1 Modelování

Jako první se sluší zmínit, že model Publish/Subscribe protokolu má jediný volitelný parametr *numUsers*, který odpovídá počtu uživatlů.

Modelovací jazyk DiViNe obsahuje nejdůležitější primitiva potřebná při modelování komunikujících procesů, některé konstrukce obsažené (například) ve Spinu zde však chybí. Jedná se především o rozhodování dle typu zprávy (konstanty) přijaté z nějakého kanálu, tedy o výrazy typu

```
if
    ::ch?get -> skip;
```

kde *get* je zmiňovaná konstanta. Takovéto chování lze částečně nasimulovat pomocí proměnné, do které se načte hodnota z kanálu a v dalším kroku se rozhodne do kterého stavu přejít.

```
q1 -> q2 {sync ch?x;},
q2 -> get {guard x == get;},
```

Tato simulace je však jen částečně věrná, protože tím vzniká nový mezistav, kdy odesílající proces si myslí, že zpráva již byla odeslána, ale příjmající proces zatím nemusí zprávu číst. Pokud by měl tento mezistav korektně (i když za cenu nárůstu počtu stavů) modelovat zmíňenou situaci, bylo by potřebné zastavit ostatní procesy a tím také donutit příjmající proces zprávu přečíst. Toho však lze dosáhnout jen velmi těžko. Při modelování Publish/Subscribe protokolu se naštěstí v tomto ohledu nevyskytly problémy. Pouze v jednom případě (userAdmin.UpdateNotify) bylo nutné přidat další synchronizaci (kanál msgSync) pro již zmíněné zablokování ostatních procesů tak, aby výsledný model odpovídal specifikaci dané LTL formulemi. Je však nutné mít na paměti, že shodného a v jistém ohledu ani podobného modelu nelze bez této konstrukce dosáhnout. Z toho také vyplývá fakt, že výsledný systém má mnohem více stavů (více možných proložení procesů).

Mezi další chybějící prvky patří strukturované dat. typy (zprávy), které jsem však nahradil dvojicí, tedy rozdělením typu `byte` na 2 části. Dále bylo třeba nahradit konstrukce *assert*. Na její místo jsem zařadil do výsledného modelu větvení umožňující jak správný tak chybný přechod a připojil jsem formuli, která zjistí, zda je možné spustit nesprávný přechod. Nakonec lze ještě zmínit absenci konstrukce `init`, což však u tohoto modelu nevadí, protože i v originále jsou všechny procesy spuštěny v jednom kroku.

Výsledný model tedy na jednu stranu odpovídá specifikaci (LTL) a v jistém smyslu i originálu napsanému v ProMeLe, na stranu druhou se liší a to především kvůli zmiňovaným problémům s kanály a konstantami. Protože navržená simulace vedla k velkému množství nových stavů, rozhodl jsem se tento stav mírně napravit a to tak, že některé mezistavy původního modelu jsem vyloučil. Jedná se zejména o případy, kdy dochází ke komunikaci a zároveň změně lokální proměnné. Např:

```
registered[0] = true;
userToCC!register, id;
```

bylo možné modelovat jako

```
ready -> doneRegister { sync userToCC!pair(register, id);
                        effect register[0] = true; },
```

Zdůrazňuji, že tento postup byl použit pouze na lokální proměnné (globální, sdílené v modelu nejsou).

Protože na modelu se 3 uživateli jsem nebyl schopný provést distrib. reachabilitu ani na 20 počítačích, snažil jsem se dále nějakým způsobem snížit počet stavů. Použil jsem kanál msgSync pro další synchronizaci při čtení zpráv (problémy s konstantami) nejen v procesu userAdmin, který již byl zmiňován. Tak vznikla mírně odlišná 2. verze, která je v souboru `ps2.mdve`. Ačkoliv se počet stavů u modelu se 2 uživateli snížil (z 1846603 na 1676307), ani v tomto případě jsem nebyl schopen model se 3 uživateli prohledat.

2 Problémy s verifikací

Při verifikaci formulí specifikujících požadované vlastnosti modelu jsem získal pochyby o správnosti formulí pojmenovaných *No illegal notify (update)*. Protože jsou obě formule identické až na záměnu notify za update a chování procesů je taktéž velmi podobné, budu dále psát jen o formuli s notify. Její znění přejímám z originálu:

$$!(!(g \vee r)\, U\, n) \wedge \Box(u \rightarrow !(!(g \vee r)\, U\, n))$$

#define	g	(User[3]@doneNotify)
#define	r	(User[3]@doneRegister)
#define	u	(User[3]@doneUnRegister)
#define	n	(UserAdmin[4]@doneNotify)

Autoři článku tvrdí, že tuto formuli lze zverifikovat, při mých pokusech však Spin označil tuto formuli za neplatnou. K vysvětlení problému je nutné ještě uvést význam at. predikátů uvedené formule. Zápis User[3]@doneGet říká, že proces User, jehož PID je 3, je ve stavu označeném návěštím *doneGet*. Stopa, kterou Spin vrátí, ukazuje, že k porušní formule dojde tak, že proces UserAdmin obdrží notifikaci (tedy přejde do stavu *doneNotify*) a tam zůstane. Následně proces User provede posloupnost příkazů vedoucích do stavu *doneUnRegister*, čímž dojde k porušení druhé části formule, kdy je v jednom okamžiku splněn jak User[3]@doneUnRegister, tak UserAdmin[4]@doneNotify. Řešením je dle mého názoru změna formule na:

$$!(!(g \vee r)\, U\, n) \wedge \Box(((n \wedge u) \vee (u \rightarrow !(!(g \vee r)\, U\, n))) \wedge !((!n \wedge u) \wedge ((!n \wedge u)\, U\, (n \wedge u))))$$

První část zůstává nezměněna. Ve druhé části je pomocí disjunkce s $(n \wedge u)$ vyloučen případ, kdy dojde k již zmiňovanému „chybovému“ stavu. Navíc přibyla ještě třetí část, která zajistí, aby nikdy nenastal případ, kdy proces User setrvá ve stavu *doneUnRegister* a současně proces UserAdmin obdrží notifikaci (což je chybové chování, které by nebylo detekováno díky změně ve druhé části formule). Případné připomínky týkající se této formule velmi rád uvítám.

3 Experimantální data

Pro představu přikládám velikosti jednotlivých produktových grafů pro model se 2 uživateli. Na jejich zjištění jsem použil sekvenčního BFS, tudíž neuvádím čas potřebný k prohledání. Velikost stavového vektoru je u všech variant stejná: 36 bytů.

formule	počet stavů
1	1977587
2	3691177
3	2643849
4	4058497
5	4042105
6	4712717
7	4718837
8	3033021
9	3033301
10	3436730
11	1846603

ps.mdve

formule	počet stavů
1	1807291
2	3350737
3	2393265
4	3695659
5	3681339
6	4250675
7	4255799
8	2727281
9	2727561
10	3141470
11	1676307

ps2.mdve