

IV101 – SMV

Jiří Barnat, David Šafránek, Vojta Řehák

SMV – Symbolic Model Verifier

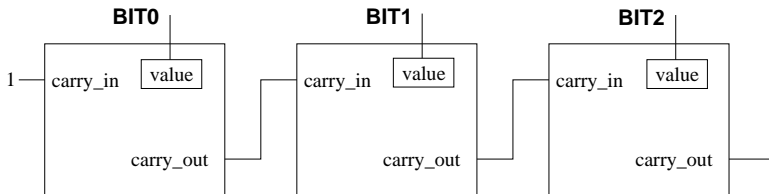
SMV

- nástroj pro symbolické ověřování modelů
- nejpoužívanější model-checker pro verifikaci návrhu hardware
- vlastní modelovací jazyk – SMV language
- první verze SMV [McMillan@CMU 1992]
- několik různých větví
 - CMU SMV – původní SMV, aktuální verze 2.5.4.3 (2001)
www.cs.cmu.edu/modelcheck/smv.html
 - Cadence SMV – Cadence Berkeley Lab
základ pro komerční tool Incisive Formal Analysis
www.cadence.com
 - NuSMV – ITC-IRST, aktuální verze 2.4.3 (05/2007)

Základní vlastnosti

- vhodný (nejen) pro popis logických obvodů
- blízký k Hardware Description Languages (Verilog, VHDL)
 - ale abstrakce – nulové zpoždění přenosu signálu
 - SMV odpovídá Synchronous Verilog
- logický obvod = kombinační logika (signály)
+ sekvenční logika (registry)
- stav obvodu = aktuální hodnoty registrů
- signály lze chápat jako “nestavové” proměnné

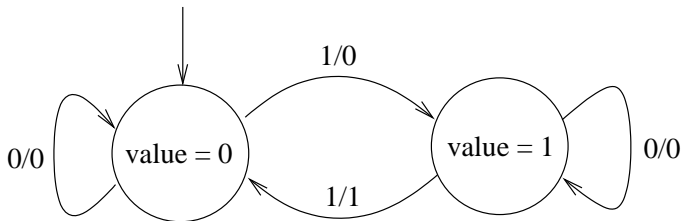
Příklad logického obvodu – 3bitový čítač – I



- pro každou buňku:
 - value – stavová proměnná
 - carry_in, carry_out – signály

Příklad logického obvodu – 3bitový čítač – II

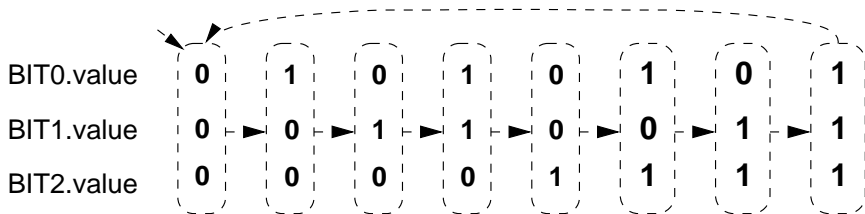
Mealy automaton popisující i -tou buňku:
(vstup – carry_in, výstup – carry_out)



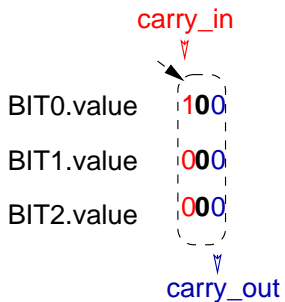
Příklad logického obvodu – 3bitový čítač – III

- algebraický popis chování i -té buňky čítače:
 1. změna stavové proměnné `value`
$$\text{next}(\text{value}) := (\text{value} + \text{carry_in}) \bmod 2$$
 2. výpočet hodnoty výstupního signálu `carry_out` (invariantní!)
$$\text{carry_out} := \text{value} \& \text{carry_in}$$
- chování 3bitového čítače je synchronním produktem všech tří komponent

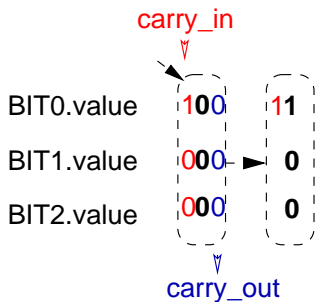
Synchronní kompozice buněk 3bitového čítače



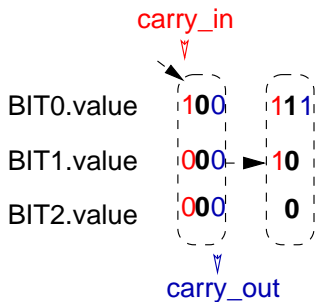
Synchronní kompozice buněk 3bitového čítače



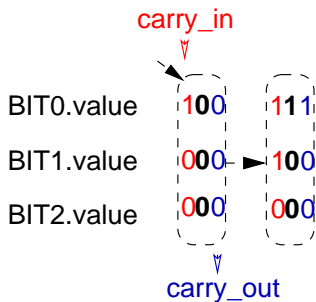
Synchronní kompozice buněk 3bitového čítače



Synchronní kompozice buněk 3bitového čítače



Synchronní kompozice buněk 3bitového čítače



Popis modelu v SMV

- globální proměnné
- moduly – parametrizované typy procesů
- proces = instance modulu
- hlavní modul `main`
 - auto-instanciacce
 - bez formálních parametrů
- paralelní kompozice konečně mnoha procesů
 - staticky definováno v `main`
 - moduly implicitně instanciovány jako synchronní
 - možno explicitně instanciovat jako asynchronní
- na úrovni jazyka nejsou kanály
 - komunikace přes sdílené proměnné
 - propojení procesů
 - hodnoty se předávají odkazem

Struktura modelu v SMV

- definice modulů (nezáleží na pořadí)

MODULE *název_modulu*(výčet_vstupních_signálů)

- v rámci modulu:

sekce VAR

- definice stavových proměnných
- definice procesů (instanciace modulů)

sekce ASSIGN

- definice přechodové relace (nad stavovými proměnnými)

sekce DEFINE

- definice signálů

sekce SPEC

- definice vlastností v CTL (typicky v main)

Zápis buňky 3bitového čítače v SMV

```
MODULE counter_cell(carry_in)
VAR
    value : boolean;
ASSIGN
    init(value) := 0;
    next(value) := value + carry_in mod 2;
DEFINE
    carry_out := value & carry_in;
```

Zápis celkového modulu 3bitového čítače v SMV

```
MODULE main
```

```
VAR
```

```
    bit0 : counter_cell(1);
```

```
    bit1 : counter_cell(bit0.carry_out);
```

```
    bit2 : counter_cell(bit1.carry_out);
```

Sekce VAR

- deklarace stavových proměnných

```
název_proměnné : typ;
```

- instanciacie synchronních procesů

```
VAR
```

```
název_instance : název_modulu(params);
```

- instanciacie asynchronních procesů

```
VAR
```

```
název_instance : process název_modulu(params);
```

Typy stavových proměnných

- `boolean` – logické hodnoty 0,1
- $\{ val_1, val_2, \dots, val_n \}$ – skalár
 - $val_1 \dots val_n$ jsou symbolické konstanty
např. `state : { idle, running, sleeping }`
- `array` $expr_1 .. expr_2$ of typ
 - $expr_1, expr_2$ celočíselné výrazy
 - typ je libovolný typ (nebo modul)
- celočíselné hodnoty reprezentovány jako `array of boolean`

Výrazy – I

- atomy: celočíselné konstanty, proměnné, signály
- operátory: *, /
 - + , -
 - mod
 - =, !=, <, >, <=, >=
 - !
 - &
 - |
 - >, <->

Výrazy – II

- přístup k prvkům pole pomocí “[]”
 - pole lze zanořovat (vícerozměrná pole, submoduly)
- přístup k proměnným procesu “.”

VAR

```
pole : array 0..5 of boolean;  
proces : A( 0, 2 );
```

ASSIGN

```
proces.x := 2; -- x proměnná v A  
pole[5 - x] := 0;
```

- hodnota proměnné x v následujícím stavu – next(x)

```
next(a) := { a, a+1 } ;  
next(b) := b + (next(a) - a) ;
```

Výrazy větvení

- výraz case

```
case
     $expr_{a1}$  :  $expr_{b1}$  ;
     $expr_{a2}$  :  $expr_{b2}$  ;
    ...
     $expr_{an}$  :  $expr_{bn}$  ;
esac
```

- $expr_{a1} \dots expr_{an}$ hodnoty typu boolean
- chová se deterministicky, vrací hodnotu prvního (od shora) výrazu $expr_{bi}$ takového, že hodnota $expr_{ai}$ je 1
- není-li žádný $expr_{ai}$ splněn, vrací 1

Množinové výrazy

- $\{ val_1 , \dots , val_n \}$ kde val_i je symbolická nebo číselná konstanta
- $expr_1$ in $expr_2$ – test na příslušnost prvku
- $expr_1$ union $expr_2$ – sjednocení

$running$ union $sleeping \equiv \{ running, sleeping \}$

$\{ running, sleeping \}$ union $idle \equiv \{ running, sleeping, idle \}$

Sekce ASSIGN

- soubor přiřazení tvaru
stavová_proměnná := výraz;
- inicializace proměnné x
init(x) := 0;
- určení hodnoty proměnné x v následujícím stavu
next(x) := x + 1;
- výraz může být množina – nedeterministické přiřazení

Sekce ASSIGN

- prováděna paralelně – nezáleží na pořadí přiřazovacích příkazů
- pokud neexistuje alespoň jedno konsistentní uspořádání SMV zahlásí sémantickou chybu
- konsistentní uspořádání = referovat lze jen již dříve přiřazenou/iniciovanou proměnnou
- v SMV nelze napsat neimplementovatelný model

Sekce DEFINE

- soubor definic signálů tvaru
název_signálu := výraz;
- signály jsou nestavové proměnné
- signály nejsou staticky typované
- definice provedeny paralelně
- sémantická kontrola implementovatelnosti podobně jak v
ASSIGN

Nedeterminismus

```
MODULE main
```

```
VAR
```

```
    bit0 : counter_cell({0, 1});
```

```
    bit1 : counter_cell(bit0.carry_out);
```

```
    bit2 : counter_cell(bit1.carry_out);
```

- systém má více možných chování
- zdroje nedeterminismu:
 - přiřazení z množiny
 - neinicializovaná proměnná
 - asynchronní procesy

Specifikace konečných automatů

```
MODULE proces
```

```
VAR
```

```
    state : { idle, entering, critical, exiting };
```

```
ASSIGN
```

```
    init(state) := idle;
```

```
    next(state) := case
```

```
        state = idle : { idle, entering };
```

```
        state = entering : critical;
```

```
        state = critical : { critical, exiting };
```

```
        state = exiting : idle;
```

```
    1 : state;
```

```
esac;
```

Asynchronní procesy

- každý modul lze instanciovat jako asynchronní proces
- asynchronní procesy spouštěny nedeterministicky (interleaving)
- asynchronní proces má definovanou proměnnou `running`
 - $A.running = 1 \Leftrightarrow$ proces A provádí přechod
 - sekce FAIRNESS v modulu X deklaruje, že uvedená podmínka musí nastat za život každé instance (procesu) nekonečněkrát

FAIRNESS

`running`

Alternativní způsob modelování – omezení (constraints)

Následující model

```
MODULE main
VAR request : boolean;
    state : ready,busy;
ASSIGN
    init(state) := ready;
    next(state) := case
        state = ready & request : busy;
        1 : ready,busy;
    esac;
```

Alternativní způsob modelování – omezení (constraints)

lze alternativně zapsat takto:

```
MODULE main
  VAR request : boolean;
      state : ready,busy;
  INIT
    state = ready
  TRANS
    (state = ready & request) -> next(state) = busy
```

Verifikace

Specifikace vlastností – I

- sekce SPEC
 - umožňuje zadat temporální podmínku, kterou musí **všechna** chování systému splňovat (chování ze všech iniciálních stavů)
 - SMV ověří, že podmínka je splněna
 - není-li splněna, vrátí protipříklad (jde-li)
 - více deklamací SPEC se bere konjunktivně
-
- SPEC – CTL specifikace
 - LTLSPEC – LTL specifikace
 - INVARSPEC – Invariant

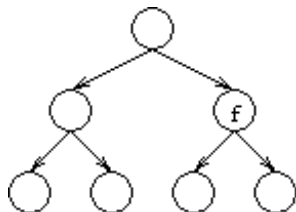
Specifikace vlastností – II

- vlastností spjaté se stavovými proměnnými určitého modulu se typicky uvádějí v rámci definice modulu
- globální vlastnosti se uvádějí v modulu `main`
- jsou-li definovány FAIRNESS pak se procházená chování omezují na ta, která splňují všechny definované fairness podmínky

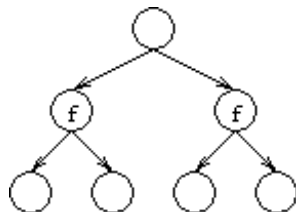
CTL – Temporální logika větvičího se času

- atomické propozice
 - výrazy, které je možné vyhodnotit nad daným stavem systému s výsledkem 0 nebo 1
 - `x = 0, proc1.state = sleeping`
 - nelze použít výraz `next(.)`
- boolovské operátory
 - `!, &, |, ->, <->`
- temporální operátory
 - `until (U), globally (G), eventually (F), next (X)`
- kvantifikátory cest (CTL)
 - existenční E, universální A
 - vždy aplikovány na podformuli s temporálním operátorem na nejnižší úrovni zanoření

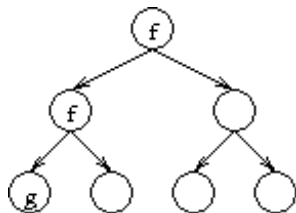
Sémantika CTL – I



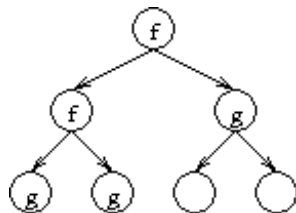
EX f



AX f

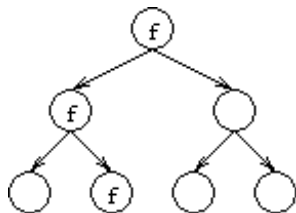


E (f U g)

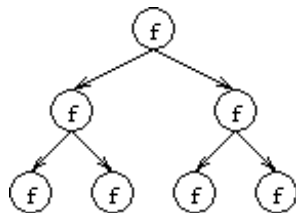


A (f U g)

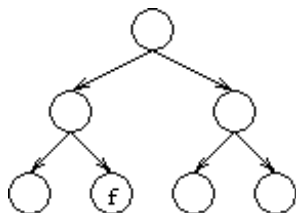
Sémantika CTL – II



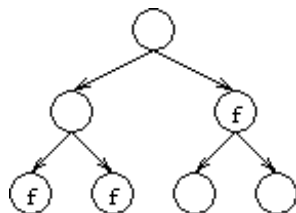
EG f



AG f



EF f



AF f

Ukázka práce s NuSMV