Alternating Bit Protocol

Cílem projektu je v CCS implementovat jednoduchý, ale efektivní protokol řešící přeposílání ztracených zpráv. Alternating-bit protokol pracuje následovně:

Odesílatel

- Obdrží zprávu na kanále *acc*
- Odešle ji s bitem 0 nebo 1. U první zprávy na bitu nezáleží (lze řešit nedeterminismem nebo např. natvrdo nastavit pro první zprávu bit 0), každá další má vždy bit jiný než ta předchozí.
- Čeká na potvrzení se stejným bitem. Pokud dorazí, je připraven obdržet (kanál acc) další zprávu a posílat ji (tedy opakuje se postup od první odrážky).
- Pokud dorazí potvrzení s jiným bitem, než čeká, přepošle zprávu znovu (se stejným bitem, na jehož potvrzení čekal). Přeposílá stejnou zprávu, tzn. nepřijímá novou z acc.
- Vypršení času čekání na potvrzení je simulováno tím, že při čekání na potvrzení je odesílatel připraven přijmout i příchozí zprávu *timer*. Na ni reaguje stejně, jako na potvrzení špatným bitem (tedy, přepošle znovu předchozí zprávu, předpokládá totiž její ztrátu cestou).

Příjemce

- Vždy, když obdrží zprávu s nějakým bitem, posílá potvrzení se stejným bitem.
- Pokud je bit opačný, než měla předchozí přijatá zpráva (jde tedy o novou zprávu a ne jen opakovaně přeposlanou předchozí), tak ji ještě před odesláním potvrzení předá dál na výstupním kanále *del*

Přenosové médium

- Samo o sobě není součástí alternating-bit protokolu. Slouží jen k jeho otestování tím, že budeme možná chování HW (kabely, routery, ...) i SW (ovladače síťových karet, firmware routerů,...) na síti mezi odesílatelem a příjemcem moci popsat CCS procesy a tak zjistit, jestli si s tím chováním protokol poradí nebo ne.
- Budeme uvažovat 3 různá média:
    - *Med1* – spolehlivé, každou zprávu, kterou dostane od odesílatele předá korektně příjemci, každé potvrzení přijaté od příjemce předá korektně odesílateli
    - *Med2* – ztrátové - u každé zprávy i každého potvrzení má nedeterministickou volbu (operátor + v CCS) zprávu/potvrzení doručit nebo ztratit. Ztráta znamená, že prostě něco medium přijalo a rovnou přešlo do stavu čekání na přijetí něčeho dalšího, aniž by proběhlo doručení.
    - *Med3* – ztrátové a duplikující - má stejné 2 možnosti jako ztrátové médium a navíc 3. možnost. Ta spočívá v tom, že obdrženou zprávu nebo potvrzení může v cyklu doručit libovolněkrát (např. jednou obdrží zprávu s bitem 0 od odesílatele a třeba 5x ji předá s bitem 0 příjemci a až potom převezme další případnou zprávu od odesílatele).
- Jednodušší je každou verzi média uvažovat jako dvě paralelní média, jedno pro směr od odesílatele k příjemci (na posílání zpráv - např. *Med1Mes, Med2Mes, Med3Mes*) a druhé pro směr od příjemce k odesílateli (na posílání potvrzení - např. *Med1Ack, Med2Ack, Med3Ack*).

Médium potom je prostě paralelní kompozicí těch dvou pomocných, např. *Med1=Med1Mes | Med1Ack;*

Timer

- Jen pomocný proces. Když neřešíme čas doručení zpráv, tak to, že reálně by vypršel limit čekání na zprávu lze řešit pomocí nedeterminismu. Tím se při testování ekvivalencí i formulí vyzkouší jak možnost, že čas mohl vypršet, tak možnost, že nevypršel.
- Definice tohoto procesu: *Timer = 'timer.Timer;*

Specifikace

- Navenek bychom čekali, že chování přenosového protokolu je takové, že mu prostě na jedné straně dáme zprávu a na druhé ji od něj spolehlivě obdržíme. Toto chování lze popsat následujícím CCS procesem:
    - *Spec = acc.'del.Spec;*

Implementace

- Reálnou implementaci protokolu vlastně tvoří jen odesílatel a příjemce. Tyto dva procesy máme proto taky vytvořené jen 1x a pro různé varianty médií je neměníme.
- Pro vyzkoušení spolehlivosti ale musíme zahrnout do systému i přenosové médium a pomocný timer.
- Implementace tedy budou následující:
    - *Impl1 = (Sender | Receiver | Med1 | Timer) \ X;*
    - *Impl2 = (Sender | Receiver | Med2 | Timer) \ X;*
    - *Impl3 = (Sender | Receiver | Med3 | Timer) \ X;*
    - kde *X* je množina zakázaných akcí, tedy těch, které mohou proběhnout jen jako synchronizace mezi procesy. V tomto zadání to typicky budou všechny, kromě *acc* a *del*
- Názvy procesů a akcí si můžete volit libovolně s výjimkou následujících, kde prosím o zachování zde uvedených názvů: *acc*, *del, timer, Timer, Spec, Impl1, Impl2, Impl3, Med1, Med2, Med3*. Takže pokud např. někomu bude vyhovovat mít procesy *Send0* a *Send1* pamatující si, který bit je aktuálně na řadě k posílání, tak není problém v implementaci místo *Sender* dát třeba *Send0*.

Verifikace

- Pro každou ze tří implementací ověřte, zda je slabě bisimulačně ekvivalentní se specifikací *Spec*.
- Popište formulí HM logiky s rekurzí vlastnost dosažitelnosti deadlocku (situace, kdy v systému nemůže proběhnout žádná akce) a otestujte tuto formuli pro všechny 3 implementace. Typicky deadlock nenastane tím, že by se dosáhl proces *Nil* ve všech paralelních komponentách, ale tím, že každá komponenta je sice schopna provést nějakou akci, jenže ty akce jsou zakázané restrikcí a každá komponenta je připravená na jiné akce, takže nemůže proběhnou ani synchronizace. Nápověda: pokud tam deadlock dosažitelný bude, zkontrolujte si, jestli to není tím, že vaše CCS svým chováním neodpovídají požadovanému chování popsanému výše.

- Popište formulí HM logiky s rekurzí vlastnost dosažitelnost livelocku (situace, kdy v systému mohou probíhat do nekonečna tau akce – systém tedy pracuje, ale navenek se tváří jako zaseknutý, protože nedělá žádné viditelné akce) a otestujte tuto formuli pro všechny 3 implementace. Nápověda: livelock může nastat např. tím, že jedna zpráva se bude pořád ztrácet a bude znovu přeposílána – tedy po *acc* se nikdy nedočkáme *del*.

Požadavky na vaše řešení:

- Výše uvedené CCS procesy popište v nástroji CAAL, všechny ve stejném projektu (proto mají varianty médií i implementací různé názvy, aby to mohlo být součástí jednoho CCS programu)
- V záložce "Verify" tohoto nástroje připravte vlastnosti pro výše zmíněné otestování slabé bisimilarity a dvou formulí HM logiky s rekurzí (3 druhy vlastností krát 3 různé implementace = 9 připravených řádků)
- Takto připravený projekt uložte do souboru (Project > Save > To File) a pošlete emailem na adresu: martin.kot@vsb.cz