Když testy nestačí

Verifikace programů a model checking

Jan Kofroň

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Department of
Distributed and
Dependable
Systems
D3S

Může být program opravdu správně?

Program může fungovat **milionkrát** správně…

Může být program opravdu správně?

Program může fungovat **milionkrát** správně…

a milionprvní běh selže.

Může být program opravdu správně?

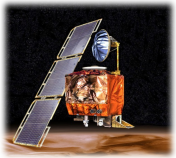
Program může fungovat **milionkrát** správně…

a milionprvní běh selže.

Jak bychom mohli získat silnější jistotu než jen „zkusili jsme to“?

Chyby nejsou jen otravné

Mars Climate Orbiter



1999: Různé části softwarového vybavení používaly různé jednotky. Výsledek: ztráta sondy.

Yorktown

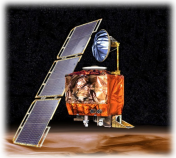
1997: Neošetřené dělení nulou vyřadilo na několik hodin část lodních systémů.

Boeing 737 MAX

2018: Řídicí software se mimo jiné spoléhal na informaci z jediného senzoru.

Chyby nejsou jen otravné

Mars Climate Orbiter



1999: Různé části softwarového vybavení používaly různé jednotky. Výsledek: ztráta sondy.

Yorktown

1997: Neošetřené dělení nulou vyřadilo na několik hodin část lodních systémů.

Boeing 737 MAX

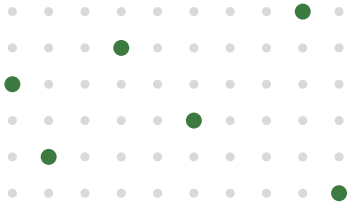
2018: Řídicí software se mimo jiné spoléhal na informaci z jediného senzoru.

Software dnes rozhoduje i o tom, co se stane ve fyzickém světě.

Testování je skvělé. Ale co přesně dokazuje?

Testování

- Vyberu konkrétní vstup nebo scénář.
- Program spustím.
- Porovnám výsledek s očekáváním.

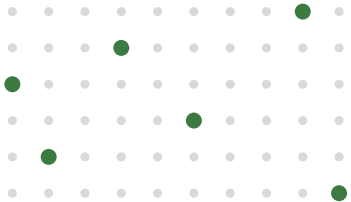


Několik vybraných bodů z mnoha možností

Testování je skvělé. Ale co přesně dokazuje?

Testování

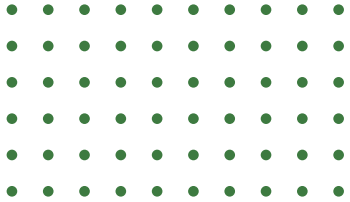
- Vyberu konkrétní vstup nebo scénář.
- Program spustím.
- Porovnám výsledek s očekáváním.



Několik vybraných bodů z mnoha možností

Verifikace

- Formuluji, co má platit.
- Matematicky analyzuji model programu.
- Snažím se pokrýt **všechny** relevantní možnosti.



Systematické pokrytí stavového prostoru

Jaké máme možnosti?

Testování
konkrétní běhy

Statická analýza
analýza kódu

Model checking
procházení možností

Důkaz
logické odvození

Jaké máme možnosti?

Testování
konkrétní běhy

Statická analýza
analýza kódu

Model checking
procházení možností

Důkaz
logické odvození

Dnes se zaměříme na **model checking.**

Model checking v jedné větě

Máme **model systému** a **vlastnost**.

Model checker rozhodne, zda vlastnost v modelu platí.

$$M \models \varphi$$

Model checking v jedné větě

Máme **model systému** a **vlastnost**.

Model checker rozhodne, zda vlastnost v modelu platí.

$$M \models \varphi$$

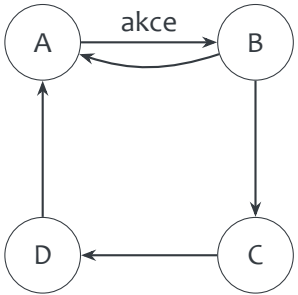
V základní podobě je to chytré a systematické procházení konečného grafu.

Co je to model?

Model zachycuje to, co je pro naši otázku důležité:

- **stavy** – co právě platí,
- **přechody** – co se může stát dál.

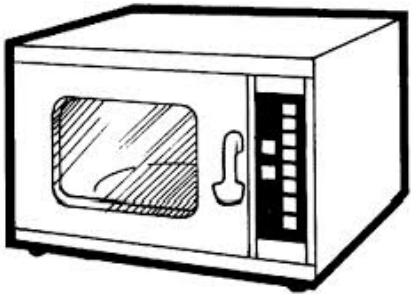
Nemusí popisovat každý šroubek. Musí ale zachytit chování, které může ovlivnit ověřovanou vlastnost.



Příklad: mikrovlnná trouba

Pro začátek ji silně zjednodušíme na tři režimy:

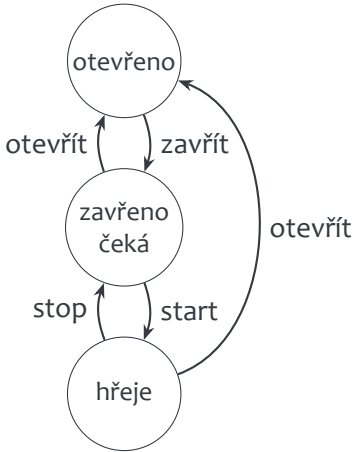
- dvířka jsou **otevřená,**
- dvířka jsou **zavřená** a trouba čeká,
- trouba **hřeje** (se zavřenými dveřmi).



Příklad: mikrovlnná trouba

Pro začátek ji silně zjednodušíme na tři režimy:

- dvířka jsou **otevřená,**
- dvířka jsou **zavřená** a trouba čeká,
- trouba **hřeje** (se zavřenými dveřmi).

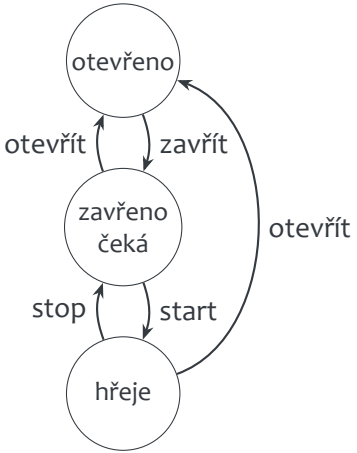


Otázka: **Které stavy (režimy) jsou přípustné a *dosažitelné*?**

Příklad: mikrovlnná trouba

Pro začátek ji silně zjednodušíme na tři režimy:

- dvířka jsou **otevřená,**
- dvířka jsou **zavřená** a trouba čeká,
- trouba **hřeje** (se zavřenými dveřmi).



Všimněte si: stav **hřeje + otevřeno** v bezpečném modelu nechceme.

Co chceme ověřit?

Bezpečnostní vlastnost (safety)

Mikrovlnka **nikdy nehřeje,** když jsou otevřená dvířka**.**

Co chceme ověřit?

Bezpečnostní vlastnost (safety)

Mikrovlnka **nikdy nehřeje,** když jsou otevřená dvířka**.**

Vlastnost postupu (liveness)

Když stisknu start za správných podmínek**, trouba začne hřát.**

Co chceme ověřit?

Bezpečnostní vlastnost (safety)

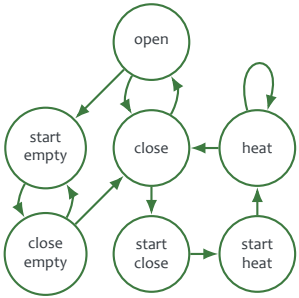
Mikrovlnka **nikdy nehřeje,** když jsou otevřená dvířka**.**

Vlastnost postupu (liveness)

Když stisknu start za správných podmínek**, trouba začne hřát.**

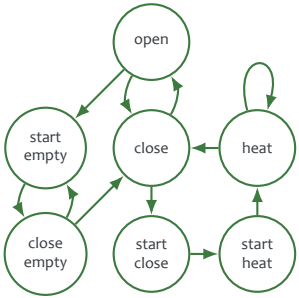
Safety: „Něco špatného se nikdy nestane.“
Liveness: „Něco dobrého se nakonec stane.“

Model checker: dva vstupy, dva možné výsledky



Model systému

Model checker: dva vstupy, dva možné výsledky

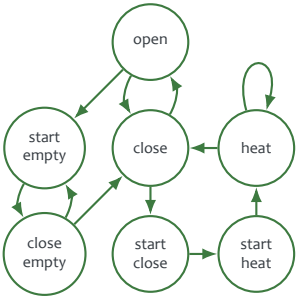


Model systému

AG (start → AF heat)

Specifikace vlastnosti

Model checker: dva vstupy, dva možné výsledky



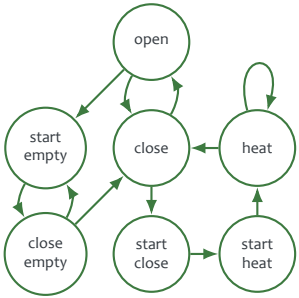
Model systému

AG (start → AF heat)

Specifikace vlastnosti

Model Checker

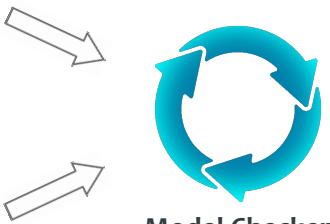
Model checker: dva vstupy, dva možné výsledky



Model systému

AG (start → AF heat)

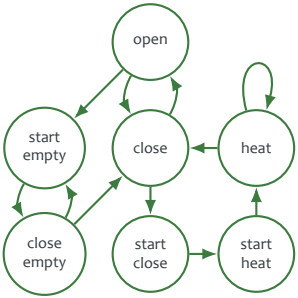
Specifikace vlastnosti



Model Checker

Vlastnost splněna

Model checker: dva vstupy, dva možné výsledky



Model systému

AG (start → AF heat)

Specifikace vlastnosti

Model Checker

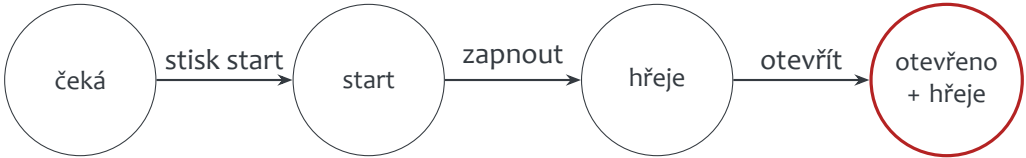
Vlastnost splněna

Vlastnost porušena (protipříklad)

Protipříklad je často nejcennější výsledek

Porušená vlastnost

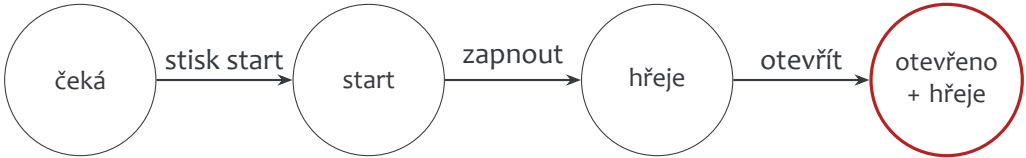
“dvířka jsou otevřená a přesto trouba hřeje.”



Protipříklad je často nejcennější výsledek

Porušená vlastnost

“dvířka jsou otevřená a přesto trouba hřeje.”



Model checker neřekne jen „ne“. Často ukáže **konkrétní cestu k chybě**.

Složitost modelu

- Modely mohou často mít stovky miliony stavů
  - ...nebo i více.
- Prozkoumat celý model pomocí model checkeru vyžaduje velké množství času
  - ... a paměti.
- Výkon moderních počítačů stále nestačí, protože verifikace je *exponenciálně náročnější* než spuštění programu.

Složitost modelu

- Modely mohou často mít stovky miliony stavů
  - ...nebo i více.
- Prozkoumat celý model pomocí model checkeru vyžaduje velké množství času
  - ... a paměti.
- Výkon moderních počítačů stále nestačí, protože verifikace je *exponenciálně náročnější* než spuštění programu.

Potřebujeme chytřejší způsoby než jen **hrubou výpočetní sílu**.

Kde se počet stavů bere?

U konkrétního programu se na počtu stavů podílí:

- kombinace hodnot vstupních proměnných (vstupní data),
- složitost (velikost) programu,
- různé naplánování vláken (procesů).

Příklad: sečtení dvou čísel ze dvou vláken.

vlákno A: 3 stavy

vlákno B: 7 stavů

2 celočíslené proměnné: 256^2

$(3 \cdot 3 + 4) \cdot 256^2 = 851\,968$ stavů

Současně běžící programy: pořadí je součást problému

```
x = 0

Vlákno A:       Vlákno B:
x = x + 1       x = x + 1
```

Na konci bychom čekali $x = 2$.

Současně běžící programy: pořadí je součást problému

```
x = 0

Vlákno A:        Vlákno B:
x = x + 1        x = x + 1
```

Na konci bychom čekali $x = 2$.

Ale instrukce se mohou proložit:

1. A načte $x = 0$
2. B načte $x = 0$
3. A spočítá $x + 1$
4. B spočítá $x + 1$
5. A uloží 1
6. B uloží 1

Současně běžící programy: pořadí je součást problému

```
x = 0

Vlákno A:        Vlákno B:
x = x + 1        x = x + 1
```

Na konci bychom čekali $x = 2$.

Ale instrukce se mohou proložit:

1. A načte $x = 0$
2. B načte $x = 0$
3. A spočítá $x + 1$
4. B spočítá $x + 1$
5. A uloží 1
6. B uloží 1

Výsledek: $x = 1$ **CHYBA**

Model checker umí prohledat i různá proložení vláken

A potom B

A: čti 0 → 0 + 1 → zapiš 1
B: čti 1 → 1 + 1 → zapiš 2

$x = 2$ **OK**

B potom A

B: čti 0 → 0 + 1 → zapiš 1
A: čti 1 → 1 + 1 → zapiš 2

$x = 2$ **OK**

Proložený běh

A: čti 0
B: čti 0
A: 0 + 1
A: zapiš 1
B: 0 + 1
B: zapiš 1

$x = 1$ **CHYBA**

Model checker umí prohledat i různá proložení vláken

A potom B

A: čti 0 → 0 + 1 → zapiš 1
B: čti 1 → 1 + 1 → zapiš 2

$x = 2$ **OK**

B potom A

B: čti 0 → 0 + 1 → zapiš 1
A: čti 1 → 1 + 1 → zapiš 2

$x = 2$ **OK**

Proložený běh

A: čti 0
B: čti 0
A: 0 + 1
A: zapiš 1
B: 0 + 1
B: zapiš 1

$x = 1$ **CHYBA**

Model checker systematicky zkoumá **všechna možná pořadí kroků,**
ne jen pár vybraných možností.

Největší problém: exploze stavového prostoru

n nezávislých bitových proměnných ⇒ 2^n kombinací

$20 \rightarrow 1\,048\,576$ stavů

$40 \rightarrow$ přibližně 10^{12} stavů

$64 \rightarrow$ přibližně 1.8×10^{19} stavů

Největší problém: exploze stavového prostoru

n nezávislých bitových proměnných ⇒ 2^n kombinací

$20 \rightarrow 1\,048\,576$ stavů
$40 \rightarrow$ přibližně 10^{12} stavů
$64 \rightarrow$ přibližně 1.8×10^{19} stavů

A to jsme ještě nepřidali stavy výpočtu ani plánování vláken.

Jak se s explozí stavů bojuje?

- **Abstrakce**: spojím stavy, které se pro danou otázku chovají stejně.

Jak se s explozí stavů bojuje?

- **Abstrakce**: spojím stavy, které se pro danou otázku chovají stejně.
- **Symbolické reprezentace**: ukládám celé množiny stavů najednou.

Jak se s explozí stavů bojuje?

- **Abstrakce**: spojím stavy, které se pro danou otázku chovají stejně.
- **Symbolické reprezentace**: ukládám celé množiny stavů najednou.
- **Redukce pořadí**: nezkouším zbytečně ekvivalentní proložení.

Jak se s explozí stavů bojuje?

- **Abstrakce**: spojím stavy, které se pro danou otázku chovají stejně.
- **Symbolické reprezentace**: ukládám celé množiny stavů najednou.
- **Redukce pořadí**: nezkouším zbytečně ekvivalentní proložení.
- **Omezené hledání**: hledám chyby jen do určité hloubky.

Jak se s explozí stavů bojuje?

- **Abstrakce**: spojím stavy, které se pro danou otázku chovají stejně.
- **Symbolické reprezentace**: ukládám celé množiny stavů najednou.
- **Redukce pořadí**: nezkouším zbytečně ekvivalentní proložení.
- **Omezené hledání**: hledám chyby jen do určité hloubky.

Model checking je kombinace
matematiky, algoritmů a chytrých optimalizací.

Jak popsat vlastnost přesně?

Model checker potřebuje vlastnost v jazyce, který nemá nejasnosti běžné řeči.

Existují proto speciální logické jazyky – **temporální logiky** –, které umějí přesně mluvit o tom, co platí

- **vždy**,
- **někdy** / **nakonec**,
- **dokud** nenastane něco jiného.

Stejné věty, ale přesně

„Mikrovlnka nikdy nehřeje s otevřenými dvířky.“

$$\mathbf{G}\,\neg(open \wedge heat)$$

„Po stisku tlačítka start nakonec začne hřát.“

$$\mathbf{G}(start \rightarrow \mathbf{F}\,heat)$$

Jak popsat vlastnost přesně?

Model checker potřebuje vlastnost v jazyce, který nemá nejasnosti běžné řeči.

Existují proto speciální logické jazyky – **temporální logiky** –, které umějí přesně mluvit o tom, co platí

- **vždy**,
- **někdy** / **nakonec**,
- **dokud** nenastane něco jiného.

Stejné věty, ale přesně

„Mikrovlnka nikdy nehřeje s otevřenými dvířky.“

$$\mathbf{G}\,\neg(open \wedge heat)$$

„Po stisku tlačítka start nakonec začne hřát.“

$$\mathbf{G}(start \rightarrow \mathbf{F}\,heat)$$

Symboly nejsou to hlavní. Důležité je, že slova jako **vždy** a **nakonec** dostanou přesný **matematický význam**.

SPIN: MODEL CHECKER V PRAXI

Spin pracuje s modely v jazyce **Promela**.

1. popíšeme možné chování systému,
2. přidáme vlastnost, která má platit,
3. Spin systematicky prohledá stavový prostor,
4. při chybě uloží **error trail** – konkrétní protipříklad.

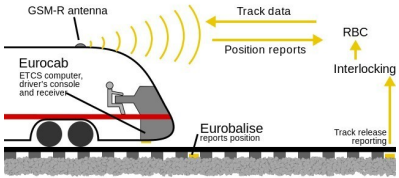
Evropská specifikace pro řízení železniční dopravy: ERTMS/ETCS

Hybrid ERTMS/ETCS Level 3 je evropská specifikace železničního zabezpečení.

- Namodelovali jsme tuto specifikaci v jazyce Promela,
- spustili jsme **Spin** pro validaci a verifikaci,
- objevili jsme **nekonzistence a jednu chybu v zabezpečení**.

Důležitá pointa

Už samotný pokus napsat požadavek přesně může odhalit problém – ještě před samotným ověřením.



#		Condition	Priority over	Section ref.
#9A		(TTD is free)		3.1.1.5
#9B		(integer train has reported to have left the VSS)	#10	4.5.1.7
	AND	(the "shadow train timer A" of the TTD was not expired at the moment of the time stamp in the position report)		
#10A		(VSS is left by all reporting trains)		3.6 3.7 3.3.4.5
#10B		(train is located on the VSS)		3.8.1
	AND	((mute timer is expired) OR (EoM))		4.2.1.2
#11A		(integer train located on the VSS reported to have left the TTD in rear)		4.5.1.4
	AND	(the "shadow train timer A" of the TTD in rear was not expired at the moment of the time stamp in the position report)		
	AND	(the reported min-safe-rear-end of this train is within the distance that can be covered at the reported speed within the "shadow train timer A" from the TTD limit)		

https://transport.ec.europa.eu/transport-modes/rail/ertms/what-ertms-and-how-does-it-work/etcs-levels-and-modes_en

Model checking není jediná verifikační technika

- Vedle model checkingu existuje řada dalších verifikačních přístupů.
- Některé techniky podporují i automatickou konstrukci modelu.
- Některé pracují nad stavovým prostorem, jiné nad zápisem programu (zdrojovým kódem).
- Výběr závisí na typu modelu/vstupu, požadovaných vlastnostech, časových možnostech, dostupném výpočetním výkonu, ...

Co už se takto ověřuje?

- Hardware a řídicí logika,
- Komunikační protokoly a distribuované algoritmy,
- Části operačních systémů a ovladače,
- Bezpečnostně kritické systémy,
- Vlastnosti programů s více vlákny.

Co už se takto ověřuje?

- Hardware a řídicí logika,
- Komunikační protokoly a distribuované algoritmy,
- Části operačních systémů a ovladače,
- Bezpečnostně kritické systémy,
- Vlastnosti programů s více vlákny.

Neověřuje se vždy celý systém najednou.
Často se verifikuje **kritická část** nebo vhodná **abstrakce** – model.

Co verifikace nezaručí sama od sebe

1. Je model věrný skutečnému programu a prostředí?
2. Ověřujeme opravdu správnou vlastnost?
3. Nezapomněli jsme na nějaký předpoklad?

Co verifikace nezaručí sama od sebe

1. Je model věrný skutečnému programu a prostředí?
2. Ověřujeme opravdu správnou vlastnost?
3. Nezapomněli jsme na nějaký předpoklad?

Výsledek verifikace je přesnou odpovědí na **přesně položenou otázku**.

Shrnutí

1. Testy kontrolují vybrané běhy; verifikace se snaží pokrýt všechny relevantní možnosti.
2. Model checking systematicky prohledává **stavový prostor** verifikovaného systému.
3. Když vlastnost neplatí, **protipříklad** často přesně ukáže cestu k chybě.
4. Model i vlastnost musejí být přesné.
5. Hlavní praktickou překážkou je **exploze stavového prostoru**.

Otázky?

Děkuji za pozornost!