

# IV113 Validace a verifikace

## Detekce akceptujícího cyklu

doc. RNDr. Jiří Barnat, Ph.D.

## Problém

- Kripkeho struktura  $M$
- LTL formule  $\varphi$
- $M \models \varphi$  ?

## Řešení pomocí Büchiho automatů

- $A_{sys}$  – automat akceptující běhy modelu
- $A_{\neg\varphi}$  – automat akceptující běhy porušující vlastnost  $\varphi$
- $L(A_{sys}) \cap L(A_{\neg\varphi}) = L(A_{sys} \times A_{\neg\varphi})$
- $L(A_{sys} \times A_{\neg\varphi}) \neq \emptyset \iff$  model má běh porušující  $\varphi$
- $L(A_{sys} \times A_{\neg\varphi}) = \emptyset \iff M \models \varphi$

## Detekce akceptujících cyklů

## Problém

- Je dán Büchiho automat  $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ .
- Je jazyk akceptovaný automatem  $\mathcal{A}$  neprázdný?

## Redukce na detekci akceptujícího cyklu v grafu

- Necht'  $G = (S, E)$ , kde
$$E = \{(u, v) \in S \times S \mid \exists a \in \Sigma \text{ takové, že } v \in \delta(u, a)\}$$
je graf Büchiho automatu.
- $L(\mathcal{A})$  je neprázdný právě když graf automatu  $\mathcal{A}$  obsahuje dosažitelný akceptující cyklus, tj. cyklus jehož alespoň jeden vrchol  $v$  je akceptující ( $v \in F$ ) a zároveň dosažitelný z iniciálního stavu ( $(s_0, v) \in E^*$ ).

## Algoritmické řešení

- 1) V grafu Büchiho automatu identifikuj všechny dosažitelné akceptující vrcholy. (Vnější procedura.)
- 2) Pro každý takto identifikovaný vrchol ověř, že není dosažitelný ze sebe sama. (Vnitřní procedura.)

## Dosažitelnost v grafu

- Standardní grafový algoritmus.
- Výpočet množiny dosažitelných, případně akceptujících dosažitelných vrcholů lze provést v čase  $\mathcal{O}(V + E)$ .
- S obecným algoritmem detekce dosažitelnosti je detekce přítomnosti akceptujícího cyklu proveditelná v čase  $\mathcal{O}(V + E + F(V + E))$ .
- Pokud ale použijeme strategii **prohledávání do hloubky**, lze dosáhnout času  $\mathcal{O}(V + E)$ .

```
proc Reachable( $V, E, v_0$ )  
  Visited =  $\emptyset$   
  DFS( $v_0$ )  
  Return(Visited)  
end
```

```
proc DFS(vertex)  
  if vertex  $\notin$  Visited  
    then /* Visits vertex */  
      Visited := Visited  $\cup$  {vertex}  
      foreach {  $v \mid (vertex, v) \in E$  } do  
        DFS( $v$ )  
      od  
      /* Backtracks from vertex */  
    fi  
  end
```

## Pozorování 1

- Každý vrchol  $v$  dosažitelný z  $v_0$  (tj.  $(v_0, v) \in E^*$ ) je procedurou  $\text{DFS}(v_0)$  nejprve navštíven a poté backtrackován.

## Pozorování 2

- Každý vrchol  $v$  dosažitelný z  $v_0$  (tj.  $(v_0, v) \in E^*$ ) je označen jako navštívený dříve, než je vrchol  $v_0$  backtrackován procedurou  $\text{DFS}(v_0)$ .

## Pozorování 3

- Pokud pro dva různé vrcholy  $v_1, v_2$  platí, že
  - $(v_0, v_1) \in E^*$ ,
  - $(v_1, v_1) \notin E^+$ ,
  - $(v_1, v_2) \in E^+$ .
- Pak procedura  $\text{DFS}(v_0)$  backtrackuje z vrcholu  $v_2$  dříve než backtrackuje z vrcholu  $v_1$ .

## DFS post-order

- Pokud  $(v, v) \notin E^+$  a  $(v_0, v) \in E^*$ , pak po skončení procedury  $\text{DFS}(v)$ , volané v rámci výpočtu  $\text{DFS}(v_0)$ , jsou všechny vrcholy  $u$  takové, že  $(v, u) \in E^+$  navštívené a backtrackované.

## Předpoklady

- Vnitřní procedura pro stav  $s$  ohlásí cyklus a ukončí výpočet algoritmu, právě když  $s$  leží na akceptujícím cyklu.
- Vnitřní procedury budou spouštěny pro akceptující vrcholy v pořadí, ve kterém z nich vnější procedura backtrackuje.

## Tvrzení

- Pokud vnitřní procedura pro vrchol  $s$  skončí aniž by ohlásila cyklus, tak podgraf dosažitelný z vrcholu  $s$  neobsahuje akceptující cyklus.

## Tvrzení

- Pokud podgraf dosažitelný z vrcholu  $s$  neobsahuje akceptující cyklus, pak žádný akceptující cyklus přes vrchol  $r$  ležící mimo podgraf dosažitelný z  $s$  neprochází stavem dosažitelným z  $s$ .

## Důsledek vedoucí k lineárnímu algoritmu

- Každou vnitřní proceduru lze omezit na vrcholy dosud nenavštívené v žádné předchozí vnitřní proceduře.

## $\mathcal{O}(V + E)$ algoritmus

- 1) Vnitřní procedury budou spouštěny pro akceptující vrcholy v pořadí, ve kterém vnější procedura z těchto vrcholů backtrackuje.
- 2) Vnitřní procedury nenavštěvují vrcholy navštívené v předchozích vnitřních procedurách.

# Detekce akceptujících cyklů v čase $\mathcal{O}(V + E)$

```
proc Detekce akceptujících cyklů
```

```
  Visited :=  $\emptyset$ 
```

```
  DFS( $v_0$ )
```

```
end
```

```
proc DFS(vertex)
```

```
  if (vertex)  $\notin$  Visited
```

```
    then Visited := Visited  $\cup$  {vertex}
```

```
      foreach { s | (vertex, s)  $\in$  E } do
```

```
        DFS(s)
```

```
      od
```

```
      if IsAccepting(vertex)
```

```
        then DetectCycle(vertex)
```

```
      fi
```

```
  fi
```

```
end
```

## Pozorování

- Během provádění procedury DFS se každý vrchol grafu nachází právě v jednom z následujících stavů:
  - nebyl navštíven (bílé vrcholy),
  - byl navštíven, ale nebyl backtrackován (šedé vrcholy),
  - byl navštíven a byl i backtrackován (černé vrcholy).
- Šedé vrcholy tvoří cestu od počátečního vrcholu k vrcholu, který je v daný okamžik algoritmem zpracováván.

## Tvrzení

- Pokud je následníkem právě zpracovávaného vrcholu šedý vrchol (tj. vrchol na zásobníku rekurzivních volání procedury *DFS*), pak graf obsahuje cyklus.

## Využití

- Ve vnořené proceduře není nutné dosáhnout přesně vrcholu, pro který je detekce cyklu volána, ale stačí dosáhnout vrcholu na zásobníku vnější procedury.

# $\mathcal{O}(V + E)$ Algoritmus

```
proc Detekce akceptujících cyklů
  Visited := Nested := in_stack :=  $\emptyset$ 
  DFS( $v_0$ )
  Exit("Nepřítomen")
end
```

```
proc DFS(vertex)
  if vertex  $\notin$  Visited
    then Visited := Visited  $\cup$  {vertex}
         in_stack := in_stack  $\cup$  {vertex}
         foreach { s | (vertex, s)  $\in$  E } do
           DFS(s)
         od
         if IsAccepting(vertex)
           then DetectCycle(vertex)
         fi
         in_stack := in_stack  $\setminus$  {vertex}
  fi
end
```

```
proc DetectCycle(vertex)
  if vertex  $\notin$  Nested
    then Nested := Nested  $\cup$  {vertex}
         foreach { s | (vertex, s)  $\in$  E } do
           if s  $\in$  in_stack
             then WriteOut(in_stack)
                  Exit("Přítomen")
             else DetectCycle(s)
           fi
         od
  fi
end
```

## Vnější procedura

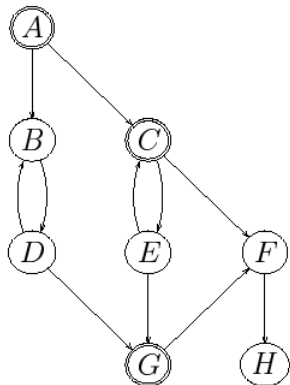
- Časová složitost:  $\mathcal{O}(V + E)$
- Prostorová složitost:  $\mathcal{O}(V)$

## Vnitřní procedury

- Celková časová složitost:  $\mathcal{O}(V + E)$
- Prostorová složitost:  $\mathcal{O}(V)$

## Složitost

- Časová složitost:  $\mathcal{O}(V + E + V + E) = \mathcal{O}(V + E)$
- Prostorová složitost:  $\mathcal{O}(V + V) = \mathcal{O}(V)$



- 1st DFS: A,B,D,B,G,F,H,H,F,G  
1st DFS stack: A,B,D,G  
visited: A,B,D,F,G,H / -
- 2nd DFS: G,F,H,H,F,G  
visited: A,B,D,F,G,H / F,G,H
- 1st DFS: G,D,B,C,E,C,G,E,F,C  
1st DFS stack: A,C  
visited: all / F,G,H
- 2nd DFS: C,E,C  
counterexample: A,C,E,C

visited state    backtrack non-accepting state    backtrack accepting state

## Typy Büchi automatů a jejich využití při verifikaci

## Terminální Büchi automaty

- Všechny akceptující cykly v automatu jsou právě ve formě smyčky nad akceptujícím stavem strážené výrazem `true`.

## Slabé Büchi automaty (weak)

- Každá silně souvislá komponenta automatu je striktně tvořena buď pouze akceptujícími stavy, nebo pouze neakceptujícími.

## Automat $A_{\neg\varphi}$

- Pro řadu LTL formulí  $\varphi$  je  $A_{\neg\varphi}$  terminální nebo slabý.
- $A_{\neg\varphi}$  je typicky velmi malý (max desítky stavů).
- Typ  $A_{\neg\varphi}$  lze zjistit před procesem verifikace.
- Typy komponent  $A_{\neg\varphi}$ 
  - **Neakceptující** – bez akceptujícího cyklu.
  - **Striktně akceptující** – každý cyklus je akceptující.
  - **Smíšené** – obsahuje akceptující i neakceptující cykly.

## Produktový automat

- Prohledávaný graf je synchronní produkt  $A_S$  a  $A_{\neg\varphi}$ .
- Typ komponent  $A_S \times A_{\neg\varphi}$  určen odpovídajícími komponentami  $A_{\neg\varphi}$ .

## $A_{\neg\varphi}$ je terminální Büchi automat

- Pro důkaz existence akceptujícího cyklu stačí prokázat dosažitelnost stavu akceptujícího ve složce  $A_{\neg\varphi}$ .
- Proces verifikace se redukuje na **analýzu dosažitelnosti**.

## „**Safety**” vlastnosti

- Vlastnost  $\varphi$ , pro které je  $A_{\neg\varphi}$  terminální BA.
- Typická slovní formulace: „Nenastane špatná událost.”
- Pro verifikaci stačí analýza dosažitelnosti.

## $A_{\neg\varphi}$ je slabý Büchi automat

- Neobsahuje smíšené komponenty.
- Pro důkaz existence akceptujícího cyklu stačí prokázat dosažitelnost cyklu v akceptující komponentě.
- Lze detekovat pomocí jednoduchého DFS.
- Zním optimální algoritmus, který nevyžaduje DFS.

## „Slabé” LTL vlastnosti

- Vlastnost  $\varphi$ , pro které je  $A_{\neg\varphi}$  slabý BA.
- Typická vlastnost je „response”:  $G(a \implies F(b))$

## Klasifikace

- Každá LTL formule patří do jedné z následujících tříd: Reactivity, Recurrence, Persistence, Obligation, Safety, Guarantee

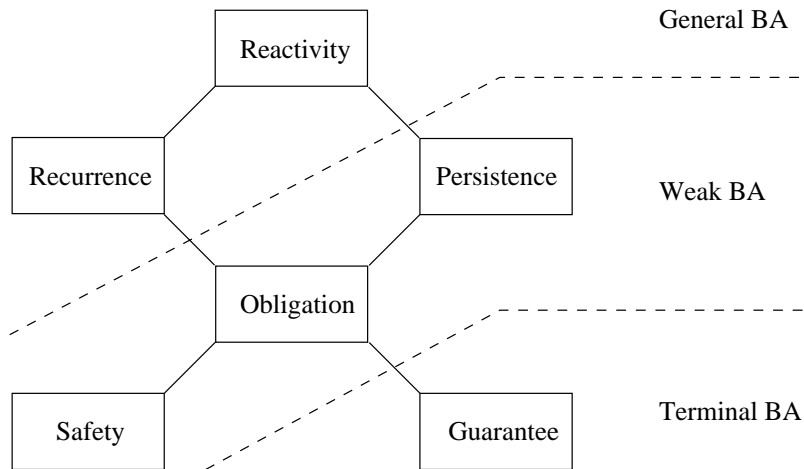
## Zajímavá fakta

- Je-li vlastnost ze třídy **Guarantee**, je popsatelná terminálním Büchi automatem.
- Je-li vlastnost ze tříd **Persistence**, **Obligation** nebo **Safety** je popsatelná slabým Büchi automatem.

## Při verifikaci se formule neguje ( $\varphi \mapsto A_{\neg\varphi}$ )

- $\varphi \in \text{Safety} \iff \neg\varphi \in \text{Guarantee}.$
- $\varphi \in \text{Recurrence} \iff \neg\varphi \in \text{Persistence}.$

# Klasifikace LTL formulí



## Boj se stavovou explozí

## Co je to stavová exploze

- Systém bývá popsán jako **paralelní kompozice procesů**.
- Vzájemným proložením možných chování jednotlivých procesů vznikají různé možné stavy systému jako celku.
- Počet dosažitelných stavů systému může být až **exponenciálně větší** než součet počtu stavů procesů.

## Důsledek

- Do operační paměti počítače nezle uložit všechny stavy produktového automatu.
- Je obtížně detekovat přítomnost akceptujícího cyklu.

## **Kompresa stavových vektorů**

- Neztrátové komprese
- Ztrátové – heuristiky

## **On-The-Fly verifikace**

## **Symbolická reprezentace stavového prostoru**

## **Redukce počtu stavů produktového automatu**

- Redukce zaváděním atomických bloků
- Redukce částečným uspořádáním akcí
- Redukce symetrií

## **Paralelní/Distribuovaná verifikace**

## Pozorování

- Graf lze zadat pomocí funkcí (tzv. implicitní definice)
  - $F\_init()$  — Vrací iniciální vrchol grafu.
  - $F\_succs(s)$  — Pro daný vrchol vrací jeho přímé následníky.
  - $Accepting(s)$  — O stavu řekne, zda je či není akceptující.

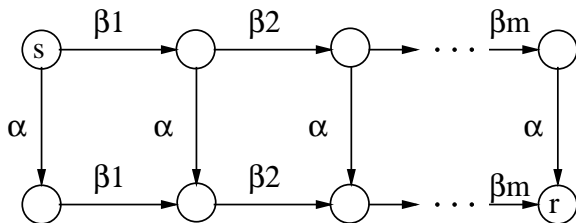
## On-the-fly verifikace

- Pokud graf obsahuje akceptující cyklus, algoritmus je schopen v některých případech detekovat přítomnost akceptujícího cyklu, aniž by přitom navštívil (a tedy uložil) všechny vrcholy grafu.
- Problém  $\mathcal{M} \models \varphi$  je někdy možné rozhodnout bez nutnosti úplné enumerace vrcholů a hran automatu  $A_{sys} \times A_{\neg\varphi}$ .
- Metoda verifikace s výše popsanou vlastností se označuje jako **on-the-fly** metoda verifikace.

## Příklad

- Uvažme systém tvořený dvěma procesy  $A$  a  $B$ .
- $A$  je tvořen jednou akcí  $\alpha$ ,  $B$  sekvencí akcí  $\beta_1, \dots, \beta_m$ .

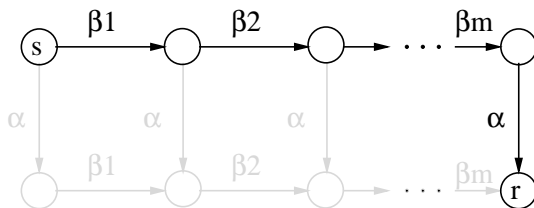
## Neredukovaný stavový prostor:



## Vlastnost: Je dosažitelný stav $r$ ?

## Pozorování

- Běhy  $(\alpha\beta_1\beta_2 \dots \beta_m)$ ,  $(\beta_1\alpha\beta_2 \dots \beta_m)$ ,  $\dots$ ,  $(\beta_1\beta_2 \dots \beta_m\alpha)$  jsou vzhledem k dané vlastnosti ekvivalentní.
- Je dostačující zvážit pouze jednoho reprezentanta z třídy ekvivalence, například  $(\beta_1\beta_2 \dots \beta_m\alpha)$ .



- Reprezentanta lze získat odkládáním akce  $\alpha$ .

## Princip redukce

- Při generování grafů se místo funkce všech přímých následníků, uvažují pouze někteří přímí následníci vrcholu.
- V důsledku toho je počet vygenerovaných stavů produktového automatu menší.

## Technická realizace

- Optimální způsob výběru následníků je obtížný.
- Nástroje implementují různé heuristiky.
- Redukovaný stavový prostor musí zachovávat přítomnost akceptujícího cyklu.
- Formule nesmí obsahovat operátor  $X$  (podtřída *LTL*).

## Anglický název

- Partial Order Reduction

## Princip

- Nesnaží se zmenšit paměťové nároky výpočtu.
- Snaží se efektivně využít větší množství výpočetních zdrojů.

## Problémy algoritmu Nested DFS

- Algoritmus přistupuje operační paměť velmi náhodně. Prosté swapování na disk nefunguje (výprask OS).
- Simulace Nested DFS algoritmu v prostředí s distribuovanou pamětí je pomalá (předávání peška).
- Není znám způsob jak DFS post-order napočítat efektivně paralelním algoritmem. (Otevřený problém.)

## Pozorování

- Místo Nested DFS algoritmu se používají jiné (časově neoptimální) algoritmy, jejichž výpočet ale lze dobře paralelizovat.

	Složitost	Optimalita	On-The-Fly
<b>Nested DFS</b>	$O(V+E)$	Ano	Ano
<b>OWCTY</b>			
obecné Büchi automaty	$O(V.(V+E))$	Ne	Ne
slabé Büchi automaty	$O(V+E)$	Ano	Ne
<b>MAP</b>	$O(V.V.(V+E))$	Ne	Částečně
<b>OWCTY+MAP</b>			
obecné Büchi automaty	$O(V.(V+E))$	Ne	Částečně
slabé Büchi automaty	$O(V+E)$	Ano	Částečně