Dowodzenie poprawności algorytmów

Zajęcia ćwiczeniowe przed egzaminem

Jak łatwo, szybko i przyjemnie udowodnić poprawność każdego algorytmu?

ŁATWO? NIE DA SIĘ.

- Procesy wykonują się równolegle, należy rozważać więc wszystkie możliwe przeploty - problem eksplozji stanów...
- Wprowadzenie możliwości awarii lub zagubienia wiadomości dodatkowo komplikuje dowody.

Najciekawsza część przetwarzania rozproszonego: nie, jak sprawić, by coś działało szybciej; ale czy coś w ogóle może działać.

Jak zwykle, wszystko zależy od założeń i tego, co chcemy udowodnić.

Co dowodzimy?

Najogólniej, interesować nas będą tylko dwie własności: żywotność i bezpieczeństwo.

- **Żywotność, postęp** (*liveness*): czy to, czego chcemy, w końcu *ostatecznie* się uda? Nb. nie interesuje nas tutaj fairness, czyli warunki typu “czy jeden proces osiągnie cel co najmniej raz, gdy wszystkie inne procesy osiągną cel *k* razy”
- **Bezpieczeństwo** (*safety*): czy to, czego nie chcemy, *nigdy* się nie stanie? Czy nie stanie się coś złego?

Zwykle dla omawianych algorytmów łatwiej jest udowodnić żywotność niż bezpieczeństwo; łatwiej też pokazać, że coś narusza warunki bezpieczeństwa, niż, że nie narusza.

Dowody

Dowód przez przykład: tylko i wyłącznie dla pokazania, że coś nie działa. *NIGDY* nie używamy dla pokazania, że coś działa.

Przykład: “Udowodnij, że jedzenie czekolady sprawi, że Danilecki będzie żyć wiecznie”

Dowód przez sprzeczność: zakładamy, że algorytm nie działa (np. nie wyznacza stanu spójnego). Pokazujemy, że aby algorytm nie działał, trzeba spełnić pewne warunki, a te warunki stoją w sprzeczności albo z logiką algorytmu, albo z założeniami.

Przykład: “Załóżmy, że działają tramwaje, Danilecki nie umiera, nikt go nie napada po drodze, nic mu nie przeszkadza w dotarciu do celu i nie idzie nieskończenie wolno. Udowodnij, że Danilecki wychodząc z domu i chcąc dotrzeć na wykład, ostatecznie na nią dotrze”

Dowody

- Wymienienie wszystkich możliwych przypadków i analiza (dowodzenie poniekąd *brute force*)

  Wykazanie, że z założeń i budowy algorytmu wynikać mogą tylko określone przebiegi, podzielenie ich na przypadki, analiza każdego z nich. Najtrudniejszy dla studentów dowód.

- Dowodzenie indukcyjne: często bardzo przydatne i proste, ale wątpliwe, by udało się je wykorzystać w dowodzeniu algorytmów podczas egzaminu.

Jak doprowadzić prowadzących do płaczu

“Ten algorytm jest podobny do X, a na wykładzie pokazaliśmy, że X działa. Tadaa!”

Pytanie: Udowodnij, że algorytm Chandy-Lamporta dla kanałów FIFO wyznacza poprawny stan spójny. Odpowiedź: “wynika to z założenia kanałów FIFO oraz budowy algorytmu”

Pytanie: Udowodnij, że algorytm Chandy-Lamporta wyznacza globalny stan spójny. Odpowiedź: “Kanały są FIFO, mój wujek w młodości grał w piłkę, a z tego wynika, że stan jest spójny” (non sequitur)

Inne: pisanie wszystkiego, co wiemy na temat (aka “słoń ma trąbę podobną do dżdżownicy, a dżdżownica…”), skróty myślowe, hasłowość - np. wymienienie punktów bez wskazania, jak z punktu A wynika punkt B. Postulowanie prawdziwości czegoś bez udowodnienia, a potem korzystania z tego w dowodzie (Zwłaszcza, gdy się jawnie nie mówi, że się to zakłada). Błędne założenia (typu “zakładam, że Danilecki jest nieśmiertelny”). Niezrozumienie tezy do udowodnienia; błędne koło...

Niepoprawność algorytmu - Chandy-Lamport bez kanałów FIFO (1)

Pytanie: Pokaż niepoprawność algorytmu Chandy-Lamporta wyznaczania spójnego stanu globalnego, gdy kanały nie zachowują własności FIFO.

UWAGA: jeżeli ktoś napisze “a bo procesy mogą się zepsuć”, to raczej nie powinien sobie gratulować.

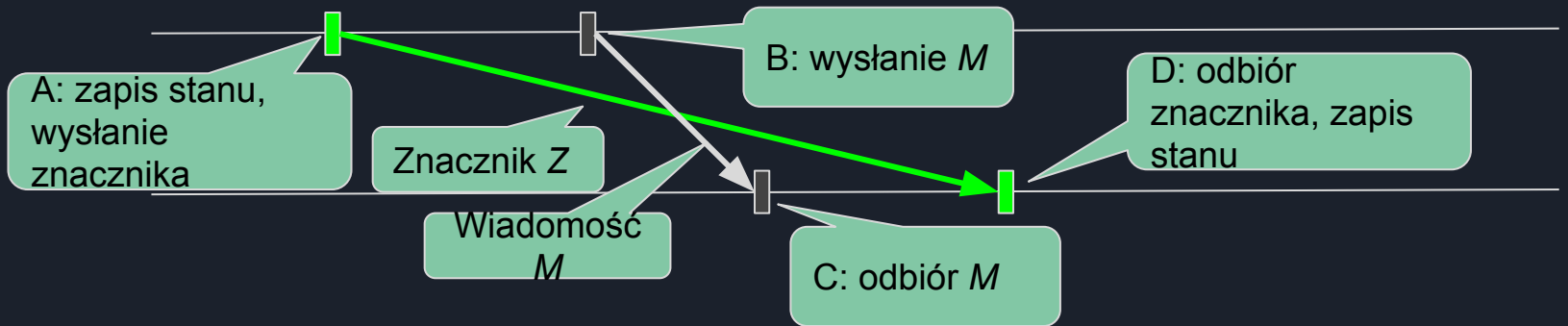
Wystarczy przykład. Musimy pokazać, że możliwe jest wyznaczenie niespójnej konfiguracji Φ. Niespójna konfiguracja pojawi się wtedy, gdy $E \in \Phi$, $E' \mapsto E$, a $E' \notin \Phi$. E to może być odbiór wiadomości *M*, a E’ - wysłanie wiadomości M. Zauważmy, że konfiguracja Φ jest wyznaczana po pierwszym otrzymaniu znacznika Z, po którym przesyłamy go natychmiast dalej.

Na następnym slajdzie pokazujemy rysunek obrazujący sytuację dozwoloną przez algorytm, prowadzącej do wyznaczenia niespójnej konfiguracji, o ile kanały nie są FIFO.

Niepoprawność algorytmu - Chandy-Lamport bez kanałów FIFO (2)

- A - zdarzenie wysłania znacznika Z następujące po zapisaniu stanu s_i^x wchodzącego w skład konfiguracji Φ
- B - zdarzenie wysłania wiadomości *M*
- C - zdarzenie odbioru *M*. Skoro kanały nie są FIFO, *M* może wyprzedzić *Z*
- D - odbiór znacznika *Z* , zapisanie stanu s_j^y wchodzącego w skład konfiguracji Φ

$C \in \Phi$, $B \mapsto C$, a $B \notin \Phi$



Niepoprawność algorytmu - Chandy-Lamport bez kanałów FIFO (3)

Ważne: sam rysunek bez opisu nic nie znaczy! W opisie musi pojawić się sakramentalne: znacznik Z może zostać wyprzedzony przez M, ponieważ kanały nie są FIFO, musi pojawić się wniosek (a więc konfiguracja jest niespójna) oraz musi pojawić się, że konfiguracja została wyznaczona przed wysłaniem/po otrzymaniu znacznika!

Poprawność algorytmu: Lai-Yang dla kanałów non-FIFO

Pytanie: Udowodnij poprawność algorytmu wyznaczania spójnego stanu globalnego Lai-Yanga dla kanałów non-FIFO.

Powinniśmy dać założenie, że kanały są niezawodne, a procesy nie ulegają awariom, ale ponieważ to nie jest algorytm zaprojektowany tak, by był odporny na awarie, więc założenie pomijamy.

Tutaj dowód przykład nie wystarczy (*vide* przykład z wiecznie żyjącym wykładowcą).

Poprawność algorytmu Lai-Yanga (2)

Żywotność: banalnie prosto (dlatego brak dowodu żywotności może nie spowodować odjęcia punktów). Czy ostatecznie wyznaczymy konfigurację?

(1) Skoro procesy nie ulegają awariom, a kanały są niezawodne, to z własności kanałów niezawodnych wynika, że każda wiadomość *M* ostatecznie dotrze do odbiorcy.
(2) Proces-inicjator wysyła znaczniki do wszystkich procesów. Z (1) wynika, że znacznik ten ostatecznie dotrze do wszystkich procesów
(3) Jeżeli jakiś proces *P* nie zapisał jeszcze stanu, najpóźniej zapisze go po otrzymaniu znacznika, a z (2) wynika, że każdy proces ostatecznie otrzyma ten znacznik

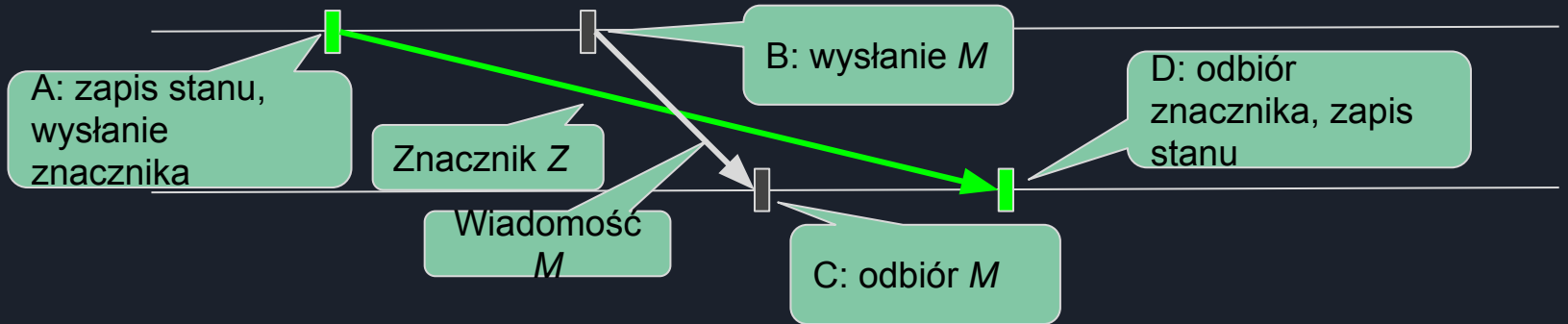
Wynika z tego, że ostatecznie każdy proces (najpóźniej w chwili otrzymania znacznika, być może wcześniej) zapisze stan, a więc ostatecznie zostanie wyznaczona jakaś konfiguracja. Za chwilę przy dowodzie bezpieczeństwa pokażemy, że będzie ona spójna.

Poprawność algorytmu Lai-Yanga (3)

Bezpieczeństwo: Algorytm się zakończył. Czy wyznaczona konfiguracja jest spójna?

Będziemy udowadniać przez sprzeczność. Aby wyznaczona konfiguracja była niespójna, musi wystąpić sytuacja jak na rysunku poniżej. Pokażemy, że konstrukcja algorytmu uniemożliwia powstanie takiej sytuacji.

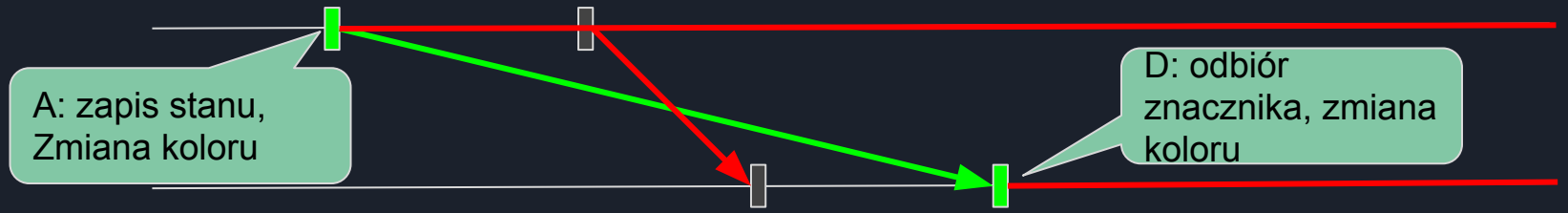
Oczywiście między A oraz B mogą wystąpić inne zdarzenia. Nie szkodzi. Ważne, by pokazać, że akurat taka sytuacja jest niemożliwa.



Poprawność algorytmu Lai-Yanga (4)

Zapis stanu powoduje zmianę koloru procesu na *Red*. Każda wiadomość wysłana w stanie *Red* także jest w stanie *Red*. Skoro wysłanie *M* następuje po zapisie stanu, *M* musi być czerwona.

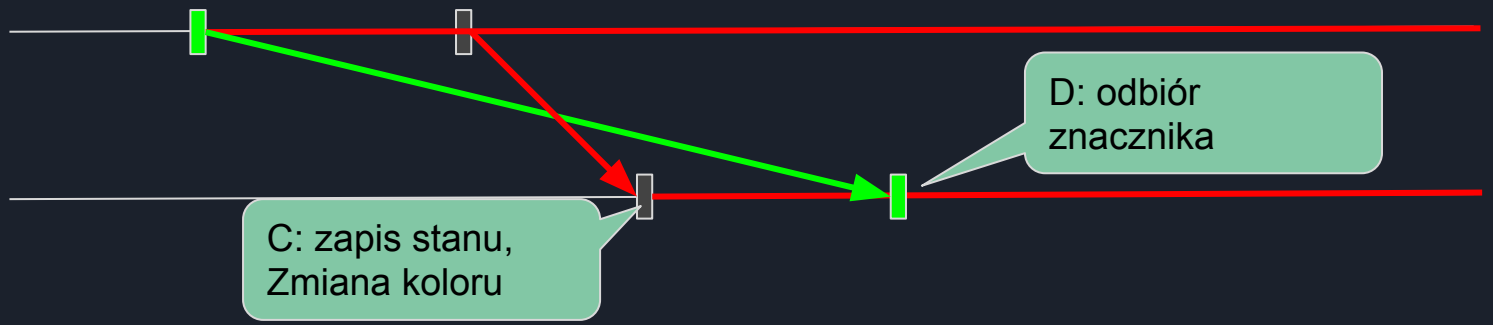
Oznacza to, że sytuacja wyznaczenia konfiguracji niespójnej powstać może tylko i wyłącznie wtedy, gdy proces *White* dostarczy wiadomość *Red*.



Poprawność algorytmu Lai-Yanga (5)

Z konstrukcji algorytmu wynika, że proces *White* otrzymując wiadomość *Red* zanim ją dostarczy, najpierw zapisuje stan i zmienia kolor na *Red*.

Oznacza to, że proces *White* nigdy nie może dostarczyć wiadomości *Red*, a więc nie może dojść do jedynej sytuacji w której algorytm wyznaczyłby konfigurację niespójną, a więc konfiguracja niespójna nigdy nie powstanie (cbdu).



Wykrywanie zakończenia w modelu synchronicznym (Dijkstra, Feijen, van Gastaren)

Procesy są połączone siecią każdy z każdym, ale na tę rzeczywistą topologię nakładamy *wirtualny pierścień*, po którym przechodzi znacznik.

Dla przypomnienia: procesy początkowo są białe, przesłanie wiadomości wstecz pierścienia “brudzi” proces, znacznik przechodzący przez zabrudzony proces również się “brudzi”. Znacznik opuszczając proces “czyści” proces. Znacznik przesyłany jest dalej tylko, jeżeli proces jest pasywny.

Kluczowe: *synchroniczny model przetwarzania*, wiadomości przesyłane są natychmiastowo!

A co by było, gdyby wiadomości jednak ***nie były*** przesyłane natychmiastowo?!

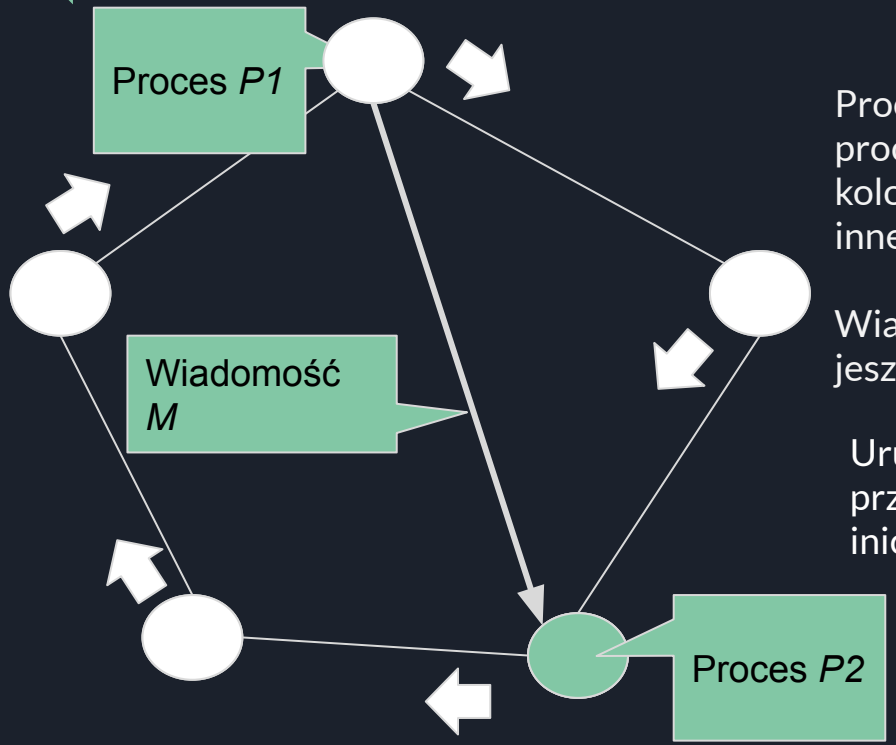
Dijkstra, Feijen, van Gastaren (2)

Założenie: kanały FIFO, niezawodne, ale zmieniamy model: niech **przesyłanie wiadomości zajmuje skończony, ale nieznany z góry czas.**

Skoro dowieść należy niepoprawność, wystarczy skonstruować przykład obrazujący, że algorytm zakończy się wykazując zakończenie, podczas gdy istnieje co najmniej jeden proces aktywny (tj. zakończenia nie ma).

Wskazówka: Procesy są aktywowane wiadomościami.

Dijkstra, Feijen, van Gastaren (2)



Proces *P1* przesyła wiadomość *M* do procesu *P2* i staje się pasywny (nie zmienia koloru, bo przesyła w przód!). Wszystkie inne procesy również stają się pasywne.

Wiadomość *M* leci w kanale, nie dociera jeszcze do *P2*

Uruchamiamy algorytm. Znacznik przechodzi przez wszystkie procesy i wraca wciąż biały do inicjatora - inicjator deklaruje zakończenie.

Wiadomość *M* dociera do *P2*, aktywuje go - zakończenia nie ma.

Dijkstra, Feijen, van Gastaren (3)

Na egzaminie: dajemy rysunek i piszemy:

Skoro wiadomości mogą przebywać w kanale dowolnie długo, przyjmijmy, że po przesłaniu wiadomości M przez P1 do P2 wszystkie procesy przechodzą w stan pasywny, a wiadomość M wciąż pozostaje w kanale. W takim przypadku P1 wciąż jest biały, bo wysłał wiadomość w przód pierścienia; w takim wypadku znacznik przejdzie przez wszystkie procesy i pozostanie biały. Inicjator zadeklaruje wykrycie zakończenia. Tymczasem po zakończeniu algorytmu wiadomość M wreszcie może dotrzeć do P2, aktywując go.

Przetwarzanie nie uległo więc zakończeniu, a więc algorytm dla modelu synchronicznego nie wykryje zakończenia w modelu, w którym przesyłanie wiadomości nie jest natychmiastowe.

Model atomowy i jednofazowy algorytm detekcji zakończenia

Pomijamy przetwarzanie lokalne; tj procesy otrzymują wiadomości, zawsze są aktywowane, wysyłają jedna lub więcej wiadomości i natychmiast zasypiają.

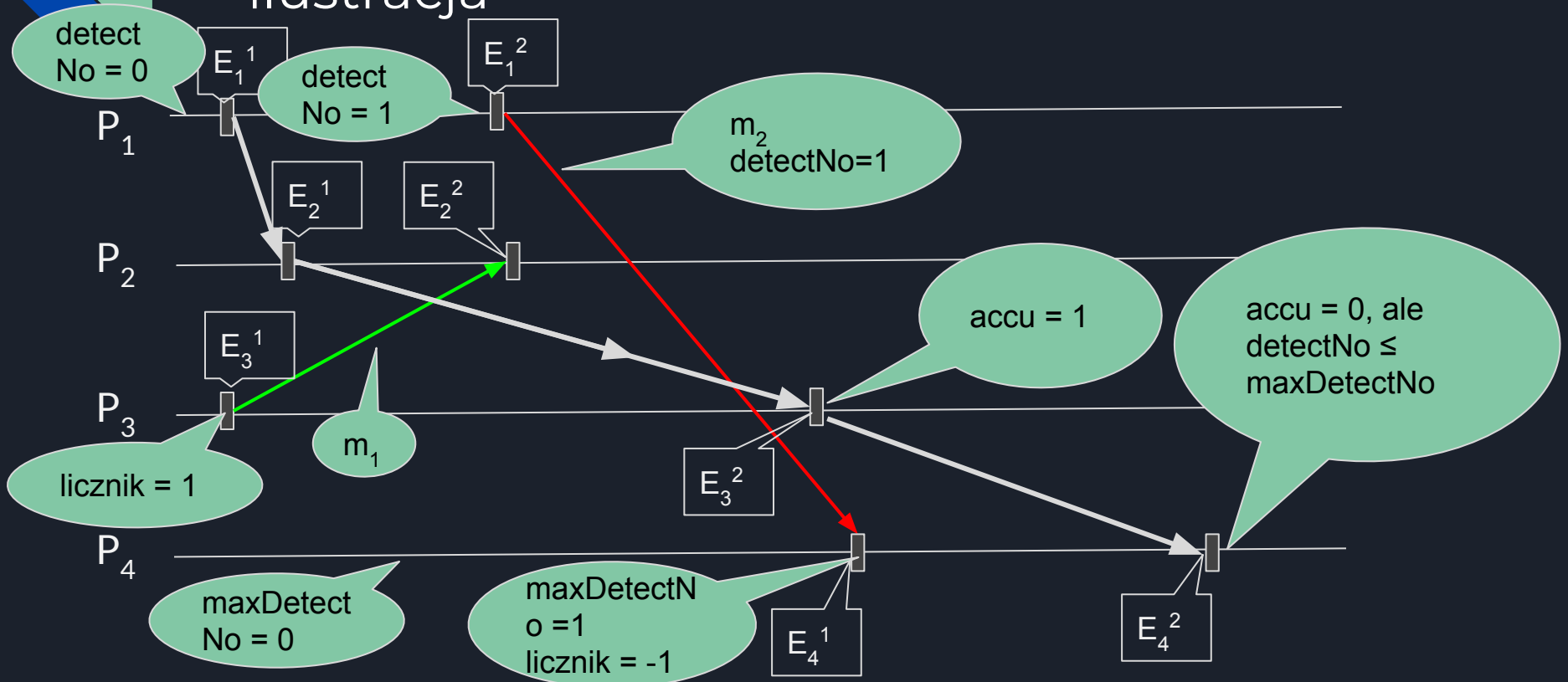
Kanały FIFO, niezawodne, procesy nie ulegają awarii.

Monitory połączone w logiczny pierścień, przetwarzanie aplikacyjne w topologii każdy z każdym

Zliczamy wiadomości wysłane i odebrane. Przesyłany w pierścieniu znacznik kumuluje sumy. Dodatkowo, sprawdzamy, czy suma jest poprawna, tj. czy przypadkiem nie było jakieś komunikacji w czasie, gdy przesyłaliśmy znacznik - każdy proces ma maxDetectNo, czyli maksymalny numer widzianej dotąd detekcji. WIadomości podbijane są maxDetectNo. Jeżeli detectNo znacznika jest ≤ maxDetectNo, znacznik oznaczany jest jako niepoprawny.

Udowodnimy teraz poprawność tego algorytmu.

Jednofazowy algorytm detekcji zakończenia - ilustracja



Jednofazowy algorytm detekcji zakończenia (2)

Żywotność (mniej ważne i prostsze): przetwarzanie się zakończyło. Czy nasz algorytm ostatecznie się zakończy, wykrywając zakończenie?

(1) Skoro kanały są niezawodne, a procesy nie ulegają awarii, każda wiadomość wysłana ostatecznie dotrze do adresata

(2) Każdy proces po otrzymaniu znacznika nie zatrzymuje go, tylko od razu przesyła dalej. A więc, z (1) wynika, że znacznik ostatecznie wróci do inicjatora

(3) Jeżeli znacznik będzie *poprawny* (valid) i *accu = 0* , to algorytm się zakończy, wykrywając zakończenie. Skoro kanały są puste (i niezawodne!), to każde +1 za wysłanie jest równoważone -1 za odebranie

(4) Znacznik może być *niepoprawny* tylko, gdyby maksymalny widziany numer detekcji był większy lub równy od numeru detekcji w znaczniku. Skoro kanały są FIFO, to taka sytuacja może się zdarzyć tylko, gdyby po odwiedzeniu jakiegoś procesu przez znacznik, proces ten później wysłał wiadomość do innego procesu, która by dotarła przed znacznikiem.

Jednofazowy algorytm detekcji zakończenia (3)

Żywotność (mniej ważne i prostsze): przetwarzanie się zakończyło. Czy nasz algorytm ostatecznie się zakończy, wykrywając zakończenie?

(5) Skoro przetwarzanie się zakończyło (przed uruchomieniem algorytmu, z założenia), to wszystkie procesy są pasywne i nie może być wiadomości w kanałach; proces pasywny nie może wysyłać wiadomości, a więc (4) jest niemożliwe

(6) Skoro tak, znacznik musi być poprawny (z (5)) docierając z powrotem do inicjatora, na pewno ostatecznie dotrze do inicjatora (z (2)), a więc algorytm ostatecznie się zakończy, wykrywając zakończenie. cbdu

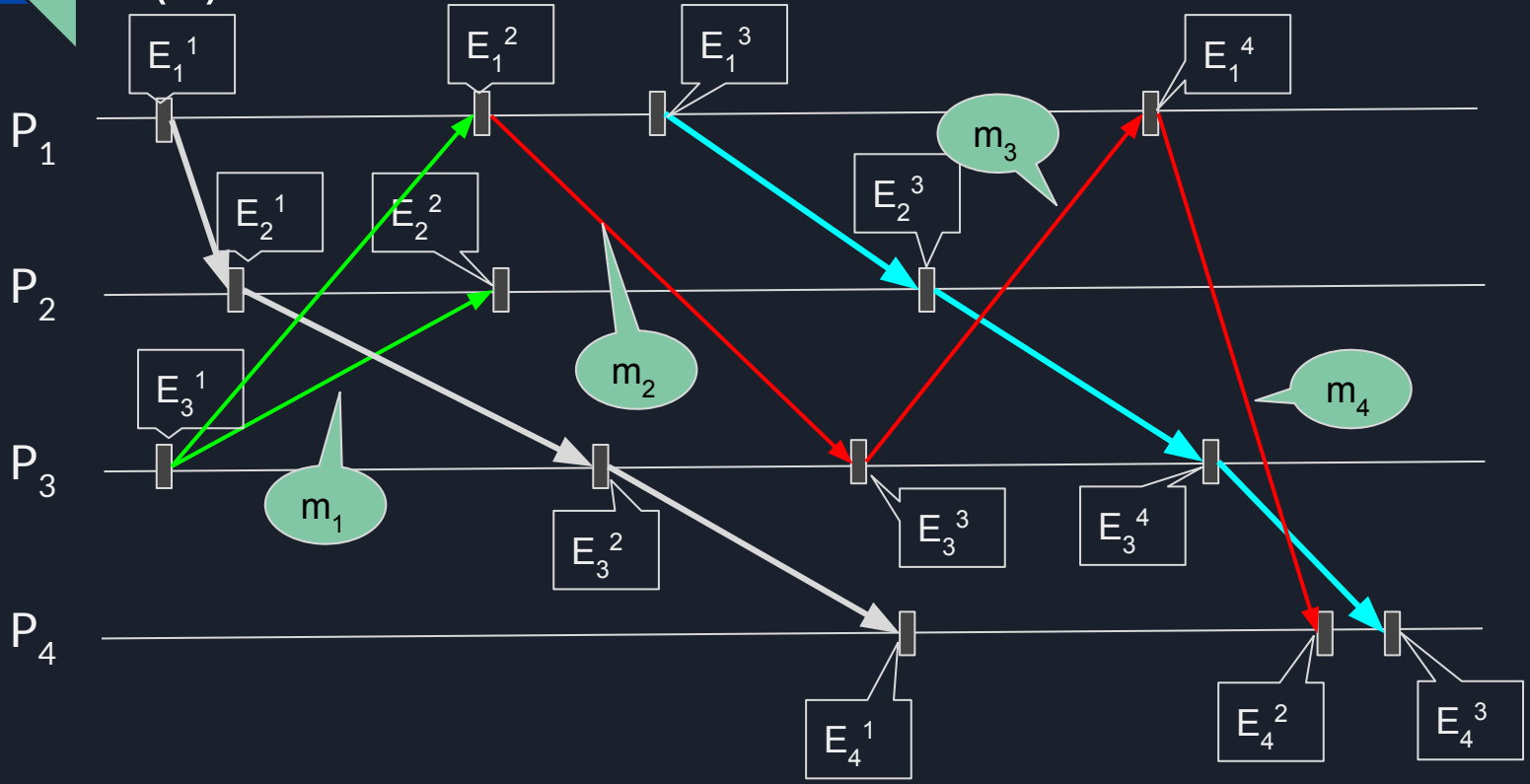
Jednofazowy algorytm detekcji zakończenia (4)

Bezpieczeństwo (trudniejsze, ważniejsze): algorytm zakończył się wykrywając zakończenie. Czy faktycznie zaszło zakończenie?

Czyli: zakładamy, że algorytm się zakończył, a więc, że licznik w znaczniku == 0 i znacznik był poprawny (*valid*).

Jak zaraz zobaczymy, dowód korzysta z dość podobnego rozumowania jak poprzednio. W jakich sytuacjach znacznik musi stać się niepoprawny? Kiedy znacznik jest poprawny?

Jednofazowy algorytm detekcji zakończenia (5)



Jednofazowy algorytm detekcji zakończenia (6)

Przypadki:

(1) Wiadomość wysłana przed przejściem znacznika, odebrana w innym procesie przed przejściem znacznika
(2) Wiadomość wysłana przed przejściem znacznika, odebrana w innym procesie po przejściu znacznika
(3) Wiadomość wysłana po przejściu znacznika, odebrana w innym procesie przed przejściem znacznika
(4) Wiadomość wysłana po przejściu znacznika, odebrana w innym procesie po przejściu znacznika - nie interesuje nas, bo możliwe tylko, gdyby zaszło co najmniej raz (2)

Jednofazowy algorytm detekcji zakończenia (7)

Przypadki:

(1) Wiadomość wysłana przed przejściem znacznika, odebrana w innym procesie przed przejściem znacznika

Licznik będzie zwiększony o jeden w procesie nadawcy, zmniejszony o jeden w procesie odbiorcy.

(a) Jeżeli odbiorca nie wyśle żadnej wiadomości, to nikogo dalej nie aktywuje, a więc jeżeli wszystkie przypadki by były takie, mielibyśmy zakończenie
(b) Jeżeli odbiorca się uaktywni i wyśle wiadomość, to jeżeli adresat ją odbierze przed znacznikiem, mamy znowu przypadek (1), jeżeli zaś po znaczniku, to przypadek (2).

To nie może zachodzić nieskończenie wiele razy, bo z założenia algorytm się zakończył i licznik w znaczniku równa się zero!

Jednofazowy algorytm detekcji zakończenia (8)

Przypadki:

(2) Wiadomość wysłana przed przejściem znacznika, odebrana w innym procesie po przejściu znacznika

W takim razie licznik będzie zwiększony u nadawcy, a nie będzie zmniejszony u odbiorcy, a więc licznik w znaczniku nie będzie równy zero, tymczasem z założenia licznik w znaczniku jest równy zero; chyba, że jakiś inny proces zmniejszył licznik odebranych wiadomości, czyli zaszedł przypadek (3)

Jednofazowy algorytm detekcji zakończenia (9)

Przypadki:

(3) Wiadomość wysłana po przejściu znacznika, odebrana w innym procesie przed przejściem znacznika

W takim wypadku wiadomość będzie miała numer detekcji co najmniej równy numerowi detekcji w znaczniku. W takim wypadku maksymalny numer detekcji w procesie w momencie nadejścia znacznika będzie co najmniej równy numerowi detekcji znacznika. W takim razie znacznik zostanie oznaczony jako *niepoprawny* (invalid). Z założenia jednak wynika, że algorytm się zakończył ze znacznikiem poprawnym, a więc przypadek (3) jest niemożliwy.

Jednofazowy algorytm detekcji zakończenia (10)

Skoro niemożliwe jest (2) i (3), to znaczy, że pozostaje nam (1). Jeżeli wszystkie przypadki to (1a), to zaszło zakończenie. (1b) redukuje się albo do (1a), albo do (2), ale (2) jest niemożliwe, czyli (1b) redukuje się tylko do (1a).

Tak więc, jeżeli nasz algorytm zakończył się wykrywając zakończenie, to faktycznie zakończenie zaszło. CBDU

Sugestie problemów do samodzielnego rozważenia

- Jak rozszerzyć te algorytmy, by działały także w systemie, w którym procesy mogą ulegać awariom?
- Jakie minimalne własności musiałby posiadać detektor uszkodzeń dla danego, tak rozszerzonego algorytmu?
- Czy algorytmy te działałyby też, gdyby kanały nie były FIFO?

Jednolite rozgłaszanie niezawodne - czy można użyć algorytmu aktywnego (*eager*)?

Proces wysyła do wszystkich i dostarcza wiadomość. Każdy inny, o ile wcześniej nie widział tej wiadomości, wysyła ją do wszystkich i dostarcza.

Algorytm ten oczywiście nie spełnia własności jednolitego *zgodnego* rozgłaszania niezawodnego. Dlaczego?

Jednolita zgodność: jeżeli *dowolny* proces dostarczył wiadomość (niezależnie od tego, czy proces jest poprawny, *czy też nie*) *wszystkie* procesy *poprawne* muszą dostarczyć wiadomość.

Zgodność: jeżeli *poprawny* proces dostarczył wiadomość (niezależnie od tego, czy proces jest poprawny, *czy też nie*) *wszystkie* procesy *poprawne* muszą dostarczyć wiadomość.

Jednolite rozgłaszanie niezawodne, algorytm aktywny (*eager*)

Jednolita zgodność: jeżeli *dowolny* proces dostarczył wiadomość (niezależnie od tego, czy proces jest poprawny, *czy też nie*) *wszystkie* procesy *poprawne* muszą dostarczyć wiadomość.

Zgodność: jeżeli *poprawny* proces dostarczył wiadomość (niezależnie od tego, czy proces jest poprawny, *czy też nie*) *wszystkie* procesy *poprawne* muszą dostarczyć wiadomość.

Czy dowolny algorytm spełniający warunki jednolitego zgodnego rozgłaszania niezawodnego, będzie też spełniał warunki zgodnego rozgłaszania niezawodnego?

Detekcja zakończenia Misra’83 (uproszczona)

- System asynchroniczny, kanały niezawodne i FIFO, procesy nie ulegają awariom
- Z góry dany cykl obejmujący wszystkie kanały (nie procesy; kanały!)
- Procesy mają kolor *white* albo *black*. Początkowo *black*: po przesłaniu znacznika proces staje się *white*, zaś ponownie *black*, jeżeli się uaktywni
- Znacznik *brudzi* się nieodwracalnie kolorem procesu (uproszczenie w oryginale bardziej skomplikowane). Proces przesyła dalej znacznik dopiero po tym, jak stanie się pasywny.
- Przesyłamy w cyklu znacznik. Musi co najmniej przejść wszystkie kanały dwukrotnie.

Misra’83

Żywotność - banalnie prosto. Zakładamy, że zakończenie zaszło. Czy algorytm ostatecznie się zakończy, to wykrywając?

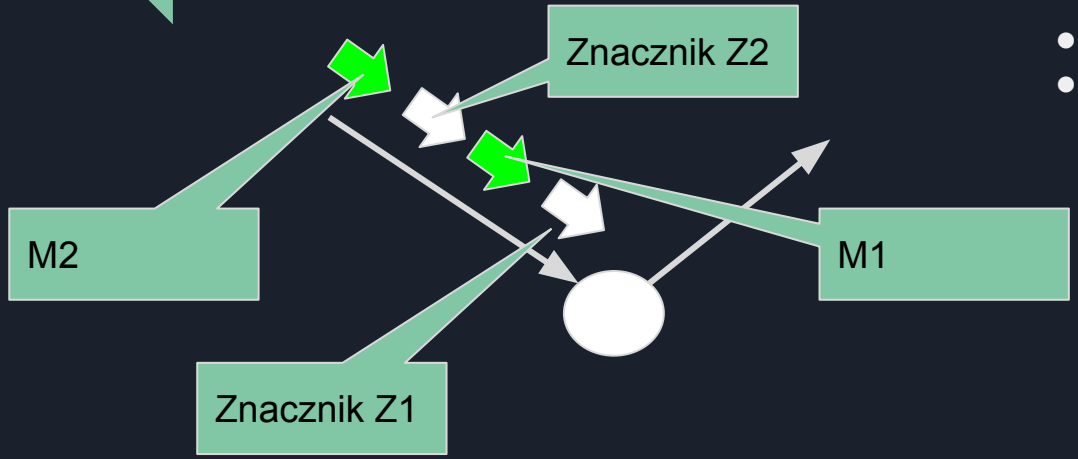
1. Skoro kanały są niezawodne i FIFO, to wszystkie wiadomości ostatecznie dojdą przed znacznikiem; znacznik nie będzie nigdy zatrzymywany, bo procesy są pasywne.
2. Znacznik ostatecznie przejdzie wszystkie kanały, zmieniając kolory wszystkich procesów na *white*.
3. Skoro mamy zakończenie, nie będzie nowych wiadomości, a więc procesy nie mogą zmienić koloru na *black*.
4. W takim razie w drugim przebiegu znacznik przejdzie wszystkie kanały, będzie wciąż *white* i zakończenie zostanie wykryte. CBDU

Misra’83

Bezpieczeństwo - nasz algorytm się zakończył i wykrył zakończenie. Czy faktycznie jest zakończenie?

(1) Znacznik musiał przejść wszystkie procesy co najmniej dwa razy (bo początkowo są *black*)
(2) Między pierwszymi odwiedzinami procesu a ostatnim, żaden proces nie został uaktywniony.
(3) Znacznik przeszedł wszystkie kanały dwukrotnie (bo musi przejść cały cykl)

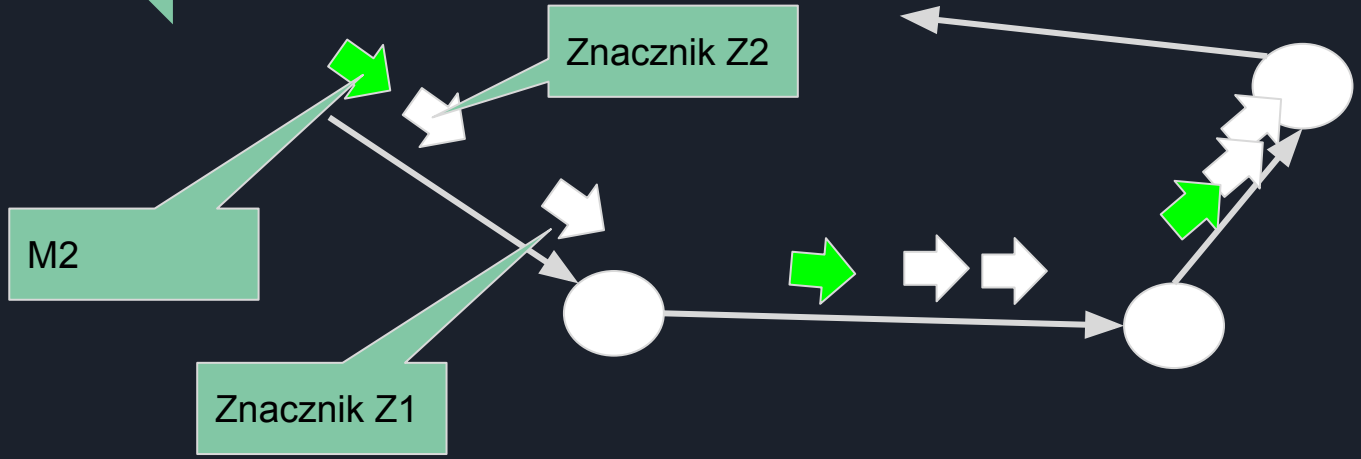
Misra’83



- Wiadomości przed Z1 nas nie interesują
- Wiadomości między Z1 a Z2 uaktywniły by proces P_i co zmieniłoby jego kolor; z założenia to nie zaszło, a więc nie mogło być wiadomości między Z1 a Z2

- Proces P_i nie wyśle więc nowych wiadomości, chyba, że uaktywni go jakaś wiadomość M2. Ale M2 musiałoby zostać wysłane przez inny aktywny proces P_j po tym, jak tamten proces został odwiedzony przez znacznik (po Z2).
- Rekurencyjnie, P_i aby wysłać M2, musiałby otrzymać jakieś MX od jakiegoś P_x i to MX również musiałby otrzymać po drugim przejściu znacznika.

Misra’83



- Kanałów jest skończenie wiele, procesów także; a więc ostatecznie zabraknie nam procesów w rekurencji - bo proces P1 musi uaktywnić wiadomość od P2, P2 uaktywni się i wyśle wiadomość tylko jak dostanie wiadomość od jakiegoś P3 itd itd - ponieważ nie możemu zbudować cyklu, więc dochodzimy do wniosku, że jest niemożliwe, by jakikolwiek proces mógł otrzymać wiadomość po przejściu drugiego znacznika, więc wszystkie będą pasywne, kanały są puste, a więc zakończenie zaszło. CBDU

Detektory uszkodzeń

Banalnie prosto zapamiętać czym jest dokładność i kompletność.

Detektor, to snajper. Na placu są zombie i brygada snajperów.

Dokładny detektor nie trafi żadnego niewinnego.

Detektor kompletny wystrzela wszystkie zombiaki.

Snajperzy nie wiedzą, który z nich kogo trafił (każdy proces ma lokalny detektor i nie wie, co mówią detektory innych procesów)

Detektory uszkodzeń (2)

Kluczowe słówka w definicjach “ostatecznie”, “nigdy” “istnieje jeden proces”, “wszystkie procesy”

Silnie dokładna brygada: ŻADEN snajper NIGDY nie zastrzeli ŻADNEGO niewinnego przechodnia (żaden poprawny proces nie jest nigdy podejrzewany przez żaden inny proces)

Słabo dokładna brygada: snajperzy zostawi przy życiu chociaż jednego niewinnego (istnieje taki poprawny proces, który nigdy nie jest podejrzewany przez żaden inny proces)

Silnie kompletna brygada: każdy zombiak oberwie od każdego snajpera (każdy poprawny proces podejrzewa *ostatecznie* każdy niepoprawny proces)

Słabo kompletna brygada snajperów - każdy zombiak zostanie trafiony przez chociaż jednego snajpera (każdy niepoprawny proces jest podejrzewany przez chociaż jeden poprawny proces)

Jednolite zgodne rozgłaszanie niezawodne z potwierdzeniami od wszystkich

(1) Inicjator wysyła wiadomość do wszystkich
(2) Każdy wysyła wiadomość do wszystkich po jej otrzymaniu
(3) Każdy dostarcza wiadomość, gdy ją otrzyma od *wszystkich* niepodejrzewanych (tych, które uznaje za poprawne) procesów

Jednolita zgodność: jeżeli *dowolny* proces dostarczył wiadomość (niezależnie od tego, czy proces jest poprawny, *czy też nie*) *wszystkie* procesy *poprawne* muszą dostarczyć wiadomość.

Jednolite zgodne rozgłaszanie niezawodne z potwierdzeniami od wszystkich

Nie czekamy na wiadomości od procesów, które uznajemy za błędne.

1. Co się stanie, gdy osłabimy własność detektora z silnej dokładności, na słabą dokładność?

   Bezpieczeństwo: podejrzewamy wszystkich oprócz jednego, dostarczamy wiadomość, ulegamy awarii - ale ten jeden poprawny rozgłosi wiadomość do innych i wszyscy poprawni dostarczą

2. Co się stanie, gdy osłabimy własność detektora z silnej kompletności, na słabą kompletność?

   Żywotność: możemy czekać na wiadomość od błędnego procesu