Analiza protokołu MSMIE

Jarosław Bąbel

26 maja 2004

Spis treści

1 Wstęp

W systemach rozproszonych współpracę między elementami umożliwiają protokoły komunikacyjne. Opiszemy protokół oparty na komunikacji przez pamięć dzieloną (inne mogą bazować np. na przesyłaniu komunikatów). Będziemy analizować protokół stworzony przez „Westinghouse" używany jako część systemu komputerowego nadzorującego pracę reaktora atomowego (Sizewell B w UK).

Zbudujemy dwa modele systemu:

1. Naturalny, w którym elementy sprzętowe będziemy modelować bezpośrednio przez agentów

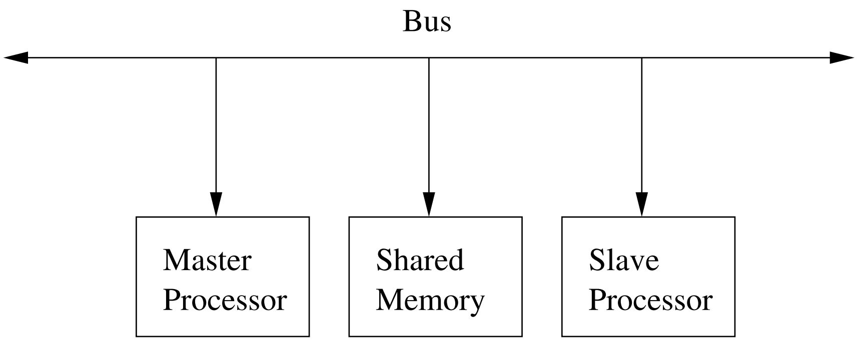
2. Abstrakcyjny bez struktury współbieżnej

Pokażemy, że te modele są w słabej bisymulacji. Następnie opiszemy oczekiwane właściwości protokołu w logice i zweryfikujemy je przy użyciu CWB. Pokażemy, że nie zachodzi jedna z ważnych własności. Przedstawimy propozycję poprawienia protokołu.

2 Opis protokołu MSMIE

MSMIE to skrót od Multiprocessor Shared-Memory Information Exchange. Jest to protokół do komunikacji pomiędzy procesorami w systemie czasu rzeczywistego. System, w którym ma działać protokół składa się z modułów. Każdy moduł posiada główny procesor wykonujący zadanie modułu i procesory pomocnicze odpowiedzialne za wejście i wyjście oraz diagnostykę. Czasem dodawane są dodatkowe główne procesory, żeby zwiększyć odporność na błędy lub uszkodzenia.

Komunikacja w module odbywa się przez pamięć dzieloną udostępnianą przez wspólną szynę. Każda komórka pamięci jest przeznaczona do komunikacji w jednym kierunku - procesor pomocniczy pisze do głównego, a główny czyta lub odwrotnie. Czytanie i pisanie jest asynchroniczne.



2.1 Wymagania

1. Uniknięcie odczytywania danych zapisanych częściowo (ang. data tearing). Zjawisko występuje w przypadku większych kawałków danych, które muszą być zapisane w kilku komórkach pamięci (tzw. buforze) w sposób nieatomowy.

2. Przerwy w komunikacji muszą być krótkie i o ustalonej długości (system czasu rzeczywistego). Np. przy użyciu semafora do kontroli dostępu, czytelnik musi czekać aż pisarz skończy i odwrotnie. Przerwy w komunikacji są zminimalizowane w MSMIE dzięki temu, że procesory piszące i czytające mogą pracować w tym samym czasie nawet jeżeli korzystają z tego samego bufora.

Uniknięcie odczytu niespójnych danych jest trudne do pogodzenia z krótkim czasem dostępu do pamięci. Zwykły semafor nie wystarcza.

Zajmiemy się komunikacją w przypadku, gdy procesor podrzędny wysyła dane do procesora głównego. Zakładamy, że każdy bufor ma jeden podrzędny procesor piszący i może mieć wiele głównych procesorów czytających.

2.2 Proponowane rozwiązanie

W MSMIE pojedynczy bufor zastępuje się trzema buforami. W ten sposób przy odpowiednim wykorzystaniu dodatkowych buforów czytanie i pisanie może odbywać się w tym samym czasie. Z każdym buforem związany jest jego stan. Zmiana stanu bufora odbywa się w sekcji krytycznej (tutaj używamy semafora). Potrzebny jest licznik procesorów głównych aktualnie czytających.

<table>
<tr><td>data $_1$</td><td>data $_2$</td><td>data $_3$</td><td rowspan="2">semaphore</td><td rowspan="2">counter</td></tr>
<tr><td>status $_1$</td><td>status $_2$</td><td>status $_3$</td></tr>
</table>

Bufory mogą mieć następujące stany:

slave procesor podrzędny pisze do tego bufora

newest procesor podrzędny skończył pisać , można czytać (najnowsze dane)

master procesor główny czyta z bufora

idle bufor może być wykorzystany do zapisu przez procesor podrzędny

Początkowo jeden bufor ma status **slave**, dwa pozostałe **idle**, a licznik czytelników jest ustawiony na zero.

2.3 Operacje protokołu

- **Slave write**

  1. Weź semafor.
  2. Jeżeli jest bufor w stanie **newest**, to zmień jego stan na **idle**.
  3. Jeżeli jest bufor w stanie **slave**, to zmień jego stan na **newest** wpp błąd.
  4. Jeżeli jest bufor w stanie **idle**, to zmień jego stan na **slave** wpp błąd.
  5. Oddaj semafor.

  Jeżeli procesor podrzędny jest gotowy do pisania, to może bez oczekiwania pisać do bufora oznaczonego **slave**. Po zakończeniu pisania procesor podrzędny wykonuje operację **Slave write**, czyli stan zapisanego bufora ustawia na **newest** i jednocześnie gwarantuje sobie możliwość przyszłego pisania przez zmianę stanu bufora **idle** na **slave**.

- **Master acquire**

  1. Weź semafor.
  2. Jeżeli jest bufor w stanie **newest**, ale nie ma bufora w stanie **master**, to zmień stan bufora **newest** na **master**.
  3. Jeżeli jest bufor w stanie **master**, to zwiększ liczbę czytelników.
  4. Zwróć semafor.

  Procesor główny wykonuje operację **Master acquire**, gdy jest gotowy do czytania. Jeżeli po wykonaniu tej operacji istnieje bufor w stanie **master**, to procesor główny czyta z niego. Jeżeli nie ma takiego bufora, to znaczy, że procesor podrzędny nigdy nie pisał i trzeba poczekać (opis protokołu nie mówi co trzeba zrobić w takiej sytuacji).

- **Master release**

  1. Weź semafor.
  2. Zmniejsz licznik czytelników.
  3. Jeżeli nie ma czytelników, to jeżeli jest bufor w stanie **newest**, to zmień stan bufora **master** na **idle** (zwalniamy, bo jest nowszy) wpp (nie ma bufora **newest**) zmień stan bufora **master** na **newest** (nie ma nowszych danych).
  4. Zwróć semafor.

  Po zakończeniu czytania główny procesor wykonuje operację **Master release**, która w przypadku, gdy nie ma innych czytelników powoduje ustawienie stanu bufora na **newest** lub **idle**.

Warto zauważyć, że opisane operacje dotyczą tylko stanów buforów. Odczyt i zapis właściwych danych odbywa się pomiędzy działaniami protokołu.

3 Model naturalny

Model umożliwia sprawdzenie poprawnej koordynacji stanów buforów. Nie uwzględnia wartości danych przechowywanych w pamięci ani operacji zapisu i odczytu do pamięci pomiędzy akcjami protokołu.

Model protokołu dostarcza „usługi” dla procesorów. Jeżeli np. procesor podrzędny chce wykonać operację **Slave write**, to synchronizuje się z agentem protokołu na odpowiednim porcie. Agent wykonuje operację protokołu. Interfejs agenta składa się z akcji: $\mathbf{slave}$, $\mathbf{ma_k}$, $\mathbf{mr_k}$, gdzie k należy do przedziału $1 \ldots n$, a n to liczba głównych procesorów.

3.1 Model obrazu pamięci

Bufor przechowuje stan i pozwala na zapis i odczyt jego wartości:

$$Buf(v) \stackrel{def}{=} \overline{\mathbf{read}}(v).Buf(v) + \mathbf{write}(v').Buf(v')$$

Buforom możemy nadać „adresy” przez przemianowanie portów związanych z pisaniem i czytaniem:

$$Buf_i(v) \stackrel{def}{=} Buf[\mathbf{read}_i/\mathbf{read}, \mathbf{write}_i/\mathbf{write}]$$

Łączymy trzy bufory równolegle. Stany **idle**, **slave**, **newest** i **master** oznaczamy odpowiednio: **i**,**s**,**n**,**m**:

$$Buffers \stackrel{def}{=} Buf_1(\mathbf{s}) \mid Buf_2(\mathbf{i}) \mid Buf_3(\mathbf{i})$$

Licznik definiujemy następująco:

$$\begin{aligned} Counter(n) \quad \stackrel{def}{=} \quad & (\text{if } n = 0 \text{ then } \mathbf{zero}.Counter(n)) + \\ & + (\text{if } n < 2 \text{ then } \mathbf{inc}.Counter(n+1)) + \\ & + (\text{if } n > 0 \text{ then } \mathbf{dec}.Counter(n-1)) \end{aligned}$$

Semafor:

$$Semaphore \stackrel{def}{=} \mathbf{take}.\mathbf{give}.Semaphore$$

Równoległe złożenie buforów, licznika i semaforów daje obraz pamięci:

$$MemImage \stackrel{def}{=} Buffers \mid Counter(0) \mid Semaphore$$

3.2 Operacja Slave write

$$
\begin{aligned}
Slave &\stackrel{def}{=} \mathbf{slave}.Sl_1 \\
Sl_1 &\stackrel{def}{=} \overline{\mathbf{take}}.\mathbf{read_1}(v_1).\mathbf{read_2}(v_2).\mathbf{read_3}(v_3).Sl_2((v_1, v_2, v_3)) \\
Sl_2(B) &\stackrel{def}{=} \sum_{i=1}^{3} \text{if } B\#i = \mathbf{n} \text{ then } Sl_3(B[i \to \mathbf{i}]) + \\
&\quad + \text{if } \mathbf{n} \notin \{B\#1, B\#2, B\#3\} \text{ then } Sl_3(B) \\
Sl_3(B) &\stackrel{def}{=} \sum_{i=1}^{3} \text{if } B\#i = \mathbf{s} \text{ then } Sl_4(B[i \to \mathbf{n}]) \\
Sl_4(B) &\stackrel{def}{=} \sum_{i=1}^{3} \text{if } B\#i = \mathbf{i} \text{ then } Sl_5(B[i \to \mathbf{s}]) \\
Sl_5(B) &\stackrel{def}{=} \overline{\mathbf{write_1}}(B\#1).\overline{\mathbf{write_2}}(B\#2).\overline{\mathbf{write_3}}(B\#3).\overline{\mathbf{give}}.Slave
\end{aligned}
$$

Błędy wychwytywane w opisie nieformalnym odpowiadają blokadom w naszym modelu. Np. w $Sl_3(B)$ jeżeli nie ma bufora ze stanem **s**, to nie można wykonać żadnej akcji, semafor pozostaje zajęty i nikt nie może czytać. Oczywiście musimy sprawdzić, czy system się nie blokuje.

3.3 Operacje Master acquire i Master release

Operacje modelujemy jako akcje $\mathbf{ma_k}$ i $\mathbf{mr_k}$, gdzie k to numer procesora głównego.

Nieformalny opis protokołu nie podaje co trzeba zrobić w sytuacji, gdy procesor podrzędny nigdy nie pisał. Model sprawdza przed wykonaniem **Master acquire**, czy coś było zapisane, jeżeli nie to operacja nie jest wykonywana (tylko bierzemy i oddajemy semafor).

Trzeba jakoś opisać naprzemienne wykonywanie przez procesor główny **Master acquire** i **Master release**. Osiągamy to w agencie $Master$, który akceptuje zdarzenia **ma** i **mr** jedno po drugim:

$$
\begin{aligned}
Master &\stackrel{def}{=} MA_1\\
MA_1 &\stackrel{def}{=} \overline{\mathbf{take}}.\mathbf{read_1}(v_1).\mathbf{read_2}(v_2).\mathbf{read_3}(v_3).MA_2((v_1,v_2,v_3))\\
MA_2 &\stackrel{def}{=} \text{ if } \mathbf{n} \in \{B\#1, B\#2, B\#3\} \wedge \mathbf{m} \notin \{B\#1, B\#2, B\#3\}\\
&\quad\ \text{then } \sum_{i=1}^{3}(\text{if } B\#i = \mathbf{n} \text{ then } MA_3(B[i \to \mathbf{m}]))\\
&\quad\ \text{else } MA_3(B)\\
MA_3(B) &\stackrel{def}{=} (\text{if } \mathbf{m} \in \{B\#1, B\#2, B\#3\} \text{ then } \mathbf{ma}.\overline{\mathbf{inc}}.MA_4(B)) +\\
&\quad\ +\overline{\mathbf{give}}.Master\\
MA_4(B) &\stackrel{def}{=} \overline{\mathbf{write_1}}(B\#1).\overline{\mathbf{write_2}}(B\#2).\overline{\mathbf{write_3}}(B\#3).\overline{\mathbf{give}}.MR_1\\
MR_1 &\stackrel{def}{=} \mathbf{mr}.\overline{\mathbf{take}}.\mathbf{read_1}(v_1).\mathbf{read_2}(v_2).\mathbf{read_3}(v_3).MR_2((v_1,v_2,v_3))\\
MR_2(B) &\stackrel{def}{=} \overline{\mathbf{dec}}.MR_3(B)\\
MR_3(B) &\stackrel{def}{=} \overline{\mathbf{zero}}.\sum_{i=1}^{3}\Big(\text{if } B\#i = \mathbf{m} \text{ then}\\
&\quad\ (\text{if } \mathbf{n} \in \{B\#1, B\#2, B\#3\} \text{ then } MR_4(B[i \to \mathbf{i}]) \text{ else } MR_4(B[i \to \mathbf{n}])\Big) +\\
&\quad\ +\overline{\mathbf{dec}}.\overline{\mathbf{inc}}.MR_4(B)\\
MR_4(B) &\stackrel{def}{=} \overline{\mathbf{write_1}}(B\#1).\overline{\mathbf{write_2}}(B\#2).\overline{\mathbf{write_3}}(B\#3).\overline{\mathbf{give}}.Master
\end{aligned}
$$

W definicji MA_3 widać, że akcja **ma** jest możliwa tylko wtedy, gdy procesor podrzędny coś zapisał. Oddanie semafora może wystąpić w sytuacji, gdy procesor główny chce czytać z bufora (próbuje zsynchronizować się z agentem protokołu na porcie **ma**). Wtedy jeszcze raz sprawdzana jest dostępność danych, a procesor ciągle czeka. Jest to związane z tym, że nie możemy „pod semaforem" czekać aż procesor główny wykona akcję **ma**. Może zdarzyć się sytuacja, w której model protokołu nieskończenie wiele razy sprawdza pamięć, nawet jeżeli procesor główny chce wykonać **Master acquire**.

W definicji MR_3 sekwencja akcji $\overline{\mathbf{dec}}.\overline{\mathbf{inc}}$ jest wykorzystywana do sprawdzenia, czy liczba czytelników jest równa zero (można dodać taką operację bezpośrednio do agenta $Counter$).

Operacje k-tego procesora możemy modelować przez przemianowanie:

$$Master_k \stackrel{def}{=} Master[\mathbf{ma_k}/\mathbf{ma}, \mathbf{mr_k}/\mathbf{mr}]$$

Możemy już skonstruować model. Agent składa się z jednego obrazu pamięci, jednego komponentu $Slave$ i dwóch $Master$. Obecność dwóch agentów $Master$ pozwala sprawdzić licznik i inne elementy protokołu, które inaczej nie byłyby używane. Z drugiej strony większa liczba procesorów głównych może pokazać złe zachowanie systemu, którego tutaj nie sprawdzimy.

$$Msmie \stackrel{def}{=} (MemImage \mid Slave \mid Master_1 \mid Master_2) \setminus L$$

Zbiór L składa się ze wszystkich akcji $\mathbf{read_i}$ i $\mathbf{write_i}$ oraz akcji ze zbioru $\{\mathbf{take}, \mathbf{give}, \mathbf{zero}, \mathbf{inc}, \mathbf{dec}\}$.

4 Model abstrakcyjny

Badamy tylko obserwowalne zachowania systemu, w związku z tym możemy zbudować prostszy model nie bazujący na nieformalnym opisie protokołu (pomijamy szczegóły operacji, uwzględniamy tylko ich wyniki). Nowy model jest sekwencyjny i sparametryzowany stanem obrazu pamięci.

Abstrakcyjny model definiujemy jako agenta $M(B, r)$. Parametr B jest zbiorem trzech par (a, l), gdzie a to stan bufora (ze zbioru $\{\mathbf{i}, \mathbf{s}, \mathbf{n}, \mathbf{m}\}$), a l to identyfikator bufora (ze zbioru $\{1, 2, 3\}$). Parametr r to zbiór czytelników (podzbiór zbioru $MI = \{1, 2\}$ identyfikatorów procesorów głównych). Opis agenta M podajemy w formie dopuszczalnych tranzycji:

- Slave write:

$$M(\{(\mathbf{s}, l), (\mathbf{n}, l'), (\mathbf{i}, l'')\},\ r) \xrightarrow{slave} M(\{(\mathbf{n}, l), (\mathbf{s}, l'), (\mathbf{i}, l'')\},\ r) \tag{1}$$

$$M(\{(\mathbf{s}, l), (\mathbf{n}, l'), (\mathbf{i}, l'')\},\ r) \xrightarrow{slave} M(\{(\mathbf{n}, l), (\mathbf{i}, l'), (\mathbf{s}, l'')\},\ r) \tag{2}$$

$$M(\{(\mathbf{s}, l), (\mathbf{i}, l')\} \cup B',\ r) \xrightarrow{slave} M(\{(\mathbf{n}, l), (\mathbf{s}, l')\} \cup B',\ r) \quad \forall l\colon (\mathbf{n}, l) \notin B' \tag{3}$$

$$M(\{(\mathbf{s}, l), (\mathbf{n}, l')\} \cup B',\ r) \xrightarrow{slave} M(\{(\mathbf{n}, l), (\mathbf{s}, l')\} \cup B',\ r) \quad \forall l\colon (\mathbf{i}, l) \notin B' \tag{4}$$

- Master acquire:

  For $k \in MI$:

$$\begin{array}{rcll} M(\{(\mathbf{m},l)\} \cup B',\ r) & \xrightarrow{ma_k} & M(\{(\mathbf{m},l)\} \cup B',\ r \cup \{k\}) & \\ & & k \notin r & (5) \\ M(\{(\mathbf{n},l)\} \cup B',\ \emptyset) & \xrightarrow{ma_k} & M(\{(\mathbf{m},l)\} \cup B',\ \{k\}) & \\ & & \forall l\colon (\mathbf{m},l) \notin B' & (6) \end{array}$$

- Master release:

  For $k \in MI$:

$$\begin{array}{rcll} M(\{(\mathbf{m},l)\} \cup B',\ r \cup \{k\}) & \xrightarrow{mr_k} & M(\{(\mathbf{m},l)\} \cup B',\ r) & \\ & & r \neq \emptyset, k \notin r & (7) \\ M(\{(\mathbf{m},l),\ (\mathbf{n},l')\} \cup B',\ \{k\}) & \xrightarrow{mr_k} & M(\{(\mathbf{i},l),\ (\mathbf{n},l')\} \cup B',\ \emptyset) & (8) \\ M(\{(\mathbf{m},l)\} \cup B',\ \{k\}) & \xrightarrow{mr_k} & M(\{(\mathbf{n},l)\} \cup B',\ \emptyset) & \\ & & \forall l\colon (\mathbf{n},l) \notin B' & (9) \end{array}$$

Początkowy stan protokołu to:

$$Msmie' \stackrel{def}{=} M(\{(\mathbf{s},1),\ (\mathbf{i},2),\ (\mathbf{i},3)\}, \emptyset)$$

Abstrakcyjny model można zdefiniować bezpośrednio:

$$M(B,r) \stackrel{def}{=} R_1 + \ldots + R_9,$$

gdzie agent R_i odpowiada i-tej tranzycji. Możemy np. zdefiniować R_3:

$$\sum_{(l,l',B')\in S} \mathbf{slave}.M(\{(\mathbf{n},l),(\mathbf{s},l'),(\mathbf{i},l'')\},\ r),$$

gdzie zbiór S to wszystkie krotki (l,l',B') takie, że $B = \{(\mathbf{s},l),(\mathbf{i},l')\} \cup B'$ i B' nie zawiera bufora w stanie **n**.

5 Analiza

Przedstawione modele są w słabej bisymulacji (CWB). Jest między nimi istotna różnica. W modelu naturalnym może zdarzyć się nieskończony ciąg sprawdzeń obrazu pamięci (tego nie rozróżnia słaba bisymulacja).

Będziemy rozważać model abstrakcyjny $Msmie'$. Model z takim prostym interfejsem pozwala na sprawdzenie kilku istotnych własności. Możemy spytać, czy system się blokuje i czy jakieś akcje nie mogą być wykonane. W tym modelu nie możemy sprawdzić, czy wartość zapisana przez procesor podrzędny zostanie poprawnie odczytana przez procesor główny.

5.1 Weryfikacja wymagań

Pierwsze wymaganie o spójności danych nie może być sprawdzone przez dostępne w naszym modelu akcje. Łatwo zauważyć, że własność zachodzi: procesor główny może czytać tylko z bufora oznaczonego przez **m**, do którego nie może pisać procesor podrzędny.

Drugie wymaganie mówi, że w komunikacji między podrzędnymi i głównymi procesorami mogą występować tylko krótkie i ustalonej długości opóźnienia. Stąd wynika, że bufor powinien być zawsze dostępny do pisania. Można to wyrazić formułą:

$$Always(\langle\langle\mathbf{slave}\rangle\rangle\mathbf{tt}),$$

która mówi, że akcja $\mathbf{slave}$ może być zawsze wykonana. $Msmie'$ spełnia tą formułę. Wynika stąd, że model nie ma blokad (zawsze można wykonać $\mathbf{slave}$) czyli protokół nie doprowadza do sytuacji błędnych zaznaczonych w opisie nieformalnym.

Bufor powinien też być zawsze dostępny do odczytu dla procesora głównego, który chce czytać. Przynajmniej od momentu przejścia przez semafor. Można to wyrazić następującą formułą (którą spełnia $Msmie'$):

$$[[\mathbf{slave}]]Always(\langle\langle\mathbf{ma_1}, \mathbf{mr_1}\rangle\rangle\mathbf{tt} \ \wedge \ \langle\langle\mathbf{ma_2}, \mathbf{mr_2}\rangle\rangle\mathbf{tt})$$

Trzeba jeszcze sprawdzić, czy wartość zapisana przez procesor podrzędny zostanie kiedyś odczytana przez procesor główny. Nie da się tego wyrazić poprzez widoczne akcje naszego modelu. Można sprawdzić, czy procesor podrzędny coś zapisał, ale odczyt przez procesor główny może dotyczyć bufora oznaczonego **m**. Poprawiamy model: dzielimy akcje ma_k na dwie klasy: wystąpienia prowadzące do odczytu najnowszego bufora oznaczamy przez ma_k'. W modelu zmieniamy tylko 6. regułę:

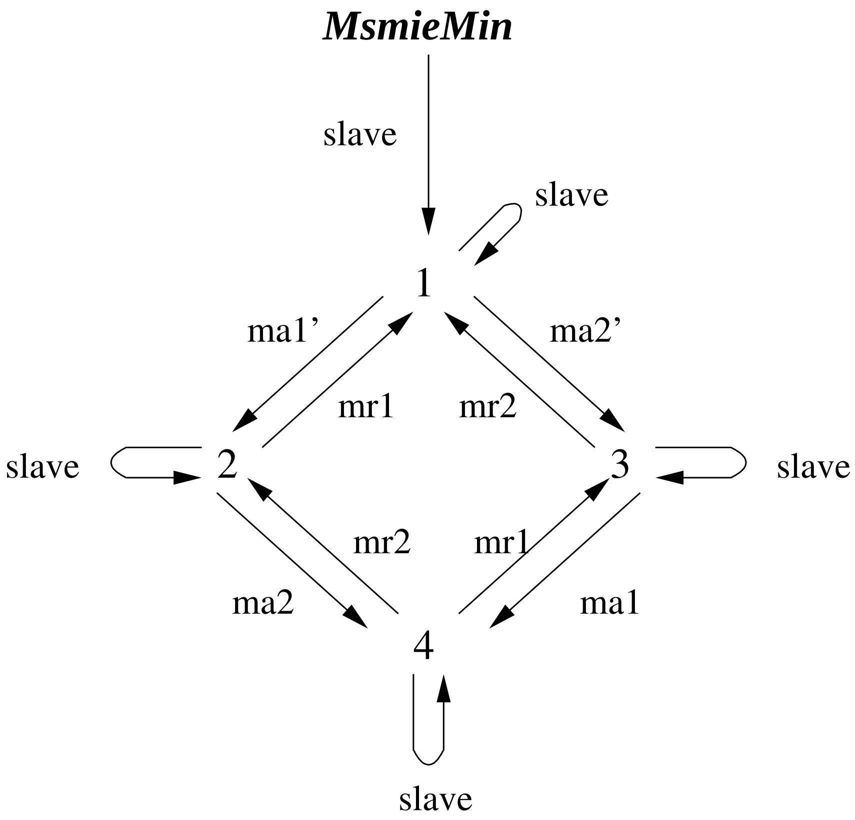
$$\begin{array}{lcl} M(\{(\mathbf{n}, l)\} \cup B', \ \emptyset) & \stackrel{ma_k'}{\longrightarrow} & M(\{(\mathbf{m}, l)\} \cup B', \ \{k\}) \\ & & \forall l \colon (\mathbf{m}, l) \notin B' \end{array}$$

Teraz możemy wyrazić sprawdzaną własność tak: jeżeli wystąpiła akcja $\mathbf{slave}$, to w końcu musi wystąpić $\mathbf{ma_k'}$ dla jakiegoś k. Taka formuła nie jest spełniona dla naszego modelu. Nie mamy żadnej gwarancji, że operacje procesora głównego będą wykonywane. Raczej chodzi nam o formułę: wartość zapisana przez procesor podrzędny zostanie w końcu odczytana przez procesor główny, pod warunkiem, że procesor główny chce w ogóle coś czytać.

Można to zapisać taką formułą:

$$\begin{array}{l} Always([[\mathbf{slave}]]\mu(X \ \nu(Y_1.\langle\langle-\epsilon\rangle\rangle\mathbf{tt} \ \wedge \ [[\mathbf{mr_1}, \mathbf{mr_2}, \mathbf{slave}, \mathbf{ma_2}]]Y_1 \ \wedge \ [[\mathbf{ma_1}]] \\ \qquad\qquad\qquad\qquad\quad \nu(Y_2.\langle\langle-\epsilon\rangle\rangle\mathbf{tt} \ \wedge \ [[\mathbf{mr_1}, \mathbf{mr_2}, \mathbf{slave}, \mathbf{ma_1}]]Y_2 \ \wedge \ [[\mathbf{ma_2}]]X)))) \end{array}$$

Nasz model nie spełnia tej formuły. Powód widać na rysunku:



Model z rysunku to najmniejszy model obserwacyjnie równoważny $Msmie'$. Ze stanu drugiego można wykonać akcję $\mathbf{slave}$, po której występuje nieskończenie wiele akcji $\mathbf{ma_2}$ i $\mathbf{mr_2}$, a nie występują $\mathbf{ma'_1}$ i $\mathbf{ma'_2}$. Czyli nigdy nie przeczytamy nowych danych.

6 Poprawianie protokołu

Wyniki analizy zostały wykorzystane przy projektowaniu protokołu. Do rozwiązania problemu użyto ograniczeń czasowych, co w praktyce wystarcza. Można zmienić protokół tak, żeby nie używać więzów czasowych. Trzeba dodać czwarty bufor i dodatkowy stan **old master**. Procesory główne mogą czytać z buforów oznaczonych **master** lub **old master**, ale do tych ostatnich nigdy nie są dodawani nowi czytelnicy.