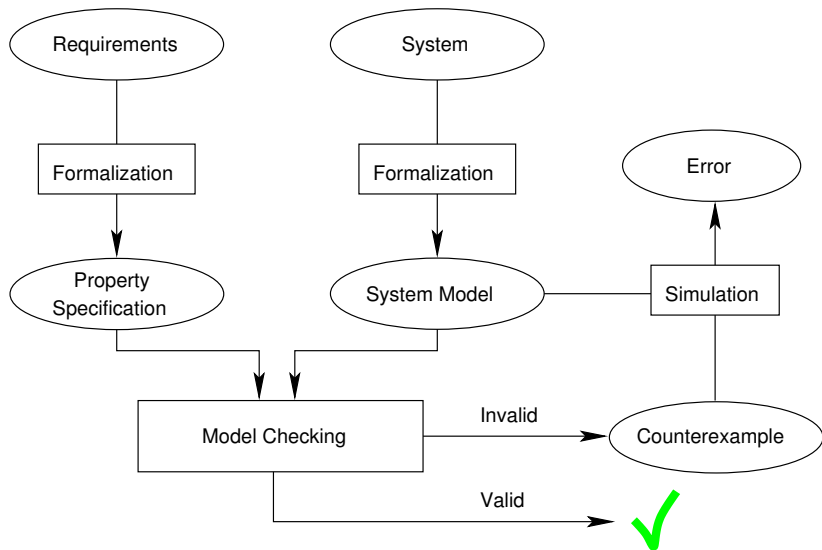
## Ověřování modelu – přehled

- Vytvoříme formální model  $\mathcal{M}$  verifikovaného systému.
- Specifikaci vyjádříme formulí  $\varphi$  zvolené temporální logiky.
- Rozhodneme, zda  $\mathcal{M} \models \varphi$ . Tj., zda  $\mathcal{M}$  je modelem formule  $\varphi$ . (Odtud název ověřování modelu.)

## Volitelné

- Postranním produktem rozhodnutí může být (případně větvící se) posloupnost stavů, dokládající rozhodnutí. Tato posloupnost se běžně označuje slovem **protipříklad** (často produkována pouze za účelem ukázání neplatnosti  $\varphi$ ).
- **Proces rozhodnutí lze v automatizovat.**

# Ověřování modelu – schéma



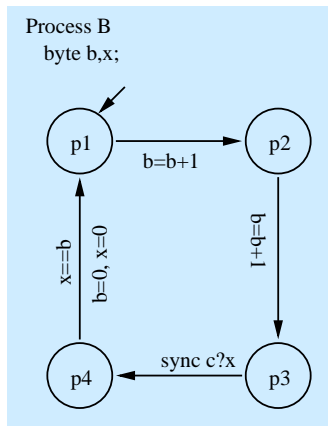
## Model-checkery

- Softwarové nástroje, které pro model systému a temporální vlastnost provedou rozhodnutí o splnění dané vlastnosti modelem.
- SPIN, UppAll, SMV, Prism, DiVinE ...

## Modelovací jazyky

- Procesy popsány jako rozšířené konečné automaty.
- Rozšíření umožňuje podmínit provedení přechodu/akce platností boolovského výrazu, případně synchronizací s akcí jiného souběžně běžícího procesu.

- Konečný automat
  - Stavy (Lokace)
  - Iničiální stav
  - Přejchody
  - (Akceptující stavy)
- Přejchody rozšířené o
  - Stráž
  - Synchronizaci s předáváním hodnot
  - Efekt (přiřazení)
- Lokální proměnné
  - integer, byte
  - channel



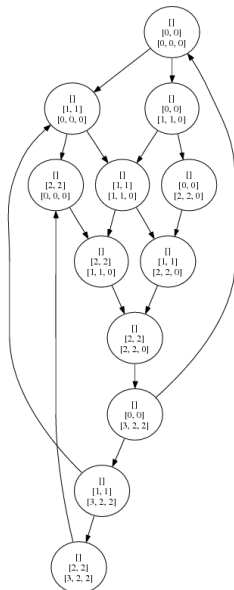
# Příklad modelu v jazyce DVE

```
channel {byte} c[0];
```

```
process A {  
  byte a;  
  state q1,q2,q3;  
  init q1;  
  trans  
  q1→q2 { effect a=a+1; },  
  q2→q3 { effect a=a+1; },  
  q3→q1 { sync c!a; effect a=0; };  
}
```

```
process B {  
  byte b,x;  
  state p1,p2,p3,p4;  
  init p1;  
  trans  
  p1→p2 { effect b=b+1; },  
  p2→p3 { effect b=b+1; },  
  p3→p4 { sync c?x; },  
  p4→p1 { guard x==b; effect b=0, x=0; };  
}
```

```
system async;
```



# Sémantika ukázána simulací

State: []; A:[q1, a:0]; B:[p1, b:0, x:0]  
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }  
1 ⟨1.0⟩: p1 → p2 { effect b = b+1; }  
Command:1

---

State: []; A:[q1, a:0]; B:[p2, b:1, x:0]  
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }  
1 ⟨1.1⟩: p2 → p3 { effect b = b+1; }  
Command:1

---

State: []; A:[q1, a:0]; B:[p3, b:2, x:0]  
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }  
Command:0

---

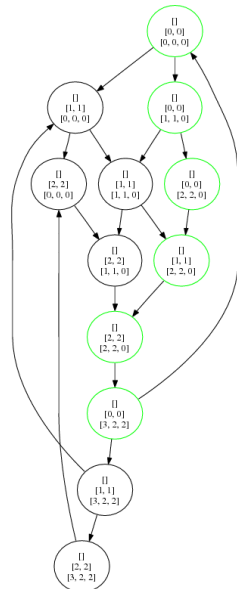
State: []; A:[q2, a:1]; B:[p3, b:2, x:0]  
0 ⟨0.1⟩: q2 → q3 { effect a = a+1; }  
Command:0

---

State: []; A:[q3, a:2]; B:[p3, b:2, x:0]  
0 ⟨0.2&1.2⟩: q3 → q1 { sync c!a; effect a = 0; }  
p3 → p4 { sync c?x; }  
Command:0

---

State: []; A:[q1, a:0]; B:[p4, b:2, x:2]



## Platnost formule

- Požadovaná vlastnost může být porušena při jednom jediném konkrétním proložení akcí.
- Rozhodnutí je podloženo analýzou grafu stavového prostoru paralelního programu.

## Stavová exploze

- Není-li řečeno jinak, je třeba uvážit všechna možná proložení akcí.
- **Počet stavů**, do kterých se může dostat paralelní program, **je exponenciálně větší než velikost** zápisu paralelního programu.

## Obecná technika aplikovatelná na různé typy systémů

- Hardware, software, zabudované systémy, ...

## Garance výsledku (matematicky podložen)

- Rozhodovací procedura tvrdí, že  $\mathcal{M} \models \varphi$ , tehdy a jen tehdy, pokud to skutečně platí.

## Existence podpůrných nástrojů – “model checkerů”

- Verifikace tzv. na kliknutí myši / zmačknutí tlačítka.
- Minimální účast uživatele v rozhodovacím procesu.
- Automatická identifikace a generování protipříkladů.

## **Vhodná pouze k verifikaci konkrétních transformací**

- Nelze použít na obecný důkaz toho, že program například pro zadané  $n$  spočítá hodnotu  $n!$ .
- Lze však ověřit, že pro hodnotu 5 program vrátí 120.

## **Verifikuje pouze model systému**

- Platnost formule, neznamená splnění specifikace systémem.

## **Velikost stavového prostoru**

- Aplikovatelné (pouze) na konečně stavové systémy.
- Vzhledem k potencionálně exponenciálnímu nárůstu počtu stavů ve stavovém prostoru, je praktická aplikovatelnost techniky omezena na relativně malé modely.

## **Verifikuje jen to, co je specifikováno**

- Co není vyjádřeno formulí, to se neverifikuje.