IV101 — SPIN + ProMeLa

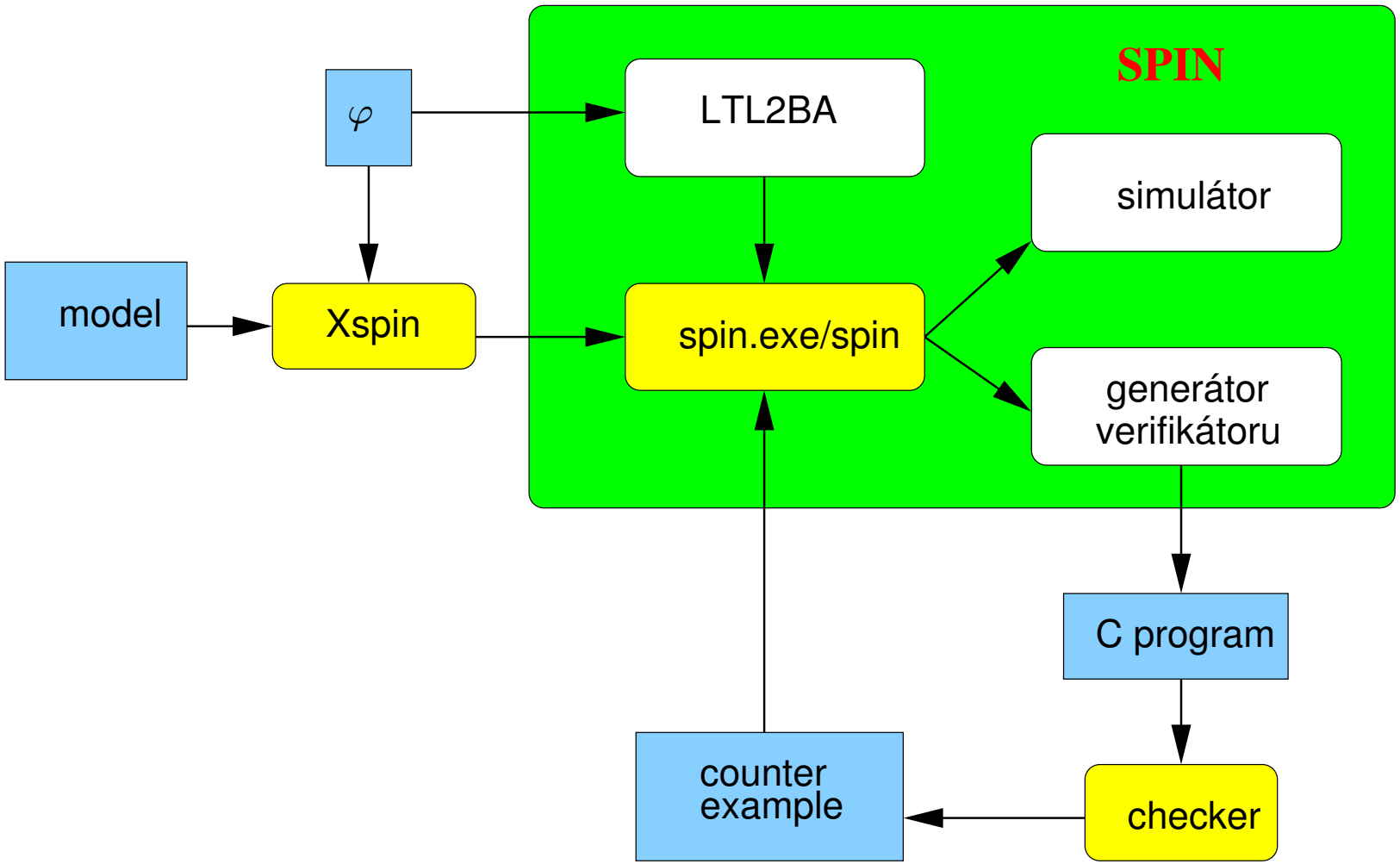
Jiří Barnat

SPIN – Simple Promela INterpreter

SPIN

- Nástroj pro analýzu logické konzistence souběžných systémů a komunikačních protokolů.
- Nejcitovanější(nejpoužívanější) model-checker.
- Vlastní modelovací jazyk – PROMELA.
- První veřejná verze `SPIN` u [Holzmann 1991].
- Současná verze: 5.1.7
- www.spinroot.com

Architektura SPINu



PROMELA – PROcess MEta LAnguage

Popis systému

- paralelní kompozice konečně mnoha procesů
- globální proměnné
- bafry (obsah komunikačních kanálů)

Proces v systému

- každý proces je konečně stavový
- procesy mění svůj stav prováděním akcí
- prokládání akcí různých procesů (interleaving)
- meziprocesová komunikace
  - sdílené proměnné
  - synchronní komunikace (handshake)
  - asynchronní komunikace (bafrované kanály)
- dynamická správa procesů
  - vytváření nových instancí procesů za běhu
  - více procesů stejného typu
  - pid procesu (`_pid`)
  - počet procesů je omezen
  - vytváření a ukončování procesů je typu LIFO

Struktura modelu

- definice typů proměnných
- definice globálních proměnných
- definice komunikačních kanálů
- definice typů procesů
- instanciace procesů
- (iniciální proces)

Koncept proveditelnosti

- každý příkaz/výraz je v kontextu globálního stavu systému buď *proveditelný* nebo *neproveditelný*
- výraz je proveditelný pokud je jeho hodnota pravdivá (nenulová)
  - `x==x`, `x-3`, `x>0`, `z=x+1`
  - příkaz přiřazení je vždy proveditelný (ne jako v C)
- speciální výraz `timeout`
  - proveditelný pokud není proveditelná jiná akce

Proměnné a typy

- základní typy
  - bit, bool, byte, short (16 bitů), int (32 bitů)
  - inicializovány na hodnotu `0`
  - `bit turn=1;`
- složené typy
  - klíčové slovo `typedef`
  - `typedef mujtyp { byte x; bit y; }`
    `mujtyp z;`
    `z.x = ...`
- pole
  - jednorozměrná
  - vícerozměrná s využitím složených typů
  - `byte b[17]; b[3]=7;`

Komunikační kanály

- definice pomocí klíčového slova `chan`
- synchronní komunikace
  - `chan jmeno_kanalu = [0] of {short,byte}`
- asynchronní komunikace
  - `chan jmeno_kanalu = [3] of {short,byte}`
- jedna zpráva může obsahovat více hodnot

Práce s asynchronnímy komunikačními kanály – 1

- kanály jsou typu First-In-First-Out (FIFO)
- posílání
  - klíčové slovo !
  - proveditelné pokud kanál není plný
  - pošle *jednu* zprávu (počet zpráv v bafru se zvýši o 1)

```
jmeno_kanalu!  výraz1, výraz2, ..., výrazN
jmeno_kanalu!  -2, 17
```

- přijímaní
  - klíčové slovo ?
  - proveditelné pokud je kanál neprázdný
  - přijme *jednu* zprávu (odstraní zprávu z bafru)

```
jmeno_kanalu?  z1, z2, ..., zN
jmeno_kanalu?  x,y
```

Práce s asynchronnímy komunikačními kanály – 2

- testování typu zprávy
  - klíčové slovo ?
  - otestuje typ následující přijaté zprávy
  - neodstraní zprávu z bafru

    ```
    jmeno_kanalu?  konst1, konst2, ..., konstN
    jmeno_kanalu?  -2, 17
    ```

  - proveditelné pokud je kanál neprázdný a následující prijímaná zpráva se shoduje v obsahu zprávy

- podmíněné přijetí zprávy
  - kombinace přijímání zprávy a testování obsahu zprávy

    ```
    jmeno_kanalu?  -2,y
    ```

- test obsazenosti kanálu (lze použít i jako atomické propozice)
  - je kanál prázdný?, neprázdný?, plný?

    `empty(ch)`, `nempty(ch)`, `full(ch)`
  - obsazenost kanálu (počet zpráv)

    `len(ch)`

Práce se synchronnímy komunikačními kanály

- komunikační kanál nedrží žádnou hodnotu
- realizuje *handshake*
- nutná participace dvou různých procesů
- obě duální akce `!` a `?` musí být proveditelné
- oba procesy změní svůj stav

synchronní kanál:	`chan q = [0] of {short,bit}`
proces `P`:	`q!3+7,false`
proces `Q`:	`q?x,false`

Strukturní příkazy – větvení

- klíčová slova `if` a `fi`
- jednotlivé možnosti větvení označeny `::`
- podmínka větvení
  - dána proveditelností prvního příkazu/výrazu za `::`
  - `else` – proveditelné pokud není proveditelná jiná větev
  - je zvykem používat oddělovač `->` místo standardního `;`
- nedeterminismus
  - z několika proveditelných větví se vykoná právě jedna

```
if
::  x>=0 -> x=0
::  x==1 -> x=x-1
::  x==0 -> x++
::  else -> x=-x
fi
```

Strukturní příkazy – cyklus

- opakované provádění volby `if–fi`
- klíčová slova `do` a `od`
- opakování je ukončeno klíčovým slovem `break`
- pokud ani jedna větev není proveditelná, není proveditelná ani konstrukce `do–od` (platí i pro jednoduché větvení `if–fi`)
- ```
  do
  ::  x>0 -> x++
  ::  x==0 -> break
  od
  ```
  ```

Strukturní příkazy – unless

- `{ blok1 } unless { blok2 }`
- postupně vykonává příkazy bloku `blok1` dokud není proveditelný první příkaz bloku `blok2`, jakmile je proveditelný první příkaz bloku `blok2`, vykonávání příkazů z bloku `blok1` je přerušeno a pokračuje se vykonáním příkazů bloku `blok2`
- simuluje *exception handling*

Strukturní příkazy – návěští a nepodmíněné skoky

- návěští – platná posloupnost znaků zakončená `:`
  - `aha:`, `@takhle#tedaňe:`
- příkaz `goto navěští`
  - následující vykonaný příkaz, je příkaz za návěštím
  - návěští musí být v rámci procesu
- každý příkaz/výraz může mít návěští
- návěští nelze umístit před volbu větve (`::`)

Strukturní příkazy – atomické sekvence

- atomicita posloupnosti příkazů redukuje velikost stavového prostoru
- `atomic {blok}`
  - příkazy bloku jsou provedeny jako jeden příkaz
  - blok může obsahovat nedeterminismus
  - mezistavy jsou generovány a ukládány
- `d_step {blok}`
  - příkazy bloku jsou provedeny jako jeden příkaz
  - blok nesmí obsahovat nedeterminismus
  - mezistavy nejsou ukládány

Typ procesu

- definován klíčovým slovem `proctype`
- má své unikátní jméno
- seznam formálních parametrů
- deklarace lokálních proměnných
- definice těla procesu

```
proctype P(chan in; chan out) {
int x;
do
  ::  x=0 -> in?x
  ::  x!=1 -> out!x
od
}
```

Spuštění procesu

- počet procesů je omezen
- spuštění procesu za běhu systému
  - klíčové slovo `run`

```
run P(ch1,ch2)
```

- procesy spuštěné iniciálně
  - klíčové slovo `active`

```
active proctype P(){...}
active [3] proctype P(){...}
```

  - spustit za běhu z procesu `init`

```
init {
run P(ch1,ch2);
run P(ch2,ch1)
}
```

```
init {
atomic { run P(ch1,ch2);
         run P(ch2,ch1) }
}
```

Ostatní konstrukce jazyka

- speciální výčtový typ `mtype`

  ```
  mtype = {pondeli, utery, sobota}
  ```

- komunikace přes `stdin/stdout`
  - výstup pomocí `printf`

    ```
    printf ("Hello world!  (%d)",x)
    ```

  - vstup pomocí předdefinovaného kanálu `STDIN`

    ```
    STDIN?c; if ::c==-1->break /*EOF*/ ::else->skip; fi
    ```

  - má význam pouze při simulaci
- prázdný proveditelný příkaz `skip`
- kompatibilita s preprocesorem cpp
- speciální macro `inline`

C v ProMeLe

- `SPIN` od verze 4
- podpora pro vložený kód v jazyce C
- kód přímo zakomponován do verifikátoru (`pan`)
  - bezpečnostní rizika
  - nutné detailnější pochopení funkce `pan`
- klíčová slova
  - `c_code`, `c_expr`, `c_decl`, `c_state`, `c_track`
- proveditelnost
  - `c_keyword { embedded C }`
  - `c_keyword [guard/assertion] { embedded C }`

C v ProMeLe

- `c_code`
  - vkládá kód v C
  - struktury ve stavu dostupné přes proměnnou `now`
  - předpokládá se, konečnost provádění vloženého C
  - `c_code [Pex->ptr != 0 && now.i < 10 && now.i >= 0] { Pex->ptr.x[now.i] = 12; }`
- `c_expr`
  - na vložený kód se nahlíží jako na výraz
  - `c_expr [Pex->ptr != NULL] { Pex->ptr->y }`
- `c_decl`
  - deklarace datových typů v C
  - definice proměnných definovaných typů pomocí `c_code`

C v ProMeLe

- `c_state`
  - `c_state` S1 S2 [S3]
  - vložení deklarace do stavu (`now`)
  - S1: typ a jméno datového objektu
  - S2: mód vložení (Global, Local where, Hidden)
  - S3: iniciální hodnota
  - ```
    c_state "Rendez R1" "Global"
    c_state "Rendez R2" "Local ex2" "now.R1"
    c_state "extern Proc H1" "Hidden"
    ```
- `c_track`
  - `c_track` S1 S2
  - vyhrazuje v `now` místo o velikosti S2 pro strukturu na adrese S1
  - `c_track "&RR" "sizeof(Rendez)"`
  - kopíruje se z a do `now` při každém přechodu
    ```

Verifikace

ProMeLa a verifikace – 1

- klíčové slovo `assert`
  - umožňuje zadat podmínku, která má platit v daném okamžiku
  - SPIN ověří, že žádný `assert` nebyl porušen

```
do
::  x++; assert (x>0)
od
```

- prefixy návěští
  - `end`
    - slouží k označení platných koncových stavů
    - SPIN umí najít neplatné koncové stavy
  - `progress`
    - SPIN umí hledat neprogresivní cyckly
  - `accept`
    - SPIN umí hledat akceptující cyckly

ProMeLa a verifikace – 2

- proces `never`
  - vykazuje synchronní chování vzhledem ke zbývající části systému
    - s každou akcí systému se provede jedna akce procesu `never`
    - není-li žádná akce proveditelná stav systému nemá následníky
- použití procesu `never` při verifikaci
  - vyjadřuje špatná chování systému
  - lze detekovat s využitím detekce akceptujících cyklů
  - může být vygenerován pro každou formuli LTL

LTL – Lineární Temporální Logika

- atomické propozice
  - výrazy, které je možné vyhodnotit na daným stavem systému
  - `x==0`, `P.label`
- boolovské operátory
  - $\wedge$, $\vee$, $\rightarrow$, !
- temporální operátory
  - until (`U`), release (`V`), globally (`[]`), eventually (<>)

Sémantika temporálních operátorů LTL

- until ($\varphi U \psi$)
  - $\varphi \rightarrow \varphi \rightarrow \cdots \rightarrow \varphi \rightarrow \wedge\psi$
- release ($\varphi V \psi$)
  - $\varphi V \psi \equiv \neg(\neg\varphi U \neg\psi)$
  - $\psi \rightarrow \psi \rightarrow \psi \rightarrow \cdots$
  - $\psi \rightarrow \psi \rightarrow \cdots \rightarrow \psi \rightarrow \psi \wedge \varphi$
- globally ($[]\varphi$)
  - $\varphi \rightarrow \varphi \rightarrow \varphi \rightarrow \cdots$
- eventually ($\langle\rangle\varphi$)
  - $\rightarrow\rightarrow\rightarrow \cdots \rightarrow \varphi$

Ukázka práce se SPINem

Architektura SPINu