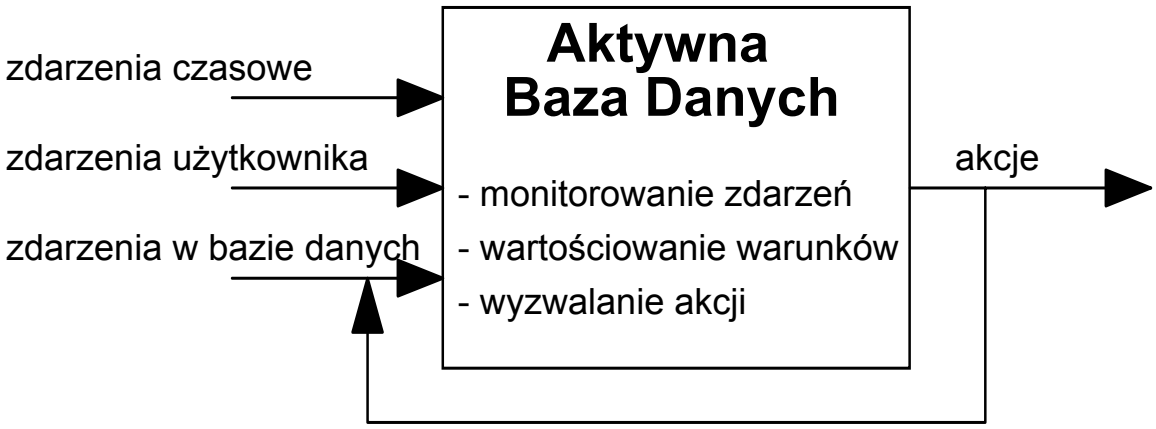
Ogólna architektura aktywnej bazy danych

Rozszerzenie własności pasywnej bazy danych o nową funkcjonalność:

- monitorowania zdarzeń
- ewaluacji dodatkowych warunków
- wyzwalania potencjalnie złożonych akcji

umożliwiającą podejmowanie przez system bazy danych autonomicznej aktywności w obszarze zastrzeżonym dotąd dla aplikacji bazy danych.



Aktywne bazy danych

Model (*E*vent (i)-*C*ondition (i, ii) -*A*ction (ii))

Trzy składowe:

1. wystąpienie zdarzenia,
2. weryfikacja warunku,
3. „odpalenie” akcji;

definiujące ***aktywną regułę*** (***trigger***)

są realizowane w dwóch fazach (potencjalnie tożsamych):

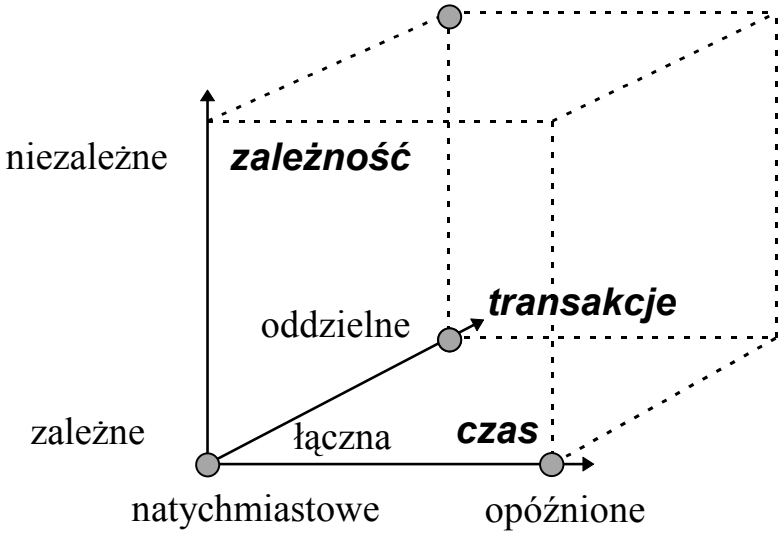
i. w momencie wystąpienie zdarzenia,
ii. w momencie odpalenia akcji.

Własności aktywnych baz danych

- **Schematy aktywności** – umożliwiają zdefiniowanie różnych zależności czasowych i przyczynowo-skutkowych składowych aktywnych reguł
- **Zdarzenia elementarne** – zbiór typów zdarzeń, które mogą być podstawą definiowania aktywnych reguł
- **Operatory zdarzeniowe** – umożliwiają specyfikację złożonych wyrażeń zdarzeniowych
- **Kontekst definiowania reguł** – określa zakres definicji reguły, na przykład: pojedynczy obiekt, zbiór obiektów (relacja), cała baza danych

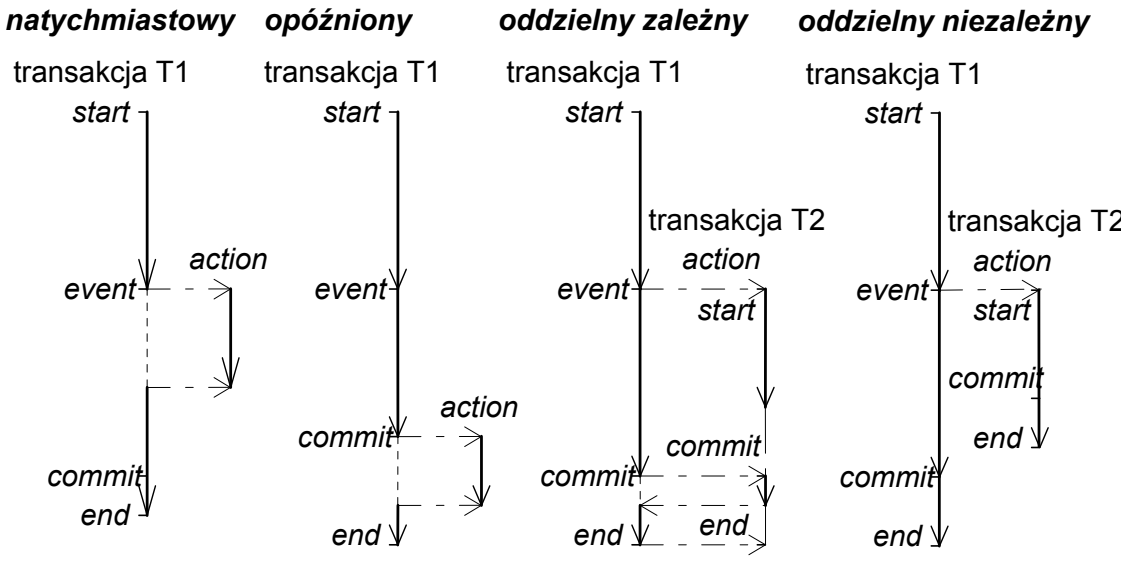
Schematy aktywności

1. **Czasowa**
   - ***bezpośrednia*** - faza druga występuje bezpośrednio po pierwszej
   - ***opóźniona*** - faza druga jest przesunięta w czasie
2. **Transakcyjna**
   - ***połączona*** - zdarzenie i akcja są elementami tej samej transakcji
   - ***oddzielna*** - zdarzenie i akcja należą do różnych transakcji
3. **Zależności**
   - ***zależna*** - wykonanie drugiej fazy jest zależne od pomyślnego zakończenia transakcji pierwszej
   - ***niezależna*** - wykonanie drugiej fazy jest niezależne od pomyślnego zakończenia transakcji pierwszej



Przykłady schematów aktywności

Różne zastosowania wymagają różnych schematów aktywności, na przykład:



1. utrzymanie danych wywiedzionych
2. weryfikacja założonych więzów integralności, operacje w świecie rzeczywistym (odpalenie rakiet)
3. utrzymywanie systemowych logów
4. monitorowanie pracy użytkowników bazy danych

Zastosowania aktywnych baz danych

Konstrukcja pośredniczącej warstwy systemowej (middleware) o nowej funkcjonalności

Wewnętrzne (systemowe) – wsparcie DBMS

- Weryfikacja więzów integralności (statyczne, dynamiczne, wbudowane, ogólne)
- Utrzymywanie danych wywiedzionych (systemy OLAP)
- Rozproszone bazy danych: fragmentacja i replikacja danych
- Zarządzanie przepływami pracy

Zewnętrzne (aplikacyjne) – przeniesienie reguł biznesowych z aplikacji do bazy danych

- Zarządzanie procesami przemysłowymi (systemy czasu rzeczywistego), np. energią, sieciami telekomunikacyjnymi, sieciami komputerowymi
- Automatyzacja procesów biznesowych (giełda, gospodarka magazynowa)
- Bazy wiedzy

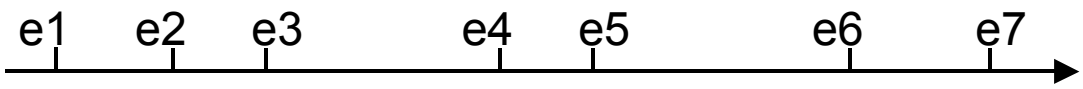
Zbiór zdarzeń elementarnych

1. **Zdarzenia związane ze stanem danych**
   a. utworzenie obiektu: *CREATE*; *INSERT*
   b. usunięcie obiektu: *DELETE*;
   c. modyfikacja, odczyt lub dowolny dostęp do obiektu: *UPDATE* , *READ* , *ACCESS* ;
2. **Zdarzenia związane z operacjami na schemacie bazy danych**

   CREATE, *ALTER*, *DROP*, ...
3. **Zdarzenia związane z uaktywnieniem semantycznej operacji na obiekcie** *awansuj, zmiana_kursu, …*;
4. **Zdarzenia związane ze zmianami stanu transakcji**
   a. rozpoczęcie transakcji: *TBEGIN*;
   b. punkt zatwierdzenia transakcji: *TCOMPLETE*;
   c. zatwierdzenie transakcji: *TCOMMIT*;
   d. wycofanie transakcji: *TABORT*;
4. **Zdarzenia temporalne**
   a. określony punkt czasu: *AT TIME (timestamp);*
   b. po upływie czasu: *AFTER TIME (interval);*
   c. periodycznie co określony czas: *EVERY TIME (interval)*;
5. **Zdarzenia w systemie bazy danych**
   LOGIN, LOGOFF, STARTUP, SHUDOWN, ERROR
6. **Zdarzenia zdefiniowane przez użytkownika**
   a. *nazwa_zdarzenia*

Wyrażenia zdarzeniowe (ang. event expression)

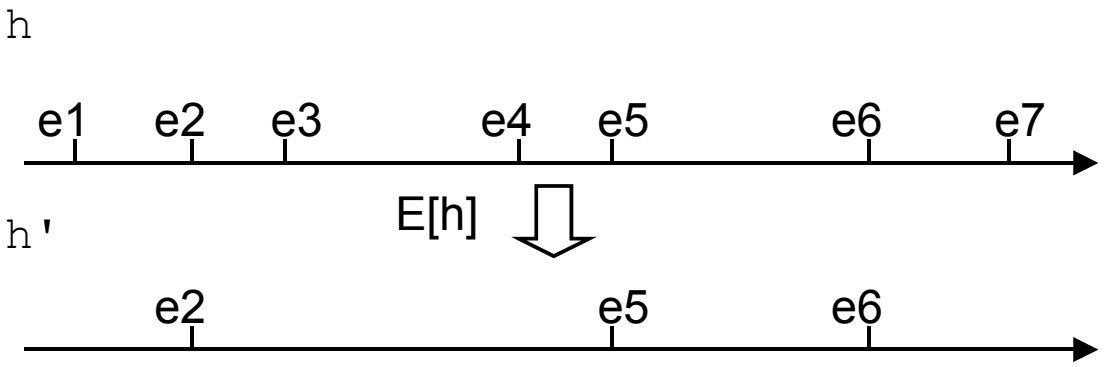
Wystąpienie zdarzenia (lub krócej zdarzenie) jest parą (*typ zdarzenia*, *znacznik czasowy - eid*). Znaczniki czasowe umożliwiają globalne uporządkowanie zdarzeń. Historia zdarzeń *h* danego obiektu jest skończonym zbiorem zdarzeń, w którym żadne dwa zdarzenia nie mogą mieć tego samego znacznika czasowego.



Wyrażenia zdarzeniowe umożliwiają definiowanie zdarzeń elementarnych (predefiniowanych w systemie) oraz zdarzeń złożonych (definiowanych przez użytkowników systemu). Wyrażenie zdarzeniowe *E[h]* jest odwzorowaniem, dla którego dziedziną i przeciw-dziedziną jest historia zdarzeń;

$$E : h \rightarrow h, \text{ gdzie: } E[h] \subseteq h$$

Wyrażenie zdarzeniowe *E* określone w historii *h* wyznacza podzbiór tych zdarzeń historii *h*, dla których jest spełnione zdarzenie złożone *E*. Mówimy, że zdarzenie złożone *E* ma miejsce w zdarzeniu elementarnym e historii *h* jeżeli ***e*** $\in$ *E*[*h*].



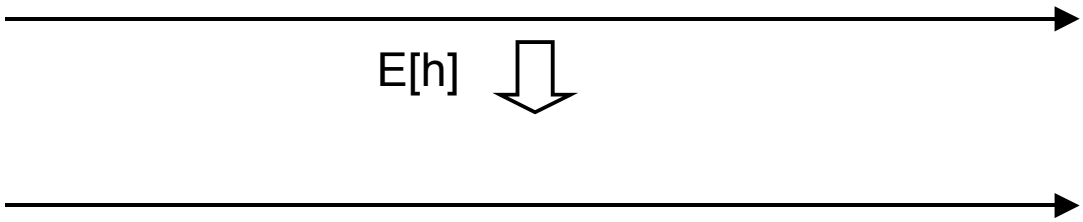
Zdarzenie `e1` poprzedza w historii zdarzenie `e2`, jeżeli `e1` ma mniejszy znacznik czasowy od `e2`. Dwa zdarzenia o takiej samej wartości znaczników czasowych są identyczne.

Operatory zdarzeniowe

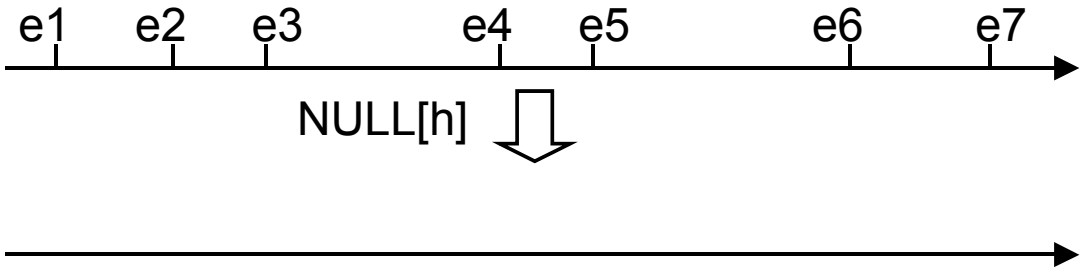
Wyrażenie zdarzeniowe są tworzone za pomocą zdarzeń elementarnych i operatorów zdarzeniowych. Wyrażenie zdarzeniowe może być puste (NULL), być zdarzeniem elementarnym lub wyrażeniem zbudowanym za pomocą jednego z podstawowych operatorów zdarzeniowych: $\frown$ (iloczynu), *!* (negacji) , *relative*, *relative*+.

Semantyka podstawowych operatorów zdarzeniowych jest następująca:

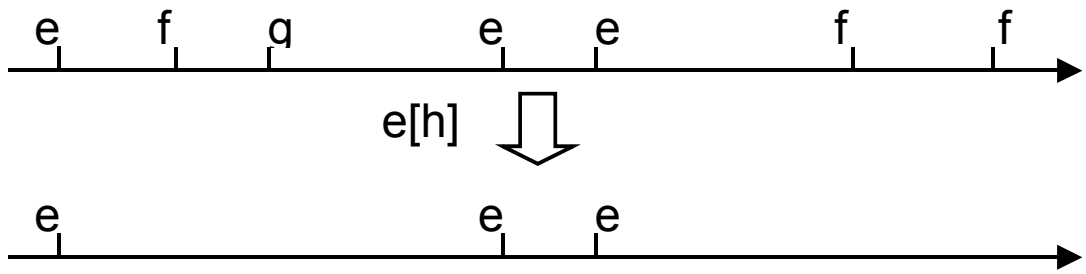
1. *E[null]=null* dla dowolnego wyrażenia *E*



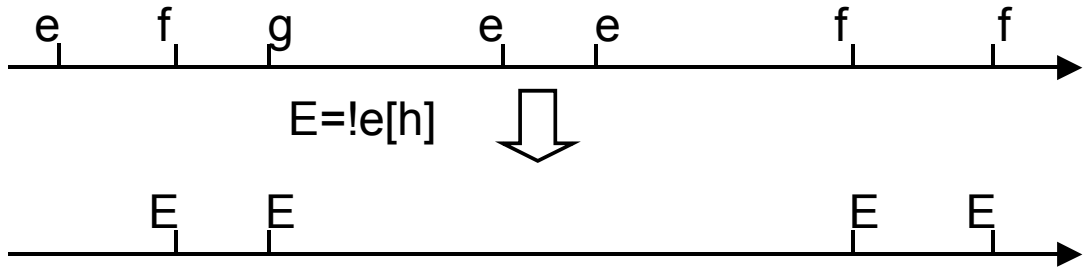
2. *NULL*[*h*]=*null* wyznacza pusty zbiór zdarzeń elementarnych dla dowolnej historii *h*.



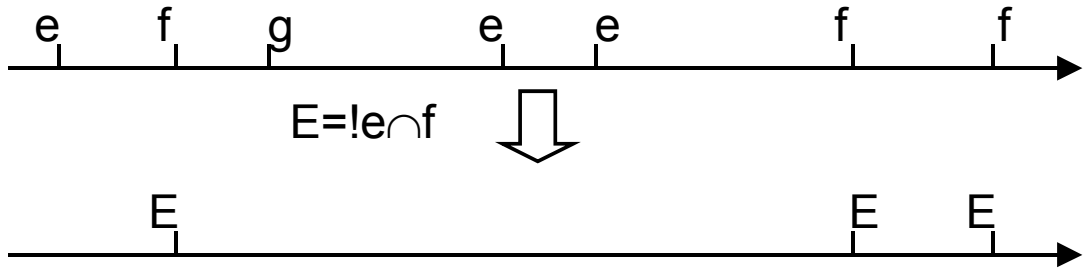
3. *a[h]*, gdzie *a* jest zdarzeniem elementarnym; wyznacza zbiór wszystkich wystąpień zdarzenia *a* w historii *h*.



4. Negacja zdarzenia: *(!E)[h] = (h – E[h])*.

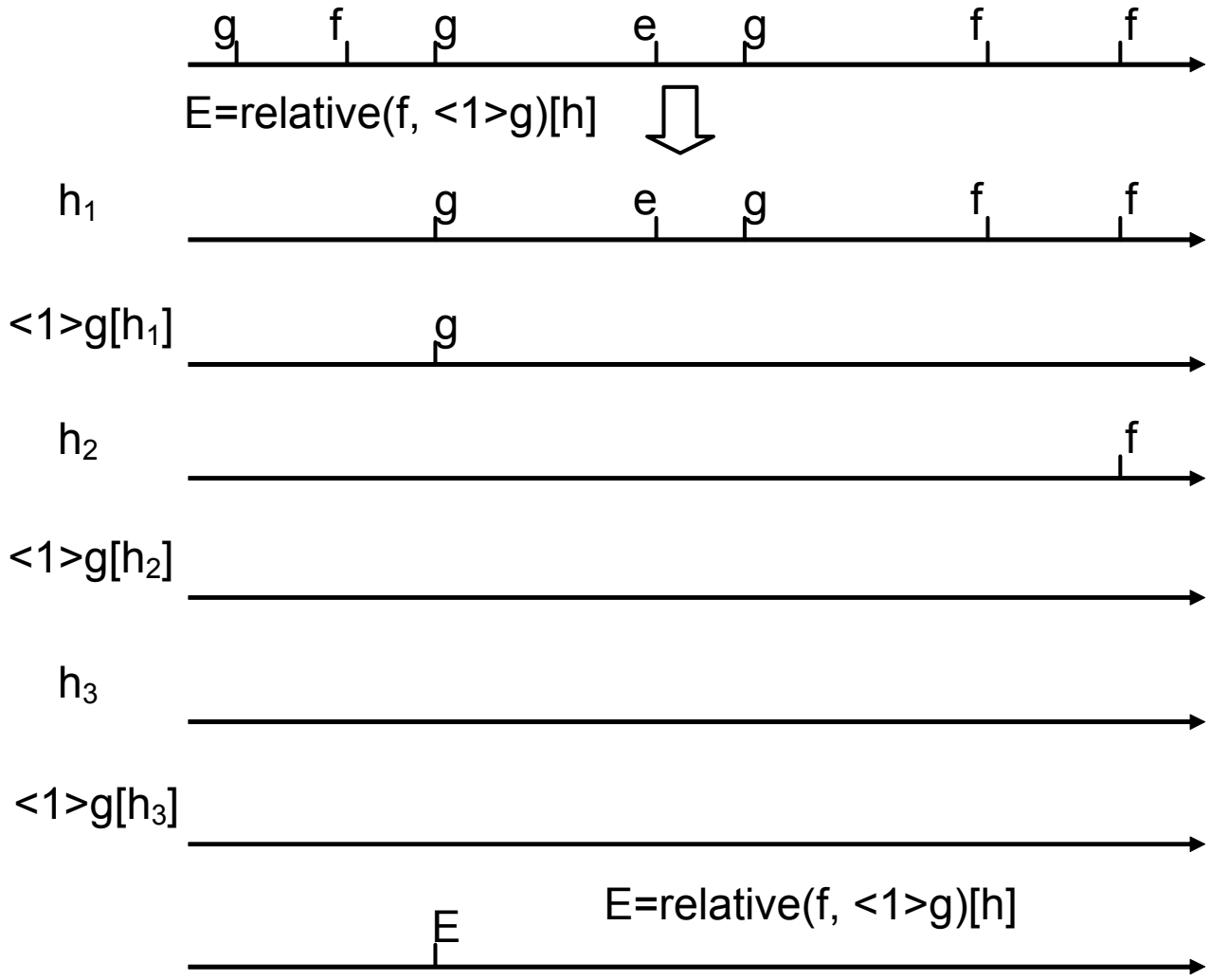


5. Iloczyn zdarzeń: *(E ∩ F)[h] = h_1 ∩ h_2*, gdzie: *h_1 = E[h] i h_2 = F[h].*



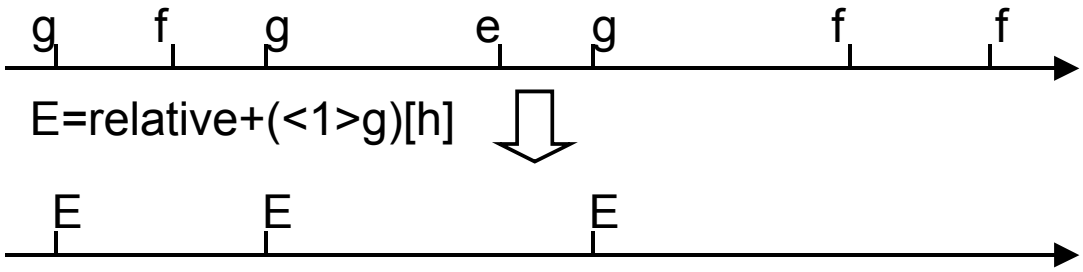
6. Operacja *relative (E, F)[h]*, jest podzbiorem zdarzeń *F* w historii *h* zdefiniowane jest następująco:

   Niech $E^i[h]$ będzie *i*-tym wystąpieniem zdarzenia *E* w historii *E[h],* i niech h_i będzie historią zdarzeń uzyskaną z *h* w wyniku usunięcia z niej wszystkich zdarzeń, których znaczniki czasowe są mniejsze lub równe od znacznika czasowego $E^i[h]$. Wtedy *relative (E, F)[h]* = $\bigcup_i F[h_i]$, gdzie *i* przyjmuje wartości od 1 do mocy zbioru *E[h]*.



7. Operacja ***relative+(E)[h]*** = $\bigcup_{i=1}^{\infty} relative^{i}(E)[h]$, gdzie:

$relative^{1}(E) = E$,
$relative^{i}(E) = relative(relative^{i-1}(E),E)$

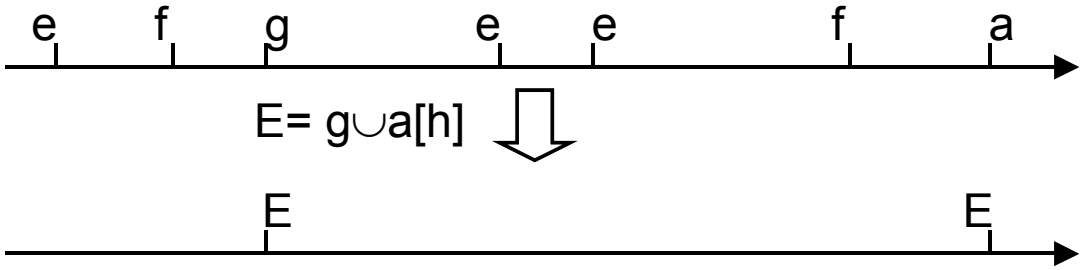


Operator *relative+* pozwala na definiowanie nieskończonych zdarzeń repetycyjnych. Na przykład:

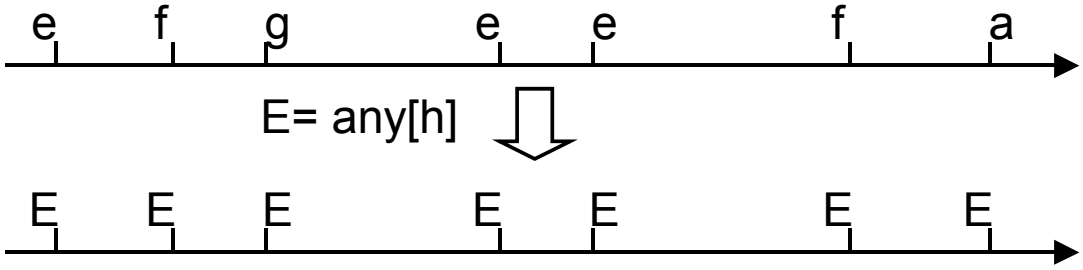
relative+(after time=1 hour)

Operatory pomocnicze:

1. Suma zdarzeń: *(E ∪ F)[h] = h_1 ∪ h_2*, gdzie: *h_1 = E[h] i h_2 = F[h].*

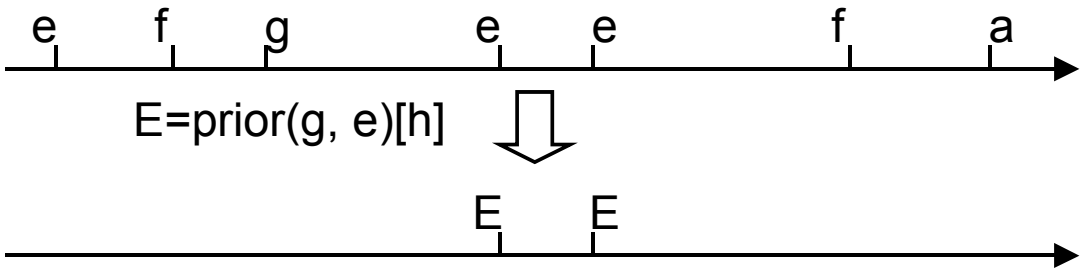


2. *any* oznacza zbiór wszystkich zdarzeń elementarnych w historii za wyjątkiem zdarzenia start.



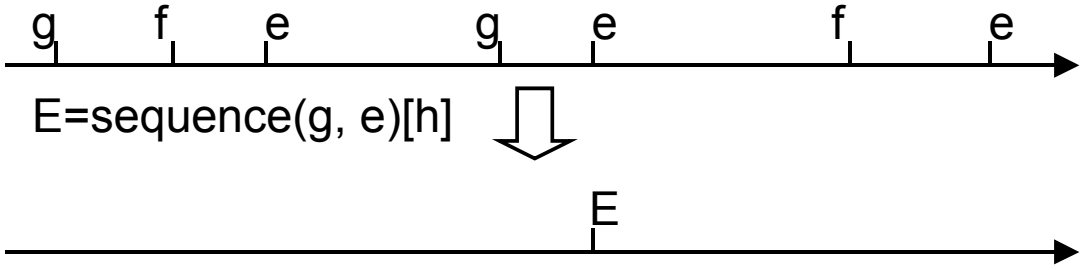
3. Operator *prior(E, F)[h]* jest zbiorem wszystkich zdarzeń *F*, których znaczniki czasowe są większe od jakiegokolwiek zdarzenia *E* w historii *h*.

Formalnie, *prior(E, F) = relative (E, any) ∩ F.*

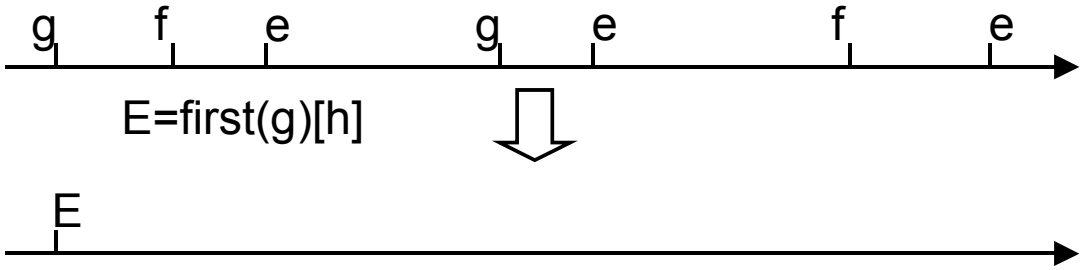


4. Operator *sequence* definiuje bezpośrednie następstwo zdarzeń. Formalnie:

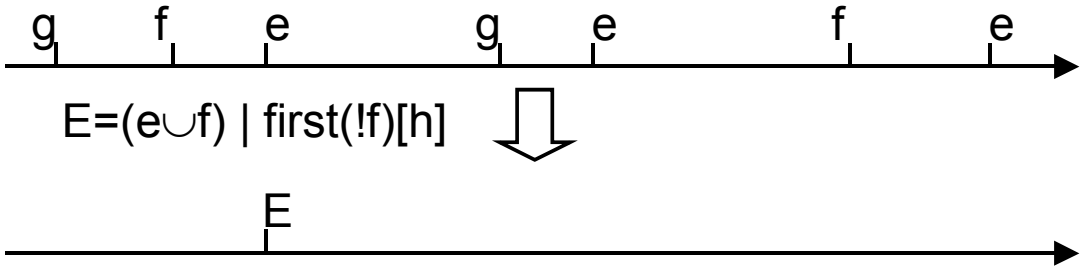
sequence(E, F) = relative(E, !(relative(any, any)))∩ F.



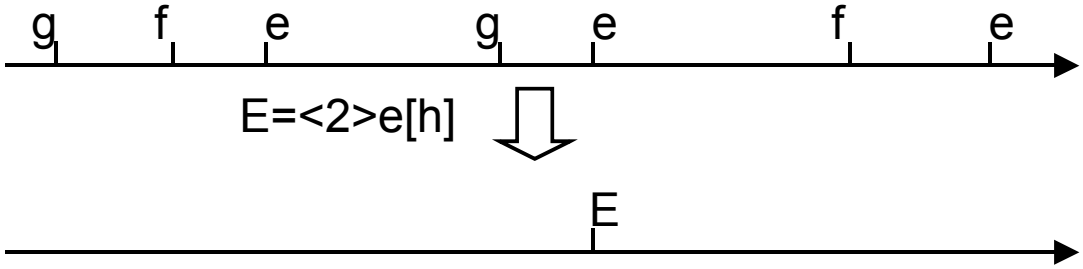
5. Operator *first* identyfikuje pierwsze zdarzenie w historii po zdarzeniu start: *first = !relative(any, any)*.



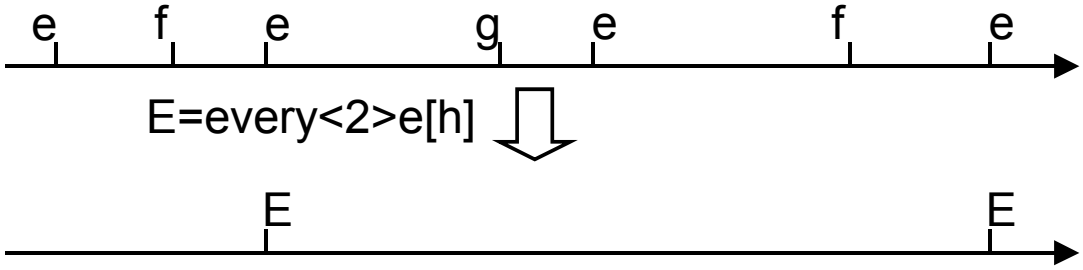
6. Operator potokowy *(E|F)[h] = F(E(h))*.



7. Operator *(<n>E)* określa *n*-te wystąpienie zdarzenia *E*.

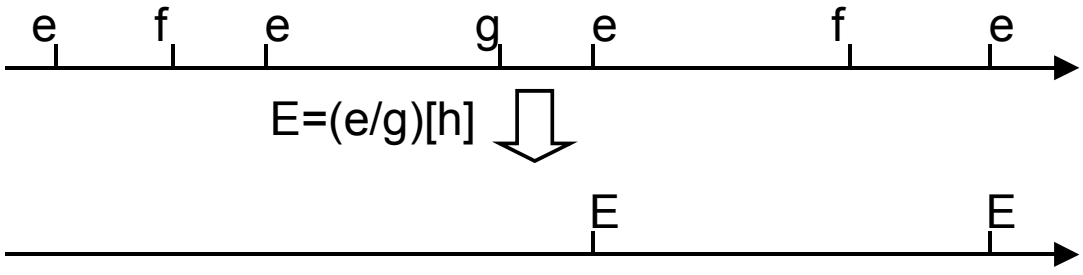


8. Operator *(every <n>E)* określa każde kolejne *n*-te wystąpienie zdarzenia *E*.



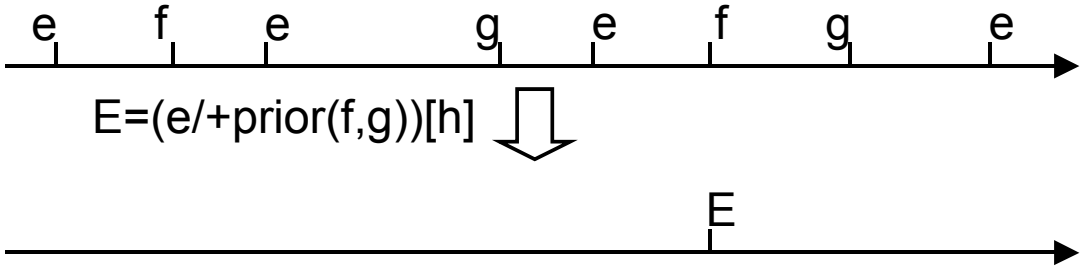
9. Operator *(F/E)[h] = F[h']* gdzie *h'* jest puste jeżeli *E[h]* jest puste; w przeciwnym wypadku *h'* jest historią uzyskana z *h* przez usunięcie z niej zdarzenia *(<1>E)[h]* i wszystkich zdarzeń je poprzedzających.

   Formalnie, *(F/E)[h] = relative((E | first), F).*



10. Operator *(F/+E)[h]* = $\bigcup_{i=1}^{n} F\,[h'_i]$, gdzie *n* jest liczbą wystąpień zdarzenia *E.* Historia h'_i , $1 \leq i < n - 1$, jest wynikiem usunięcia z historii *h* zdarzenia *(<i>E)[h]* i wszystkich zdarzeń je poprzedzających oraz zdarzenia *(<i+1>E)[h]* i wszystkich zdarzeń po nim następujących. Historia h'_n jest wynikiem usunięcia z historii *h* zdarzenia *(<n>E)[h]* i wszystkich zdarzeń je poprzedzających.

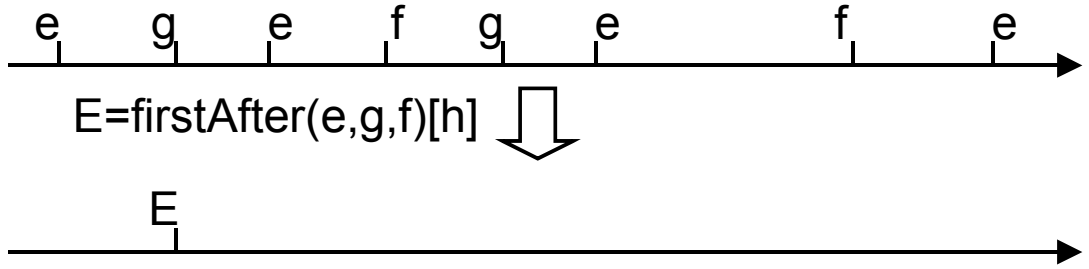
    Zdarzenie *E* pełni rolę ogranicznika, który umożliwia podział historii *h* na zbiór pod-historii ograniczonych wystąpieniami *E*.



11. Operator *firstAfter(E, F, G)[h]* wyznacza w historii *h* zbiór zdarzeń F, takich że F występuje po ostatnim zdarzeniu *E* (*relative*) i zdarzenia te nie są rozdzielone (*prior*) wystąpieniem zdarzenia *G*, które występuje po zdarzeniu *E* (*relative*).

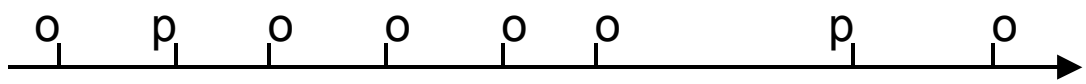
Formalnie:

firstAfter(E, F, G) = (F ∩ !prior(G, any))/+E.

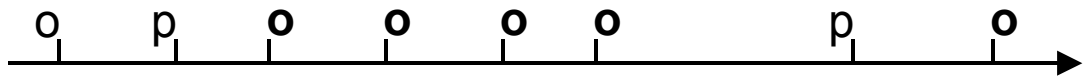


Przykłady wyrażeń zdarzeniowych

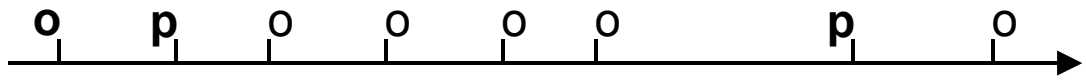
Zdarzenie: seria co najmniej trzech kolejnych obniżek stóp procentowych nie przedzielona podwyżką stóp



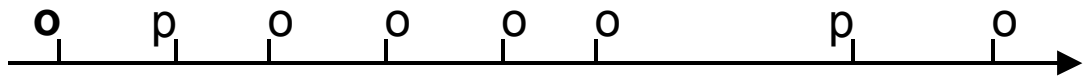
`prior(p,o)` – obniżki po podwyżce



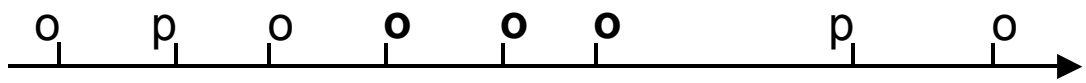
`!prior(p,o)` – wszystkie zdarzenie, które nie są obniżkami po podwyżce



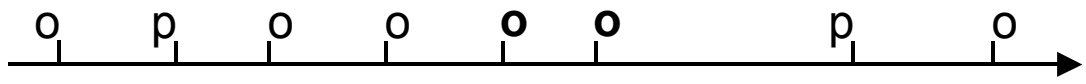
`!prior(p,o)∩o` – wszystkie obniżki, które nie są poprzedzone przez podwyżki



`relative(o,!prior(p,o)∩o)` – wszystkie pary obniżek nie przedzielonych podwyżkami



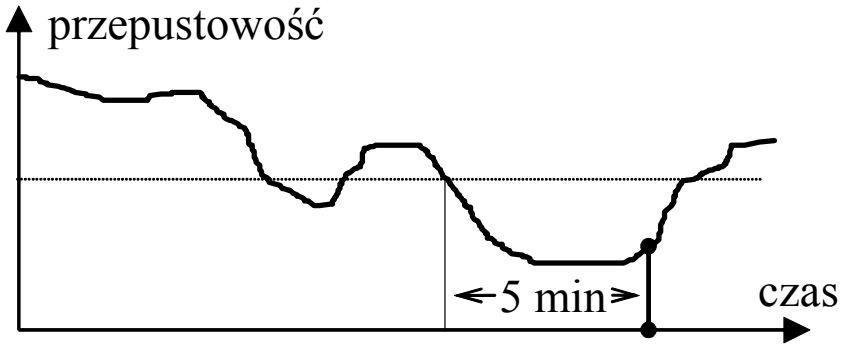
`relative(relative(o,!prior(p,o)∩o,!prior(p,o)∩o)` – wszystkie trójki obniżek nie przedzielonych podwyżkami



Inne rozwiązanie:

`(o∪p) | sequence(o,o o)`

Przykłady definiowania reguł (ODE)




```
#define ZmianaPoniżejProgu AFTER UPDATE && przepustowość<10
#define ZmianaPowyżejProgu AFTER UPDATE && przepustowość>10
class Łącze {
private:
   float przepustowość;
   ...
public:
   boolean Przełącz(Łącze);
   void Alarm (Status );
   ...
trigger:
   SpadekPrzepustowości():perpetual separate dependent
      firstAtfter(
         relative(
            ZmianaPowyżejProgu,
            <1>( ZmianaPoniżejProgu)),
         AFTER TIME (M=5),
         ZmianaPowyżejProgu)
      ==> Przełącz (this);
};
Łącze::Łącze ( )
{
   ...
   SpadekPrzepustowości ( );
   BrakPołączenia ( )
}
```

Przykłady definiowania reguł (ODE)

```
#define dayBegin   at time( HR=9 )
#define dayEnd at time( HR=17 )
class Magazyn {
    ...
public:
    Magazyn ( );
    wydanie ( Towar t, ilość i, Osoba o );
    zamówienie ( Towar t, ilość i);
    bilans_dnia ( );
    raport ( );
    rejestr ( Towar t, ilość i);
    ...
trigger:
    T1( ) : perpetual dayEnd
        ==> bilans_dnia();
    T2( ) : perpetual wydanie(t, i, o)) && i > 100
        ==> log ( t, i, o );
    T3( ) : perpetual firstAtfter ( dayBegin,
        prior(choose 5 (after tcommit),after tcommit),
        dayBegin ) ==> raport ( );}

Magazyn::Magazyn ( )
{
    ...
    T1( ); T2( ); T3( );
    ...
}
```

Implementacja

Wydajna implementacja wymaga przyrostowej weryfikacji zdefiniowanych zdarzeń złożonych – automaty skończone.

Zastosowania języka zdarzeń

- Aktywne bazy danych
- Software configuration management
- Cooperative work
- Złożone przeszukiwanie tekstu
- Analizy historii w temporalnych bazach danych

Metodyka projektowania zbiorów aktywnych reguł

Projektowanie pojedynczych reguł jest stosunkowo proste. Projekt dużego zbioru potencjalnie interferujących ze sobą reguł wymaga zagwarantowania pewnych cech zachowania reguł:

- **ukończenia** (ang. termination) – dla danego stanu początkowego bazy danych i skończonego zbioru modyfikacji stanu bazy danych, przetwarzanie akcji uaktywnionych reguł zostanie zakończone w skończonym czasie
- **determinizmu stanu** (ang. confluence) – kolejność wykonania akcji reguł uaktywnionych tym samym zdarzeniem nie ma wpływu na końcowy stan bazy danych
- **determinizmu obrazu** (ang. observable determinism) - kolejność wykonania akcji reguł uaktywnionych tym samym zdarzeniem nie jest widoczna z zewnątrz (przez użytkowników systemu)

Przykłady

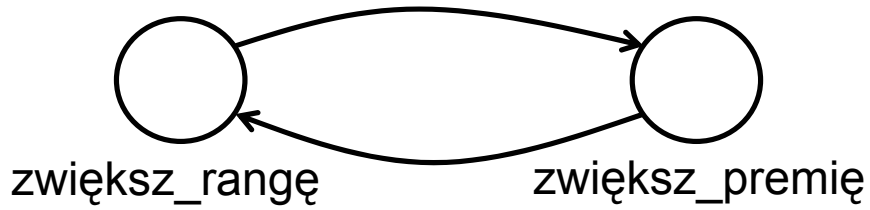
Dana jest baza danych o następującym schemacie:

```
Pracownicy(nr, nazwisko, ranga, płaca)
Premie(nr_prac, wysokość)
Transakcje(nr_prac, miesiąc, numer)
```

Przykład 1 (brak ukończenia)

```
create trigger zwiększ_rangę
    after update of wysokość on Premie
    for each row
    when new.wysokość - old.wysokość > 1000
    begin
        update Pracownicy
        set ranga = ranga + 1
        where nr = new.nr_prac;
    end

create trigger zwiększ_premię
    after update of ranga on Pracownicy
    for each row
    begin
        update Premie
        set wysokość = wysokość+100*new.ranga
        where nr_prac = new.nr;
    end
```



Przykład 2 (ukończenie, brak determinizmu stanu)

```
create trigger zwiększ_płacę
    after insert on Transakcje
    for each row
    when new.numer > 50
    begin
        update Pracownicy
        set płaca = płaca + 100
        where nr = new.nr_prac;
    end
create trigger zwiększ_rangę_2
    after insert on Transakcje
    for each row
    when new.numer > 100
    begin
        update Pracownicy
        set ranga = ranga + 1
        where nr_prac = new.nr;
    end
create trigger zwiększ_płacę_2
    before update of klasa on Pracownicy
    for each row
    when new.ranga = 15
    begin
        new.płaca = 1.1 * płaca;
    end
```

Końcowa wartość płacy pracownika zależy od kolejności uaktywnienia reguł:

zwiększ_ płacę $\rightarrow$ zwiększ_ rangę_2 $\rightarrow$ zwiększ_ płacę_2
zwiększ_ rangę_2 $\rightarrow$ zwiększ_ płacę_2 $\rightarrow$ zwiększ_płacę

$$(1000+100) * 1.1 \neq 1.1 * 1000+100$$

Przykład 3 (ukończenie, determinizm stanu, brak determinizmu obrazu)

```
create trigger nowa_ranga
    after update of ranga on Pracownicy
    for each row
    begin
        send_email(new.nr,
        "gratulacje, nowa ranga: "
        ||new.ranga
        ||", nowa płaca: "
        ||new.płaca);
    end
```

Dla zbioru reguł: `zwiększ_rangę_2`, `zwiększ_płacę_2` i `nowa_ranga`, stan bazy danych jest deterministyczny, ale obraz widziany przez obserwatorów zewnętrznych zależy od kolejności przetwarzania tych reguł.

Kolejność reguł:

zwiększ_ rangę_2 $\rightarrow$ *zwiększ_ płacę_2* $\rightarrow$ *nowa_ranga*

przekazuje w e-mailu nową rangę i nową płacę.

Kolejność:

zwiększ_ rangę_2 $\rightarrow$ *nowa_ranga* $\rightarrow$ *zwiększ_ płacę_2*

przekazuje w e-mailu nową rangę i starą płacę.

Statyczna analiza reguł

Automatyzacja weryfikacji reguł ukończenia oraz determinizmu stanu i obrazu jest w ogólności trudna. Istnieją jednak proste lecz konserwatywne algorytmy statycznej analizy reguł, które na podstawie składni definiowania aktywnych reguł w bazie danych pozwalają:

- określić czy dany zbiór aktywnych reguł charakteryzuje się cechą ukończenia, czy **może** nie posiadać tej cechy;
- określić czy dany zbiór aktywnych reguł charakteryzuje się cechą determinizmu stanu, czy **może** nie posiadać tej cechy;
- określić czy dany zbiór aktywnych reguł charakteryzuje się cechą determinizmu obrazu, czy **może** nie posiadać tej cechy.

Statyczna weryfikacja własności ukończenia

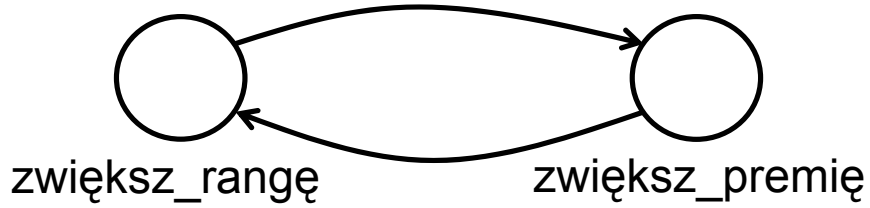
Do statycznej weryfikacji własności ukończenia można zastosować ***graf wyzwalania*** (TG). Węzły grafu reprezentują zbiór aktywnych reguł. Dwa węzły grafu TG, reprezentujące reguły R_1 i R_2 są połączone krawędzią skierowaną od węzła R_1 do R_2, kiedy wykonanie akcji reguły R_1 może spowodować uaktywnienie reguły R_2.

Reguła R_1 może spowodować uaktywnienie reguły R_2, jeżeli kod akcji reguły R_1 zawiera instrukcje manipulacji danymi na relacji (klasie), której modyfikacja jest zdarzeniem uaktywniającym regułę R_2.

Definicja

> Brak cyklu w grafie wyzwalania oznacza, że analizowany zbiór reguł posiada własność ukończenia. Występowanie cyklu w grafie wyzwalania oznacza, że zbiór reguł może nie posiadać własności ukończenia.

Dla zbioru reguł `zwiększ_rangę` i `zwiększ_premię` graf wyzwalania wygląda następująco.



Statyczna weryfikacja własności determinizmu stanu

Podstawą analizy determinizmu stanu zbioru aktywnych reguł jest komutatywność reguł. Dwie reguły r_i i r_j są komutatywne, jeżeli nie jest spełniony żaden z poniższych warunków:

1. akcje reguły r_i nie uaktywniają reguły r_j, a akcje reguły r_j nie uaktywniają reguły r_i;
2. operacje wykonywane w ramach akcji reguł r_i i r_j dotyczą innych relacji (zbiorów obiektów);
3. operacje wykonywane w ramach akcji reguły r_i dotyczą innych relacji (zbiorów obiektów) niż dane weryfikowane w warunku reguły r_j; (i symetrycznie dla reguł r_j i r_i).

Definicja

Zbiór aktywnych reguł R posiada własność determinizmu stanu, jeżeli każda para reguł z tego zbioru jest komutatywna. Brak komutatywności choć jednej z par oznacza, że zbiór reguł może nie posiadać własności determinizmu stanu.