



**POLITECHNIKA WROCLAWSKA  
WYDZIAŁ INFORMATYKI I ZARZĄDZANIA  
WYDZIAŁOWY ZAKŁAD INFORMATYKI**

**WERYFIKACJA DYNAMICZNYCH WIĘZÓW  
INTEGRALNOŚCI SPECYFIKOWANYCH W LOGICE  
TEMPORALNEJ Z OPERATORAMI METRYCZNYMI**

Wojciech Macyna

**Promotor: Prof. dr hab. Zygmunt Mazur**

**Rozprawa doktorska**

**Wrocław 2003**



**WROCLAW UNIVERSITY OF TECHNOLOGY  
FACULTY OF COMPUTER SCIENCE AND MANAGEMENT  
FACULTY DIVISION OF COMPUTER SCIENCE**

**VERIFICATION OF DYNAMIC INTEGRITY CONSTRAINTS SPECIFIED IN  
THE TEMPORAL LOGIC WITH METRIC OPERATORS**

**Abstract**

The complexity of modern databases systems needs effective tools for controlling of semantic data correctness. The lack of such tools may lead to anomalies between data. Dynamic integrity constraints are logical restrictions imposed on data from different database state.

This work proposes temporal logic with metric operators as a formal language for specification of dynamic integrity constraints. Three different kinds of logic are considered: PTL-, FTL- and MTL - logic, which use past, future and mixed temporal operators, respectively. For each type of these logics a new verification method is proposed. These methods use bitemporal graphs and, in the case of FTL and MTL formula verification, three-valued logic

This work describes also a use of temporal logic for the verification of dynamic integrity constraints in databases. Two kinds of a database model are considered: nontemporal - and temporal database model. Nontemporal database is seen as a state sequence, but only the current state is available. In contrary, a temporal database registers all object history, so every database state is accessible.



## Spis treści

|                                                                                   |     |
|-----------------------------------------------------------------------------------|-----|
| WSTĘP .....                                                                       | 5   |
| 1.1.1 Aktualny stan badań .....                                                   | 7   |
| 1.1.2 Cel i zakres pracy .....                                                    | 8   |
| 2. PODSTAWY LOGIKI TEMPORALNEJ .....                                              | 11  |
| 2.1 Podstawowe pojęcia .....                                                      | 11  |
| 2.1.1 Struktura czasu .....                                                       | 12  |
| 2.1.2 Składnia logiki temporalnej .....                                           | 12  |
| 2.2 Operatory temporalne czasu przeszłego PTL .....                               | 14  |
| 2.2.1 Semantyka operatorów PTL .....                                              | 14  |
| 2.2.2 Własności wnioskowania operatorów PTL bez ograniczeń metrycznych .....      | 15  |
| 2.2.3 Własności operatorów PTL z ograniczeniami metrycznymi .....                 | 16  |
| 2.3 Operatory temporalne czasu przyszłego FTL .....                               | 23  |
| 2.3.1 Semantyka operatorów FTL .....                                              | 23  |
| 2.3.2 Własności operatorów FTL bez ograniczeń metrycznych .....                   | 24  |
| 2.3.3 Własności definicji operatorów FTL z ograniczeniami metrycznymi .....       | 25  |
| 2.4 Mieszana logika temporalna MTL .....                                          | 31  |
| 3. METODY WERYFIKACJI FORMUŁ TEMPORALNYCH .....                                   | 33  |
| 3.1 Weryfikacja formuł PTL .....                                                  | 34  |
| 3.1.1 Grafy przejść logiki czasu przeszłego (PGP) .....                           | 34  |
| 3.1.1.1 Definicja grafu PGP .....                                                 | 34  |
| 3.1.1.2 Metoda konstrukcji grafu PGP .....                                        | 37  |
| 3.1.2 Weryfikacja formuł PTL .....                                                | 40  |
| 3.1.2.1 Założenia wstępne .....                                                   | 40  |
| 3.1.2.2 Algorytm weryfikacji formuł PTL .....                                     | 40  |
| 3.1.2.3 Warunki uruchamiania algorytmu 2.3 .....                                  | 46  |
| 3.1.2.4 Opis algorytmu 2.3 .....                                                  | 46  |
| 3.1.2.5 Przykłady weryfikacji .....                                               | 48  |
| 3.2 Weryfikacja formuł logiki FTL i MTL .....                                     | 52  |
| 3.2.1 Grafy FGP i MPG .....                                                       | 52  |
| 3.2.1.1 Grafy logiki czasu przyszłego (FGP) .....                                 | 52  |
| 3.2.1.2 Definicja grafu FGP .....                                                 | 54  |
| 3.2.1.3 Konstrukcja grafu FGP .....                                               | 55  |
| 3.2.1.4 Bitemporalne grafy przejść (MGP) .....                                    | 59  |
| 3.2.2 Weryfikacja formuł logiki FTL i MTL .....                                   | 60  |
| 3.2.2.1 Założenia wstępne .....                                                   | 60  |
| 3.2.2.2 Algorytm weryfikacji formuł MTL .....                                     | 62  |
| 3.2.2.3 Warunki uruchamiania metody MTLVerification .....                         | 71  |
| 3.2.2.4 Opis algorytmu weryfikacji formuł MTL .....                               | 72  |
| 3.2.2.5 Formuły zależne .....                                                     | 74  |
| 3.2.3 Reprezentacja i manipulowanie dodatkowymi danymi .....                      | 77  |
| 3.2.3.1 Algorytm PTLVerification .....                                            | 77  |
| 3.2.3.2 Algorytm MTLVerification .....                                            | 77  |
| 3.2.3.3 Manipulowanie danymi przez algorytm .....                                 | 79  |
| 3.2.4 Metody weryfikacji formuł temporalnych w pojedynczym punkcie .....          | 81  |
| 3.2.4.1 Metoda StateVerification .....                                            | 82  |
| 3.2.4.2 Opis metody StateVerification .....                                       | 86  |
| 3.3 Złożoność pamięciowa i obliczeniowa wszystkich metod weryfikacji .....        | 88  |
| 3.3.1 Złożoność pamięciowa .....                                                  | 88  |
| 3.3.2 Złożoność obliczeniowa .....                                                | 89  |
| 4. ZASTOSOWANIE METOD WERYFIKACJI FORMUŁ LOGIKI TEMPORALNEJ W BAZACH DANYCH ..... | 91  |
| 4.1 Dynamiczne więzy integralności w nietemporalnych bazach danych .....          | 91  |
| 4.1.1 Podstawowe definicje bazy danych .....                                      | 91  |
| 4.1.2 Formuły temporalne - założenia wstępne .....                                | 93  |
| 4.1.3 Przykładowy schemat bazy .....                                              | 96  |
| 4.1.4 Przykłady dynamicznych więzów integralności .....                           | 97  |
| 4.1.5 Weryfikacja temporalnych więzów integralności .....                         | 100 |

|                                                                          |     |
|--------------------------------------------------------------------------|-----|
| 4.2 Więzy integralności w temporalnych bazach danych .....               | 102 |
| 4.2.1 Podstawowe założenia temporalnego modelu danych.....               | 103 |
| 4.2.2 Weryfikacja więzów integralności w temporalnych bazach danych..... | 105 |
| 4.2.2.1 Weryfikacja formuł prostych FTL .....                            | 105 |
| 4.2.2.2 Weryfikacja formuł złożonych FTL .....                           | 112 |
| ZAKOŃCZENIE .....                                                        | 116 |
| LITERATURA.....                                                          | 122 |
| DODATEK A. WYBRANE PRAWA LOGIKI TEMPORALNEJ.....                         | 127 |
| DODATEK B. LOGIKA TRÓJWARTOŚCIOWA .....                                  | 130 |

## WSTĘP

### Więzy integralności w bazach danych

Złożoność współczesnych baz danych wymaga efektywnych narzędzi do kontrolowania poprawności semantycznej danych. Brak tych narzędzi może doprowadzić do zgubnych w skutkach anomalii pomiędzy danymi. Dlatego integralność jest weryfikowana we wszystkich znanych dzisiaj modelach baz danych. Więzy integralności są logicznymi restrykcjami położonymi na dane, dostarczającymi informacje, które są przydatne przy określaniu najbardziej odpowiedniej struktury bazy danych.

#### Typy więzów integralności

Wyróżniamy kilka rodzajów typów więzów integralności:

- integralność dziedziny - zakłada, że każdy atrybut obiektu bądź relacji posiada swoją dziedzinę. Dziedziną może być na przykład podzbiór zbioru liczb naturalnych bądź zbiór wyliczeniowy możliwych wartości atrybutu. Przyjęcie przez atrybut wartości spoza dziedziny jest naruszeniem integralności dziedziny.
- integralność referencyjna - powstaje w wyniku złączenia relacji w modelu relacyjnym, bądź zależności obiekt - podobiekt w modelu obiektowy. Naruszeniem integralności referencyjnej może być usunięcie krotki relacji do których istnieje odwołanie w innej relacji, bądź usunięcie podobiektu, którego identyfikator jest wartością atrybutu innego obiektu.
- integralność statyczna - nakłada dowolne restrykcje na wartości obiektów w tym samym stanie bazy.
- integralność tranzytywna - jest zdefiniowana na dwóch postępujących po sobie stanach bazy: na stanie z przed transakcji i stanie po transakcji. Określa zależność pomiędzy wartościami atrybutów relacji bądź obiektów w tych dwóch stanach. Przykładem może być nałożenie warunku, że płaca pracownika w stanie przed transakcją nie może być mniejsza od jego płacy w stanie po transakcji.
- integralność dynamiczna - jest zdefiniowana na sekwencji stanów. Określa zależności pomiędzy wartościami atrybutów obiektów w całych sekwencjach stanów. Więcej o integralności dynamicznej będzie powiedziane w dalszej części pracy.
- integralność rozmyta - jest stosowanych w rozmytych systemach bazodanowych operujących na rozmytych bądź niekompletnych danych. Specyfikacja tej integralności wymaga zastosowania języka zawierającego rozmyte operatory.

Wraz z przedstawionymi powyżej rodzajami więzów integralności związane są następujące problemy:

- specyfikacja więzów integralności - czyli sposób przedstawiania ograniczeń integralnościowych. Pierwsze co się nasuwa podczas przeglądu różnych specyfikacji jest silne uzależnienie ich od modelu danych bazy. Wyróżniamy więzy integralności oparte na algebrze relacji, formułach logicznych, triggerach itd.

- weryfikacja więzów integralności - czyli weryfikacja formuł, jak również sprawdzanie poprawności syntaktycznej i semantycznej, usuwanie redundantnych zbiorów więzów integralności oraz unikanie sprzeczności pomiędzy nimi.
- translacja więzów integralności - ma na celu tłumaczenie wymagań użytkownika bazy na formuły zrozumiałe dla języka systemu.
- optymalizacja więzów integralności - czyli obniżanie kosztów weryfikacji.
- zapobieganie naruszeniom więzów integralności - czyli metody niedopuszczenia do łamania więzów integralności np. cofanie transakcji, odpalanie triggerów itd.

Rozszerzenie wyżej przedstawionych problemów oraz przegląd więzów integralności w istniejących systemach modelu relacyjnego można znaleźć w pracy [Grefen i Apers 1993].

## **Dynamiczne więzy integralności**

### **1. Znaczenie i zastosowanie więzów integralności.**

O ile integralność dziedziny oraz integralność referencyjna jest rozwiązana w prawie każdym systemie baz danych, a integralność statyczna stosunkowo dobrze zrozumiała i wyrażana za pomocą logiki predykatowej pierwszego rzędu, to duże problemy sprawia integralność dynamiczna.

Dynamiczne więzy integralności występują w wielu rodzajach aplikacji. Pojawiają się wszędzie tam, gdzie muszą być spełnione warunki pomiędzy wartościami atrybutów obiektów w różnych punktach czasu istnienia bazy. Typowymi aplikacjami, które zawierają owe więzy integralności mogą być systemy bankowe, systemy obsługujące zbiory kartotek sądowniczych, systemy kredytowania banków i wiele innych.

Tradycyjnie, projektowanie bazy danych jest prawie wyłącznie projektowaniem struktur danych; więzy integralności natomiast pozostawiane są programowi aplikacyjnemu. Sens zachowania takiego stanu rzeczy jest jednak wątpliwy przy tworzeniu wysoko specjalistycznych baz danych. W tym przypadku logiczne jest pozostawienie całego projektowania bazy danych człowiekowi, który jest ekspertem w danej dziedzinie, który projektując dane powinien projektować także ograniczenia na tych danych. Poza tym umieszczenie specyfikacji więzów integralności w modelu danych umożliwia projektantowi bazy danych skupienie się na tym, co ma być zrobione, a więc jakie zależności powinny być nałożone na dane, a nie na tym jak to ma być zrobione, czyli jak działa mechanizm weryfikujący dane formuły. Upraszcza to program aplikacyjny, który jest bardziej modułarny dzięki wyeliminowaniu sprawdzenia integralności oraz umożliwia zmianę więzów integralności w czasie bez zmiany schematu bazy i programu aplikacyjnego.

Mimo, że zaproponowano wiele strategii rozwiązujących problem weryfikacji i monitorowania dynamicznych więzów integralności, żadna z nich nie została powszechnie zastosowana w systemach komercyjnych. Jak do tej pory tylko niektóre badawcze prototypy systemów gwarantują pełną weryfikację dynamicznych więzów integralności.

### **2. Przykład dynamicznych więzów integralności**

Dynamiczne więzy integralności to zależności pomiędzy wartościami atrybutów obiektów bazy w całych sekwencjach stanów bazy. Przykładem formuł integralnościowych mogą być zapewnienia, że:

- (1) *Cena usługi nie może być większa od 200 PLN w przeciągu pierwszych 6 miesięcy od zamówienia.*
- (2) *Jeżeli klient złożył zamówienie, to musi zapłacić w ciągu 10 dni, a firma realizuje zamówienie w ciągu 5 dni po zapłacie.*

Powyższe zapewnienia mogą być zapisane w sposób formalny za pomocą logiki temporalnej w następujący sposób:

- (1)  $\forall u \in \text{type}(\text{Usługa}), \sim((u.\text{cena} > 200 \text{ PLN}) \text{ since } <_6 (u.\text{begin\_state}))$
- (2)  $\forall u \in \text{type}(\text{Usługa}), u.\text{zamówienie} = \text{true} \Rightarrow \text{sometime-future}_{<10} ( u.\text{zapłata}=\text{true} \wedge \text{sometime-future}_{<5} u.\text{realizacja}=\text{true})$

### 3. Motywacja wyboru logiki temporalnej jako matematycznej podstawy weryfikacji

Powyższe warunki są w istocie warunkami temporalnymi, tzn. odnoszą się do danych z różnych stanów bazy. Wobec tego ich weryfikacja może polegać na analizowaniu całej historii bazy. Jeżeli jednak formuły te są narzucone z góry (a priori), wówczas nie ma potrzeby rejestrowania całej historii obiektów. Należy zarejestrować tylko te informacje historyczne, które są wykorzystywane do obliczenia pewnych warunków. Po tym procesie informacje te są usuwane, gdyż ich obecność jest zbędna i mogłaby przekroczyć zasoby systemu. Tak więc zakres tych informacji historycznych przechowywanych w bazie jest minimalny. Logika temporalna ma tę zaletę, że dzięki własnościom rekurencji potrafi wnioskować o przeszłości jedynie na podstawie danych teraźniejszych oraz bardzo małej ilości danych przeszłych. Tak więc jest to pierwszy powód, dla którego powstały metody weryfikujące więzy integralności w oparciu o logikę temporalną. Drugim powodem jest zysk na złożoności obliczeniowej. Znacznie bardziej opłaca się prowadzić weryfikację formuł temporalnych w każdym stanie bazy wnioskując jedynie z danych ze stanu teraźniejszego, niż za każdym razem zadawać zapytania (np. w temporalnych wersjach SQLa) o całą historię. To drugie podejście prowadzi do ewidentnej redundancji obliczeniowej.

Oczywiście mając tak ograniczoną ilość informacji z przeszłości nie można odpowiadać na zapytania ad-hoc. Do tego celu służą jednak odpowiednie języki zapytań ([Snodgrass 1996], [Snodgrass 1987]).

#### 1.1.1 Aktualny stan badań

Do tej pory powstało kilka prac związanych ze specyfikacją i weryfikacją dynamicznych więzów integralności. W pracy [Lipeck i Saake 1987] przedstawione zostały algorytmiczne podstawy monitorowania więzów integralności wyrażonych za pomocą logiki temporalnej z operatorami przyszłości. Jako podstawowego narzędzia użyto grafów przejść. Formuły więzów integralności są oparte o temporalny rachunek zdań rozszerzony o predykaty logiki pierwszego rzędu. Jako że podejście jest stosowane do konwencjonalnego modelu bazy danych, a weryfikacja odbywa się w oparciu o dane z różnych stanów historii bazy, zachowywane są pewne potrzebne do weryfikacji informacje. Informacje te są składowane w specjalnie do tego celu utworzonych relacjach pomocniczych.

Kontynuacją powyższego podejścia jest praca [Gertz i Lipeck 1996]. Przedstawia ona metody umożliwiające tworzenie triggerów dla każdej takiej operacji w bazie i dla każdej sytuacji, które mogłyby spowodować niespełnienie formuły integralnościowej. W tym przypadku triggerzy pełnią funkcję „strażników”, których rola polega na cofaniu transakcji mogących naruszyć integralność.

Metodę weryfikacji formuł logiki z operatorami przeszłości opartą na grafach przejść dla zaproponowano w pracy [Schwiderski i inni 1994], a koncepcję grafów bitemporalnych jako narzędzi do weryfikacji formuł logiki temporalnej złożonych z operatorów przeszłości i przyszłości w [Zhou 1994].

Chomicki w pracy [Chomicki 1995] rozważa użycie logiki pierwszego rzędu z operatorami temporalnymi czasu przeszłego. W przeciwieństwie do podejść rozważanych poprzednio Chomicki może stosować w sposób dowolny kwantyfikatory logiki pierwszego rzędu. Jednak istotnym ograniczeniem w jego pracy jest fakt, że weryfikowalny w tym przypadku jest jedynie taki zbiór formuł  $f$ , że każda podformuła w  $f$  jest niezależna od dziedziny. Wobec tego istnieje spora grupa formuł, których weryfikacja jest niemożliwa dzięki temu podejściu. Chomicki wykorzystuje także relacje pomocnicze. Z każdą podformułą każdej formuły, która jest weryfikowana skojarzona jest taka relacja. Instancje relacji pomocniczych są definiowane indukcyjnie, tzn. instancja w stanie bazy  $i + 1$  jest określona na podstawie instancji w stanie  $i$  oraz wartości relacji bazowej w stanie  $i + 1$ . W związku z tym podejście to jest ściśle związane z modelem relacyjnym, co ogranicza jego stosowanie w innych modelach baz danych. Zastosowanie operatorów metrycznych związanych z formułami (także [Chomicki 1995]) daje możliwość stosowania tej teorii w bazach danych czasu rzeczywistego.

Problemy pokrewne w stosunku do wyżej wymienionych omawiane są w pracy [Sistla i Wolfson 1995]. Chociaż praca ta dotyczy aktywnych baz danych, kluczowym zagadnieniem jest weryfikacja formuł temporalnych. Rozważania są prowadzone dla dwóch przypadków: języka formuł temporalnych czasu przeszłego oraz języka formuł temporalnych czasu przyszłego. Jednak zamiast standardowych kwantyfikatorów logiki pierwszego rzędu używany jest niestandardowy kwantyfikator „zamrożenia” (*freeze quantifier*). Kwantyfikator ten służy do przypisywania zmiennym pojedynczych lub złożonych wartości w dowolnych punktach czasu istnienia bazy. Co więcej można dzięki niemu realizować ograniczenia metryczne nałożone na operatory temporalne. Dzięki temu języki rozpatrywane w tej pracy mają dużą siłę ekspresji, choć nie jest ona porównywalna z poprzednim podejściem [Chomicki 1995], gdyż nie można porównać języka umożliwiającego stosowanie formuł z tradycyjnymi kwantyfikatorami (uniwersalnym i szczegółowym) z językiem, który zamiast tego oferuje kwantyfikator zamrażający.

### 1.1.2 Cel i zakres pracy

Po przeanalizowaniu przedstawionych powyżej koncepcji nasuwają się następujące wnioski:

- We wszystkich przypadkach mamy do czynienia z konwencjonalnymi bazami danych, a więc takimi, w których mamy dostęp jedynie do aktualnych informacji. Jeżeli chodzi o informacje przeszłe zachowywane jest tylko niezbędne minimum takich informacji, umożliwiające weryfikację formuł integralnościowych.
- Do weryfikacji użyto formuł logiki temporalnej czasu przyszłego lub logiki temporalnej czasu przeszłego bez ograniczeń metrycznych. Jedynie Chomicki w [Chomicki 1995] wspomina o możliwości zastosowania ograniczeń metrycznych, ale tylko w formułach PTL niezależnych od dziedziny.
- Brak jest efektywnej weryfikacji więzów integralności w temporalnych bazach danych.

#### Cele pracy:

- Przedstawienie składni i semantyki trzech rodzajów logik temporalnych: logiki temporalnej czasu przeszłego (PTL) umożliwiającej stosowanie operatorów temporalnych czasu



przeszłego, logiki temporalnej czasu przyszłego (FTL) umożliwiające stosowanie logiki temporalnej czasu przeszłego bądź logiki mieszanej (MTL) umożliwiające stosowanie operatorów czasu przyszłego i przeszłego jednocześnie.

- Zaproponowanie składni i semantyki nowego rodzaju logiki: logiki temporalnej z operatorami metrycznymi będącej rozszerzeniem logiki PTL, FTL oraz MTL o operatory metryczne.
- Stworzenie i udowodnienie nowych własności wnioskowania dla logiki temporalnej z operatorami metrycznymi.
- Zmodyfikowanie grafów przejść jako narzędzia weryfikacji formuł temporalnych z operatorami metrycznymi.
- Zaprojektowanie i porównanie metod umożliwiających efektywną weryfikację formuł logiki temporalnej z operatorami metrycznymi. Metody te korzystają ze wspomnianych w poprzednim podpunkcie własności wnioskowania oraz ze zmodyfikowanych grafów przejść pochodzących z [Zhou 1994]. Są dwie klasy metod: metody umożliwiające wnioskowanie na sekwencji stanów (*PTLVerification*, *FTLVerification*, *MTLVerification*) oraz w pojedynczym stanie (*StateVerification*). Wnioskowanie w algorytmach FTL- i MTLVerification odbywa się na logice trójwartościowej. Formuły mogą przyjąć jedną z trzech wartości: *true*, *false* lub *unknown*.
- Określenie złożoności pamięciowej i obliczeniowej wszystkich zaproponowanych w pracy metod weryfikacji.
- Przedstawienie najbardziej optymalnego modelu danych przechowywania dodatkowych informacji przydatnych w procesie weryfikacji formuł temporalnych.
- Wykorzystanie formuł temporalnych jako formalnego zapisu dynamicznych więzów integralności w bazach danych.
- Przedstawienie najbardziej optymalnej metody weryfikacji dynamicznych więzów integralności w konwencjonalnych bazach danych.
- Zaproponowanie metody weryfikacji dynamicznych więzów integralności w temporalnych bazach danych o modelu opartym o czas rzeczywisty (ang. *valid time*).
- Porównanie uzyskanych z innymi pracami, wskazanie innych obszarów zastosowań nowych metod weryfikacji oraz naszkicowanie kierunków dalszych badań.

**Zakresem** pracy jest głównie weryfikacja formuł logiki temporalnej będących podstawą dynamicznych więzów integralności w bazach danych. Rozpatrzono dwa rodzaje baz danych: bazy temporalne (zachowujące całą historię obiektów) oparte o czas rzeczywisty (ang. *valid time*) oraz bazy nietemporalne niezachowujące historii i zawierające informacje o obiektach jedynie w chwili aktualnej. Należy podkreślić, że w kręgu zainteresowań autora tej pracy jest tylko weryfikacja dynamicznych więzów integralności. Problemy związane ze specyfikacją są poza zakresem tej pracy. Również modele danych nie są istotne w dalszych rozważaniach. Metody weryfikacji powinny być jak najbardziej uniwersalne i niezależne od modeli danych.

W związku z tym niniejsza rozprawa doktorska wygląda w sposób następujący:

- 1) Rozdział I wprowadza podstawowe założenia logiki temporalnej wykorzystywane w dalszej części pracy. Skupia się na definiowaniu syntaktyki oraz semantyki logiki temporalnej z operatorami metrycznymi. Rozpatrywane są trzy odmiany logiki temporalnej: logika temporalna czasu przeszłego (PTL) zakładająca stosowanie operatorów temporalnych czasu przeszłego, logika temporalna czasu przyszłego (FTL) zakładająca

stosowanie operatorów temporalnych czasu przyszłego oraz logika mieszana (MTL) umożliwiające stosowanie operatorów temporalnych czasu przyszłego i przeszłego w jednej formule. Oprócz tego rozdział I wprowadza i udowadnia własności wnioskowania, umożliwiające szybkie wnioskowanie wartości rozpatrywanej formuły w sekwencji stanów.

- 2) Rozdział II wprowadza kilka metod weryfikacji formuł temporalnych. W zależności od rodzaju stosowanej logiki temporalnej wyróżnia się następujące metody: *FTLVerification* (weryfikacja formuł FTL), *PTLVerification* (weryfikacja formuł PTL) oraz *MTLVerification* (weryfikacja formuł MTL). Metody te umożliwiają szybką weryfikację formuł temporalnych w sekwencji stanów. Rozdział ten wprowadza także metodę *StateVerification*, która pozwala na szybką weryfikację formuł prostych w pojedynczym stanie. Każda z wyżej wymienionych metod korzysta z grafów przejść. Grafy te są zmodyfikowaną wersją grafów z pracy [Zhou 1994]. W rozdziale tym rozpatrzono także sposób przechowywania dodatkowych informacji przydatnych w procesach weryfikacji. Jest to jedyny aspekt pracy zależny od modelu danych w bazie. Wreszcie ostatnia część rozdziału przedstawia złożoność obliczeniową i pamięciową wprowadzonych metod weryfikacji.
- 3) Rozdział III opisuje wykorzystanie przedstawionych w rozdziale II metod w bazach danych. Główny nacisk położony jest na zastosowanie w bazach nietemporalnych oraz temporalnych opartych o czas rzeczywisty. Weryfikacja w temporalnych bazach danych odbywa się z pominięciem pewnych sekwencji stanów nie mających żadnego wpływu na wartość weryfikowanej formuły.
- 4) W podsumowaniu przedstawione są wnioski wynikające z badań nad problemami poruszonymi w niniejszej rozprawie, porównanie z innymi pracami o podobnej tematyce oraz szkic kierunków dalszych badań.



## 2. PODSTAWY LOGIKI TEMPORALNEJ

Celem tego rozdziału jest wprowadzenie logiki temporalnej jako matematycznej podstawy do weryfikacji dynamicznych więzów integralności w bazach danych. Przedstawiono trzy rodzaje logik temporalnych: PTL - logika temporalna czasu przeszłego, FTL - logika temporalna czasu przyszłego oraz MTL - logika mieszana. Aby zwiększyć siłę ekspresji tych logik, każdą z nich rozszerzono o operatory metryczne. Z kolei konsekwencją obecności operatorów metrycznych jest potrzeba zdefiniowania nowych reguł będących podstawą wnioskowania dla takich formuł. W związku z tym, w niniejszym rozdziale oprócz pojęć podstawowych związanych z różnymi rodzajami logik, wprowadzono składnię, semantykę oraz wyprowadzono i udowodniono pewne własności definicji operatorów temporalnych trzech rodzajów logik temporalnych. Własności te są podstawą logiczną do prowadzenia weryfikacji formuł temporalnych.

### 2.1 Podstawowe pojęcia

Logika temporalna jest rozszerzeniem logiki klasycznej o operatory temporalne. Podstawowa różnica pomiędzy logiką temporalną a logiką klasyczną jest taka, że formuły logiki klasycznej służą do opisu zależności w tej samym czasie, a formuły logiki temporalnej opisują zależności na sekwencji punktów czasu lub przedziałów czasowych.

Badając logikę temporalną odnosi się wrażenie, że logika ta będąc silnie związana z czasem pozwala na wnioskowanie czasu spełnienia formuł. Niestety tak nie jest. Klasyczna logika temporalna nie nakłada żadnych czasowych ograniczeń na zdarzenia informując jedynie o sekwencjach i przebiegach spełnialności formuł. Dlatego też kolejne podrozdziały niniejszej pracy wprowadzają metryczną logikę temporalną. Jej użycie zwiększa siłę ekspresji logiki temporalnej przez co wydaje się być ona bardziej odpowiednim formalizmem do badania więzów dynamicznych zachowań bazy danych. W związku z tym można wnioskować już nie tylko o zależnościach pomiędzy sekwencjami spełnialności formuł, ale można również nałożyć ograniczenia na czas trwania tych sekwencji.

Mówiąc w ogóle o logice należy rozpocząć od określenia rzędu danej logiki. Istnieje kilka rzędów logik:

- rachunek zdań (ang. propositional logic) - bazą dla takiej logiki jest zbiór faktów elementarnych (formuł atomowych) połączonych operatorami logicznymi ( $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Leftrightarrow$ ,  $\Rightarrow$ ). Każda formuła przyjmuje wartość: true lub false.
- logika pierwszego rzędu (ang. first-order logic) - formuły budowane są na dziedzinie ( $D$ ) elementów,  $n$ -arna relacja  $R_i$  definiowana jest jako podzbiór  $D^n$ ,  $n$ -arny predykat  $p_i$  związany jest z każdą  $n$ -arną relacją  $R_i$ . Predykat jest funkcją, która dla każdego elementu  $D^n$  daje wartość true, jeżeli element ten należy do relacji  $R$ . Są zdefiniowane kwantyfikatory:  $\forall, \exists$ , które operują na zmiennych z dziedziny  $D$ .
- logika drugiego rzędu (ang. second-order logic) - kwantyfikowane mogą być całe predykaty, a nie tylko zmienne np.:  $\forall P, \exists x P(x)$ , gdzie  $P$  jest predykatem, a  $x$  zmienną.

Tak samo jest w przypadku logiki klasycznej, logika temporalna może być również traktowana jako rachunek zdań, logikę pierwszego bądź drugiego rzędu, w zależności od tego, do jakiej logiki klasycznej zostaną dołączone operatory temporalne.

Rozpatrywana w tej pracy logikę to temporalny rachunek zdań rozszerzony o możliwość wprowadzenia symboli predykatowych, bądź jako logikę temporalną pierwszego rzędu, w której nie można stosować kwantyfikatorów.

### 2.1.1 Struktura czasu

Logiki temporalne dzięki swojej naturze pozwalają na wprowadzenie wnioskowania uwzględniającego czas. Przedstawiając zagadnienia związane z czasem można skorzystać z pojęcia struktury czasowej  $T = (T, <)$ , gdzie  $T$  jest zbiorem punktów czasowych, zaś  $<$  binarną relacją poprzedzania zdefiniowaną na poszczególnych elementach należących do  $T$  np.  $t_1 < t_2$ ,  $t_3 < t_4$ , ...

#### Czas dyskretny i czas ciągły

Wyróżnia się dwa rodzaje struktur czasowych: czas dyskretny i czas ciągły.

#### **Definicja 1.1**

Strukturę czasową nazywamy ciągłą, jeżeli dla każdych  $t_1, t_2 \in T$  istnieje  $t \in T$  takie, że  $t_1 < t < t_2$ .

Z drugiej jednak strony często mówi się o strukturze dyskretnej czasu. Już sam kalendarzowy podział czasu każe nam traktować czas jako sekwencję dni, tygodni, miesięcy, lat itd. Przedstawiamy to formalnie w następujący sposób:

#### **Definicja 1.2**

Strukturę czasową  $T$  nazywamy dyskretną, jeżeli istnieją takie  $t_1, t_2 \in T$ , przy czym  $t_1 \neq t_2$  oraz pomiędzy tymi punktami nie istnieje żaden inny punkt.

$\exists t_1, t_2 \in T, t_1 < t_2 \wedge \neg \exists t, t_1 < t < t_2$

Wobec tego punkt  $t_2$  jest następnikiem punktu  $t_1$ , gdy zachodzi zależność:  
 $t_2 = \text{succ}(t_1)$ .

W niniejszej pracy czas jest rozpatrywany jako struktura dyskretna. Struktura taka odpowiada modelowi baz danych widzianemu jako sekwencja stanów.

### 2.1.2 Składnia logiki temporalnej

W niniejszej pracy rozpatrywana jest odmiana logiki temporalnej będąca logicznym rachunkiem zdań rozszerzonym o operatory temporalne.

Niech  $\Omega$  oznacza przestrzeń stanów. Wówczas  $\underline{\sigma} \in \Omega$  jest skończoną sekwencję stanów oznaczoną:  $\underline{\sigma} = \langle \sigma_0, \sigma_1, \dots, \sigma_n \rangle$ ,  $n \in \mathbb{N}$ , gdzie  $\sigma_i$  jest pojedynczym stanem tej przestrzeni.

Dla  $n = 0$ ,  $\underline{\sigma}$  jest pustą sekwencją stanów oznaczoną jako  $\underline{\lambda}$ . Dolny indeks przy poszczególnych stanach oznacza jego numer. Tak więc stan  $\sigma_k$  jest stanem o numerze  $k$ .

$|\underline{\sigma}|$  definiuje długość sekwencji stanów  $\underline{\sigma}$ , a więc  $|\langle \sigma_0, \sigma_1, \dots, \sigma_n \rangle| = n+1$ .

Oczywiście możliwe jest także zdefiniowanie nieskończonej sekwencji stanów:  $\underline{\sigma} = \langle \sigma_0, \sigma_1, \dots, \sigma_k, \dots \rangle$ . Wówczas  $|\langle \sigma_0, \sigma_1, \dots, \sigma_k, \dots \rangle| = \infty$ .

Pojęciem wymaganym przy określaniu semantyki formuł temporalnych jest wartościowanie. Wartościowanie jest przypisaniem wartości zmiennym wolnym w formule. Dokładna definicja tego pojęcia zostanie przedstawiona w rozdziale II. Każda formuła

powinna być interpretowana wraz z odpowiednim wartościowaniem. Jednak w celu uproszczenia dowodów przedstawionych w dalszej części tego rozdziału, wartościowanie zostało opuszczone. Nie ma to oczywiście wpływu na same metody dowodzenia.

### Definicja 1.3

Alfabet logiki temporalnej składa się z następujących elementów:

1. Zbioru formuł atomowych;
2. Zbioru połączeń logicznych  $\neg, \Leftrightarrow, \Rightarrow, \wedge, \vee$ ;
3. Zbioru operatorów temporalnych czasu przeszłego: *always-past, sometime-past, previous, since*;
4. Zbioru operatorów temporalnych czasu przyszłego: *always-future, sometime-future, next, until*;
5. Nawiasów: ( , )

■

### Definicja 1.4

Składnia logiki temporalnej

Składnia logiki temporalnej definiowana jest następująco:

1. Formuła atomowa jest poprawną syntaktycznie formułą logiki temporalnej;
2. Jeśli A i B są poprawnymi formułami logiki temporalnej wówczas: A, B,  $\neg A$ ,  $A \Leftrightarrow B$ ,  $A \Rightarrow B$ ,  $A \wedge B$ ,  $A \vee B$ ,  $A \delta B$  (gdzie  $\delta$  - dwuargumentowy operator logiki temporalnej) oraz  $\delta A$  (gdzie  $\delta$  - jednoargumentowa operator logiki temporalnej) są także poprawnymi formułami logiki temporalnej

■

Formuła, która nie posiada operatorów temporalnych jest formułą nietemporalną, a formuła posiadająca przynajmniej jeden operator temporalny to formułą temporalną.

Logika umożliwiająca stosowanie operatorów temporalnych czasu przyszłego określana jest jako logika temporalna czasu przyszłego (FTL), logika zawierająca operatory temporalne czasu przeszłego jest logiką temporalną czasu przeszłego (PTL), natomiast logika pozwalającą na stosowanie zarówno operatorów czasu przeszłego jak i przyszłego jest logiką temporalną mieszaną (MTL).

### Definicja 1.5

Formuła temporalna jest formułą zależną, jeżeli argumentem przynajmniej jednego operatora temporalnego występującego w tej formule jest formuła temporalna. W przeciwnym wypadku formuła jest formułą niezależną.

■

### Przykład 1.1

Przykładem formuły zależnej może być formuła :  $\varphi_1 = \text{sometime\_future } (A \wedge B \text{ since } C)$ , a przykładem formuły niezależnej może być formuła  $\varphi_2 = \text{sometime\_future } (A \wedge B)$

## 2.2 Operatory temporalne czasu przeszłego PTL

Podrozdział ten wyjaśnia semantykę operatorów PTL zarówno ograniczonych metrycznie jak i bez takich ograniczeń oraz wprowadza i uzasadnia własności wnioskowania potrzebne w dalszej części pracy.

### 2.2.1 Semantyka operatorów PTL

Wprowadzenie operatorów metrycznych do logiki PTL powoduje rozszerzenie semantyki operatorów temporalnych tej logiki. Znaczenie poszczególnych operatorów temporalnych jest zależne od tego, jaki znak występuje przy operatorze metrycznym ( $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ). Definicja 1.6 przedstawia semantykę operatorów PTL.

#### Definicja 1.6

Niech  $\varphi$ ,  $\varphi_1$ ,  $\varphi_2$  będą formułami temporalnymi,  $\rho$  formułą atomową,  $\underline{\sigma}$  sekwencją stanów, w których formuły te są rozpatrywane,  $k$  - punktem z przedziału  $\langle 0, |\underline{\sigma}|-1 \rangle$ , w którym określana jest prawdziwość formuły,  $s$  - operatorem metrycznym.

Semantyka standardowych operatorów logicznych zdefiniowana jest w sposób następujący:

- |                                                                  |                                                                                                   |
|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| 1a) $[\underline{\sigma}, k] \models \rho$                       | wtw. $\sigma_k \models \rho$                                                                      |
| 1b) $[\underline{\sigma}, k] \models \neg\varphi$                | wtw. $[\underline{\sigma}, k] \not\models \varphi$                                                |
| 1c) $[\underline{\sigma}, k] \models \varphi_1 \wedge \varphi_2$ | wtw. $[\underline{\sigma}, k] \models \varphi_1$ oraz $[\underline{\sigma}, k] \models \varphi_2$ |
| 1d) $[\underline{\sigma}, k] \models \varphi_1 \vee \varphi_2$   | wtw. $[\underline{\sigma}, k] \models \varphi_1$ lub $[\underline{\sigma}, k] \models \varphi_2$  |

Semantyka operatorów PTL bez operatorów metrycznych zdefiniowana jest w sposób następujący:

- |                                                                          |                                                                                                                                                                            |
|--------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2a) $[\underline{\sigma}, k] \models \text{always-past } \varphi$        | wtw. $\forall i. 0 \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi$                                                                                      |
| 2b) $[\underline{\sigma}, k] \models \text{sometime-past } \varphi$      | wtw. $\exists i. 0 \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi$                                                                                      |
| 2c) $[\underline{\sigma}, k] \models \text{previous } \varphi$           | wtw. $k > 0 \Rightarrow [\underline{\sigma}, k-1] \models \varphi$                                                                                                         |
| 2d) $[\underline{\sigma}, k] \models \varphi_1 \text{ since } \varphi_2$ | wtw. $\exists j. 0 \leq j \leq k \wedge [\underline{\sigma}, j] \models \varphi_2 \wedge \forall i. j \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1$ |

Semantyka operatorów PTL z operatorami metrycznymi zdefiniowana jest dla dwóch przypadków:

Dla  $s \leq k$  semantyka operatorów PTL zdefiniowana jest następująco:

- |                                                                                       |                                                                                                                                                                                  |
|---------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3a) $[\underline{\sigma}, k] \models \text{always-past } <_{(\leq)} s \varphi$        | wtw. $\forall p. k-s < (\leq) p \leq k \Rightarrow [\underline{\sigma}, p] \models \varphi$                                                                                      |
| 3b) $[\underline{\sigma}, k] \models \text{sometime-past } <_{(\leq)} s \varphi$      | wtw. $\exists p. k-s < (\leq) p \leq k \Rightarrow [\underline{\sigma}, p] \models \varphi$                                                                                      |
| 3c) $[\underline{\sigma}, k] \models \varphi_1 \text{ since } <_{(\leq)} s \varphi_2$ | wtw. $\exists p. k-s < (\leq) p \leq k \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1$ |
| 3d) $[\underline{\sigma}, k] \models \text{always-past } >_{(\geq)} s \varphi$        | wtw. $\forall p. 0 \leq p < (\leq) k-s \Rightarrow [\underline{\sigma}, p] \models \varphi$                                                                                      |
| 3e) $[\underline{\sigma}, k] \models \text{sometime-past } >_{(\geq)} s \varphi$      | wtw. $\exists p. 0 \leq p < (\leq) k-s \Rightarrow [\underline{\sigma}, p] \models \varphi$                                                                                      |

3f)  $[\underline{\sigma}, k] \models \varphi_1 \text{ since } >_{(\geq)} s \varphi_2$

wtw.  $\exists p. 0 \leq p \leq (<) k - s \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i, p \leq i \leq k \Rightarrow [\underline{\sigma}, k] \models \varphi_1$

Dla  $s > k$  semantyka operatorów PTL zdefiniowana jest następująco:

4a)  $[\underline{\sigma}, k] \models \text{always-past } <_{(\leq)} s \varphi$

przyjmuje wartość false

4b)  $[\underline{\sigma}, k] \models \text{sometime-past } <_{(\leq)} s \varphi$

wtw.  $\exists p. 0 \leq p \leq k \Rightarrow [\underline{\sigma}, p] \models \varphi$

4c)  $[\underline{\sigma}, k] \models \varphi_1 \text{ since } <_{(\leq)} s \varphi_2$

wtw.  $\exists p. 0 \leq p \leq k \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1$

4d)  $[\underline{\sigma}, k] \models \text{always-past } >_{(\geq)} s \varphi$

przyjmuje wartość false

4e)  $[\underline{\sigma}, k] \models \text{sometime-past } >_{(\geq)} s \varphi$

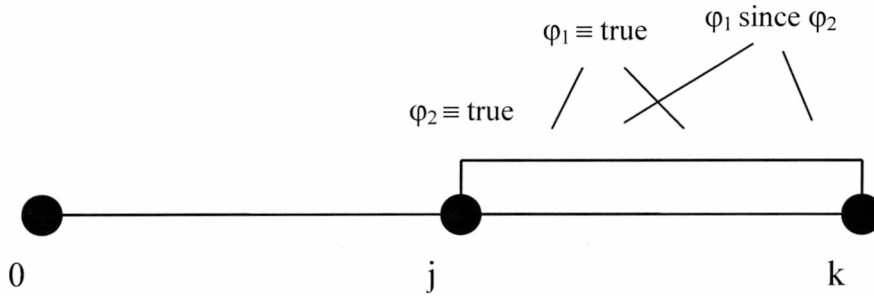
przyjmuje wartość false

4f)  $[\underline{\sigma}, k] \models \varphi_1 \text{ since } >_{(\geq)} s \varphi_2$

przyjmuje wartość false

### Przykład 1.2

Rysunek 1.1 ilustruje semantykę formuły:  $[\underline{\sigma}, k] \models \varphi_1 \text{ since } \varphi_2$



Stan początkowy

Rys.1.1 Spełnienie formuły  $[\underline{\sigma}, k] \models \varphi_1 \text{ since } \varphi_2$

Formuła  $[\underline{\sigma}, k] \models \varphi_1 \text{ since } \varphi_2$  jest spełniona w sekwencji stanów  $\underline{\sigma}$  w punkcie k, jeżeli formuła  $\varphi_2$  spełniona jest w punkcie j, a formuła  $\varphi_1$  spełniona jest w każdym punkcie i, takim, że  $j \leq i \leq k$ .

■

### 2.2.2 Własności wnioskowania operatorów PTL bez ograniczeń metrycznych

Jednym z celów niniejszej pracy jest wykorzystanie w formułach temporalnych operatorów metrycznych. Nic jednak nie stoi na przeszkodzie, aby była możliwość stosowania zarówno operatorów temporalnych nieograniczonych metrycznie jak i ograniczonych metrycznie w jednej formule. Wobec tego warto wykorzystać wprowadzone w pracy [Zhou 1994a] własności 1.1a - 1.1c.

#### Własność 1.1a

$$\varphi_1 \text{ since } \varphi_2 \Leftrightarrow \varphi_1 \wedge (\varphi_2 \vee \text{previous}(\varphi_1 \text{ since } \varphi_2))$$

**Własność 1.1b**

$$\text{sometime-past } \varphi \quad \Leftrightarrow \quad \varphi \vee \text{previous sometime-past } \varphi$$

**Własność 1.1c**

$$\text{always-past } \varphi \quad \Leftrightarrow \quad \varphi \wedge \text{previous always-past } \varphi$$

■

Dowody własności 1.1a - 1.1c znajdują się z pracy [Zhou 1994a].

**2.2.3 Własności operatorów PTL z ograniczeniami metrycznymi**

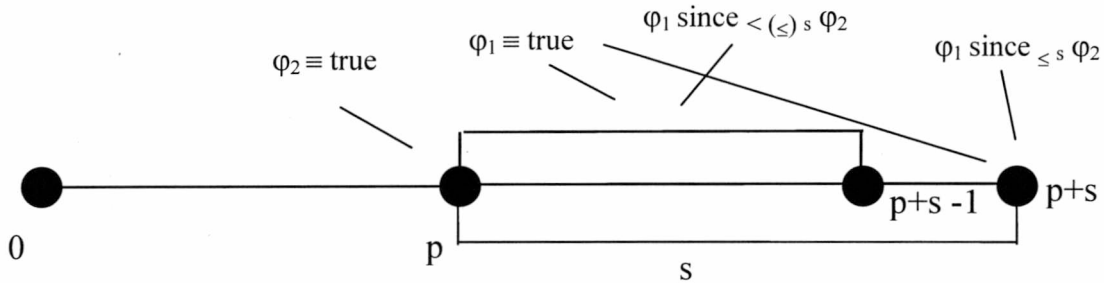
Celem tego podrozdziału jest przedstawienie i udowodnienie własności operatorów temporalnych wykorzystanych później w algorytmach weryfikacji w rozdziale II. Algorytmy weryfikacji formuł PTL bez ograniczeń metrycznych wykorzystują w procesie wnioskowania jedynie własności 1.1a - 1.1c. Wprowadzenie operatorów metrycznych implikuje konieczność zdefiniowania nowych reguł wynikających z własności definicji operatorów temporalnych PTL. W związku z tym ten podrozdział definiuje i udowadnia takie własności. Większość dowodów własności prowadzonych jest metodą „nie wprost”.

**Własności operatora since****Własność 1.2a**

$$\exists p. k - p < (\leq) s \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ since } < (\leq) s \varphi_2$$

**Dowód 1.2a**

Własność ta jest innym zapisem definicji operatora **since**<sub><( $\leq$ )s</sub>.



Rys.1.2. Spełnienie formuły  $\varphi_1 \text{ since } < (\leq) s \varphi_2$

Rysunek 1.2a obrazuje własność 1.2a. Z faktu, że formuła  $\varphi_2$  jest spełniona w punkcie p, a formuła  $\varphi_1$  w każdym punkcie między punktem p a punktem k, który jest bliżej punktu p niż s stanów wynika, że spełniona jest formuła :  $\varphi_1 \text{ since } <_s \varphi_2$  w każdym punkcie  $k \in \langle p, p+s-1 \rangle$ . Ponadto formuła  $\varphi_1 \text{ since } \leq_s \varphi_2$  jest spełniona także przy tych założeniach dodatkowo w punkcie  $p+s$ . Własność 1.2a jest równoważna własności następującej:

### Własność 1.2b

$\exists p. k - s < (\leq) p \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1 \text{ since } \varphi_2) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ since } < (\leq) s \varphi_2$ .

### Dowód 1.2b

Aby udowodnić tę własność wystarczy zauważyć, że:

$$[\underline{\sigma}, k] \models \varphi_1 \text{ since } \varphi_2 \Rightarrow [\underline{\sigma}, k] \models \varphi_1$$

Dowód tej implikacji wynika bezpośrednio z semantyki since. Jeżeli bowiem:

$$[\underline{\sigma}, k] \models \varphi_1 \text{ since } \varphi_2 \equiv \exists p. 0 \leq k-s < (\leq) p \leq k \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1 \Rightarrow [\underline{\sigma}, k] \models \varphi_1$$

W związku z tym lewą stronę implikacji własności 1.2b można zapisać:

$$\exists p. k - s < (\leq) p \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1 \text{ since } \varphi_2) \Rightarrow$$

$\exists p. k - s < (\leq) p \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ since } < (\leq) s \varphi_2$ ,  
co prowadzi do własności 1.2a

■

### Własność 1.2c

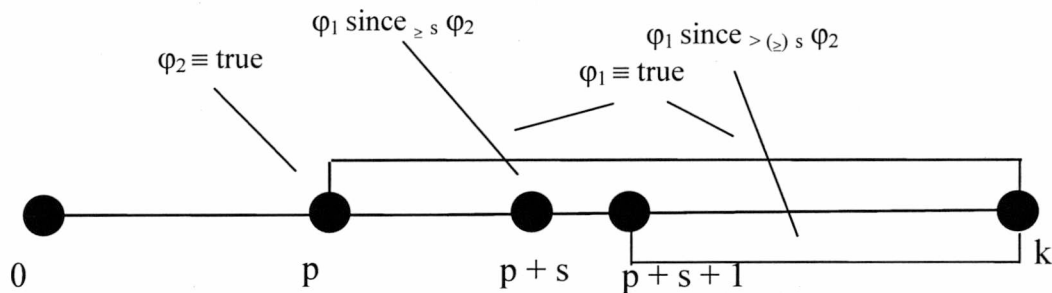
$\exists p. [\underline{\sigma}, p] \models \varphi_2 \wedge k - p > (\geq) s \wedge \forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1 \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ since } > (\geq) s \varphi_2$

■

### Dowód 1.2c

Własność ta jest innym zapisem definicji operatora  $\text{since}_{>(\geq)s}$ .

■



Rys.1.3 Spełnienie formuły  $\varphi_1 \text{ since}_{>(\geq)s} \varphi_2$

Jeżeli formuła  $\varphi_2$  jest spełniona w punkcie p oraz w każdym punkcie z przedziału  $\langle p, k \rangle$  spełniona jest formuła  $\varphi_1$  to formuła  $\varphi_1 \text{ since}_{>s} \varphi_2$  jest spełniona w każdym punkcie z przedziału  $\langle p+s+1, k \rangle$  (przy czym  $p+s \leq k$ ). Dodatkowo formuła  $\varphi_1 \text{ since}_{\geq s} \varphi_2$  jest spełniona także w punkcie p+s.

Własność 1.2c jest równoważna następującej:



### Własność 1.2d

$\exists p. [\underline{\sigma}, p] \models \varphi_2 \wedge k - p > (\geq) s \wedge \forall i. p \leq i \leq k \Rightarrow [\underline{\sigma}, i] \models \varphi_1 \text{ since } \varphi_2 \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ since } >(\geq)_s \varphi_2$  ■

Dowód 1.2d

Analogicznie jak w przypadku 1.2b.

### Własności operatora sometime-past

#### Własność 1.3a

$\exists p. [\underline{\sigma}, p] \models \varphi \wedge p \leq k < (\leq) p+s \Rightarrow [\underline{\sigma}, k] \models \text{sometime-past } <(\leq)_s \varphi$  ■

#### Dowód 1.3a

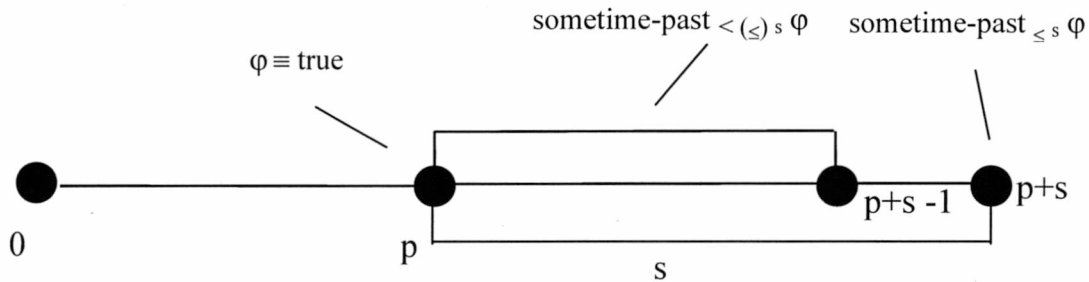
Dowód „nie wprost”. Należy doprowadzić do sprzeczności zdanie:

$\exists p. [\underline{\sigma}, p] \models \varphi \wedge p \leq k < (\leq) p+s \wedge \neg [\underline{\sigma}, k] \models \text{sometime-past } <(\leq)_s \varphi \equiv (\text{def})$

$\exists p. [\underline{\sigma}, p] \models \varphi \wedge p \leq k < (\leq) p+s \wedge \neg \exists m. k-s < (\leq) m \leq k \Rightarrow [\underline{\sigma}, m] \models \varphi \equiv$

$\exists p. [\underline{\sigma}, p] \models \varphi \wedge p \leq k < (\leq) p+s \wedge \forall m. k-s < (\leq) m \leq k \wedge \neg [\underline{\sigma}, m] \models \varphi$

Sprzeczność, gdyż z jednej strony:  $\exists p. [\underline{\sigma}, p] \models \varphi \wedge k - s < (\leq) p \leq k$ , a z drugiej:  $\forall m. k-s < (\leq) m \leq k \wedge \neg [\underline{\sigma}, m] \models \varphi$ . Twierdzenie jest prawdziwe. ■



Rys.1.4 Ilustracja własności 1.3a

Przedstawiona na rysunku 1.4 ilustracja graficzna wskazuje, że jeżeli w punkcie  $p$  spełniona jest formuła  $\varphi$  to formuła  $\text{sometime-past } <_s \varphi$  spełniona jest we wszystkich stanach aż do punktu  $p+s-1$ , a formuła  $\text{sometime-past } \leq_s \varphi$  we wszystkich sekwencjach stanów aż do punktu  $p+s$ .

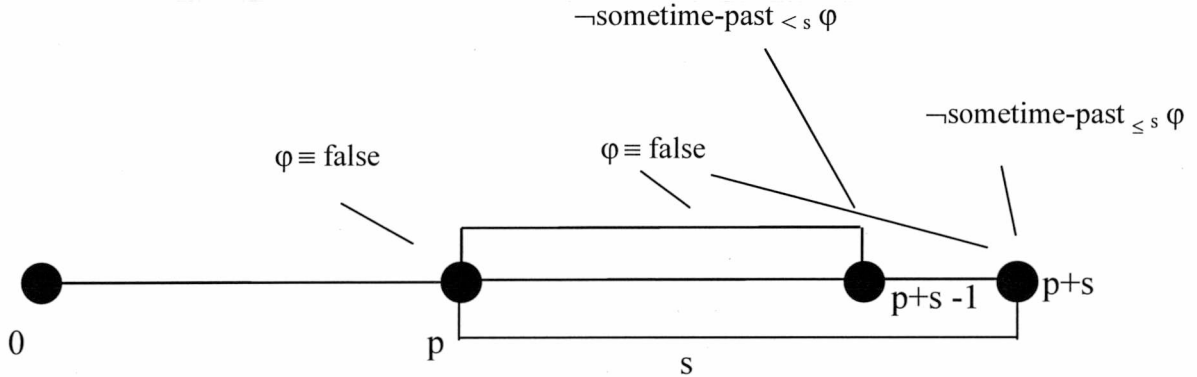
#### Własność 1.3b

$(\forall k. p \leq k < (\leq) p+s \wedge \neg [\underline{\sigma}, k] \models \varphi) \Rightarrow \neg [\underline{\sigma}, p+s-1(p+s)] \models \text{sometime-past } <(\leq)_s \varphi$  ■

#### Dowód 1.3b

Dowód jest prowadzony analogicznie jak w przypadku własności 1.3a. Należy udowodnić sprzeczność w poniższym zdaniu:

$(\forall k. p \leq k < (\leq) p+s \wedge \neg[\underline{\sigma}, k] \models \varphi) \wedge [\underline{\sigma}, p+s-1(p+s)] \models \text{sometime-past } < (\leq) s \varphi \equiv (\text{def})$   
 $(\forall k. p \leq k < (\leq) p+s \wedge \neg[\underline{\sigma}, k] \models \varphi) \wedge (\exists m. p+s-1(p+s)-s < (\leq) m \leq p+s-1(p+s) \Rightarrow \neg[\underline{\sigma}, m] \models \varphi)$   
 $\equiv (\forall k. p \leq k < (\leq) p+s \wedge \neg[\underline{\sigma}, k] \models \varphi) \wedge (\exists m. p-1(p) < (\leq) m \leq p+s-1(p+s) \Rightarrow \neg[\underline{\sigma}, m] \models \varphi) \equiv$   
 $(\forall k. p \leq k < (\leq) p+s \wedge \neg[\underline{\sigma}, k] \models \varphi) \wedge (\exists m. p \leq m < (\leq) p+s \Rightarrow \neg[\underline{\sigma}, m] \models \varphi)$   
 co prowadzi do oczywistej sprzeczności, która kończy dowód własności.



Rys.1.5 Ilustracja własności 1.3b

Rysunek 1.5 jest ilustracją graficzną własności 1.4b. Z faktu, że formuła  $\varphi$  nie jest spełniona w żadnym punkcie przedziału  $\langle p, p+s-1 \rangle$  można wnioskować, że formuła  $\text{sometime-past } < s \varphi$  nie jest spełniona w punkcie  $p+s-1$ . Jeżeli formuła  $\varphi$  nie jest dodatkowo spełniona w punkcie  $p+s$ , to formuła  $\text{sometime-past } \leq s \varphi$  nie jest spełniona w tym punkcie.

### Własność 1.3c

$\exists p. [\underline{\sigma}, p] \models \varphi \wedge k \geq (>) p+s \Rightarrow [\underline{\sigma}, k] \models \text{sometime-past } > (\geq) s \varphi$

### Dowód 1.3c

Należy doprowadzić do sprzeczności zdanie:

$\exists p. [\underline{\sigma}, p] \models \varphi \wedge k \geq (>) p+s \wedge \neg[\underline{\sigma}, k] \models \text{sometime-past } > (\geq) s \varphi \equiv (\text{def})$

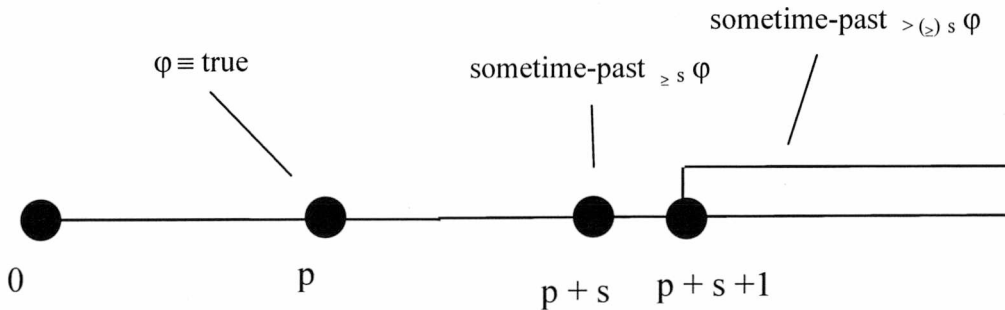
$\exists p. [\underline{\sigma}, p] \models \varphi \wedge k \geq (>) p+s \wedge \neg \exists m. 0 \leq m \leq (<) k-s \Rightarrow [\underline{\sigma}, m] \models \varphi \equiv$

$\exists p. [\underline{\sigma}, p] \models \varphi \wedge k \geq (>) p+s \wedge \forall m. 0 \leq m \leq (<) k-s \wedge [\underline{\sigma}, m] \models \varphi$

Sprzeczność, gdyż z jednej strony:  $\exists p. [\underline{\sigma}, p] \models \varphi \wedge k-s \geq (>) p$ , a z drugiej:

$\forall m. m \leq (<) k-s \wedge [\underline{\sigma}, m] \models \varphi$

Wynikła stąd sprzeczność potwierdza prawdziwość twierdzenia.



Rys.1.6 Ilustracja własności 1.3c

Ilustracja graficzna własności 1.3c z rysunku 1.6 wskazuje, że jeżeli formuła  $\varphi$  jest spełniona w punkcie  $p$ , to formuła  $\text{sometime-past } >_s \varphi$  jest spełniona w punkcie większym niż  $p+s$ , a formuła  $\text{sometime-past } \geq_s \varphi$  w każdym punkcie większym lub równym  $p+s$

### Własności operatora always-past

#### Własność 1.4a

$$\exists p. \neg [\underline{\sigma}, p] \models \varphi \wedge 0 \leq k - s < (\leq) p \leq k \Rightarrow \neg [\underline{\sigma}, k] \models \text{always-past } < (\leq)_s \varphi$$

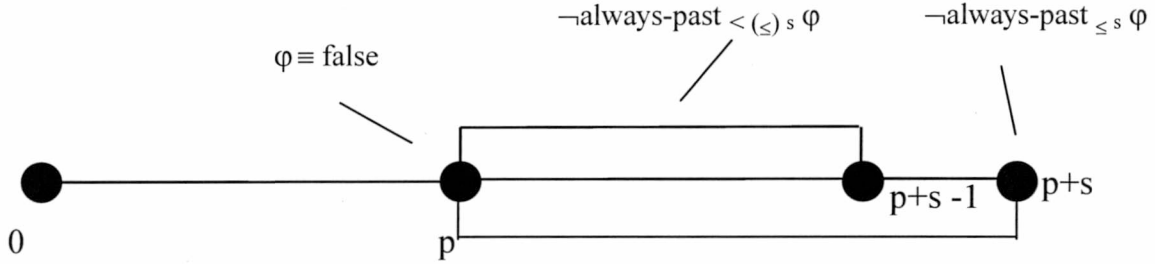
#### Dowód 1.4a:

Twierdzenie jest udowadnianie w sposób analogiczny jak w przypadku własności poprzednich. Wykazana zostanie sprzeczność poniższego zdania:

$$\exists p. \neg [\underline{\sigma}, p] \models \varphi \wedge 0 \leq k - s < (\leq) p \leq k \wedge [\underline{\sigma}, k] \models \text{always-past } < (\leq)_s \varphi \equiv$$

$$\exists p. \neg [\underline{\sigma}, p] \models \varphi \wedge 0 \leq k - s < (\leq) p \leq k \wedge (\forall m. 0 \leq k - s < (\leq) m \leq k \Rightarrow [\underline{\sigma}, m] \models \varphi)$$

Sprzeczność gdyż z jednej strony  $\exists p. 0 \leq k - s < (\leq) p \leq k \wedge \neg [\underline{\sigma}, p] \models \varphi$ , a z drugiej:  $\forall m. 0 \leq k - s < (\leq) m \leq k \Rightarrow [\underline{\sigma}, m] \models \varphi$ . Wobec powyższego twierdzenie jest spełnione.



Rys.1.7 Ilustracja własności 1.4a

Rysunek 1.7 jest ilustracją graficzną własności 1.4a. Z faktu, że formuła  $\varphi$  nie jest spełniona w punkcie  $p$  wynika, że formuła  $\text{always-past } <_s \varphi$  nie jest spełniona w żadnej punkcie z przedziału  $\langle p, p+s-1 \rangle$ . Dodatkowo formuła  $\text{always-past } \leq_s \varphi$  nie jest także spełniona w punkcie  $p+s$ . ■

#### Własność 1.4b

$$(\forall k. p \leq k < (\leq) p+s \wedge [\underline{\sigma}, k] \models \varphi) \Rightarrow [\underline{\sigma}, p+s-1(p+s)] \models \text{always-past } < (\leq)_s \varphi$$

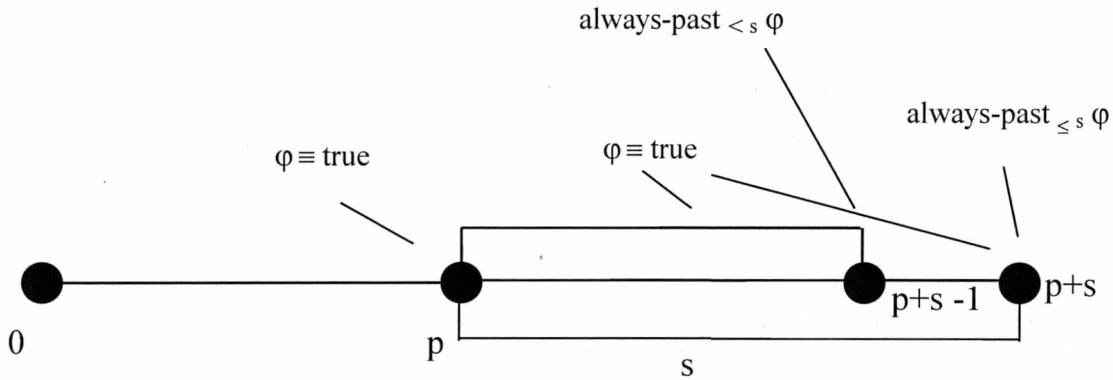
#### Dowód 1.4b

Własność udowadniania jest poprzez doprowadzenie do sprzeczności zdania:

$$(\forall k. p \leq k < (\leq) p+s \wedge [\underline{\sigma}, k] \models \varphi) \wedge \neg [\underline{\sigma}, p+s-1(p+s)] \models \text{always-past } < (\leq)_s \varphi \equiv (\text{def})$$

$$(\forall k. p \leq k < (\leq) p+s \wedge [\underline{\sigma}, k] \models \varphi) \wedge \neg (\forall m. p-1(p) < (\leq) m \leq p+s-1(p+s) \Rightarrow [\underline{\sigma}, m] \models \varphi) \equiv (\forall k. p \leq k < (\leq) p+s \wedge [\underline{\sigma}, k] \models \varphi) \wedge \neg (\forall m. p \leq m < (\leq) p+s \Rightarrow [\underline{\sigma}, m] \models \varphi)$$

Sprzeczność udowadnia własność 1.4b



Rys.1.8 Ilustracja własności 1.4b

Rysunek 1.8 jest ilustracją graficzną własności 1.4b. Jeżeli formuła  $\phi$  jest spełniona w każdym punkcie z przedziału  $\langle p, p+s-1 \rangle$  to formuła  $\text{always-past}_{<_s} \phi$  jest spełniona w punkcie  $p+s-1$ . Dodatkowo, jeżeli formuła  $\phi$  jest spełniona także w punkcie  $p+s$ , to formuła  $\text{always-past}_{\leq_s} \phi$  jest spełniona także w tym punkcie.

#### Własność 1.4c

$$\exists p. \neg[\underline{\sigma}, p] \models \phi \wedge k - p \geq (>) s \Rightarrow \neg[\underline{\sigma}, k] \models \text{always-past}_{>(\geq)_s} \phi$$

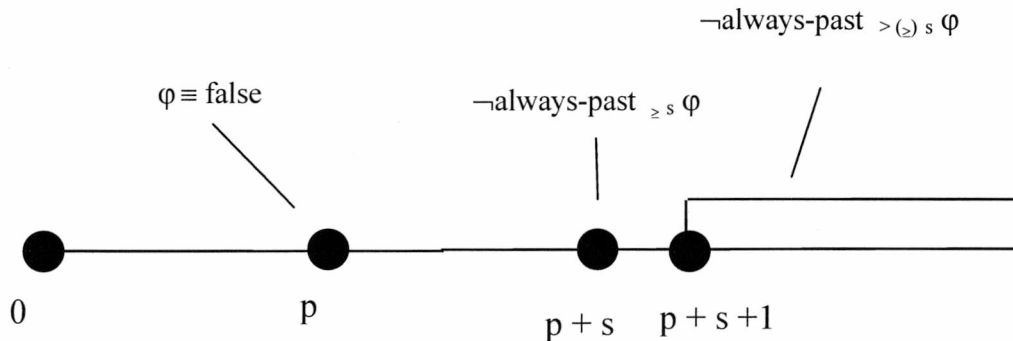
#### Dowód 1.4c:

Należy wykazać nieprawdziwość zdania:

$$\exists p. \neg[\underline{\sigma}, p] \models \phi \wedge k - p \geq (>) s \wedge [\underline{\sigma}, k] \models \text{always-past}_{>(\geq)_s} \phi \equiv (\text{def})$$

$$\exists p. \neg[\underline{\sigma}, p] \models \phi \wedge k - p \geq (>) s \wedge (\forall q. 0 \leq q \leq (<) k - s \Rightarrow [\underline{\sigma}, q] \models \phi)$$

Sprzeczność bo z jednej strony:  $\exists p. k - s \geq (>) p \wedge \neg[\underline{\sigma}, p] \models \phi$ , a z drugiej  $\forall q. k - s \geq (>) q \Rightarrow [\underline{\sigma}, q] \models \phi$ . Sprzeczność udowadnia prawdziwość twierdzenia.



Rys.1.9 Ilustracja własności 1.4c

Rysunek 1.9 ilustruje własność 1.4c. Jeżeli formuła  $\varphi$  nie jest spełniona w punkcie  $p$ , to formuła  $\text{always-past } >_s \varphi$  nie jest spełniona w żadnym punkcie większym od  $p+s$ , a formuła  $\text{always-past } \geq_s \varphi$  w żadnym punkcie większym lub równym  $p+s$ .

## 2.3 Operatory temporalne czasu przyszłego FTL

Podrozdział ten wyjaśnia definicje i semantykę operatorów FTL zarówno ograniczonych metrycznie jak i bez takich ograniczeń oraz wprowadza i udowadnia pewne własności wynikające z definicji operatorów FTL potrzebne w dalszej części pracy.

### 2.3.1 Semantyka operatorów FTL

Wprowadzenie operatorów metrycznych do logiki FTL powoduje rozszerzenie semantyki operatorów temporalnych tej logiki. Znaczenie poszczególnych operatorów temporalnych jest zależne od tego, jaki znak występuje przy operatorze metrycznym ( $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ). Definicja 1.7 przedstawia semantykę operatorów FTL.

#### Definicja 1.7

Niech  $\varphi$ ,  $\varphi_1$ ,  $\varphi_2$  będą formułami temporalnymi,  $\rho$  formułą atomową,  $\underline{\sigma}$  sekwencją stanów, w których formuły te są rozpatrywane,  $k$  - punktem z przedziału  $\langle 0, |\underline{\sigma}| - 1 \rangle$ ,  $s$  - operatorem metrycznym.

Semantyka standardowych operatorów logicznych zdefiniowana jest w sposób następujący:

- |                                                                  |                                                                                                   |
|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| 1a) $[\underline{\sigma}, k] \models \rho$                       | wtw. $\sigma_k \models \rho$                                                                      |
| 1b) $[\underline{\sigma}, k] \models \neg \varphi$               | wtw. $[\underline{\sigma}, k] \not\models \varphi$                                                |
| 1c) $[\underline{\sigma}, k] \models \varphi_1 \wedge \varphi_2$ | wtw. $[\underline{\sigma}, k] \models \varphi_1$ oraz $[\underline{\sigma}, k] \models \varphi_2$ |
| 1d) $[\underline{\sigma}, k] \models \varphi_1 \vee \varphi_2$   | wtw. $[\underline{\sigma}, k] \models \varphi_1$ lub $[\underline{\sigma}, k] \models \varphi_2$  |

Semantyka operatorów FTL bez operatorów metrycznych zdefiniowana jest w sposób następujący:

- |                                                                          |                                                                                                                                                                                                   |
|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2a) $[\underline{\sigma}, k] \models \text{always-future } \varphi$      | wtw. $\forall i. k \leq i \leq  \underline{\sigma}  - 1 \Rightarrow [\underline{\sigma}, i] \models \varphi$                                                                                      |
| 2b) $[\underline{\sigma}, k] \models \text{sometime-future } \varphi$    | wtw. $\exists i. k \leq i \leq  \underline{\sigma}  - 1 \Rightarrow [\underline{\sigma}, i] \models \varphi$                                                                                      |
| 2c) $[\underline{\sigma}, k] \models \text{next } \varphi$               | wtw. $k <  \underline{\sigma}  - 1 \Rightarrow [\underline{\sigma}, k+1] \models \varphi$                                                                                                         |
| 2d) $[\underline{\sigma}, k] \models \varphi_1 \text{ until } \varphi_2$ | wtw. $\exists j. k \leq j \leq  \underline{\sigma}  - 1 \wedge [\underline{\sigma}, j] \models \varphi_2 \wedge \forall i. k \leq i \leq j \Rightarrow [\underline{\sigma}, i] \models \varphi_1$ |

Semantyka operatorów FTL z operatorami metrycznymi zdefiniowana jest w sposób następujący:

Dla  $k+s \leq |\underline{\sigma}| - 1$  semantyka operatorów FTL zdefiniowana jest następująco:

- |                                                                                       |                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3a) $[\underline{\sigma}, k] \models \text{always-future } <_{(\leq)} s \varphi$      | wtw. $\forall p. k \leq p <_{(\leq)} k+s \Rightarrow [\underline{\sigma}, p] \models \varphi$                                                                                           |
| 3b) $[\underline{\sigma}, k] \models \text{sometime-future } <_{(\leq)} s \varphi$    | wtw. $\exists p. k \leq p <_{(\leq)} k+s \Rightarrow [\underline{\sigma}, p] \models \varphi$                                                                                           |
| 3c) $[\underline{\sigma}, k] \models \varphi_1 \text{ until } <_{(\leq)} s \varphi_2$ | wtw. $\exists p. k \leq p <_{(\leq)} k+s \Rightarrow [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1$ |

- 3d)  $[\underline{\sigma}, k] \models \text{always-future } >_{(\geq)} s \varphi$  wtw.  $\forall p. k + s < (\leq) p \Rightarrow [\underline{\sigma}, p] \models \varphi$ .
- 3e)  $[\underline{\sigma}, k] \models \text{sometime-future } >_{(\geq)} s \varphi$  wtw.  $\exists p. k + s < (\leq) p \Rightarrow [\underline{\sigma}, p] \models \varphi$ .
- 3f)  $[\underline{\sigma}, k] \models \varphi_1 \text{ until } >_{(\geq)} s \varphi_2$  wtw.  $\exists p. k + s < (\leq) p \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1$

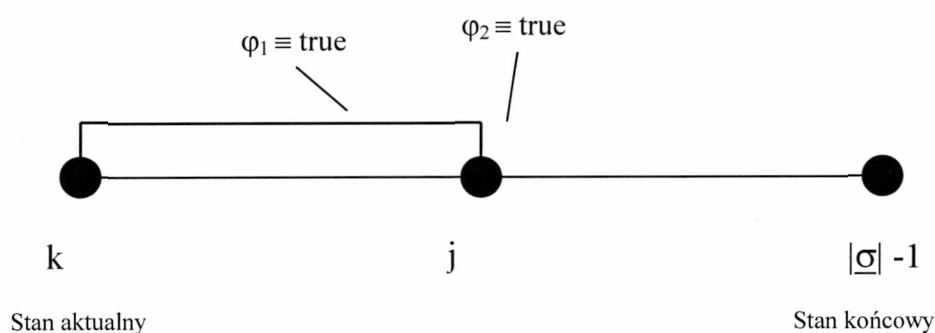
■

Dla  $k+s > |\underline{\sigma}| - 1$  semantyka operatorów FTL zdefiniowana jest następująco:

- 4a)  $[\underline{\sigma}, k] \models \text{always-future } <_{(\leq)} s \varphi$  przyjmuje wartość a false
- 4b)  $[\underline{\sigma}, k] \models \text{sometime-future } <_{(\leq)} s \varphi$  wtw.  $\exists p. k \leq p \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, p] \models \varphi$
- 4c)  $[\underline{\sigma}, k] \models \varphi_1 \text{ until } <_{(\leq)} s \varphi_2$  wtw.  $\exists p. k \leq p \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1$
- 4d)  $[\underline{\sigma}, k] \models \text{always-future } >_{(\geq)} s \varphi$  przyjmuje wartość false
- 4e)  $[\underline{\sigma}, k] \models \text{sometime-future } >_{(\geq)} s \varphi$  przyjmuje wartość false
- 4f)  $[\underline{\sigma}, k] \models \varphi_1 \text{ until } >_{(\geq)} s \varphi_2$  przyjmuje wartość false

### Przykład 1.3

Poniższy rysunek ilustruje semantykę formuły:  $[\underline{\sigma}, k] \models \varphi_1 \text{ until } \varphi_2$



Rys.1.10 Spełnienie formuły  $[\underline{\sigma}, k] \models \varphi_1 \text{ until } \varphi_2$

Formuła  $[\underline{\sigma}, k] \models \varphi_1 \text{ until } \varphi_2$  spełniona jest w punkcie  $k$ , jeżeli formuła  $\varphi_2$  spełniona jest w punkcie  $j \geq k$  oraz w każdym punkcie pomiędzy  $k$  i  $j$  spełniona jest formuła  $\varphi_1$ .

■

### 2.3.2 Własności operatorów FTL bez ograniczeń metrycznych

Logika FTL daje możliwość stosowania zarówno operatorów temporalnych ograniczonych metrycznie jak i nieograniczonych metrycznie w jednej formule. Wobec tego warto wykorzystać wprowadzone w pracy [Zhou 1994a] własności rekurencyjne 1.5a - 1.5c.

Dla operatorów FTL zachodzą następujące własności rekurencyjne:

**Własność 1.5a**

$$\varphi_1 \text{ until } \varphi_2 \Leftrightarrow \varphi_1 \wedge (\varphi_2 \vee \text{next}(\varphi_1 \text{ until } \varphi_2))$$

**Własność 1.5b**

$$\text{sometime-future } \varphi \Leftrightarrow \varphi \vee \text{next sometime-future } \varphi$$

**Własność 1.5c**

$$\text{always-future } \varphi \Leftrightarrow \varphi \wedge \text{next always-future } \varphi$$

Dowody własności 1.5a - 1.5c znajdują się z pracy [Zhou 1994a].

### 2.3.3 Własności definicji operatorów FTL z ograniczeniami metrycznymi

Celem tego podrozdziału jest przedstawienie i udowodnienie własności wnioskowania wykorzystanych później w algorytmach weryfikacji w rozdziale II. Algorytmy weryfikacji formuł PTL bez ograniczeń metrycznych wykorzystują w procesie wnioskowania jedynie własności 1.5a - 1.5c. Wprowadzenie operatorów metrycznych implikuje konieczność zdefiniowania nowych własności wnioskowania dla operatorów temporalnych FTL. W związku z tym ten podrozdział definiuje i udowadnia takie własności. Większość dowodów własności prowadzonych jest metodą „nie wprost”.

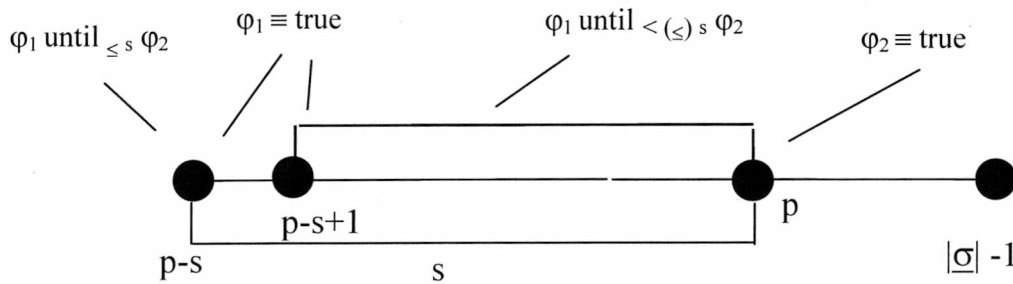
#### Własności operatora until

**Własność 1.6a**

$$\exists p. p - k < (\leq) s \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ until } < (\leq) s \varphi_2$$

**Dowód 1.6a**

Własność wynika bezpośrednio z definicji operatora  $\text{until}_{< (\leq) s}$ . ■



Rys.1.11. Spełnienie formuły  $[\underline{\sigma}, k] \models \varphi_1 \text{ until } < (\leq) s \varphi_2$



Rysunek 1.11 obrazuje własność 1.6a. Z faktu, że formuła  $\varphi_2$  jest spełniona w punkcie  $p$  pomiędzy punktami  $k$  i  $k+s$ , a formuła  $\varphi_1$  w każdym punkcie pomiędzy punktem  $k$  a punktem  $p$ , który jest bliżej punktu  $k$  niż  $s$  stanów wynika, że spełniona jest formuła :  $\varphi_1 \text{ until } <_s \varphi_2$  w każdym punkcie  $k \in \langle p-s+1, p \rangle$ . Ponadto jeżeli formuła  $\varphi_1$  spełniona jest w punkcie  $p-s$ , to formuła  $\varphi_1 \text{ until } \leq_s \varphi_2$  jest spełniona także w punkcie  $p-s$ .

#### Własność 1.6b

Własność 1.6a jest równoważna:

$$\exists p. p - k < (\leq) s \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ until } < (\leq)_s \varphi_2$$

#### Dowód 1.6b

Aby udowodnić tę własność wystarczy zauważyć, że:

$$[\underline{\sigma}, k] \models \varphi_1 \text{ until } \varphi_2 \Rightarrow [\underline{\sigma}, k] \models \varphi_1$$

Dowód tej implikacji wynika bezpośrednio z semantyki until. Jeżeli bowiem:

$$[\underline{\sigma}, k] \models \varphi_1 \text{ until } \varphi_2 \equiv \exists p. k \leq p \leq |\underline{\sigma}| - 1 \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge \forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1 \\ \Rightarrow [\underline{\sigma}, k] \models \varphi_1$$

W związku z tym lewą stronę implikacji własności 1.6b można zapisać:

$$\exists p. p - k < (\leq) s \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1) \Rightarrow$$

$$\exists p. p - k < (\leq) s \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ until } < (\leq)_s \varphi_2,$$

co prowadzi do własności 1.6a

■

Analogiczne własności można wyprowadzić w przypadku przeciwnych znaków przy operatorze metrycznym.

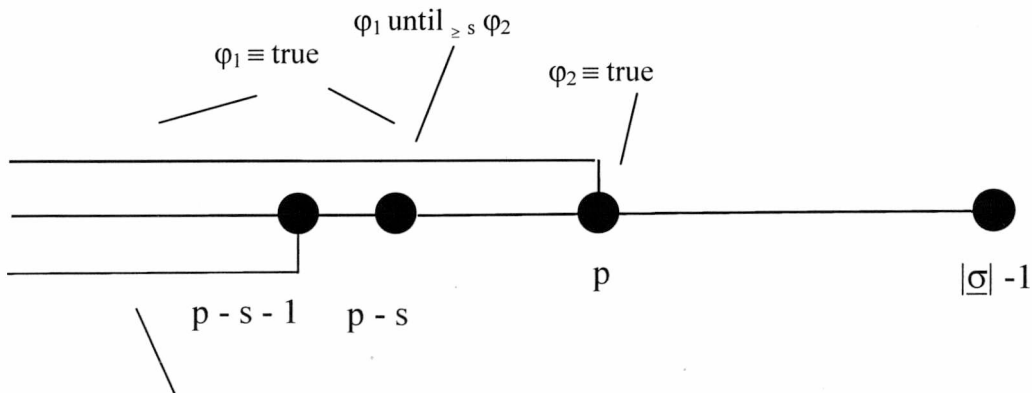
#### Własność 1.6c

$$\exists p. k \leq p \leq |\underline{\sigma}| - 1 \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ until } > (\geq)_s \varphi_2$$

#### Dowód 1.6c

Własność jest inaczej zapisaną definicją operatora  $\text{until } > (\geq)_s$ .

■



$$\varphi_1 \text{ until } >_{(\leq)} s \varphi_2$$

Rys.1.12. Spełnienie formuły  $\varphi_1 \text{ until } >_{(\leq)} s \varphi_2$

Rysunek 1.12 jest ilustracją graficzną własności 1.6c. Z faktu, że formuła  $\varphi_2$  jest spełniona w stanie  $p$  oraz, że w każdym punkcie pomiędzy  $k$  i  $p$  spełniona jest formuła  $\varphi_1$  wynika, że formuła  $\varphi_1 \text{ until } >_s \varphi_2$  spełniona jest w każdym punkcie z przedziału  $\langle k, p-s-1 \rangle$  (oczywiście, jeżeli  $k \leq p-s-1$ ). Formuła  $\varphi_1 \text{ until } \geq_s \varphi_2$  jest przy tych założeniach jeszcze dodatkowo spełniona w stanie  $p-s$ .

Na podstawie powyższej własności można wnioskować:

#### Własność 1.6d

$$\exists p. k \leq p \leq |\underline{\sigma}| - 1 \wedge [\underline{\sigma}, p] \models \varphi_2 \wedge (\forall i. k \leq i \leq p \Rightarrow [\underline{\sigma}, i] \models \varphi_1 \text{ until } \varphi_2) \Rightarrow [\underline{\sigma}, k] \models \varphi_1 \text{ until } >_{(\leq)} s \varphi_2$$

#### Dowód 1.6d

Dowód jest analogiczny jak w przypadku własności 1.6b

#### Własności definicji operatora sometime-future

##### Własność 1.7a

$$\exists p. [\underline{\sigma}, p] \models \varphi \Rightarrow (\forall k. p-s < (\leq) k \leq p \Rightarrow [\underline{\sigma}, k] \models \text{sometime-future } <_{(\leq)} s \varphi)$$

#### Dowód 1.7a

Aby udowodnić własność 1.7a należy doprowadzić do sprzeczności zdanie:

$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \neg (\forall k. p-s < (\leq) k \leq p \Rightarrow [\underline{\sigma}, k] \models \text{sometime-future } <_{(\leq)} s \varphi) \equiv$$

$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \exists k. p-s < (\leq) k \leq p \wedge \neg [\underline{\sigma}, k] \models \text{sometime-future } <_{(\leq)} s \varphi \equiv (\text{def})$$

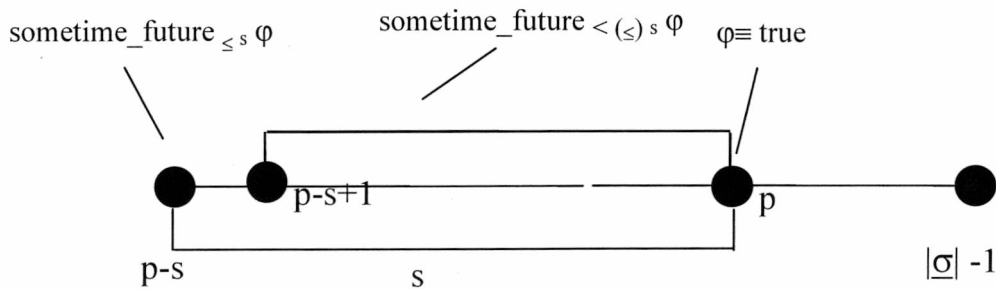
$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \exists k. p-s < (\leq) k \leq p \wedge \neg (\exists q. k \leq q < (\leq) k+s \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, q] \models \varphi) \equiv$$

$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \exists k. p-s < (\leq) k \leq p \wedge \forall q. k \leq q < (\leq) k+s \leq |\underline{\sigma}| - 1 \wedge \neg [\underline{\sigma}, q] \models \varphi$$

Sprzeczność bo z jednej strony  $\exists p. p < (\leq) k+s \wedge [\underline{\sigma}, p] \models \varphi$ , a z drugiej:

$$\forall q. q < (\leq) k+s \wedge \neg [\underline{\sigma}, q] \models \varphi$$

■



Rys.1.13. Ilustracja własności 1.7a

Z faktu, że  $\varphi$  jest spełniona w punkcie  $p$  wynika, że formuła  $\text{sometime\_future}_{<s} \varphi$  jest spełniona w każdym punkcie z przedziału  $\langle p-s+1, p \rangle$ , a formuła  $\text{sometime\_future}_{\leq s} \varphi$  w każdym punkcie z przedziału  $\langle p-s, p \rangle$ . ■

#### Własność 1.7b

$$(\forall k. p-s \leq k < (\leq) p \leq |\underline{\sigma}| - 1 \wedge \neg[\underline{\sigma}, k] \models \varphi) \Rightarrow \neg[\underline{\sigma}, p-s] \models \text{sometime\_future}_{<(\leq)s} \varphi$$

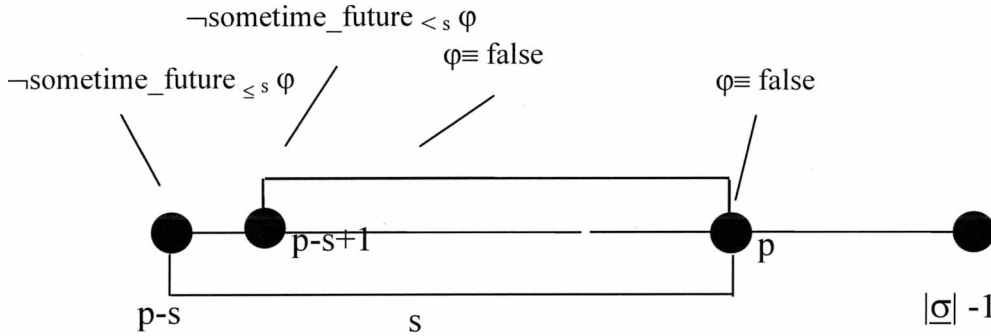
#### Dowód 1.7b

Sposób prowadzenia dowodu jest analogiczny jak w przypadku własności 1.7a. Należy doprowadzić do sprzeczności:

$$(\forall k. p-s \leq k < (\leq) p \leq |\underline{\sigma}| - 1 \wedge \neg[\underline{\sigma}, k] \models \varphi) \wedge [\underline{\sigma}, p-s] \models \text{sometime\_future}_{<(\leq)s} \varphi \equiv (\text{def})$$

$$(\forall k. p-s \leq k < (\leq) p \leq |\underline{\sigma}| - 1 \wedge \neg[\underline{\sigma}, k] \models \varphi) \wedge (\exists q. p-s \leq q < (\leq) k \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, q] \models \varphi)$$

Ewidencyjna sprzeczność udowadnia własność. ■



Rys.1.14. Ilustracja własności 1.7b

Rysunek 1.14 jest ilustracją graficzną własności 1.7b. Jeżeli formuła  $\varphi$  nie jest spełniona w każdym punkcie z przedziału  $\langle p-s+1, p \rangle$ , to formuła  $\text{sometime\_future}_{<s} \varphi$  nie jest spełniona w punkcie  $p-s-1$ . W przypadku znaku  $\leq$  przy ograniczeniu metrycznym, jeżeli  $\varphi$  nie jest spełniona w każdym punkcie z przedziału  $\langle p-s, p \rangle$ , to formuła  $\text{sometime\_future}_{\leq s} \varphi$  nie jest spełniona w punkcie  $p-s$ .

#### Własność 1.7c

$$\exists p. [\underline{\sigma}, p] \models \varphi \Rightarrow (\forall k. 0 \leq k < (\leq) p-s \Rightarrow [\underline{\sigma}, k] \models \text{sometime\_future}_{>(\geq)s} \varphi)$$

#### Dowód 1.7c

Dowód własności opiera się na doprowadzeniu do sprzeczności następującego zdania:

$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \neg(\forall k. 0 \leq k < (\leq) p-s \Rightarrow [\underline{\sigma}, k] \models \text{sometime\_future}_{>(\geq)s} \varphi) \equiv (\text{def})$$

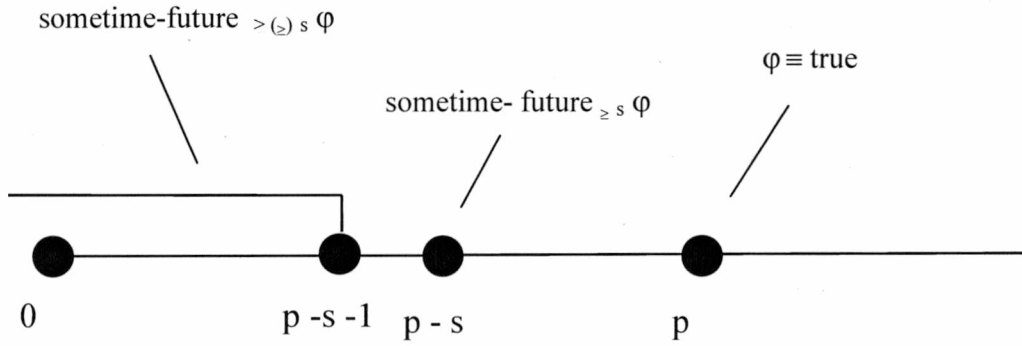
$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \exists k. 0 \leq k < (\leq) p-s \wedge \neg[\underline{\sigma}, k] \models \text{sometime\_future}_{>(\geq)s} \varphi \equiv$$

$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \exists k. 0 \leq k < (\leq) p-s \wedge \neg(\exists q. k+s < (\leq) q \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, q] \models \varphi) \equiv$$

$$\exists p. [\underline{\sigma}, p] \models \varphi \wedge \exists k. 0 \leq k < (\leq) p-s \wedge \forall q. k+s < (\leq) q \leq |\underline{\sigma}| - 1 \wedge \neg[\underline{\sigma}, q] \models \varphi$$

Dochodzimy do sprzeczności, gdyż z jednej strony:  $\exists p. k+s < (\leq) p$ ,  $\exists p. [\underline{\sigma}, p] \models \varphi$ , a z drugiej  $\forall m. k+s < (\leq) m \wedge \neg[\underline{\sigma}, q] \models \varphi$ .

Wobec tego sprzeczność udowadnia spełnienie twierdzenia. ■



Rys.1.15. Ilustracja własności 1.7c

Przedstawiona na rysunku 1.15 ilustracja graficzna wskazuje, że jeżeli formuła  $\varphi$  spełniona jest w punkcie  $p$ , to formuła  $\text{sometime-future } >_{(s)} \varphi$  spełniona jest w każdym punkcie mniejszym lub równym  $p - s - 1$ . Formuła  $\text{sometime-future } \geq_s \varphi$  spełniona jest dodatkowo jeszcze w punkcie  $p - s$ .

#### Własności definicji operatora always-future

##### Własność 1.8a

$$\exists p. \neg[\underline{\sigma}, p] \models \varphi \Rightarrow (\forall k. 0 \leq p-s < (\leq) k \leq p \Rightarrow \neg[\underline{\sigma}, k] \models \text{always-future } <_{(s)} \varphi)$$

##### Dowód 1.8a

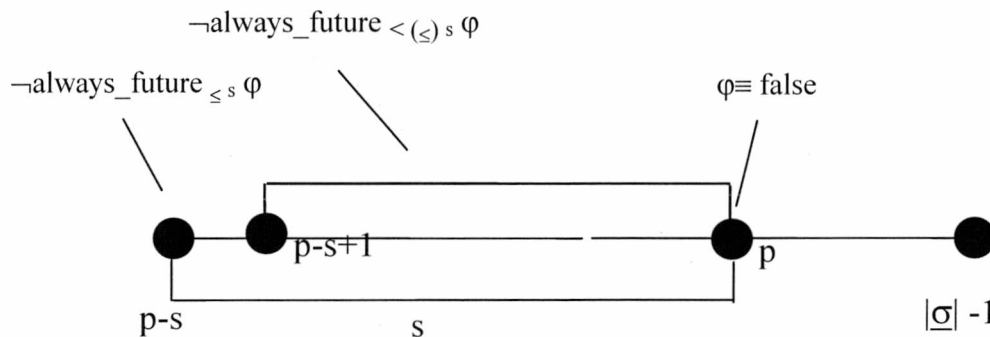
Metoda dowodzenia jest taka sama, jak w przypadku poprzedniej własności. Należy udowodnić sprzeczność powyższego zdania:

$$\exists p. \neg[\underline{\sigma}, p] \models \varphi \wedge \neg(\forall k. 0 \leq p-s < (\leq) k \leq p \Rightarrow \neg[\underline{\sigma}, k] \models \text{always-future } <_{(s)} \varphi) \equiv$$

$$\exists p. \neg[\underline{\sigma}, p] \models \varphi \wedge \exists k. 0 \leq p-s < (\leq) k \leq p \wedge [\underline{\sigma}, k] \models \text{always-future } <_{(s)} \varphi \equiv (\text{def})$$

$$\exists p. \neg[\underline{\sigma}, p] \models \varphi \wedge \exists k. 0 \leq p-s < (\leq) k \leq p \wedge \forall q. k \leq q < (\leq) k+s \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, q] \models \varphi$$

Dochodzimy do sprzeczności, gdyż z jednej strony:  $\exists p, p < (\leq) k + s$  oraz  $\neg[\underline{\sigma}, p] \models \varphi$ , a z drugiej  $\forall q. q < (\leq) k + s \Rightarrow [\underline{\sigma}, q] \models \varphi$ . Sprzeczność udowadnia poprawność twierdzenia. ■



Rys.1.16. Ilustracja własności 1.8a

Rysunek 1.16 jest ilustracją graficzną własności 1.9a. Jeżeli formuła  $\varphi$  nie jest spełniona w punkcie  $p$ , to formuła  $\text{always\_future}_{<s}\varphi$  nie jest spełniona w żadnym punkcie przedziału  $\langle p-s+1, p \rangle$ . Dodatkowo formuła  $\text{always\_future}_{\leq s}\varphi$  nie jest spełniona także w punkcie  $p-s$ .

#### Własność 1.8b

$$(\forall k. p-s \leq k < (\leq) p \wedge [\underline{\sigma}, k] \models \varphi) \Rightarrow [\underline{\sigma}, p-s] \models \text{always\_future}_{<(\leq)s}\varphi$$

#### Dowód 1.8b

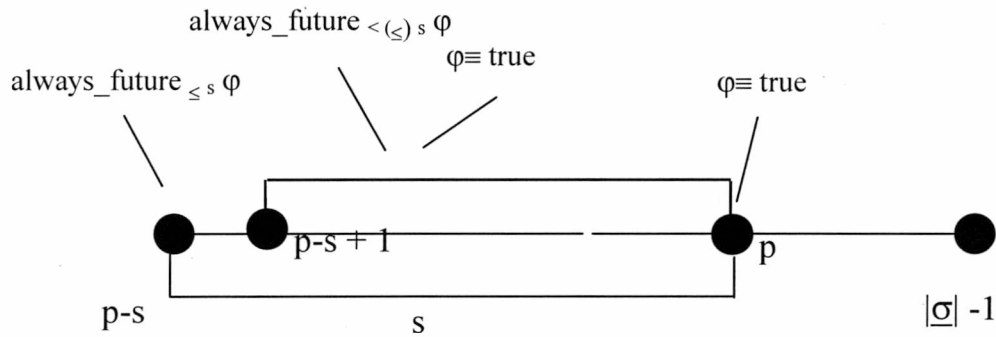
Dowód polega na doprowadzeniu do sprzeczności zdania:

$$(\forall k. p-s \leq k < (\leq) p \wedge [\underline{\sigma}, k] \models \varphi) \wedge \neg [\underline{\sigma}, p-s] \models \text{always\_future}_{<(\leq)s}\varphi \equiv (\text{def})$$

$$(\forall k. p-s \leq k < (\leq) p \wedge [\underline{\sigma}, k] \models \varphi) \wedge (\neg \forall q. p-s \leq q < (\leq) p \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, q] \models \varphi)$$

Sprzeczność udowadnia poprawność własności.

■



Rys.1.17. Ilustracja własności 1.8b

Na rysunku 1.17 widać, że jeżeli formuła  $\varphi$  jest spełniona w każdym punkcie z przedziału  $\langle p-s+1, p \rangle$  wynika, że formuła  $\text{always\_future}_{<s}\varphi$  jest spełniona w punkcie  $p-s+1$ . Formuła  $\text{always\_future}_{\leq s}\varphi$  jest spełniona także w punkcie  $p-s$ .

#### Własność 1.8c

$$\exists p. \neg [\underline{\sigma}, p] \models \varphi \Rightarrow (\forall k. 0 \leq k < (\leq) p-s \Rightarrow \neg [\underline{\sigma}, k] \models \text{always\_future}_{>(\geq)s}\varphi)$$

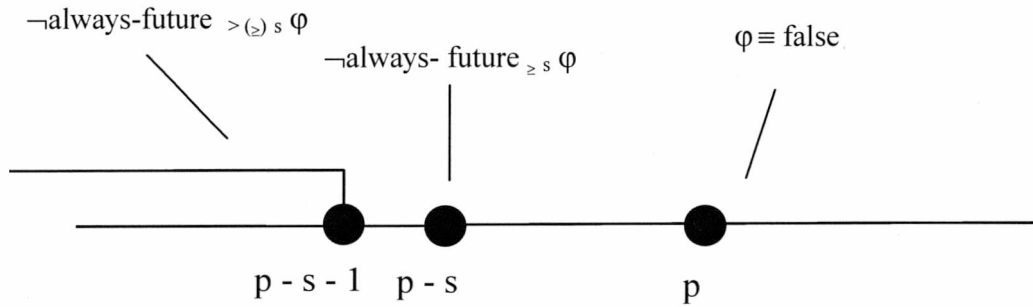
#### Dowód 1.8c

W związku z tym musi być doprowadzone do sprzeczności zdanie:

$$\exists p. \neg [\underline{\sigma}, p] \models \varphi \wedge \neg (\forall k. 0 \leq k < (\leq) p-s \Rightarrow \neg [\underline{\sigma}, k] \models \text{always\_future}_{>(\geq)s}\varphi) \equiv$$

$\exists p. \neg[\underline{\sigma}, p] \models \varphi \wedge \neg(\forall k. 0 \leq k < (\leq) p - s \Rightarrow \neg[\underline{\sigma}, k] \models \text{always-future}_{>(\geq)s} \varphi) \equiv$   
 $\exists p. \neg[\underline{\sigma}, p] \models \varphi \wedge \exists k. 0 \leq k < (\leq) p - s \wedge [\underline{\sigma}, k] \models \text{always-future}_{>(\geq)s} \varphi \equiv (\text{def})$   
 $\exists p. \neg[\underline{\sigma}, p] \models \varphi \wedge \exists k. 0 \leq k < (\leq) p - s \wedge \forall q. k + s < (\leq) q \leq |\underline{\sigma}| - 1 \Rightarrow [\underline{\sigma}, q] \models \varphi$   
 Sprzeczność, bo z jednej strony  $\exists p. k + s < (\leq) p \wedge \neg[\underline{\sigma}, p] \models \varphi$ , a z drugiej:  
 $\forall q. k + s < (\leq) q \Rightarrow [\underline{\sigma}, q] \models \varphi$

■



Rys.1.18. Ilustracja własności 1.8c

Rysunek 1.18 jest ilustracją graficzną własności 1.8c. Jeżeli formuła  $\varphi$  nie jest spełniona w stanie  $p$ , to formuła  $\text{always-future}_{>s} \varphi$  nie jest spełniona w każdym punkcie mniejszym lub równym  $p-s-1$ , a formuła  $\text{always-future}_{\geq s} \varphi$  dodatkowo jeszcze w punkcie  $p-s$ .

## 2.4 Mieszana logika temporalna MTL

Trzecim i ostatnim rodzajem logiki rozpatrywanym w niniejszej pracy jest mieszana logika temporalna MTL. Logika ta daje możliwość używania zarówno operatorów logiki temporalnej czasu przeszłego jak i przyszłego w jednej formule.

Formuły logiki MTL są rozpatrywane w sekwencji stanów  $\underline{\sigma} = \langle \sigma_0, \sigma_1, \dots, \sigma_k, \dots \rangle$ . Zachodzi jednak pewna istotna różnica pomiędzy tymi formułami a formułami FTL oraz PTL.

Formuła PTL w sekwencji stanów  $\underline{\sigma}$  w punkcie czasu  $k$  jest właściwie rozpatrywana w sekwencji stanów  $\underline{\sigma}_k = \langle \sigma_0, \sigma_1, \dots, \sigma_k \rangle$ . Pozostałe stany z sekwencji stanów  $\underline{\sigma}$  nie mają wpływu na formułę PTL.

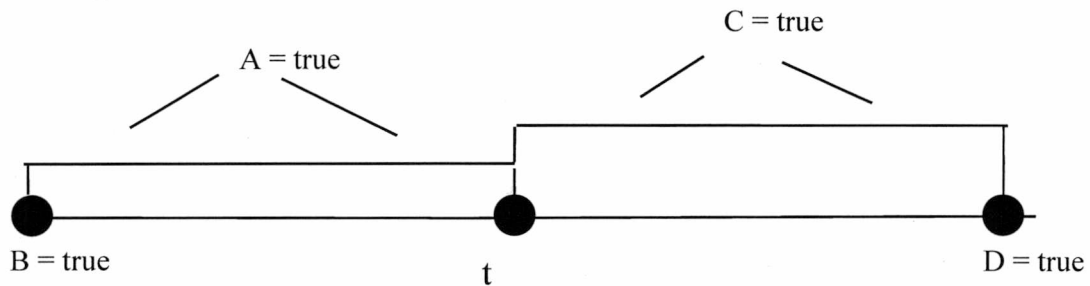
Podobnie jest z formułami typu FTL. Formuła FTL w punkcie  $k$  jest rozpatrywana w sekwencji stanów  $\underline{\sigma}_k = \langle \sigma_k, \sigma_{k+1}, \dots \rangle$ .

W przypadku formuł MTL sprawa wygląda w sposób nieco inny. Z uwagi na to, że składnia i semantyka formuł MTL jest złożeniem składni i semantyki formuł FTL i PTL, sekwencje stanów, w których formuła taka jest rozpatrywana jest również złożeniem stanów rozpatrywania formuł PTL i FTL. W związku z tym formuły MTL rozpatrywane są w całej sekwencji stanów  $\underline{\sigma}$ .

Formuły MTL wprowadzają także koncepcję punktu odniesienia [Klimek 1999]. Punkt odniesienia w tym znaczeniu to punkt w którym można odnieść się jednocześnie do

przeszłości jak i do przyszłości. Możliwości takiej nie dają ani formuły PTL ani FTL. Przykładem formuły MTL może być:  $\varphi \equiv A \text{ since } B \wedge C \text{ until } D$ .

Formuła taka jest spełniona w punkcie  $t$ , jeżeli wartości formuł  $A$ ,  $B$ ,  $C$  i  $D$  są takie jak na rysunku 1.19.



Rys. 1.19. Warunki spełnienia formuły  $A \text{ since } B \wedge C \text{ until } D$  w punkcie  $t$ .

Taką formułę nie jest łatwo zastąpić formułą PTL ani formułą FTL. W związku z tym logika MTL przydaje się właśnie w sytuacjach, gdy istnieje potrzeba spojrzenia z danej chwili jednocześnie w przeszłość i przyszłość.

Z uwagi na fakt, że logika MTL jest złączeniem logik FTL i PTL, wszystkie własności wnioskowania wprowadzone w poprzednich rozdziałach zarówno dla logiki FTL jak i PTL mają również zastosowanie do logiki MTL.

### 3. METODY WERYFIKACJI FORMUŁ TEMPORALNYCH

Niniejszy rozdział zawiera metody weryfikacji formuł temporalnych. W rozdziale I przedstawiono trzy rodzaje logik temporalnych: PTL, FTL oraz MTL, przy czym każdą z nich można jeszcze podzielić na dwa rodzaje: logikę z ograniczeniami metrycznymi oraz logikę bez ograniczeń metrycznych. Rozdział I zakłada również, że logika temporalna z ograniczeniami metrycznymi jest rozszerzeniem logiki temporalnej bez ograniczeń metrycznych. W związku z tym również metodę weryfikacji formuł logiki temporalnej z ograniczeniami metrycznymi można traktować jako rozszerzenie metody weryfikacji formuł bez ograniczeń metrycznych. W związku z tym rozdział II przedstawia w rzeczywistości sześć metod weryfikacji formuł temporalnych w zależności od tego, jakiego typu są te formuły. Dodatkową siódmą metodą jest metoda *MTLStateVerification*, która jest metodą weryfikacji formuł prostych logiki MTL z operatorami metrycznymi

Metody weryfikacji opierają się na grafach przejść. Sam pomysł wykorzystania grafów przejść do weryfikacji więzów integralności został po raz pierwszy przedstawiony w pracy [Lipeck i Saake 1987], a następnie modyfikowany w pracach [Schwiderski i inni 1994] oraz [Zhou 1994], którego bitemporalne grafy przejść mają postać zbliżoną do tych rozpatrywanych w tej pracy. Grafy bitemporalne w [Zhou 1994] zbudowane są za pomocą własności rekurencyjnych 1.1a - 1.1c oraz 1.6a - 1.6c. Są więc wystarczającym mechanizmem wnioskowania w przypadku formuł logiki temporalnej bez ograniczeń metrycznych. Same grafy bitemporalne nie wystarczają jednak do weryfikacji formuł z ograniczeniami metrycznymi. Do tego celu potrzebne są jeszcze własności definicji operatorów temporalnych ograniczonych metrycznie z rozdziału I. Koncepcja grafów przejść przedstawiona w tej pracy różni od koncepcji z [Zhou 1994]. Różnice te są ukazane w podrozdziałach 3.1.1 i 3.2.1.

Rozdział II został podzielony na kilka części. W pierwszej przedstawione są grafy przejść czasu przeszłego (PGP), które mogą być wykorzystane do weryfikacji formuł logiki temporalnej czasu przeszłego (PTL). W części drugiej przedstawione zostaną mieszane grafy przejść (MGP) oraz ich wykorzystania do weryfikacji formuł MTL. Problemy weryfikacji formuł FTL i MTL są na tyle podobne, że zdecydowano się opisać je w jednym podrozdziale. W części trzeciej przedstawiona jest uproszczona, przydatna w praktyce metoda *MTLStateVerification*. Dwa ostatnie podrozdziały opisują problemy związane z kosztami wszystkich metod. Badana jest złożoność pamięciowa i obliczeniowa wszystkich metod. Poruszane są problemy jak najbardziej efektywnego przechowywania dodatkowych danych przydatnych w procesie weryfikacji.



### 3.1 Weryfikacja formuł PTL

Podrozdział ten opisuje metodę weryfikacji formuł PTL. Składa się z dwóch podrozdziałów: pierwszy definiuje grafy PGP, drugi przedstawia metodę weryfikacji wraz z opisem.

#### 3.1.1 Grafy przejść logiki czasu przeszłego (PGP)

Grafy PGP są tworzone na bazie własności rekurencyjnych 1.1a-1.1c wprowadzonych w rozdziale I. Mimo, że sam pomysł został zaczerpnięty z pracy [Zhou 1994], grafy PGP różnią się nieco od tych z cytowanej powyżej pracy w sposób następujący:

- istnieje większa liczba typów wierzchołków;
- w przypadku operatorów *always\_past* i *sometime\_past* ograniczonych metrycznie nie jest potrzebne przechowywanie wierzchołka *previous*.

Grafy PGP tworzone są w sposób rekurencyjny na podstawie formuły typu PTL. Na tak zdefiniowanym grafie można następnie przeprowadzić weryfikację formuły.

##### 3.1.1.1 Definicja grafu PGP

###### Definicja 2.1

Niech  $\phi$  formuła PTL. Graf przejść czasu przeszłego (PGP) składa się z następujących komponentów:

1) skończonej liczby wierzchołków  $V_{PGP} = V_{\wedge} \cup V_{\vee} \cup V_{\neg} \cup V_{Prev} \cup V_{Atom} \cup V_{SMid} \cup V_{Past} \cup V_{True} \cup V_{False}$  przy czym:

- $V_{\wedge}$  - zbiór wierzchołków koniunkcji;
- $V_{\vee}$  - zbiór wierzchołków dysjunkcji;
- $V_{\neg}$  - zbiór wierzchołków negacji;
- $V_{Past}$  - zbiór wierzchołków czasu przeszłego (typu *Past*), przy czym  $V_{Past} = V_{StPast} \cup V_{AlPast} \cup V_{Since}$
- $V_{StPast}$  - zbiór wierzchołków typu *Past* skojarzony z operatorem *sometime\_past*;
- $V_{AlPast}$  - zbiór wierzchołków *Past* skojarzony z operatorem *always\_past*;
- $V_{Since}$  - zbiór wierzchołków typu *Past* skojarzony z operatorem *since*;
- $V_{Prev}$  - zbiór wierzchołków typu *Previous*, przy czym  $V_{Prev} = V_{StPrev} \cup V_{AlPrev} \cup V_{Sprev}$
- $V_{StPrev}$  - zbiór wierzchołków typu *Previous* skojarzony z operatorem *sometime\_past*;
- $V_{AlPrev}$  - zbiór wierzchołków *Previous* skojarzony z operatorem *always\_past*;
- $V_{Sprev}$  - zbiór wierzchołków typu *Previous* skojarzony z operatorem *since*;
- $V_{Atom}$  - zbiór wierzchołków atomowych;
- $V_{SMid}$  - wierzchołek typu *Middle* skojarzony z operatorem *since*;
- $V_{False}$  - zbiór wierzchołków typu *false*;
- $V_{True}$  - zbiór wierzchołków typu *true*;
- $V_{Ext}$  - zbiór wierzchołków wykorzystywany do tworzenia grafu

2) skończonej liczby krawędzi  $E = E_B \cup E_T$ , przy czym:

- $E_B \subseteq V \times V$  - krawędzie obliczania;

-  $E_T \subseteq V \times V_{Prev}$  - krawędzie przejść;

- 3) wierzchołka początkowego  $v_0 \in V$ ;
- 4) opisu wierzchołków  $v : V \rightarrow \text{string}$ ;

Opis wierzchołka ma postać podformuły związanej z tym wierzchołkiem. Służy do tworzenia podgrafów grafu formuły  $\phi$ . Przykładowo opisem wierzchołka początkowego  $v_0$  jest cała formuła  $\phi$ . W przypadku stosowania operatora metrycznego w podgrafie wprowadzone są jeszcze:

- 5) funkcji operatora metrycznego wierzchołka  $met: V_{Past} \rightarrow \text{integer} \cup \text{NULL}$ ;
- 6) funkcji znaku przy operatorze wierzchołka  $sign: V_{Past} \rightarrow \{<, \leq, >, \geq, \text{NULL}\}$ ;
- 7) zmiennej pomocniczej  $threshold: V_{Past} \rightarrow \text{integer}$ ;
- 8) zmiennej pomocniczej  $verified: V_{Past} \rightarrow \{\text{true}, \text{false}, \text{unknown}\}$ ;
- 9) zmiennej pomocniczej  $verified: v_0 \rightarrow \{\text{true}, \text{false}, \text{unknown}\}$ ;

■

W grafach PGP wierzchołki są podzielone na pięć różnych klas. Do klasy  $V_{\wedge}$  należą wierzchołki odpowiadające iloczynowi podformuł danej formuły,  $V_{\vee}$  wierzchołki sumy podformuł,  $V_{\neg}$  - wierzchołki negacji.  $V_{Atom}$  związany jest z podformułą atomową, a  $V_{Past}$  z operatorem temporalnym. Wierzchołki typu *Past* dzielą się na trzy kategorie w zależności od rodzaju operatora, któremu są przypisane. Klasa wierzchołków  $V_{Prev}$  zawiera wierzchołki typu *Previous*, a klasa  $V_{SMid}$  jest specjalną klasą wierzchołków wprowadzonych dla ułatwienia wnioskowania metrycznego operatora *since*. Każdemu wierzchołkowi typu *Past* odpowiada dokładnie jeden wierzchołek typu *Previous*, a każdemu wierzchołkowi typu *Previous* jeden wierzchołek typu *Past*. Wyjątek od tej zasady stanowią operatory *always\_past* i *sometime\_past* z operatorami metrycznymi, które nie potrzebują przechowywać wartości z poprzedniego stanu.

W grafie PGP wyróżnia się także dwie klasy krawędzi. Są to  $E_B$  - krawędzie obliczania oraz  $E_T$  - krawędzie przejść. Każda krawędź przejść jest połączeniem pomiędzy wierzchołkiem typu *Previous*, a wierzchołkiem odpowiadającym formule rozpatrywanej  $V_{Past}$  lub wierzchołkiem podformuły będącej argumentem formuły typu *Previous*. Każda inna krawędź grafu jest krawędzią obliczania. Krawędzie obliczania służą do udostępniania argumentów potrzebnych wierzchołkom do obliczania ich wartości logicznych, a krawędzie przejść przekazują wartość logiczną obliczoną w poprzednim stanie bazy, która jednocześnie służy do obliczania wartości w stanie obecnym.

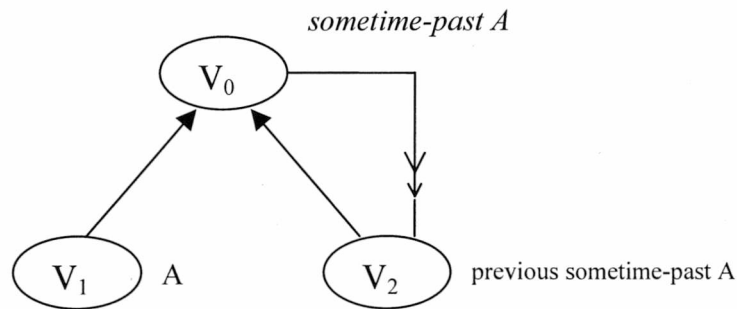
Z każdym wierzchołkiem typu *Past* mogą być skojarzone są dwie funkcje *met* i *sign*. Funkcja *met* zwraca wartość liczbową operatora metrycznego, a funkcja *sign* znak przy tej wartości. Te dwie wyżej wspomniane funkcje przyjmują wartość NULL, jeżeli z wierzchołkiem *Past* nie jest związany operator metryczny. Dodatkowo z każdym wierzchołkiem typu *Past* związana jest zmienna pomocnicza *threshold* oraz *verified*. Zmienna *verified* związana jest dodatkowo z wierzchołkiem  $v_0$ . Ich zastosowanie będzie wyjaśnione później.

■

Przykład 2.1 ilustruje intuicyjny sposób tworzenia grafu PGP dla formuły *sometime-past A*. Formuła nie zawiera operatorów metrycznych, wobec czego sposób tworzenia grafu jest identyczny jak w pracy [Zhou 1994].

### Przykład 2.1

Dana niech będzie formuła  $\varphi = \text{sometime-past } A$ . Graf odpowiadający tej formule tworzony jest na podstawie własności:  $\text{sometime-past } A \Leftrightarrow A \vee \text{previous sometime-past } A$ . Odpowiedni graf PGP składa się z trzech wierzchołków:  $V_v = \{v_0\}$ , który odpowiada formule rozpatrywanej  $\text{sometime-past } A$ ,  $V_{\text{Atom}} = \{v_1\}$  przy czym  $v(v_1) = A$  oraz  $V_{\text{Prev}} = \{v_2\}$ , a także z trzech krawędzi: jednej krawędzi przejść  $E_T = \{\langle v_0, v_2 \rangle\}$  i dwóch krawędzi obliczania  $E_B = \{\langle v_1, v_0 \rangle, \langle v_2, v_0 \rangle\}$ . Odzwierciedla to rysunek 2.1. Tak więc, by zweryfikować formułę  $\text{sometime-past } A$  należy znać wartość podformuły atomowej  $A$  oraz wartość poprzednią  $\text{sometime-past } A$ .

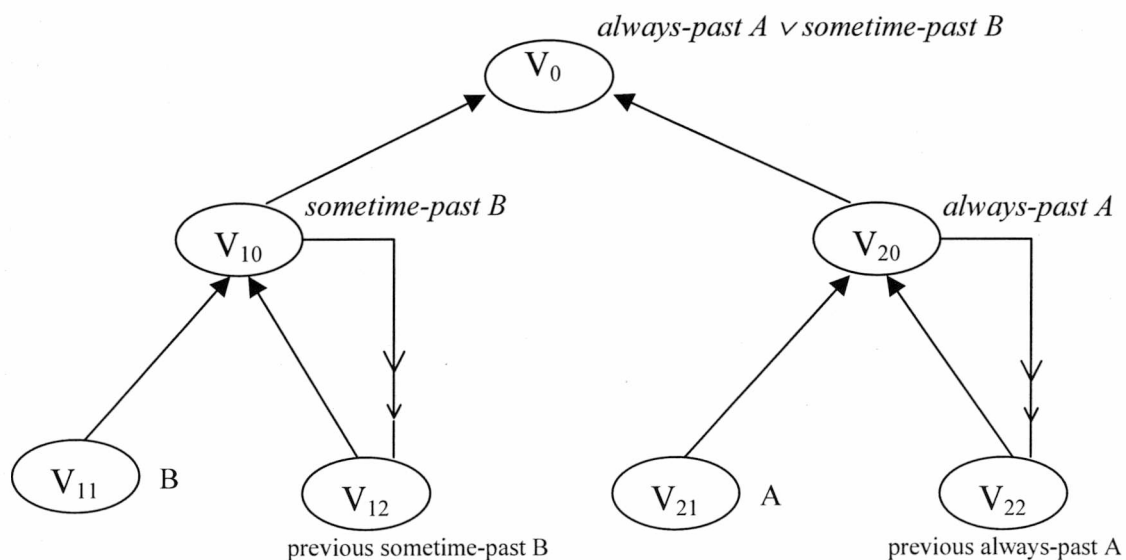


Rys. 2.1. Graf formuły  $\varphi = \text{sometime-past } A$

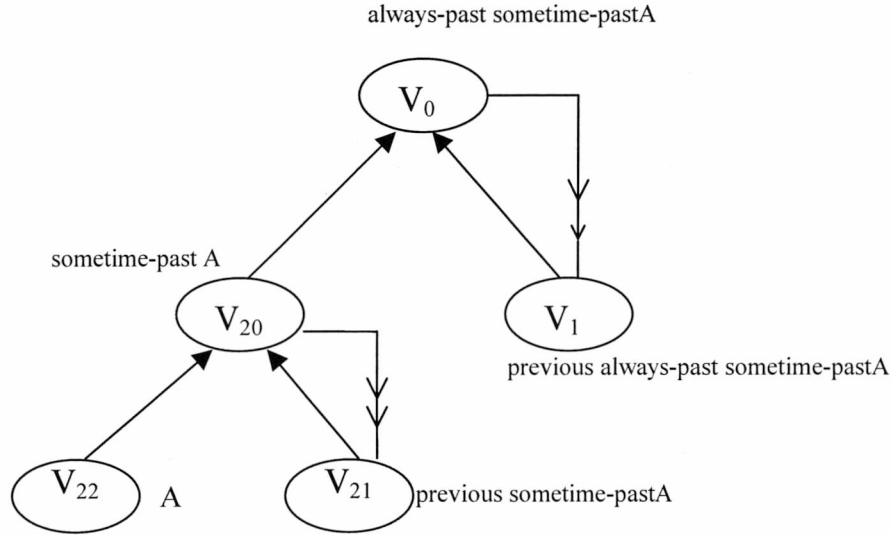
Grafy PGP są konstruowane w sposób indukcyjny. Wierzchołek  $v_0$  odpowiada zawsze najwyższemu operatorowi połączenia w hierarchii połączeń.

### Przykład 2.2

Dla przykładu graf formuły  $\varphi_1 = \text{always-past } A \vee \text{sometime-past } B$  składa się z dwóch podgrafów formuł:  $\text{always-past } A$  i  $\text{sometime-past } B$ , przy czym każdy z nich połączony jest poprzez krawędź obliczania do wierzchołka  $v_0$ , gdzie  $V_v = \{v_0\}$ . Ilustruje to rysunek 2.2. Również formułę bazującą na złożeniu operatorów temporalnych daje się łatwo przedstawić za pomocą takiego grafu. Rysunek 2.3 przedstawia formułę  $\varphi_2 = \text{always-past sometime-past } A$



Rys. 2.2 Graf formuły  $\varphi_1 = \text{always-past } A \vee \text{sometime-past } B$



Rys. 2.3 Graf formuły  $\varphi_2 = \text{always-past sometime-past } A$

### 3.1.1.2 Metoda konstrukcji grafu PGP

Podrozdział prezentuje metody tworzenia grafu PGP dla formuły  $\varphi$ . Graf taki tworzony jest w sposób indukcyjny. Poprawnie semantycznie zdefiniowana formuła  $\varphi$  jest parsowana na podformuły. Następnie dla każdej takiej podformuły uruchamiana jest metoda *SubPGPCreation*, a w przypadku napotkania operatora temporalnego metoda *TemporalPGPCreation*. Algorytmy te są zaczerpnięte z pracy [Zhou 1994], lecz możliwość wykorzystania operatorów metrycznych w niniejszej pracy implikuje modyfikację podgrafu związanego z operatorem temporalnym *always-past* i *sometime-past*.

#### Algorytm 2.1

Konstrukcja podgrafu PGP:

Niech  $\varphi$  formuła logiki temporalnej czasu przeszłego postaci: *always-past*  $\varphi_1$ , *sometime-past*  $\varphi_1$ ,  $\varphi_1$  *since*  $\varphi_2$ . Odpowiedni graf PGP  $T = \langle V, E, v, v_0 \rangle$  dla formuły  $\varphi$  jest budowany w sposób następujący:

#### procedure TemporalPGPCreation (formula $\varphi$ , graph T)

/\*  $s$  - oznacza operator metryczny ( $s \in \text{integer} \cup \text{NULL}$ ) a  $sign$  - znak przy tym operatorze:

$sign \in \{<, \leq, >, \geq, \text{NULL}\}$  \*/

case  $\varphi$  of

always-past<sub>sg s</sub>  $\varphi_1$  : AlwaysPast ( $\varphi_1, T, sign, s$ );

sometime-past<sub>sg s</sub>  $\varphi_1$  : SometimePast ( $\varphi_1, T, sign, s$ );

$\varphi_1$  since<sub>sg s</sub>  $\varphi_2$  : Since ( $\varphi_1, \varphi_2, T, sign, s$ );

```

endcase;
endproc;

```

**procedure AlwaysPast (formula  $\varphi$ , graph T, char sg, integer s)**

**begin**

*/\* dodawane są nowe wierzchołki \*/*

$V_{AlPast} := V_{AlPast} \cup \{v_{i0}\}; \quad V_{Ext} := V_{Ext} \cup \{v_{i1}\};$

*/\* Przyporządkowanie wartości zmiennym związanym z ograniczeniami metrycznymi \*/*

$sign(v_{i0}) := sg; \quad met(v_{i0}) := s;$

**if**  $sign(\varphi) \neq \text{NULL}$  **then**

$V_{True} := V_{True} \cup \{v_{i2}\};$

$v(v_{i2}) := \text{true}; \quad v(v_{i1}) = \varphi;$

$v(v_{i0}) = \text{always-past } \varphi;$

$E_B = E_B \cup \{< v_{i1} \ v_{i0} >, < v_{i2} \ v_{i0} >\};$  */\* Krawędzie obliczania\*/*

**else**

$V_{AlPrev} := V_{AlPrev} \cup \{v_{i2}\};$

$v(v_{i0}) = \varphi_1 \wedge \text{previous always-past } \varphi_1; \quad v(v_{i1}) = \varphi_1;$

$v(v_{i2}) = \text{previous always-past } \varphi_1;$

$E_B = E_B \cup \{< v_{i1} \ v_{i0} >, < v_{i2} \ v_{i0} >\};$  */\* Krawędzie obliczania\*/*

$E_T = E_T \cup \{< v_{i0} \ v_{i2} >\};$  */\* Krawędzie przejść\*/*

**endif**

**endproc;**

**procedure SometimePast (formula  $\varphi$ , graph T, char sg, integer s)**

**begin**

*/\* dodawane są nowe wierzchołki \*/*

$V_{StPast} := V_{StPast} \cup \{v_{i0}\}; \quad V_{Ext} := V_{Ext} \cup \{v_{i1}\};$

*/\* Przyporządkowanie wartości zmiennym związanym z ograniczeniami metrycznymi \*/*

$sign(v_{i0}) := sg; \quad met(v_{i0}) := s;$

**if**  $sign(\varphi) \neq \text{NULL}$  **then**

$V_{False} := V_{False} \cup \{v_{i2}\};$

$v(v_{i2}) := \text{false}; \quad v(v_{i1}) = \varphi;$

$v(v_{i0}) = \text{sometime-past } \varphi;$

$E_B = E_B \cup \{< v_{i1} \ v_{i0} >, < v_{i2} \ v_{i0} >\};$  */\* Krawędzie obliczania\*/*

**else**

$V_{StPrev} := V_{StPrev} \cup \{v_{i2}\};$

$v(v_{i0}) = \varphi \wedge \text{previous sometime-past } \varphi; \quad v(v_{i1}) = \varphi;$

$v(v_{i2}) = \text{previous sometime-past } \varphi;$

$E_B = E_B \cup \{< v_{i1} \ v_{i0} >, < v_{i2} \ v_{i0} >\};$  */\* Krawędzie obliczania\*/*

$E_T = E_T \cup \{< v_{i0} \ v_{i2} >\};$  */\* Krawędzie przejść\*/*

**endif;**

**endproc;**

**procedure Since (formula  $\varphi_1$ , formula  $\varphi_2$ , graph T, char sg, integer s)**

**begin**

/\* dodawane są nowe wierzchołki \*/

$V_{\text{Since}} := V_{\text{Since}} \cup \{v_{i0}\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_{i1}, v_{i3}\}; V_{\text{SMid}} := V_{\text{SMid}} \cup \{v_{i2}\}; V_{\text{SPrev}} := V_{\text{SPrev}} \cup \{v_{i4}\};$

/\* Przyporządkowanie wartości zmiennym związanym z ograniczeniami metrycznymi \*/  
 $\text{sign}(v_{i0}) := \text{sg}; \text{met}(v_{i0}) := s;$

$v(v_{i0}) = \varphi_1 \wedge (\varphi_2 \vee \text{previous } \varphi_1 \text{ since } \varphi_2); v(v_{i1}) = \varphi_1;$

$v(v_{i2}) = \varphi_2 \vee \text{previous } \varphi_1 \text{ since } \varphi_2; v(v_{i3}) = \varphi_2; v(v_{i4}) = \text{previous } \varphi_1 \text{ since } \varphi_2;$

$E_B = \{ \langle v_{i1} v_{i0} \rangle, \langle v_{i2} v_{i0} \rangle, \langle v_{i3} v_{i2} \rangle, \langle v_{i4} v_{i2} \rangle \};$  /\* Krawędzie obliczania\*/

$E_T = \{ \langle v_{i0} v_{i4} \rangle \};$  /\* Krawędzie przejść\*/

**endproc;**

## Algorytm 2.2

Konstrukcja grafu PGP

Niech  $\varphi$  formuła logiki temporalnej. Odpowiedni graf PGP  $T = \langle V, E, v, v_0 \rangle$  dla formuły  $\varphi$  jest budowany w sposób następujący:

**procedure PGPCreation (formula  $\varphi$ , graph T)**

**begin**

$v(v_0) := \varphi$  /\* utwórz  $v_0$  \*/

SubPGPCreation(  $v_0$ , T );

**endproc;**

**procedure SubPGPCreation ( node v, graph T)**

**begin**

**case  $v(v)$  of**

$\varphi_1 \wedge \varphi_2$  : /\* utwórz dwa nowe węzły  $v_i$  i  $v_j$  \*/

$V_{\wedge} := V_{\wedge} \cup \{v\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_i, v_j\};$

$E_B := E_B \cup \{ \langle v_i, v \rangle, \langle v_j, v \rangle \};$

$v(v_i) := \varphi_1; v(v_j) := \varphi_2;$

$\varphi_1 \vee \varphi_2$  : /\* utwórz dwa nowe węzły  $v_i$  i  $v_j$  \*/

$V_{\vee} := V_{\vee} \cup \{v\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_i, v_j\};$

$E_B := E_B \cup \{ \langle v_i, v \rangle, \langle v_j, v \rangle \};$

$v(v_i) := \varphi_1; v(v_j) := \varphi_2;$

$\neg \varphi_1$  : /\* utwórz nowy węzeł  $v_i$  \*/

$V_{\neg} := V_{\neg} \cup \{v\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_i\};$

$E_B := E_B \cup \{ \langle v_i, v \rangle \}; v(v_i) := \varphi_1;$

$A_i$  : /\*  $A_i$  jest formułą atomową \*/

```

 $V_{\text{atom}} := V_{\text{atom}} \cup \{v\}; v(v) := A_i;$ 

previous  $\varphi_1$  : /* utwórz nowy węzeł  $v_i$  */
 $V_{\text{Prev}} := V_{\text{Prev}} \cup \{v\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_i\};$ 
 $E_B := E_B \cup \{ \langle v_i, v \rangle \}; E_T := E_T \cup \{ \langle v, v_i \rangle \}; v(v_i) := \varphi_1;$ 

else : /* utwórz podgraf dla operatora temporalnego */
TemporalPGPCreation ( $v(v)$ ,  $T$ )

endcase;
endproc;
```

### 3.1.2 Weryfikacja formuł PTL

Podrozdział opisuje metodę weryfikacji formuł PTL z ograniczeniami metrycznymi. Jest on napisany w sposób następujący: Pierwsza część zawiera definicje i założenia wstępne przydatne w procesie weryfikacji. Część druga to algorytm weryfikacji. Część trzecia wyjaśnia kiedy algorytm powinien być uruchamiany. Kolejne dwie części to odpowiednio: opis metod algorytmu oraz przykłady weryfikacji formuł PTL.

#### 3.1.2.1 Założenia wstępne

Cechą charakterystyczną formuł PTL (szczególnie w kontekście baz danych) jest to, że ich weryfikacja odbywa się na podstawie danych z przeszłości. Dane z przeszłości są znane, a więc formuła PTL może być na ich podstawie zweryfikowana. Tak więc do zweryfikowania formuły PTL w stanie  $k$  korzysta się z danych w stanach mniejszych od  $k$ .

#### Definicja 2.3

##### Przedział weryfikacji

Przedziałem weryfikacji formuły  $\varphi$  przy waluacji  $\theta$  jest sekwencja stanów w której odbywa się weryfikacja formuły  $\varphi$ , przy czym  $\text{begin\_state}(\varphi, \theta)$  oraz  $\text{end\_state}(\varphi, \theta)$  oznaczają odpowiednio punkty czasu początku i końca weryfikacji. ■

Wszystkie algorytmy weryfikacji formuł temporalnych korzystają z funkcji markujących. Weryfikacji formuł PTL opiera się na logice dwuwartościowej. W związku z tym funkcja  $m$  przybiera wartość *true* lub *false*. Do weryfikacji formuł FTL i MTL wykorzystywana jest logika trójwartościowa. W związku z tym podrozdział wprowadza funkcję  $m'$ , która przyjmuje wartości *true*, *false* i *unknown*.

#### Definicja 2.4

##### Markowanie

Markowanie  $m_{v,\theta}$  wierzchołka  $v$  grafu  $T$  formuły  $\varphi$  jest funkcją  $m_{v,\theta}: k \rightarrow \{ \text{true}, \text{false} \}$ , gdzie  $k$  - indeks stanu w którym obliczana jest wartość formuły

### 3.1.2.2 Algorytm weryfikacji formuł PTL

#### Algorytm 2.3

## Weryfikacja formuł PTL

Do weryfikacji formuł PTL wykorzystywane są dane z przeszłości. Tak więc w dowolnym punkcie wiadomo, czy weryfikowana formuła jest spełniona czy nie. W związku z tym, od razu można podjąć odpowiednie kroki jako reakcję na spełnienie bądź niespełnienie formuły. Metoda weryfikacji jest rozszerzeniem metody z pracy [Zhou 1994] o możliwość weryfikacji formuł z operatorami metrycznymi. Czcionką *Courier Italics* zaznaczone są te sekcje kodu, które są brane pod uwagę przy weryfikacji formuł z operatorami metrycznymi. Tak więc jeżeli rozpatrywane są formuły bez ograniczeń metrycznych wygląd metody staje się prostszy.

**procedure PTLVerification (formula  $\phi$ , integer  $k$ , valuation  $\theta$ )**  
**begin**

*/\* Zmienna globalna rejestrująca rozpatrywaną formułę \*/*  
*var: formula  $\phi_v$ ;*  
 *$\phi_v := \phi$ ;*

*PTLMarkNode( $v_0(\phi)$ ,  $k$ ,  $\theta$ );*  
**endproc;**

**procedure PTLMarkNode (node  $v$ , integer  $k$ , valuation  $\theta$ )**  
**begin**

**case  $v$  of**

$v \in V_{\text{Atom}}$  : if  $\gamma(v) = \text{true}$  then

*$m_{v,\theta}(k) := \text{true}$ ;*

else

*$m_{v,\theta}(k) := \text{false}$ ;*

endif;

$v \in V_{\neg}$  : if  $\langle v', v \rangle \in E$  then

*PTLMarkNode ( $v'$ ,  $k$ ,  $\theta$ );*

*$m_{v,\theta}(k) := \neg m_{v',\theta}(k)$ ;*

endif;

$v \in V_{\wedge}$  : if  $\langle v', v \rangle \in E$  and  $\langle v'', v \rangle \in E$  then

*PTLMarkNode ( $v''$ ,  $k$ ,  $\theta$ );*

*PTLMarkNode ( $v'$ ,  $k$ ,  $\theta$ );*

*$m_{v,\theta}(k) := m_{v',\theta}(k) \wedge m_{v'',\theta}(k)$ ;*

endif;

$v \in V_{\vee}$  : if  $\langle v', v \rangle \in E$  and  $\langle v'', v \rangle \in E$  then

*PTLMarkNode ( $v''$ ,  $k$ ,  $\theta$ );*

*PTLMarkNode ( $v'$ ,  $k$ ,  $\theta$ );*

*$m_{v,\theta}(k) := m_{v',\theta}(k) \vee m_{v'',\theta}(k)$ ;*

endif;



```

v ∈ VPrev :   if <v', v> ∈ ET then
                mv, θ(k) := mv', θ(k-1);
                endif;

v ∈ VAlPast : if <v', v> ∈ E and <v'', v> ∈ E then
                PTLMarkNode (v'', k, θ);
                PTLMarkNode (v', k, θ);
                mv, θ(k) := mv', θ(k) ∧ mv'', θ(k);

                if met(v) = NULL then
                    if v' ∈ VPrev then mv', θ(k) := mv, θ(k);
                    else if v'' ∈ VPrev then mv'', θ(k) := mv, θ(k);
                    else PTLMetricVerification (v, k, θ);
                    endif;
                endif;

v ∈ VStPast :   if <v', v> ∈ E and <v'', v> ∈ E then
                PTLMarkNode (v'', k, θ);
                PTLMarkNode (v', k, θ);
                mv, θ(k) := mv', θ(k) ∨ mv'', θ(k);

                if met(v) <> NULL then
                    if v' ∈ VPrev then mv', θ(k) := mv, θ(k);
                    else if v'' ∈ VPrev then mv'', θ(k) := mv, θ(k);
                    else PTLMetricVerification (v, k, θ);
                    endif;
                endif;

v ∈ VSince :   if <v', v> ∈ E and <v'', v> ∈ E then
                PTLMarkNode (v'', k, θ);
                PTLMarkNode (v', k, θ);
                mv, θ(k) := mv', θ(k) ∧ mv'', θ(k);

                if v' ∈ VSMid then threshold(v, θ) := threshold(v', θ);
                else if v'' ∈ VSMid then threshold(v, θ) := threshold(v'', θ);
                if met(v) <> NULL then PTLMetricVerification (v, k, θ);
                endif;

                if <v, v'''> ∈ ET then
                    mv''', θ(k) := mv, θ(k);
                endif;

v ∈ VSMid :   if <v', v> ∈ E and <v'', v> ∈ E then
                PTLMarkNode (v'', k, θ);
                PTLMarkNode (v', k, θ);
                mv, θ(k) := mv', θ(k) ∨ mv'', θ(k);

```

```

if sign(v)='<' or sign(v)='≤' then
  if v' ∉ VPrev and mv',θ(k) = true then threshold(v, θ) := k;
  else if v'' ∉ VSPrev and mv'',θ(k) = true then threshold(v, θ) := k;
  endif;
endif;
else if sign(v)='>' or sign(v)='≥' then
  if v' ∉ VPrev and mv',θ(k) = true and mv'',θ(k) = false then
    threshold(v, θ) := k;
  else if v'' ∉ VPrev and mv'',θ(k) = true and mv',θ(k) = false then
    threshold(v, θ) := k;
  endif;
endif;
endcase;
endproc;

```

// Procedura przeprowadza końcową weryfikację uwzględniającą operatory metryczne

**procedure PTLMetricVerification (node v, integer k, valuation θ)**

**begin**

case v of

v ∈ V<sub>StPast</sub> :

if sign (v) = „ < ” (or „ ≤ ”) then

if m<sub>v,θ</sub>(k) = true then threshold(v, θ) := k;

else if m<sub>v,θ</sub>(k) = false then

if threshold(v, θ) = 0 then verified(v, θ) = false;

else if k - threshold(v, θ) < (or ≤ ) met(v) then

m<sub>v,θ</sub>(k) := true; // własność 1.3a

else verified(v, θ) = false; // własność 1.3b

endif;

// m<sub>v,θ</sub>(k) pozostaje false

endif;

if sign (v) = „ > ” (or „ ≥ ”) then

if m<sub>v,θ</sub>(k) = true then

if threshold(v, θ) = 0 then threshold(v, θ) := k;

endif;

if k - met(v) < begin\_state(φ<sub>v</sub>, θ) then

m<sub>v,θ</sub>(k) := false; //definicja 1.6 4e

else if threshold(v, θ) = 0 then

m<sub>v,θ</sub>(k) := false;

verified(v, θ) := false;

else if k - threshold(v, θ) < (or ≤ ) met(v) then

m<sub>v,θ</sub>(k) := false;

else m<sub>v,θ</sub>(k) := true; // własność 1.3c

// zawsze true

endif;

endif;

/\* W przypadku spełnienia ostatniego z powyższych warunków: k - threshold(v, θ) ≥ (or >) met(v), co oznacza, że dla każdego stanu o indeksie i > k, m<sub>v,θ</sub>(i) = true, a więc wartość ta jest już niezmienna na przestrzeni dalszej weryfikacji. Fakt ten może zostać zapisany, bądź można

stworzyć nowy graf, w którym podgraf operatora *sometime-past* zostanie zredukowany do wierzchołka typu *True*. \*/

```

V ∈ VAlPast :
  if (sign (v)= " < " (" ≤ ")) then
    if mv,θ(k) = false then threshold(v,θ) := k + 1; // własność 1.4a
    endif;
    // mv,θ(k) pozostaje false
    if k - met(v) < begin_state(φv,θ) then
      mv,θ(k) := false; //definicja 1.6 4a
    else if mv,θ(k) = true and k- threshold(v,θ) ≥ (>) met(v) then
      mv,θ(k) = true;
      verified(v,θ) = true; // własność 1.4b
    endif;
  if sign (v)= " > " (" ≥ ") then
    if mv,θ(k) = false then
      if threshold(v,θ)=0 then threshold(v):=k;
    endif;

    if k - met(v) < begin_state(φv,θ) then
      mv,θ(k) := false; //definicja 1.6 4d

    else if threshold(v,θ)=0 then
      mv,θ(k) := true;
      verified(v,θ):=true;

    else if k- threshold(v,θ) < (or ≤) met(v) then
      mv,θ(k) := true; // własność 1.4c
    else mv,θ(k) := false; // zawsze false
    endif;
  endif;
endif;

```

/\* W przypadku spełnienia ostatniego z powyższych warunków:  $k - \text{threshold}(v, \theta) \geq (\text{or } >)$   $\text{met}(v)$ , co oznacza, że dla każdego stanu o indeksie  $i > k$ ,  $m_{v,\theta}(i) = \text{false}$ , a więc wartość ta jest już niezmienna na przestrzeni dalszej weryfikacji. Fakt ten może zostać zapisany bazy, bądź można stworzyć nowy graf, w którym podgraf operatora *always\_past* zostanie zredukowany do wierzchołka typu *False*. \*/

```

V ∈ VSince :
  if sign (v)= " < " (or " ≤ ") then
    if mv,θ(k) = true then
      if k - threshold(v,θ) < (or ≤) met(v) then
        mv,θ(k) = true; // własność 1.2b
      else
        mv,θ(k) = false;
        verified(v,θ) = false;
      endif;
    else verified(v,θ) = false; // mv,θ(k) pozostaje false
    endif;
  endif;
endif;

```

```

    if sign (v)=" > " (or " ≥ ") then
        if  $m_{v,\theta}(k) = \text{true}$  then
            if  $k - \text{threshold}(v,\theta) > (\text{or } \geq) \text{met}(v)$  then
                 $m_{v,\theta}(k) = \text{true};$  // własność 1.2d
                 $\text{verified}(v,\theta) = \text{true};$ 
            endif;
            else  $m_{v,\theta}(k) = \text{false};$ 
        else  $\text{verified}(v,\theta) = \text{false};$  //  $m_{v,\theta}(k)$  pozostaje false
        endif;
    endif;
endcase;

// zapis tylko w przypadku, gdy formuła PTL jest podformułą FTL
insert_data( $m_v(k)$ , k);
endproc;

```

/\* Funkcja nadaje wartości wierzchołkom typu *Previous* w pierwszym weryfikowanym stanie. W zależności o typu wierzchołka nadawana jest odpowiednia wartość. Te wartości początkowe są zdefiniowane w pracy Zhou [1994]\*/

**procedure InitPrevious (node v) : boolean**

**begin**

**case v of**

$V_{AlPrev}$  : return true;

$V_{StPrev}$  : return false;

$V_{SPrev}$  : return false;

**endcase;**

**endfunction;**

/\* Nadawanie wartości początkowej zmiennej *threshold* oraz początkowe markowanie wierzchołków *Previous* \*/

**procedure InitValues (formula  $\varphi$ , valuation  $\theta$ , integer k)**

**begin**

for each  $v \in V_{Prev}$  do

$m_{v,\theta}(k) := \text{InitPrevious}(v);$

endfor;

for each  $v \in V_{Past}$  do

$\text{threshold}(v,\theta) := 0;$

endfor;

**endif;**

**endproc;**

/\* Procedura zwalnia zmienną *verified* używaną do optymalizacji weryfikacji (więcej na ten temat w następnym podrozdziale) \*/

**procedure unlock (formula  $\varphi$ , valuation  $\theta$ )**

**begin**

for each  $v \in V_{Past}$  do

$\text{verified}(v,\theta) := \text{unknown};$

endfor

```

    verified( $v_0$ ,  $\theta$ ) := unknown;
endproc;

```

### 3.1.2.3 Warunki uruchamiania algorytmu 2.3

Dwa czynniki mają wpływ na wartość formuły temporalnej z operatorami metrycznymi: zmiana wartości jej podformuł atomowych oraz upływający czas. Model czasu w rozpatrywanej logice jest dyskretny. Dlatego też weryfikacja musi się odbywać co jednostkę czasu. Metoda *PTLMonitor* uruchamiana jest jako odpowiedź na to zmiany czasu. Dość często zdarzają się jednak sytuacje, kiedy to upływający czas nie jest w stanie zmienić wartości formuły. W takim przypadku algorytm weryfikacyjny nie musi być uruchamiany do momentu zmiany jednej z podformuł atomowych ( $trans(\varphi, \theta)$ ). W tym celu została wprowadzona zmienna  $verified(v, \theta)$ . Zmienna ta jest skojarzona z każdym wierzchołkiem typu *Past* oraz z wierzchołkiem początkowym. Jej wartość (*true* lub *false*) określa wartość podformuły do momentu zmiany wartości jednej z podformuł atomowych występujących w formule. Jeżeli  $\varphi$  jest formułą PTL, a  $\theta$  waluacją, to algorytm monitorujący wygląda następująco.

```

procedure PTLMonitor(integer k)
begin
  for each  $\varphi$  do
    for each  $\theta$  do
      if  $k = \text{begin\_state}(\varphi, \theta)$  then InitValues ( $\varphi$ ,  $\theta$ , k);
      if  $trans(\varphi, \theta)$  in k then
        unlock ( $\varphi, \theta$ ) ;
        PTLVerification( $\varphi$ , k,  $\theta$ );
      else if  $verified(v_0(\varphi), \theta) = \text{unknown}$  then
        PTLVerification ( $v_0(\varphi)$ , k,  $\theta$ ) ;
      endif;
    endfor;
  endfor;
endproc;

```

### 3.1.2.4 Opis algorytmu 2.3

Opis algorytmu 2.3 składa się z opisu dwóch procedur występujących w tym algorytmie. Są to procedury *PTLMarkNode* oraz *PTLMetricVerification*.

#### Procedura PTLMarkNode

Rekurencyjna procedura *PTLMarkNode* służy do markowania wierzchołków grafu PTG odpowiednimi wartościami. Rozpatruje ona kilka przypadków w zależności od typu wierzchołka w grafie.

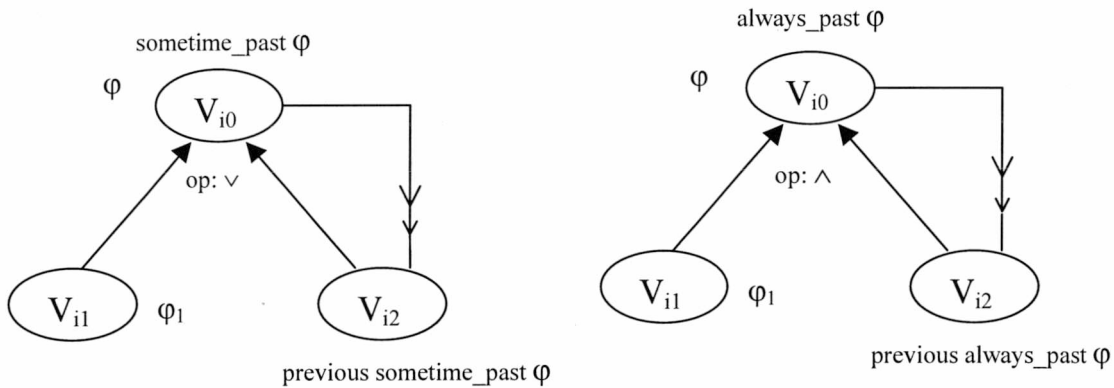
W przypadku wierzchołków formuł atomowych, ich wartości pobierane są z aktualnego stanu. Wartość wierzchołka typu „negacja” jest zaprzeczeniem wartości wierzchołka „negowanego”. Gdy rozpatrywany wierzchołek jest wierzchołkiem koniunkcji, jego wartość jest logicznym iloczynem wartości wierzchołków będących jego argumentami.

Analogicznie jest w przypadku wierzchołka alternatywy. Jego wartość jest logiczną sumą wartości wierzchołków będących jego argumentami.

Dalsza analiza jest już znacznie bardziej skomplikowana. Można ją podzielić na dwa przypadki. Pierwszy z nich rozpatruje występowanie formuł typu PTL bez operatorów metrycznych. Własności 1.1a - 1.1c (rozdział I) są podstawą do utworzenia grafów PGP dla każdego z operatorów PTL. Wówczas wierzchołek typu *AlPast* może być traktowany jak wierzchołek koniunkcji, a wierzchołek *StPast* jako wierzchołek alternatywy. Obliczona w ten sposób wartość jest przypisana do odpowiedniego wierzchołka typu *Prev*. Mając do dyspozycji grafy wraz z przypisanymi do każdego z ich działaniami, łatwo już zdefiniować odpowiednie specyfikacje dla każdego z wierzchołków. W takim przypadku weryfikacja odbywa się analogicznie jak w pracy [Zhou 1994]. Przykład 2.1 wyjaśnia dokładnie całą procedurę weryfikacji.

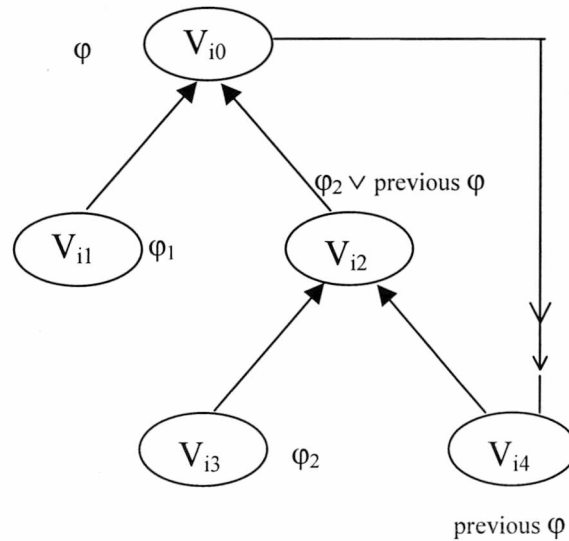
Drugi przypadek zachodzi, gdy przy operatorach PTL występuje operator metryczny.

W przypadku wierzchołków *StPast* i *AlPast* weryfikacja przekazywana jest do *PTLMetricVerification*. W przypadku wierzchołka *SMid* rejestrowany jest czas, w którym wartość wierzchołka nie będącego wierzchołkiem typu *Prev* jest prawdziwa. Wartości te są następnie wykorzystane w procedurze *PTLMetricVerification*, gdzie dla każdego wierzchołka typu *Past* odbywa się weryfikacja końcowa. Rysunki 2.4a - 2.4c przedstawiają grafy PTG operatorów temporalnych PTL.



Rys. 2.4a Graf  $\psi \equiv \text{sometime\_past } \phi$

Rys. 2.4b Graf  $\psi \equiv \text{always\_past } \phi$



### Procedura PTLMetricVerification

Procedura *PTLMetricVerification* służy do końcowej weryfikacji wartości poszczególnych wierzchołków wynikającej z obecności operatorów metrycznych. Procedura ta wykorzystuje zdefiniowane w rozdziale I własności wnioskowania. Parametrami wejściowymi są: wierzchołek grafu, odpowiednia waluacja oraz weryfikowany stan. W zależności od typu wierzchołka weryfikacja odbywa się w sposób następujący:

- wierzchołek typu *StPast* (*sometime-past*  $\varphi$ )
  1. operator  $<$  lub  $\leq$ . Jeżeli wstępna weryfikacja (algorytm *PTLMarkNode*) dała wartość *true*, czyli podformuła  $\varphi$  jest spełniona, w zmiennej *threshold* rejestrowany jest czas kiedy to miało miejsce, a wierzchołek markowany jest wartością *true*. W przeciwnym razie wartość końcowa w wierzchołku obliczana jest na podstawie własności 1.3a oraz 1.3b.
  2. operator  $>$  lub  $\geq$ . Jeżeli stan, w którym markowana jest wartość wierzchołka jest zbyt blisko stanu początkowego, to na podstawie własności 1.7c wierzchołek markowany jest wartością *false*. Jeżeli wynik wstępnej weryfikacji z *PTLMarkNode* nie dał w żadnym z wcześniejszych stanów wartości *true*, to wierzchołek markowany jest także wartością *false*. W przeciwnym razie zmienna *threshold* jest określona, a więc może być wykorzystywana jest własność 1.3c.
- wierzchołki typu *AlPast* (*always-past*  $\varphi$ )
  1. operator  $<$  lub  $\leq$ . Jeżeli wynik wstępnej weryfikacji z *PTLMarkNode* wynosi *false* (podformuła  $\varphi$  nie jest spełniona), to jest to rejestrowanie w zmiennej *threshold*, a wierzchołek jest markowany wartością *false*. Gdy wstępna weryfikacja dała wartość *true*, a zmienna *threshold* jest określona, to ostateczna weryfikacja jest obliczona na podstawie własności wnioskowania 1.4a i 1.4b. W przypadku, gdy stan, w którym markowana jest wartość wierzchołka jest zbyt blisko stanu początkowego, to wierzchołek jest markowany wartością *false* zgodnie z definicją 1.6 4a.
  2. operator  $>$  lub  $\geq$ . Jeżeli stan, w którym markowana jest wartość wierzchołka jest zbyt blisko stanu początkowego, to na podstawie definicji 1.6 4d wierzchołek markowany jest wartością *false*. Jeżeli wynik wstępnej weryfikacji z *PTLMarkNode* nie dał w żadnym z wcześniejszych stanów wartości *false* oraz stan markowania jest wystarczająco daleko punktu początkowego, to wierzchołek markowany jest także wartością *true*. W przeciwnym razie zmienna *threshold* jest określona, a więc może być wykorzystywana jest własność 1.4c.
- wierzchołki typu *Since*. Wnioskowanie odbywa się na podstawie własności 1.2b oraz 1.2d.

#### 3.1.2.5 Przykłady weryfikacji

Podrozdział przedstawia dwa przykłady weryfikacji. W formule z przykładu 2.3 nie występują operatory metryczne. Proces weryfikacji korzysta jedynie z własności 1.1a - 1.1c wprowadzonych w [Zhou 1994]. Formuła w przykładzie 2.4 zawiera operatory metryczne. W

związku z tym do weryfikacji potrzebne są jeszcze własności wnioskowania wprowadzone w podrozdziale 1.2.1.

### Przykład 2.3

Niech dana będzie formuła nie zawierająca operatorów metrycznych:

$$\varphi = B \wedge (\neg \text{always-past } (A \wedge C) \vee \text{sometime-past } D)$$

1. Zgodnie z własnościami rekurencji (własności 1.1a - 1.1c) podformuły mogą być zapisane w sposób następujący:

$$\text{always-past } (A \wedge C) \Leftrightarrow (A \wedge C) \wedge \text{previous always-past } (A \wedge C) \text{ oraz}$$

$$\text{sometime-past } D \Leftrightarrow D \vee \text{previous sometime-past } D$$

Na podstawie takiego zapisu tworzony jest graf formuły  $\varphi$ .

2. Zgodnie z algorytmem najpierw wartościowane są podformuły typu *Prev* na początku procesu weryfikacji formuły  $\varphi$ . W obu tych przypadkach uzyskiwane są następujące wartości początkowe  $\text{sometime-past } D = \text{false}$  oraz  $\text{always-past } (A \wedge C) = \text{true}$ . Wartości te są jednocześnie wartościami formuł  $\text{previous sometime-past } D = \text{false}$  oraz  $\text{previous always-past } (A \wedge C) = \text{true}$  w pierwszym stanie bazy.

3. W pierwszym stanie znana jest wartość formuł A, B, C i D. Wobec tego korzystając z kroków 1 i 2 można obliczyć wartość całej formuły  $\varphi$  w pierwszym stanie bazy. Należy przy tym zauważyć, że przy przejściu do drugiego stanu bazy wartości  $\text{always-past } (A \wedge C)$  oraz  $\text{sometime-past } D$  obliczone dla pierwszego stanu bazy w kroku pierwszym są jednocześnie wartościami odpowiednio  $\text{previous always-past } (A \wedge C)$  i  $\text{previous sometime-past } D$  dla drugiego stanu. Wobec tego w stanie drugim znowu możliwe jest obliczenie wartości formuły  $\varphi$ .

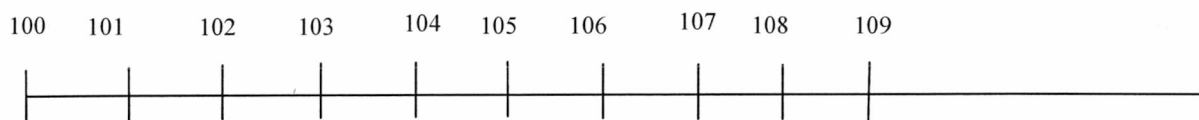
4. Ten sposób postępowania powtarzany jest przy każdej zmianie stanu bazy, aż do końca weryfikacji formuły  $\varphi$ .

Należy wspomnieć też o tym, że przy każdorazowej dynamicznej zmianie stanu bazy kroki 2 i 3 są powtarzane tylko raz, a więc przy przejściu ze stanu  $i$  do stanu  $i+1$  obliczana jest wartość formuły  $\varphi$  na podstawie wartości formuł A, B, C, D w stanie  $i+1$  oraz na podstawie wartości  $\text{always-past } (A \wedge C)$  i  $\text{sometime-past } D$  przekazanych ze stanu  $i$ . Dzięki temu metoda ta jest bardzo efektywna w porównaniu do metod weryfikacji opartych o języki zapytań, które do obliczenia wartości formuły w stanie  $i$  wymagałyby „cofnięcia się” o odpowiednią liczbę stanów. Następnie, w stanie  $i+1$  bazy owo „cofnięcie się” musiałoby być powtórzone, co wprowadziłoby niepotrzebną redundancję obliczeniową. Metoda wprowadzona w tej pracy pozwala na uniknięcie tej redundancji, przez co algorytm działa nieporównywalnie szybciej. ■

### Przykład 2.4

Rozpatrzmy formułę:  $\varphi = \text{sometime\_past } <_3 A \text{ since } <_3 B$

Niech rysunek ukazuje waluację podformuł atomowych A i B





|                         |                          |
|-------------------------|--------------------------|
| $A, B$                  | $\neg A, \neg B$         |
| $\varphi = \text{true}$ | $\varphi = \text{false}$ |

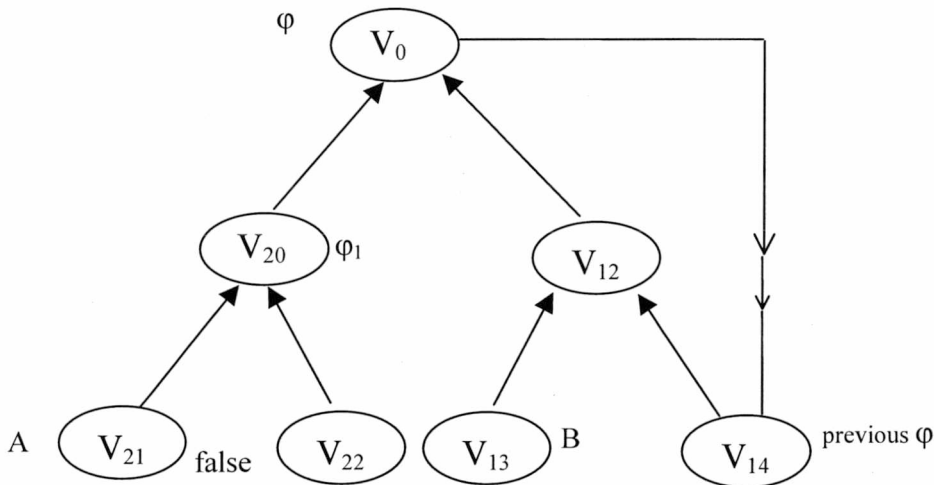
Rys. 2.5. Waluacja podformuł atomowych A i B

Niech :

$$\varphi_1 = \text{sometime\_past}_{<3} A$$

Niech waluacja  $\theta$  wygląda tak jak na rysunku 2.5. Tak więc od stanu 100 do 102 podformuły A i B przyjmują wartość *true* a w stanach 103 – 109 podformuły A i B przyjmują wartość *false*.

Poniższy graf ukazuje graf weryfikacji rozpatrywanej formuły.



Rys.2.6 Graf formuły  $\varphi$

Stan 100

Skoro  $A = \text{true}$  to według algorytmu wartość  $m$  na wierzchołku  $v_{20}$  jest również *true*, gdyż w tym stanie  $\text{threshold}(v_{20}, \theta) = 100$ , a więc jest spełniony warunek  $k\text{-threshold}(v_{20}, \theta) < 3$ . Również wartość  $m$  na wierzchołku  $v_{20}$  w tym stanie jest *true*, gdyż  $B = \text{true}$ , a więc wartość  $m$  na wierzchołku  $v_{13}$  wynosi *true*, a co za tym idzie  $m$  na wierzchołku  $v_{12}$  jest *true*. Wreszcie wartość na wierzchołku  $v_0$  jest prawdziwa, gdyż  $k\text{-threshold}(v_0, \theta) = < 3$ . Tak więc wartość formuły  $\varphi$  jest w stanie 100 prawdziwa.

Stan 101 i 102

Analogicznie jak w stanie 100.

Stan 103 i 104

Sytuacja nie ulega zbyt dużej zmianie. Pomimo, że  $A = false$ ,  $m$  na wierzchołku  $v_{20}$  jest nadal  $true$ , gdyż  $k\text{-threshold}(v_{20}, \theta) < 3$ , bo  $threshold(v_{20}, \theta) = 102$ . Daje to w efekcie wartość  $true$  dla wierzchołka  $v_0$ .

Stan 105

W tym stanie wartość  $m$  na wierzchołku  $v_{20}$  jest  $false$ , gdyż nie jest spełniona nierówność  $k\text{-threshold}(v_{20}, \theta) < 3$ . Warto zauważyć, że nierówność ta nie będzie spełniona dopóki nie będzie stanu, w którym  $A = true$ , czyli dopóki nie zajdzie kolejna transakcja zmieniająca  $A$  na  $true$ . Sam upływ czasu nie zdoła zmienić wartości  $\varphi_1$  na  $true$ . Wobec tego można zapisać, że  $verified(v_{20}, \theta) = false$ . Konsekwencją faktu, że wartość  $m$  na wierzchołku  $v_{20}$  wynosi  $false$  jest wartość  $false$  na wierzchołku  $v_0$ . I znów można zapisać, że  $verified(v_0, \theta) = false$ .

Stany  $> 105$

Weryfikacja w kolejnych stanach nie ma już sensu, gdyż nie zachodzi już żadna transakcja, a wartość  $verified(v_0, \theta)$  jest określona, w związku z czym ani czas ani transakcja nie mogą zmienić wartości formuły  $\varphi$ .



Porównując weryfikację formuły  $\varphi$  z przykładu 2.4 z taką samą formułą, ale bez operatorów metrycznych ( $\varphi' = \text{sometime\_past } A \text{ since } B$ ), otrzymuje się następujące różnice:

- wierzchołek grafu  $v_{22}$  jest typu *Previous*, a nie typu *False*;
- algorytm zostanie uruchamiany jedynie w stanach 100 oraz 103, a nie jak w przypadku formuły  $\varphi$  stanach 100 - 105;
- pamiętane są wartości w zmiennych związanych z wierzchołkami typu *Previous*, podczas gdy podczas weryfikacji formuły  $\varphi$  należy pamiętać dwie zmienne *threshold* i *verified* oraz wartość z poprzedniego stanu wierzchołka typu *Since*.

## 3.2 Weryfikacja formuł logiki FTL i MTL

Podrozdział ten wprowadza metodę weryfikacji formuł FTL oraz MTL. Składa się z dwóch podrozdziałów: podrozdział 3.2.1 definiuje grafy FGP oraz MGP jako narzędzie do weryfikacji, natomiast podrozdział 3.2.2 wprowadza metodę weryfikacji formuł MTL wraz z opisem, opisuje warunki uruchomienia metody oraz omawia problemy związane z weryfikacją formuł złożonych.

### 3.2.1 Grafy FGP i MPG

#### 3.2.1.1 Grafy logiki czasu przyszłego (FGP)

W tej części przedstawione zostaną grafy FGP, będące podstawą do weryfikacji formuł FTL. Mimo, że pomysł tworzenia grafów FGP został zaczerpnięty z pracy [Zhou 1994], grafy FGP różnią się od tych z cytowanej powyżej pracy w sposób zasadniczy. Owa różnica wynika przede wszystkim z tego, że grafy przejść z pracy [Zhou 1994] nie są przystosowane do weryfikacji formuł logiki FTL z operatorami metrycznymi, gdyż operator temporalny FTL nie zawsze jest związany z jednym wierzchołkiem typu *Future*. W związku z tym nie można skojarzyć operatora metrycznego z operatorem temporalnym, co uniemożliwia weryfikację formuł logiki FTL z ograniczeniami metrycznymi.

Wprowadzony w niniejszej pracy nowy rodzaj grafów FGP pozwala na weryfikację zarówno formuł ograniczonych jak i nieograniczonych metrycznie.

Poniższy przykład ilustruje sposób tworzenia grafu FGP oraz wyjaśnia metodę weryfikacji formuły FTL bez ograniczeń metrycznych w oparciu o taki graf.

#### Przykład 2.5

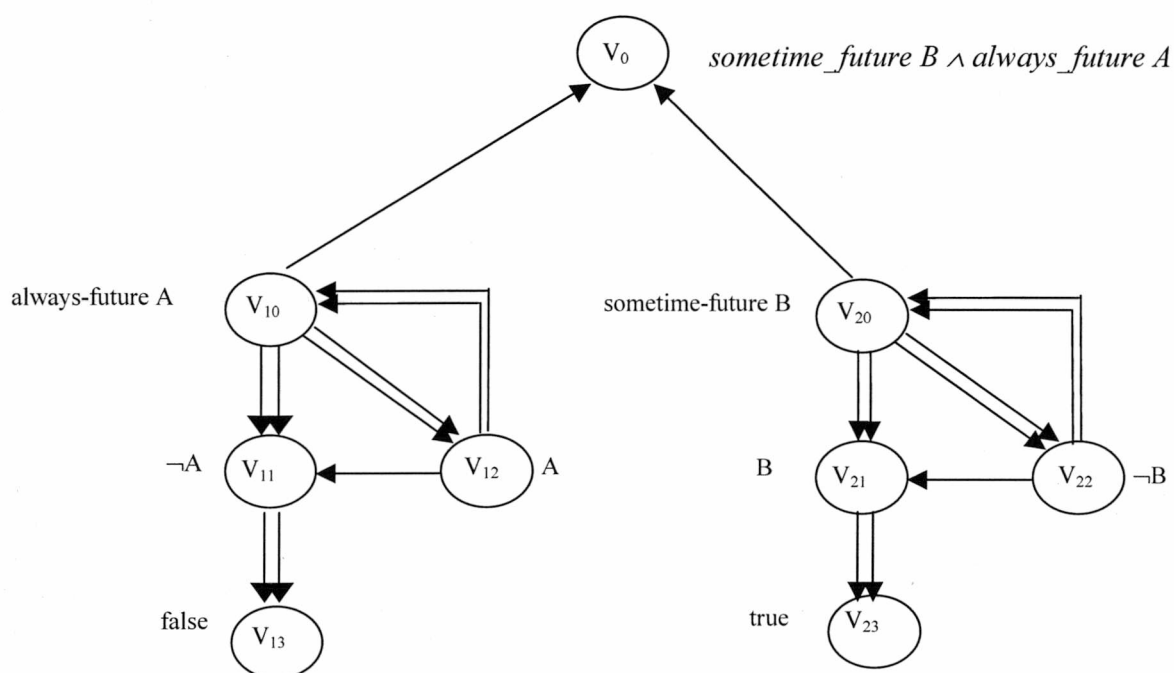
Dana niech będzie formuła typu FTL bez operatorów metrycznych:

$$\varphi = \textit{sometime\_future } B \wedge \textit{always\_future } A$$

Według własności 1.5 a - 1.5c (rozdział I) formułę tę można zapisać w sposób następujący;

$$\varphi' = (A \wedge \textit{next always\_future } A) \wedge (B \vee \textit{next sometime\_future } B)$$

Graf zbudowany na bazie tej formuły wygląda w sposób następujący:



Rys. 2.7 Graf FTG formuły  $\varphi$

Jedną z cech odróżniających grafy FTG od grafów PTG jest występowanie łańcuchów. Na łańcuch składają się trzy wierzchołki połączone podwójną linią. W procesie weryfikacji wartość funkcji markującej wierzchołka na początku łańcucha zastępowana jest przez wartość wierzchołka na końcu łańcucha, jeżeli wartość w wierzchołku środkowym jest *true*.

A oto intuicyjna weryfikacja formuły  $\varphi$  w oparciu o powyższy graf: Algorytm weryfikacyjny uruchamiany jest w każdym stanie, który mógł zmienić wartość formuł atomowych  $A$  lub  $B$ . Wierzchołki  $v_{10}$  i  $v_{20}$  grafu są wierzchołkami typu *Future*. Przed każdym uruchomieniem algorytmu weryfikacji wierzchołkom tym przypisana jest wartość *unknown*. Następnie w zależności od wartości formuły atomowej, ta przypisana wartość zostaje zastąpiona przez wartość znajdującą się na końcu odpowiedniego łańcucha. I tak, jeżeli  $A$  przyjmie wartość *false* to wierzchołkowi  $v_{10}$  zostanie przypisana wartość wierzchołka  $v_{13}$ , która jest *false*. Tak więc wartość podformuły *always\_future A* w tym stanie wynosi *false*. Natomiast gdy  $A$  przyjmie wartość *true*, wartość wierzchołka  $v_{10}$  nie zmieni się, gdyż, jak to ukazuje graf, łańcuch  $\langle v_{10}, v_{12}, v_{10} \rangle$  rozpoczyna się i kończy w tym samym wierzchołku. Analogicznie jest w przypadku podformuły *sometime\_future B*.

Metoda weryfikacji w pracy [Zhou 1994] zakłada inicjowanie korzeni podgrafów operatorów FTL wartością *true*. Wynika to ze stosowania w tej pracy założenia o potencjalnej spełnialności. Niniejsza praca umożliwia weryfikację formuł temporalnych z ograniczeniami metrycznymi, a potencjalna spełnialność nie jest tutaj zastosowana.

Na wartość formuły  $\varphi$  wpływają wartości  $A$  i  $B$  w stanie rozpatrywanym oraz wartości *always\_future A* i *sometime\_future B* w stanie następnym. Te ostatnie nie są jednak jeszcze znane w chwili obecnej, wobec tego wierzchołki im odpowiadające ( $v_{10}$  i  $v_{20}$ ) muszą być zainicjowane wartością *unknown*.

W rozważaniach tych należy uwzględnić jednak pewien szczególny stan: stan końcowy weryfikacji. W stanie końcowym nie są określone formuły *next always\_future A* i

*next sometime\_future B*. Muszą one być określone w sposób sztuczny. Idąc za pracą [Zhou 1994] funkcja *Endmark(v)* podaje wartości dla poszczególnych typów wierzchołków *Future*. Dlatego też po stanie ostatnim weryfikacji wierzchołków, jeżeli wartość na wierzchołku  $v_{10}$  pozostaje *unknown*, to jest ona zastępowana wartością *true*, a w przypadku wierzchołków  $v_{20}$  wartością *false*. Czyli jeżeli *B* przyjmie wartość *false*, to wartością wierzchołków  $v_{20}$  będzie wartość *unknown*, która zostanie zastąpiona wartością *false*. Jest to zgodne z semantyką *sometime\_future  $\psi$* , który przyjmuje wartość *true*, jeżeli formuła  $\psi$  jest prawdziwa, w jednym ze stanów przyszłych. Skoro jednak jest tylko jeden stan przyszły, wobec tego formuła  $\psi$  musi być prawdziwa w tym stanie. Również jeżeli *A* przyjmie wartość *true* w ostatnim stanie weryfikacji, to zgodnie z semantyką *always\_future A* w ostatnim stanie jest również *true*.

Tak prowadzona weryfikacja jest poprawna przy założeniu, że weryfikowane są formuły FTL bez ograniczeń metrycznych. Jednak jednym z celów tej pracy jest weryfikacja formuł FTL zarówno ograniczonych jak i nieograniczonych metrycznie. W związku z tym taka metoda weryfikacji jak w przykładzie 2.5 okazuje się niewystarczająca.

### 3.2.1.2 Definicja grafu FGP

Grafy FGP są tworzone na bazie własności 1.5a-1.5c wprowadzonych w rozdziale I. Zarówno definicja jak i sposób tworzenia tego typu grafów jest inny niż w pracy [Zhou 1994]. Graf w niniejszej pracy ma znacznie więcej typów wierzchołków oraz związanych zmiennych. Formalna definicja grafu FGP ma postać następującą:

#### Definicja 2.6

Niech  $\phi$  formuła FTL oraz  $S(\phi)$  ilość wszystkich podformuł formuły  $\phi$ . Graf przejść (FGP) składa się z następujących komponentów:

- 1) skończonej liczby wierzchołków  $V_{FGP} = V_{\wedge} \cup V_{\vee} \cup V_{\neg} \cup V_{Future} \cup V_{Future'} \cup V_{Atom} \cup V_{Next} \cup V_{True} \cup V_{False}$ , przy czym:

- $V_{Atom}$  - zbiór wierzchołków atomowych;
- $V_{\wedge}$  - zbiór wierzchołków koniunkcji;
- $V_{\vee}$  - zbiór wierzchołków dysjunkcji;
- $V_{\neg}$  - zbiór wierzchołków negacji;
- $V_{Future}$  - zbiór wierzchołków typu *Future*:  $V_{Future} = V_{StFuture} \cup V_{AlFuture} \cup V_{Until}$
- $V_{StFuture}$  - zbiór wierzchołków typu *Future* skojarzony z operatorem *sometime-future*;
- $V_{AlFuture}$  - zbiór wierzchołków typu *Future* skojarzony z operatorem *always-future*;
- $V_{Until}$  - zbiór wierzchołków typu *Future* skojarzony z operatorem *until*;
- $V_{Next}$  - zbiór wierzchołków typu *Next*;
- $V_{Future'}$  - zbiór wierzchołków typu *Future'*;
- $V_{False}$  - zbiór wierzchołków typu *False*;
- $V_{True}$  - zbiór wierzchołków typu *True*;

- 2) skończonej liczby krawędzi  $E_{FGP} = E_B \cup E_T \cup E_K$ , przy czym:

- $E_K \subseteq V \times V$  - kontrolne krawędzie przejść;
- $E_B \subseteq V \times V$  - krawędzie obliczania;

- 3) opisu wierzchołków  $v : V \rightarrow \text{string}$ ;

Opis wierzchołka ma postać podformuły związanej z tym wierzchołkiem. Służy do tworzenia podgrafów grafu formuły  $\varphi$ . Przykładowo opisem wierzchołka początkowego  $v_0$  jest cała formuła  $\varphi$ .

- 4) funkcji operatora metrycznego wierzchołka  $met: V_{Future} \rightarrow \text{integer} \cup \text{NULL}$ ;
- 5) funkcji znaku przy operatorze wierzchołka  $sign: V_{Future} \rightarrow \{<, \leq, >, \geq, \text{NULL}\}$ ;
- 6) łańcuchów przejść  $K \subset V_{Future} \times V \times V$
- 7) wierzchołków obliczania  $V_m \subseteq V$
- 8) wierzchołka początkowego  $v_0 \in V$
- 9) zmiennej pomocniczej  $verified: V_{Future} \rightarrow \{\text{true}, \text{false}, \text{unknown}\}$ ;
- 10) zmiennej pomocniczej  $verified: v_0 \rightarrow \{\text{true}, \text{false}, \text{unknown}\}$ ;

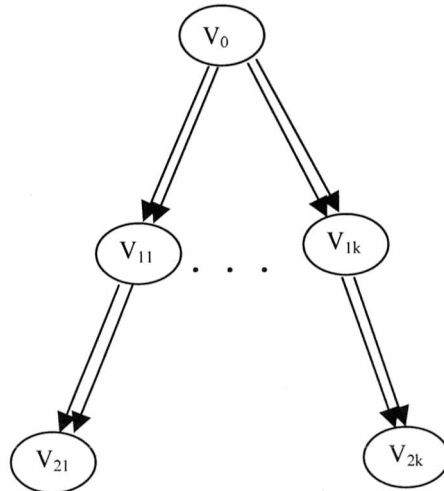
■

$V_{\wedge}, V_{\vee}, V_{\neg}$  są wierzchołkami, w których zachodzą operacje odpowiednio koniunkcji dysjunkcji i negacji. Wierzchołki typu *Future* służą do końcowej weryfikacji podformuł FTL. Dla przykładu na rysunku 2.7 wierzchołki  $\{v_{10}, v_{20}\} \in V_{Future}$ . Klasa kontrolnych krawędzi przejść ( $E_K$ ) zawiera krawędzie prowadzące z wierzchołka typu *Future* do wierzchołka typu *Future*, typu *True* lub *False*. Wobec tego na rysunku krawędzie  $\langle v_{10}, v_{12} \rangle, \langle v_{12}, v_{10} \rangle, \langle v_{10}, v_{11} \rangle, \langle v_{11}, v_{13} \rangle$  są kontrolnymi krawędziami przejść, a łańcuchy  $\langle v_{10}, v_{12}, v_{10} \rangle$  oraz  $\langle v_{10}, v_{11}, v_{13} \rangle$  są łańcuchami przejść.

### 3.2.1.3 Konstrukcja grafu FGP

Algorytm konstrukcji grafu FGP jest algorytmem rekurencyjnym. Weryfikacja przedstawiona w tej pracy nie korzysta z pojęcia potencjalnej spełnialności. W związku z tym grafy FGP są nieco inne niż w [Zhou 1994]. Także metoda tworzenia grafu musi być różna. Poprawnie semantycznie zdefiniowana formuła  $\varphi$  typu FGP jest parsowana na podformuły. Następnie dla każdej takiej podformuły uruchamiana jest metoda *SubFGPCreation*, a w przypadku napotkania operatora temporalnego metoda *TemporalFGPCreation*.

Podgraf FGP dla operatora temporalnego FTL składa się z łańcuchów przejść zakorzenionych w wierzchołku typu *Future*. Można przedstawić w sposób następujący:



Rys. 2.8 Podgraf wynikowy FGP

Wierzchołek  $v_0$  jest wierzchołkiem operatora temporalnego FTL. Rodzaj wierzchołków  $v_{11} \dots v_{1k}$  zależy od rodzaju operatora FTL, natomiast wierzchołki  $v_{21} \dots v_{2k}$  przyjmują są wierzchołkami typu *True* lub *False*. Z tak zdefiniowanego szablonu korzysta algorytm budowania grafu FGP dla poszczególnych operatorów temporalnych (algorytm 2.4)

#### Algorytm 2.4

**procedure AlwaysFuture (formula  $\varphi$ , graph T, char sg, int s)**

**begin**

/\* dodawane są nowe wierzchołki \*/

$V_{\text{AlwaysFuture}} := V_{\text{AlwaysFuture}} \cup \{v_{i0}\}; V_{\neg} := V_{\neg} \cup \{v_{i11}\};$

$V_{\text{False}} := V_{\text{False}} \cup \{v_{i21}\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_{i11}\};$

$v(v_{i0}) := \text{always-future } \varphi;$

$v(v_{i11}) := \neg \varphi; v(v_{i21}) := \text{false};$

$v(v_{i22}) := \text{always-future } \varphi;$

$v_{i22} := v_{i0};$

/\* dodawane są nowe łańcuchy \*/

$K := K \cup \{ \langle v_{i0}, v_{i11}, v_{i21} \rangle, \langle v_{i0}, v_{i12}, v_{i22} \rangle \}$

/\* dodawana jest nowa krawędź \*/

$E_B := E_B \cup \{ \langle v_{i11}, v_{i12} \rangle \}$

/\* jeżeli  $\varphi$  jest formułą FTL, dodawany jest wierzchołek pomocniczy \*/

if  $\varphi$  is FTL then

$V_{\text{Future}'} := V_{\text{Future}'} \cup \{v_{i12}\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_{i3}\};$

$v(v_{i3}) := \varphi;$

$E_B := E_B \cup \{ \langle v_{i3}, v_{i11} \rangle \}$

else  $v(v_{i12}) := \varphi;$

endif;

/\* Przyporządkowanie wartości zmiennym związanym z ograniczeniami metrycznymi \*/

$\text{sign}(v_{i0}) := \text{sg}; \text{met}(v_{i0}) := s;$

**endproc;**

**procedure SometimeFuture (formula  $\varphi$ , graph T, char sg, int s)**

**begin**

/\* dodawane są nowe wierzchołki \*/

$V_{\text{StFuture}} := V_{\text{StFuture}} \cup \{v_{i0}\}; V_{\neg} := V_{\neg} \cup \{v_{i12}\};$

$V_{\text{True}} := V_{\text{True}} \cup \{v_{i21}\}; V_{\text{Ext}} := V_{\text{Ext}} \cup \{v_{i11}\};$

$v(v_{i0}) := \text{sometime-future } \varphi; v(v_{i21}) := \text{true};$

```

v (vi12) := ¬φ ; v (vi22) := sometime-future φ;
vi22 := vi0;

/* dodawane są nowe łańcuchy */
K := K ∪ { < vi0, vi11, vi21>, < vi0, vi12, vi22> }

/* dodawana jest nowa krawędź */
EB := EB ∪ { < vi12, vi11> }

/* jeżeli φ jest formułą FTL, dodawany jest wierzchołek pomocniczy */
if φ FTL then
    VFuture' = VFuture' ∪ { vi11 }; VExt = VExt ∪ { vi3 };
    v (vi3) := φ;
    EB := EB ∪ { < vi11, vi3> }
else v (vi11) := φ;
endif;
/* Przyporządkowanie wartości zmiennym związanym z ograniczeniami metrycznymi */
sign(vi0) := sg; met(vi0) := s;
endproc;

procedure Until (formula φ1, formula φ2, graph T, char sg, int s)
begin
    /* dodawane są nowe wierzchołki */
    VUntil := VUntil ∪ { vi0 }; V∧ := V∧ ∪ { vi11 } ∪ { vi12 } ∪ { vi13 } ∪ { vi14 };
    VTrue := VTrue ∪ { vi21 }; VFalse := VFalse ∪ { vi23 } ∪ { vi24 };
    V¬ := V¬ ∪ { vi3 } ∪ { vi4 }; VExt := VExt ∪ { vi5 } ∪ { vi6 };

    v (vi0) := φ1 until φ2;
    v (vi11) := φ1 ∧ φ2 ; v (vi21) := true ;
    v (vi12) := φ1 ∧ ¬φ2; v (vi22) := φ1 until φ2;
    v (vi13) := ¬φ1 ∧ φ2; v (vi23) := false;
    v (vi14) := ¬φ1 ∧ ¬φ2; v (vi24) := false;
    v (vi3) := ¬φ1; v (vi4) := ¬φ2;
    v (vi5) := φ1; v (vi6) := φ2;
    vi22 := vi0;

    /* dodawane są nowe krawędzie obliczania */
    EB := EB ∪ { < vi5, vi11> } ∪ { < vi6, vi11> } ∪ { < vi5, vi12> } ∪ { < vi4, vi12> }
    EB := EB ∪ { < vi3, vi13> } ∪ { < vi6, vi13> } ∪ { < vi3, vi14> } ∪ { < vi4, vi14> }

    /* dodawane są nowe łańcuchy */
    K := K ∪ { < vi0, vi11, vi21>, < vi0, vi12, vi22>, < vi0, vi13, vi23>, < vi0, vi14, vi24> };

    /* Przyporządkowanie wartości zmiennym związanym z ograniczeniami metrycznymi */
    sign(vi0) := sg; met(vi0) := s;
endproc

```



**procedure Next (formula  $\varphi$ , graph T)**

**begin**

$v(v_{i1'}) := \varphi$ ;

$V_{Next} = V_{Next} \cup \{v_{i1'}\}$ ;

$E_B := E_B \cup \{<v_{i1'}, v_{i1'}>\}$

**endproc;**

**procedure TemporalFGPCreation( node v, graph T)**

**begin**

*/\* s - oznacza operator metryczny ( $s \in \text{integer} \cup \text{NULL}$ ), a sg - znak przy tym operatorze:*

*sg  $\in \{<, \leq, >, \geq, \text{NULL}\}$  \*/*

**case v(v) of**

always-future  $sg \ s \ \varphi$  : AlwaysFuture ( $\varphi$ , T, sg, s);

sometime-future  $sg \ s \ \varphi$  : SometimeFuture ( $\varphi$ , T, sg, s);

$\varphi_1$  until  $sg \ s \ \varphi_2$  : Until( $\varphi_1$ ,  $\varphi_2$ , T, sg, s);

next  $\varphi$  : Next( $\varphi$ , T);

**endcase;**

**endproc;**

//Konstrukcja grafów typu FTG

**Procedure SubFTGCreation( node v, graph T)**

**begin**

**case v(v) of**

$\varphi_1 \wedge \varphi_2$  : */\* utwórz dwa nowe węzły  $v_i$  i  $v_j$  \*/*

$V_{\wedge} := V_{\wedge} \cup \{v\}$ ;  $V_{Ext} := V_{Ext} \cup \{v_i, v_j\}$ ;

$E_B := E_B \cup \{<v_i, v>, <v_j, v>\}$ ;

$v(v_i) := \varphi_1$ ;  $v(v_j) := \varphi_2$ ;

$\varphi_1 \vee \varphi_2$  : */\* utwórz dwa nowe węzły  $v_i$  i  $v_j$  \*/*

$V_{\vee} := V_{\vee} \cup \{v\}$ ;  $V_{Ext} := V_{Ext} \cup \{v_i, v_j\}$ ;

$E_B := E_B \cup \{<v_i, v>, <v_j, v>\}$ ;

$v(v_i) := \varphi_1$ ;  $v(v_j) := \varphi_2$ ;

$\neg \varphi_1$  : */\* utwórz nowy węzeł  $v_i$  \*/*

$V_{\neg} := V_{\neg} \cup \{v\}$ ;  $V_{Ext} := V_{Ext} \cup \{v_i\}$ ;

$E_B := E_B \cup \{<v_i, v>\}$ ;  $v(v_i) := \varphi_1$ ;

$A_i$  : */\*  $A_i$  jest formułą atomową \*/*

$V_{atom} := V_{atom} \cup \{v\}$ ;  $v(v) := A_i$ ;

*/\* tworzenie podgrafu dla operatora temporalnego FTL \*/*

FTL\_operator : TemporalFGPCreation ( v, T);

```

// tylko w przypadku tworzenia grafu MTG
// tworzenie podgrafu dla operatora temporalnego PTL

PTL_operator : TemporalPGPCreation ( v, T);
endcase;
endproc;

procedure FGPKonstrukcja ( formula  $\varphi$ )
  v( v0) :=  $\varphi$  /* utwórz v0 */
  SubFTGCreation(v0,T)

endproc;

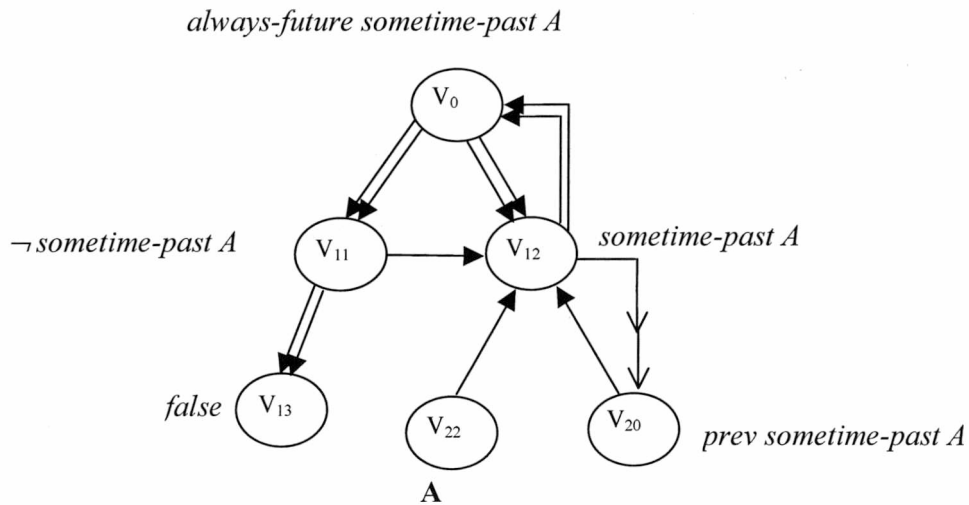
```

### 3.2.1.4 Bitemporalne grafy przejść (MGP)

Jak to zostało wspomniane w rozdziale I formuły MTL, to formuły w których mogą występować jednocześnie operatory PTL i FTL. Skoro zostały już stworzone grafy PTG i FTG, służące do reprezentacji odpowiednio operatorów PTL i FTL, nic nie stoi na przeszkodzie, by na ich podstawie utworzyć graf MTG, który reprezentowałby formuły MTL. Wyszpecyfikowanie metody tworzenia grafu MTG polega na dodaniu do algorytmu 2.4 procedury PGPCreation konstruującej podgrafy PGP. Oto przykład takiego grafu:

#### Przykład 2.6

Niech  $\varphi_I = \text{always-future sometime-past } A$ . Zgodnie z powyższym algorytmem najpierw tworzony jest podgraf dla podformuły  $\varphi_{I1} = \text{sometime-past } A$ , co pozwala na utworzenie grafu  $\varphi_I = \text{always-future } \varphi_{I1}$ . Efektem takiego postępowania jest następujący graf:



Rysunek 2.9 Przykładowy graf dla formuły  $\varphi = \text{always-future sometime-past } A$

Grafy MTG są prostym złożeniem grafów FTL i PTL. Wobec tego graf taki składa się z komponentów charakterystycznych dla grafów FTL i PTL.

Mając do dyspozycji tak zdefiniowane grafy można przejść do stworzenia algorytmów weryfikacji formuł FTL i MTL operujących na tych grafach.

### 3.2.2 Weryfikacja formuł logiki FTL i MTL

Podrozdział wprowadza nową metodę weryfikacji formuł FTL i MTL z ograniczeniami metrycznymi. Metody weryfikacji tych dwóch rodzajów formuł mają tak wiele cech wspólnych, że zdecydowano się przedstawić jedną metodę - weryfikację formuł MTL. Jak wiadomo z rozdziału I formuła typu MTL jest złożona z podformuł PTL i FTL. Problemy związane z weryfikacją formuł PTL zostały opisane w podrozdziale 3.1. W tym rozdziale najpierw zostaną przeanalizowane cechy charakterystyczne dla weryfikacji formuł FTL, a następnie problemy związane z weryfikacją podformuł PTL w formule MTL.

Podrozdział ten jest więc napisany w sposób następujący: Pierwsza część zawiera definicje i założenia wstępne przydatne w procesie weryfikacji. Część druga to algorytm weryfikacji formuł typu MTL. Część trzecia wyjaśnia kiedy algorytm powinien być uruchamiany. Kolejne dwie części to odpowiednio: opis metod algorytmu oraz przykłady weryfikacji formuł MTL.

#### 3.2.2.1 Założenia wstępne

Weryfikacja formuł temporalnych FTL jest znacznie bardziej skomplikowana niż weryfikacja PTL. W przypadku formuł PTL weryfikacja w dowolnym punkcie  $k$  odbywa się na podstawie danych już znanych niezależnie od stopnia złożoności formuły. Natomiast w przypadku formuł FTL dane te jeszcze nie są znane. Wobec tego weryfikacja odbywa się „wstecznie”. Pociąga to za sobą pewne konsekwencje. W celu wyjaśnienia pewnych zależności zachodzących przy weryfikacji formuł FTL zostaną wprowadzone definicje następujących zmiennych.

#### Definicja 2.7

##### Stan weryfikacji

Stan weryfikacji formuły temporalnej  $\varphi$  z którą związany jest wierzchołek  $v$  przy waluacji  $\theta$  oznaczony jako  $ver\_state(v, \theta)$ , to największy punkt czasu, w którym jest już określona wartość formuły  $\varphi$ . ■

#### Definicja 2.8

##### Stan rozpatrywania

Stan rozpatrywania formuły temporalnej  $\varphi$  z którą związany jest wierzchołek  $v$  przy waluacji  $\theta$  oznaczony jako  $prog\_state(v, \theta)$ , to największy punkt czasu, w którym jest określone są wszystkie podformuły formuły  $\varphi$ . ■

#### Lemat 2.1

Niech  $\varphi$  jest formułą postaci  $op_{\leq(<)} s(\varphi_1, \varphi_2)$  gdzie  $op$  operator FTL, a  $\varphi_1$  i  $\varphi_2$  podformuły  $\varphi$ . Niech  $v$  wierzchołek związany z formułą  $\varphi$ , a  $v_1$  i  $v_2$  wierzchołki związane odpowiednio z podformułami  $\varphi_1$  i  $\varphi_2$ . Wartość formuły  $\varphi$  w punkcie  $k$  przy waluacji  $\theta$  można określić

najpóźniej w punkcie  $k+s$  ( $k+s-1$ ), przy założeniu, że wartości podformuł  $\varphi_1$  i  $\varphi_2$  są znane w każdym punkcie przedziału  $\langle k, k+s \rangle$  ( $\langle k, k+s-1 \rangle$ ).

### Dowód lematu 2.1

Poprawność lematu 2.1 wynika bezpośrednio z semantyki operatora  $op$  z ograniczeniem metrycznym  $\leq(<)$ . ■

### Wniosek:

Z lematu wynika bezpośrednio:  $prog\_state(v, \theta) - ver\_state(v, \theta) \leq (<) s$ . Co więcej wartość formuły  $\varphi$  w punktach pomiędzy  $prog\_state(v, \theta)$ ,  $ver\_state(v, \theta)$  nie jest znana (*unknown*). ■

Sytuacja jest prosta, jeżeli  $\varphi_1$  i  $\varphi_2$  są atomowymi podformułami formuły  $\varphi$ . Może się jednak zdarzyć, że podformuła  $\varphi_1$  formuły  $\varphi$  jest postaci:  $op_{1 \leq (<) s_1}(\varphi_{11}, \varphi_{12})$  a podformuła  $\varphi_2$  postaci  $op_{2 \leq (<) s_2}(\varphi_{21}, \varphi_{22})$ , gdzie  $op_1$  i  $op_2$  operatory temporalne FTL. Wówczas zachodzi wprowadzić  $prog\_state(v_1, \theta) - ver\_state(v_1, \theta) \leq (<) s_1$  oraz  $prog\_state(v_2, \theta) - ver\_state(v_2, \theta) \leq (<) s_2$  co przy równości  $prog\_state(v, \theta) = \min(ver\_state(v_1, \theta), ver\_state(v_2, \theta))$ , ale w rezultacie wystąpi zależność:  $prog\_state(v, \theta) - ver\_state(v, \theta) \leq (<) s + \max(s_1, s_2)$

Tak więc wartość formuły  $\varphi$  w stanie  $k$  można stwierdzić najpóźniej w stanie  $k + s + s_1$  (oczywiście znów przy założeniu, że wartości podformuł:  $\varphi_1, \varphi_2, \varphi_{11}, \varphi_{12}, \varphi_{21}, \varphi_{22}$  itd. są ustalone w każdym punkcie przedziału  $\langle k, k+s+s_1 \rangle$  ( $\langle k, k+s+s_1-2 \rangle$ ). Rozumowanie takie można prowadzić dalej w zależności od tego, jak daleki jest stopień zależności formuły  $\varphi$ .

Z powyższego rozumowania można wywnioskować, że im większy jest stopień zależności formuły FTL, tym większa może być różnica pomiędzy stanem rozpatrywania najbardziej zagnieżdżonego operatora temporalnego a stanem weryfikacji formuły rozpatrywanej.

Przykład 2.6 ukazuje zależność pomiędzy stanem weryfikacji a stanem rozpatrywania.

### Przykład 2.6

Dana niech będzie formuła  $\varphi = \text{always-future}_{\leq 4} A$ . a  $\theta$  waluacja podformuły  $A$ .

Jeżeli wartość podformuły  $A$  w punkcie  $k$  jest *true* co zapisujemy ( $A[k]=\text{true}$ ) i tylko w tym punkcie znana jest wartość podformuły  $A$ , to zgodnie z semantyką operatora  $\text{always-future}_{\leq s}$  wartość formuły  $\varphi$  w punkcie  $k$  wynosi *unknown*. Dopiero jeżeli  $A[k+1]=\text{true}$ ,  $A[k+2]=\text{true}$ ,  $A[k+3]=\text{true}$ ,  $A[k+4]=\text{true}$ , to dopiero wtedy można wywnioskować, że wartość formuły  $\varphi$  w punkcie  $k$  jest spełniona. Wobec tego, gdy postęp weryfikacji znajdzie się w punkcie  $k+4$  ( $prog\_state(v, \theta) = k+4$ ), stan weryfikacji formuły przesunie się dopiero do punktu  $k$  ( $ver\_state(v, \theta) = k$ ), gdzie  $v$  oznacza wierzchołek grafu związany z formułą  $\varphi$ . ■

Sytuacja zmienia się diametralnie w przypadku znaku  $\geq$  lub  $>$  przy operatorze metrycznym.

### Lemat 2.2

Niech  $\varphi$  jest formułą postaci  $op_{\geq(>)s}(\varphi_1, \varphi_2)$  gdzie  $op$  operator FTL, a  $\varphi_1$  i  $\varphi_2$  podformuły  $\varphi$ . Wartość formuły  $\varphi$  w punkcie  $k$  przy waluacji  $\theta$  można określić najpóźniej w ostatnim punkcie przedziału rozpatrywania  $end\_state(\varphi, \theta)$ .

### Dowód lematu 2.2

Dowód lematu 2.2 wynika bezpośrednio z semantyki ograniczenia metrycznego  $\geq(>)s$ . ■

Tak jak w lemacie 2.1 należy przyjąć założenie, że podformuła  $\varphi_1$  jest określona w każdym punkcie przedziału  $\langle k, end\_state(\varphi, \theta) \rangle$ . W tym przypadku sytuacja jest taka sama niezależnie od stopnia złożoności formuły. Jeżeli bowiem formuła  $\varphi_1$  jest określona w każdym punkcie przedziału  $\langle k, end\_state(\varphi, \theta) \rangle$ , to jej podformuła także musi być określona w każdym punkcie tego przedziału. Tak więc największe opóźnienie weryfikacji wynosi  $end\_state(\varphi, \theta)$ .

### Wnioski z lematów 2.1 i 2.2

Z powyższych rozważań można wywnioskować, że:

- weryfikacja formuły FTL może odbywać się ze znacznym opóźnieniem w stosunku do aktualnego stanu bazy.
- w najgorszym przypadku formuła w stanie  $k$  może być zweryfikowana w końcowym stanie weryfikacji.

Do weryfikacji formuł FTL i MTL wykorzystywana jest logika trójwartościowa. W związku z tym podrozdział wprowadza funkcję markującą  $m'$ , która przyjmuje wartości *true*, *false* lub *unknown*.

### Definicja 2.10

Markowanie  $m'_{v,\theta}$  wierzchołka  $v$  grafu  $T$  typu MTG jest funkcją  $m'_{v,\theta}: k \rightarrow \{true, false, unknown\}$ , gdzie  $k$  - punkt weryfikacji formuły, a  $\theta$  waluacja.

### 3.2.2.2 Algorytm weryfikacji formuł MTL

Mając tak zdefiniowane założenia wstępne można przystąpić do tworzenia metod weryfikacji formuł FTL i MTL.

Algorytm weryfikacji formuł MTL nie jest algorytmem tak efektywnym jak algorytm weryfikujący formuły logiki czasu przeszłego (PTL). Wynika to z faktu, że muszą być przechowywane pewne dane historyczne potrzebne do procesu weryfikacji. Metody zapisu tych dodatkowych informacji przedstawione są w rozdziale 2.2.3.

Z poniższej metody można wyodrębnić metodę weryfikacji formuł FTL poprzez nie branie pod uwagę sekcji specyfikacji napisanych czcionką Courier oraz metodę weryfikacji formuł bez operatorów metrycznych (opuszczenie sekcji opisanych czcionką *Courier Italics*).

Tak więc z poniższej metody można wyodrębnić cztery metody stosowane w zależności od potrzeb. Są to: metoda weryfikacji formuł typu FTL bez operatorów metrycznych, metoda weryfikacji formuł typu FTL z operatorami metrycznymi, metoda MTL bez operatorów metrycznych oraz metoda MTL z operatorami metrycznymi.

**procedure MTLVerification (formula  $\varphi$ , integer  $k$ , valuation  $\theta$ )**

**begin**

/\* Zmienna globalna rejestrująca rozpatrywaną formułę \*/

var:  $\varphi_v$  formula;

$\varphi_v := \varphi$ ;

var: node  $v_0$ ;

/\* przypisanie zmiennej początkowego wierzchołka  $v_0$  grafu związanego z formułą  $\varphi$  \*/

$v_0 := \varphi(v_0)$ ;

/\* rozpoczęcie nowej weryfikacji w stanie aktualnym \*/

MTLMarkNode( $v_0, act\_state, \theta, true$ );

/\* Jeżeli nie można zweryfikować formuły w stanie aktualnym z powodu braku danych wartości podformuł ze stanów poprzednich, stan  $act\_state$  oznaczany jest jako nie zweryfikowany (*unknown*) i jest dołączany do temporalnej bazy stanów nie zweryfikowanych wierzchołka  $v_0$ . \*/

if  $m'_{v_0, \theta}(act\_state) = unknown$  then

insert\_unknown( $v_0, act\_state, \theta$ );

/\* Efektem próby weryfikacji formuły w stanie aktualnym może być możliwość obliczenia wartości formuły (czyli weryfikacja wierzchołka  $v_0$ ) w którymś ze stanów wcześniejszych zapisanych jako *unknown*. W związku z tym następuje próba zweryfikowania formuły w najwcześniejszym niezwerfikowanym stanie zapisanym w temporalnej bazie niezwerfikowanych stanów. Jeżeli weryfikacja zakończy się sukcesem, stan zostaje usuwany z tej bazy i następuje próba zweryfikowania formuły w stanie następnym itd. Obliczenia te zostają powtórzone do momentu dojścia do stanu obecnego lub gdy wartość w którymś ze stanów jest nieokreślona. \*/

/\* Pobieranie najwcześniejszego niezwerfikowanego stanu \*/

$k := \min(\text{MinStateDB}(v_0, \theta))$ ;

while  $k \leq act\_state$  do

MTLMarkNode( $v_0, k, \theta, true$ );

if  $m'_{v_0, \theta}(k) \neq unknown$  then delete\_unknown( $v_0, k, \theta$ );

$k := k + 1$ ;

if  $m'_{v_0, \theta}(k) = unknown$  then break;

endwhile;

**endproc;**

**procedure MTLMarkNode(node  $v$ , integer  $k$ , valuation  $\theta$ , boolean subver)**

**begin**

/\* Rekurencyjna procedura weryfikująca wierzchołki w grafie FTG \*/

**case  $v$  of**

$v \in V_{\text{True}} : m'_{v,\theta}(k) := \text{true};$   
 $v \in V_{\text{False}} : m'_{v,\theta}(k) := \text{false};$

/\* W przypadku klasy formuł atomowych, jeżeli rozpatrywana jest formuła w stanie wcześniejszym niż stan aktualny, jej wartość jest już na pewno zapisana w bazie, a więc można tę wartość uzyskać. Jeżeli rozpatrywana jest formuła w stanie aktualnym, jej wartość jest zapisywana do bazy w celu dalszej weryfikacji. \*/

$v \in V_{\text{Atom}} : \text{if } k < \text{act\_state} \text{ then}$   
      $\text{GetValueDB}(m'_{v,\theta}(k));$  // Pobieranie wcześniej zapisanej wartości  
   else  
     if  $\gamma(v) = \text{true}$  then  $m'_{v,\theta}(k) := \text{true};$   
     else  $m'_{v,\theta}(k) := \text{false};$   
     endif;  
      $\text{insert\_data}(v, \gamma(v), k, \theta);$   
   endif;

$v \in V_{\neg} :$    if  $\langle v', v \rangle \in E$  then  
      $\text{MTLMarkNode}(v', k, \theta, \text{subver});$   
      $m_{v,\theta}(k) := \neg m_{v',\theta}(k);$   
   endif;

$v \in V_{\wedge} :$    if  $\langle v', v \rangle \in E$  and  $\langle v'', v \rangle \in E$  then  
      $\text{MTLMarkNode}(v'', k, \theta, \text{subver});$   
      $\text{MTLMarkNode}(v', k, \theta, \text{subver});$   
      $m_{v,\theta}(k) := m_{v',\theta}(k) \wedge m_{v'',\theta}(k);$   
   endif;

$v \in V_{\vee} :$    if  $\langle v', v \rangle \in E$  and  $\langle v'', v \rangle \in E$  then  
      $\text{MTLMarkNode}(v'', k, \theta, \text{subver});$   
      $\text{MTLMarkNode}(v', k, \theta, \text{subver});$   
      $m_{v,\theta}(k) := m_{v',\theta}(k) \vee m_{v'',\theta}(k);$   
   endif;

$v \in V_{\text{Future}}' :$  if  $\langle v', v \rangle \in E$  then  
      $\text{MTLMarkNode}(v', k, \theta, \text{subver});$   
      $m'_{v,\theta}(k) := m'_{v',\theta}(k);$   
   endif;

$v \in V_{\text{Next}} :$    if  $\langle v', v \rangle \in E$  then  
      $\text{MTLMarkNode}(v', k+1, \theta, \text{subver});$   
      $m'_{v,\theta}(k) := m'_{v',\theta}(k+1);$   
   endif;

$v \in V_{\text{Past}} :$    if exists  $(m'_{v,\theta}(i))$  then  
     // pobieranie wcześniej zapisanej wartości  
      $\text{GetValueDB}(m'_{v,\theta}(i));$

```

        else if subver=true then
            PTLAddVerification(v, k,  $\theta$ );
        endif;

v  $\in$  VSMid :    PTLMarkNodeInMTL(v, k,  $\theta$ );

v  $\in$  VFuture :    if exists ( $m'_{v, \theta}(i)$ ) then GetValueDB( $m'_{v, \theta}(i)$ );
                  else if subver=true then
                    FTLAddVerification(v, k,  $\theta$ );
                  endif;

endcase;
endproc;

// Tylko w przypadku formuł typu MTL.
procedure PTLAddVerification (node v, int k, valuation  $\theta$ )
begin
    var: p integer;
    PTLMarkNode(v, k,  $\theta$ , true);

    /*Cześć algorytmu uruchamiana jest, jeżeli formuła FTL jest
    podformułą PTL. Następuje próba weryfikacji na wierzchołku typu
    Past, jeżeli są jakieś zaległości, czyli jeżeli stan do którego
    posunęła się weryfikacja jest mniejszy od stanu rozpatrywanego.
    Procedura PTLMarkNodeInMTL wywoływana jest z wartością false
    argumentu subver, co oznacza, że po napotkaniu wierzchołka typu
    Past lub Future proces weryfikacji spróbuje uzyskać wartość
    tego wierzchołka w stanie p zamiast wgłębiać się w weryfikację
    podgrafu związanego z tym wierzchołkiem. Jeżeli wartość nie
    zostanie znaleziona, nastąpi natychmiastowe wyjście z pętli
    while */

    if (prog_state(v,  $\theta$ ) < k-1)
        p := prog_state(v,  $\theta$ ) + 1;
        while p <= k do
            PTLMarkNodeInMTL(v, p,  $\theta$ , false);
            p := p + 1;
            if  $m'_{v, \theta}(p)$  = unknown then exit;
        endwhile;
    endif;
endproc;

procedure FTLAddVerification (node v, int k, valuation  $\theta$ )
begin
    var: p integer;

    FTLReplaceValue(v, k,  $\theta$ , true);

    /* Cześć algorytmu uruchamiana jest, jeżeli występują formuły zależne typu FTL. Następuje
    próba weryfikacji na wierzchołku typu Future, jeżeli są jakieś zaległości, czyli jeżeli stan do
    którego posunęła się weryfikacja jest mniejszy od stanu rozpatrywanego, nastąpi próba

```



dodatkowej weryfikacji na tej samej zasadzie co w przypadku procedury *PTLAddVerification*.\*/

```

if (prog_state(v,θ) < k-1)
    p:= prog_state(v,θ) +1;
    while p<=k do
        FTLReplaceValue(v, p, θ, false)
        p:=p+1;
        if m'_{v,θ}(p)= unknown then exit;
    endwhile;
endif;
endproc;

```

**procedure PTLMarkNodeInMTL (node v, int k, valuation θ, bool subver)**  
**begin**  
**case v of**

```

v ∈ VPrev :   if <v', v> ∈ ET then
                m_{v,θ}(k) := m_{v',θ}(k-1);
                endif;

v ∈ VAlPast : if <v', v> ∈ E and <v'', v> ∈ E then
                begin
                    MTLMarkNode (v'', k, θ, subver);
                    MTLMarkNode (v', k, θ, subver);
                    m_{v,θ}(k) := m_{v',θ}(k) ∧ m_{v'',θ}(k);
                    if m'_{v',θ}(k) <> unknown and m'_{v'',θ}(k) <> unknown then
                        if met(v)=NULL then
                            if v' ∈ VPrev then m_{v',θ}(k) := m_{v',θ}(k);
                            else if v'' ∈ VPrev then m_{v'',θ}(k) := m_{v'',θ}(k);
                            endif;
                        else PTLMetricVerification (v,k,θ);
                        endif;
                        // zapis tylko w przypadku, gdy formuła PTL jest
                        // podformułą FTL
                        insert_data(v, m_{v,θ}(k), k, θ);
                    endif;
                endif;

v ∈ VStPast :   if <v', v> ∈ E and <v'', v> ∈ E then
                MTLMarkNode (v'', k, θ, subver);
                MTLMarkNode (v', k, θ, subver);
                m_{v,θ}(k) := m_{v',θ}(k) ∨ m_{v'',θ}(k);

                // warunek jest sprawdzany tylko w przypadku, gdy
                //formuła FTL jest podformułą PTL
                if m'_{v',θ}(k) <> unknown and m'_{v'',θ}(k) <> unknown then

```

```

    if met(v) <> NULL then
        if v' ∈ VPrev then mv',θ(k) := mv,θ(k);
        else if v'' ∈ VPrev then mv'',θ(k) := mv,θ(k);
        endif;
    else PTLMetricVerification(v, k, θ);
    endif;
    // zapis tylko w przypadku, gdy formuła PTL jest
    // podformułą FTL
    insert_data(v, mv(k), k, θ);
endif;
endif;

```

v ∈ V<sub>Since</sub> : if <v', v> ∈ E and <v'', v> ∈ E then  
 MTLMarkNode(v'', k, θ, true);  
 MTLMarkNode(v', k, θ, true);

/\* sprawdzanie tylko w przypadku, gdy formuła FTL jest  
 podformułą PTL \*/

```

if m'v',θ(k) <> unknown and m'v'',θ(k) <> unknown then
    m'v,θ(k) := m'v',θ(k) ∧ m'v'',θ(k);
endif;

```

```

if v' ∈ VPrev then m'v',θ(k) := m'v,θ(k);
if v'' ∈ VPrev then m'v'',θ(k) := m'v,θ(k);

```

```

if v' ∈ VSMid then threshold(v, θ) := threshold(v', θ);
else if v'' ∈ VSMid then threshold(v, θ) := threshold(v'', θ);
endif;

```

```

if <v, v'''> ∈ ET then
    mv''',θ(k) := mv,θ(k);
endif;

```

```

if met(v) <> NULL then PTLMetricVerification(v, k, θ);
else insert_data(v, mv,θ(k), k, θ);
/* zapis do bazy tylko w przypadku, gdy formuła PTL
jest podformułą FTL */
endif;
endif;

```

v ∈ V<sub>SMid</sub> : if <v', v> ∈ E and <v'', v> ∈ E then  
 MTLMarkNode(v'', k, θ, subver);  
 MTLMarkNode(v', k, θ, subver);

// sprawdzanie tylko w przypadku, gdy formuła FTL  
 // jest podformułą PTL

```

if m'v',θ(k) <> unknown and m'v'',θ(k) <> unknown then
    m'v,θ(k) := m'v',θ(k) ∨ m'v'',θ(k);

```

```

        if sign(v)='<' or sign(v)='≤' then
        begin
            if  $v' \notin V_{Prev}$  and  $m_{v',\theta}(k) = true$  then threshold(v,  $\theta$ ) := k;
            else if  $v'' \notin V_{Prev}$  and  $m_{v'',\theta}(k) = true$  then
            threshold(v,  $\theta$ ) := k;
            endif;
            else if sign(v)='>' or sign(v)='≥' then
            begin
                if  $v' \notin V_{Prev}$  and  $m_{v',\theta}(k) = true$  and  $m_{v',\theta}(k) = false$  then
                threshold(v,  $\theta$ ) := k;
                else if  $v'' \notin V_{Prev}$  and  $m_{v'',\theta}(k) = true$  and  $m_{v'',\theta}(k) = false$ 
                then threshold(v,  $\theta$ ) := k;
                endif;
            end;
        end;
    end;

endcase;
endproc;

```

**procedure FTLReplaceValue(node v, int i, valuation  $\theta$ , bool subver)**

/\* Zadaniem tej procedury jest zastąpienie wartości wierzchołka v przez wartość takiego wierzchołka v'' łańcucha  $\langle v, v', v'' \rangle \in K$  grafu FTG, gdzie wartość wierzchołka v' jest prawdziwa. \*/

```

begin
     $m'_{v,\theta}(i) := unknown$ ;
    if  $\langle v, v', v'' \rangle \in K$ 
    for each v' do
    begin
        FTLMarkNode(v, i,  $\theta$ , subver);
        if  $m'_{v',\theta}(i) = true$  then
             $m'_{v,\theta}(i) := m'_{v',\theta}(i)$ ;
            if met(v) <> NULL then FTLMetricVerification(v, i,  $\theta$ );
            else ver_state(v,  $\theta$ ) := i;
            endif;
        end;
    end;
    endfor;

    if i = end_state( $\phi_v, \theta$ ) and  $m'_{v,\theta}(i) = unknown$  then
        ver_state(v,  $\theta$ ) = end_state( $\phi_v, \theta$ );
         $m'_{v,\theta}(i) := endmark(v, ver\_state(v, \theta) + 1, end\_state(\phi_v, \theta), \theta)$ ;
    end;
end;
endproc;

```

/\*Końcowa weryfikacja wynikająca z występowania operatorów metrycznych \*/

**procedure FTLMetricVerification (node v, int k, val  $\theta$ )**  
**begin**

```

case v of
 $v \in V_{AlFuture}$  :
```

```

if sign(v) = "<" (" $\leq$ ") then
    if  $m'_{v,\theta}(k)$  = false then
        //własność 1.8a
        // Zapamiętanie wartości false od ver_state(v)+1 do k
        insert_data(v, false, ver_state(v, $\theta$ )+1,  $\theta$ );
        ver_state(v, $\theta$ ) := k;
        verified(v, $\theta$ ) = false; // do transakcji false
    endif;

    if  $m'_{v,\theta}(k)$  = unknown then
        if  $k - (ver\_state(v,\theta)+1) \geq met(v)-1$  ( $met(v)$ ) then
            //własność 1.8b
             $m'_{v,\theta}(ver\_state(v,\theta)+1)$  = true;
            insert_data(v, true, ver_state(v)+1,  $\theta$ );
            ver_state(v, $\theta$ ) := ver_state(v, $\theta$ ) + 1;
        endif;
    endif;

if sign(v) = ">" (" $\geq$ ") then
    if  $m'_{v,\theta}(k)$  = false then
        if  $k - begin\_state(\varphi_v,\theta) > (\geq) met(v)$  then
            //własność 1.8c
            // wszystko false od begin_state( $\varphi_v,\theta$ ) do  $k - c - 1$  ( $k - c$ )
            ver_state(v, $\theta$ ) :=  $k - c - 1$  ( $k - c$ );
            insert_data(v, false, begin_state( $\varphi_v,\theta$ ),  $\theta$ );
        endif;
    endif;
end

 $\forall v \in V_{StFuture}$  :
if sign(v) = "<" (" $\leq$ ") then
    if  $m'_{v,\theta}(k)$  = true then
        // własność 1.7a
        if  $k - (ver\_state(v,\theta) + 1) < (\leq) met(v)$  then
            // Zapamiętanie wartości true od ver_state(v, $\theta$ )+1 do k
            verified (v, $\theta$ ) = true;
            insert_data(v, true, ver_state(v, $\theta$ )+1,  $\theta$ );
            ver_state(v, $\theta$ ) := k;
        endif;
    endif;

    if  $m'_{v,\theta}(k)$  = unknown then
        if  $k - (ver\_state(v,\theta) + 1) > (\geq) met(v)$  then
            //własność 1.7b
             $m'_v(ver\_state(v,\theta)+1)$  := false;
            insert_data(v, true, ver_state(v, $\theta$ )+1,  $\theta$ );
            ver_state(v, $\theta$ ) := ver_state(v, $\theta$ ) + 1;
        endif;
    endif;
endif;

if sign(v) = ">" (" $\geq$ ") then
    if  $m'_{v,\theta}(k)$  = true then
        //własność 1.7c
        if  $k - begin\_state(\varphi_v,\theta) > (\geq) met(v)$  then

```

```

        // wszystko true od begin_state do k - met(v) - 1 (k - met(v))
        ver_state(v,  $\theta$ ) := k - met(v) - 1 (k - met(v));
        insert_data(v, true, begin_state( $\phi_v, \theta$ ),  $\theta$ );
    endif;
endif;
end;

v  $\in$  vUntil:
if sign(v) = "<" (" $\leq$ ") then
// własność 1.6b
    if m'v,  $\theta$ (k) = true then
        if k - (ver_state(v,  $\theta$ ) + 1) < ( $\leq$ ) met(v) then
            // Zapamiętanie wartości true od ver_state(v,  $\theta$ ) + 1 do k
            insert_data(v, true, ver_state(v,  $\theta$ ) + 1,  $\theta$ );
            ver_state(v,  $\theta$ ) := k;
            verified(v,  $\theta$ ) = true;
        endif;
    endif;
    if m'v,  $\theta$ (k) = false then
        if k - (ver_state(v,  $\theta$ ) + 1) < ( $\leq$ ) met(v) then
            // Zapamiętanie wartości false od ver_state(v,  $\theta$ ) + 1 do k
            insert_data(v, false, ver_state(v,  $\theta$ ) + 1,  $\theta$ );
            ver_state(v,  $\theta$ ) := k;
            verified(v,  $\theta$ ) = false;
        endif;
    endif;
    if m'v,  $\theta$ (k) = unknown then
        if k - (ver_state(v,  $\theta$ ) + 1) > ( $\geq$ ) met(v) then
            m'v(ver_state(v,  $\theta$ ) + 1) := false;
            insert_data(v, false, ver_state(v,  $\theta$ ) + 1,  $\theta$ );
            ver_state(v,  $\theta$ ) := ver_state(v,  $\theta$ ) + 1;
        endif;
    endif;

if sign(v) = ">" (" $\geq$ ") then
// własność 1.6d begin
    if m'v,  $\theta$ (k) = true then
        if k - (ver_state(v,  $\theta$ ) + 1) > ( $\geq$ ) met(v) then
            // od ver_state(v,  $\theta$ ) + 1 do k - met(v) - 1 (k - met(v)) true
            ver_state(v,  $\theta$ ) := k - met(v) - 1 (k - met(v));
            insert_data(v, true, ver_state(v,  $\theta$ ) + 1,  $\theta$ );
        endif;
    endif;
    if m'v,  $\theta$ (k) = false then
        od ver_state(v) + 1 do k false
        ver_state(v,  $\theta$ ) := k;
        insert_data(v, false, ver_state(v,  $\theta$ ) + 1,  $\theta$ );
    endif;
endif;
endcase;
endproc;

```

/\*Funkcja służy do określenia wartości podformuły FTL w ostatnim stanie przedziału weryfikacji.\*/

```

function Endmark (node v, integer beg, integer end , valuation  $\theta$ ) : boolean
begin
case v of
V  $\in$  VAlFuture :
    if sign(v) = '<' (' $\leq$ ') then
        insert_data(v, false, beg,  $\theta$ );
        return false;
    else if sign(v) = '>' (' $\geq$ ') then
        insert_data(v, true, beg,  $\theta$ );
        insert_data(v, false, end - met(v) [end - met(v) + 1] ,  $\theta$ );
        return false;
    else insert_data(v, true, beg,  $\theta$ );
        return true;
    endif;

// Pozostałe wierzchołki
default:
    insert_data(v, false, ver_state(v,  $\theta$ )+1,  $\theta$ );
    return false;

endcase;
endproc;

```

### 3.2.2.3 Warunki uruchamiania metody MTLVerification

Niech  $\varphi$  będzie formułą typu FTL, a  $\theta$  waluacja, czyli przypisanie wartości zmiennym wolnym tej formuły. Algorytm weryfikacji uruchamiany jest w przypadku zajścia następujących zdarzeń: transakcji na waluacji  $\theta$  (gdy zajdzie *insert*( $\theta$ ), *modify*( $\theta$ ) lub *delete*( $\theta$ )) oraz w przypadku takiej zmiany stanu bazy, dla której wartość formuły  $\varphi$  może się zmienić bez transakcji (*verified*( $v_0, \theta$ ) = *unknown*). Metoda *MTLVerification* uruchamiana jest przy takich samych warunkach jak metoda *PTLVerification*.

```

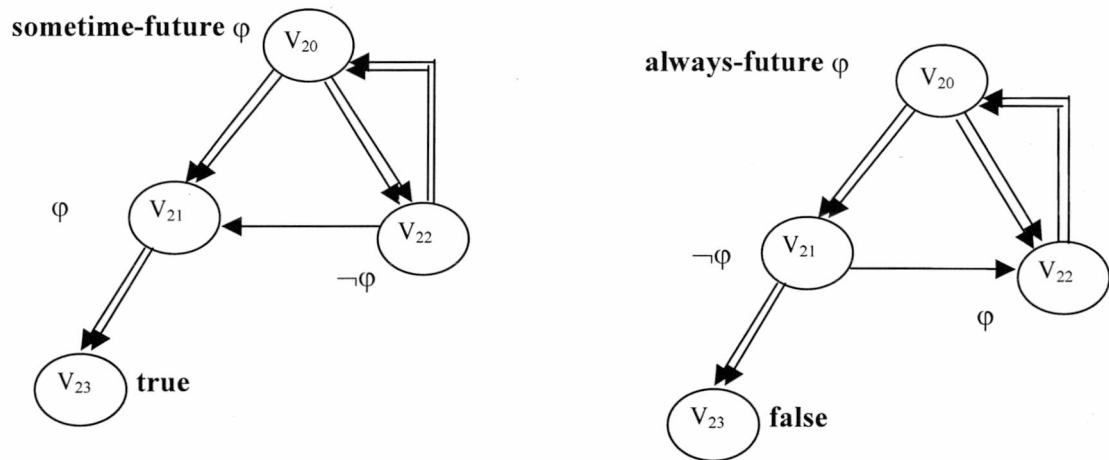
procedure MTLMonitor(integer k)
begin
for each  $\varphi$  do
    for each  $\theta$  do
        if k = begin_state( $\varphi, \theta$ ) then InitValues ( $\varphi, \theta$ );
        endif;
        if trans( $\varphi, \theta$ ) then
            unlock ( $\varphi, \theta$ ) ;
            MTLVerification( $\varphi$ , k,  $\theta$ );
        else if verified( $v_0, \theta$ ) = unknown then
            MTLVerification ( $\varphi$ , k,  $\theta$ );
        endif;
    endfor;
endfor;

```

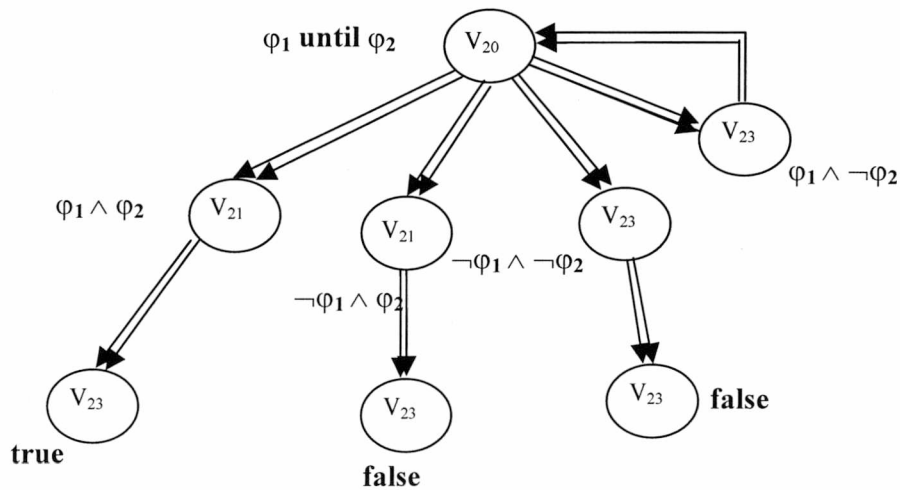
endproc;

### 3.2.2.4 Opis algorytmu weryfikacji formuł MTL

Weryfikacja formuł MTL polega na uruchomieniu rekurencyjnej procedury *MTLMarkNode*. Jeżeli algorytm napotka wierzchołek typu *Past*, uruchamiana jest metoda *PTLMarkNodeInMTL*. Na rysunku 2.10a - 2.10c przedstawione są grafy wierzchołków typu *Future*. Jeżeli algorytm napotka wierzchołek tego typu, wówczas jest on markowany wartością *unknown*. Następnie wybierany jest jeden z łańcuchów grafu, w zależności od tego, który środkowy wierzchołek łańcucha markowany jest wartością *true*. Po wczytaniu wartości ostatniego wierzchołka tego łańcucha, dalsza weryfikacja odbywa się w procedurze *FTLMetricVerification*.



Rys. 2.10a. Graf formuły *sometime-future φ* Rys. 2.10b Graf formuły *always-future φ*



Rys. 2.10c. Graf formuły  $\varphi_1 \text{ until } \varphi_2$

## Opis procedury *FTLMetricVerification*

Procedura *FTLMetricVerification* służy do końcowej weryfikacji wartości poszczególnych wierzchołków FTL uwzględniając przy tym operatory metryczne. Tak jak w przypadku formuł PTL, końcowa weryfikacja uzyskiwana jest na podstawie wprowadzonych w rozdziale I własności wnioskowania.

Opis procedury został podzielony w zależności od typu wierzchołka.

- wierzchołki typu *StFuture* (*sometime-future*  $\varphi$ )
  1. operator  $<$  lub  $\leq$ . Jeżeli wstępna weryfikacja (algorytm *FTLMarkNode*) dała wartość *true*, czyli podformuła  $\varphi$  jest spełniona, formuła jest spełniona w stanach od pierwszego nieokreślonego do rozpatrywanego (własność 1.7a). Gdy wstępna weryfikacja dała wartość *unknown*, sprawdzana jest różnica pomiędzy stanem rozpatrywanym a pierwszym nieokreślonym. Jeżeli jest ona większa od operatora metrycznego, to na podstawie własności 1.7b można wnioskować niespełnienie formuły w pierwszym stanie nieokreślonym.
  2. operator  $>$  lub  $\geq$ . Jeżeli wstępna weryfikacja dała wartość *true*, określana jest wartość formuły tak jak to wskazuje własność 1.7c. Badana jest odległość punktu rozpatrywanego od początku weryfikacji. Jeżeli odległość ta jest większa od operatora metrycznego związanego z wierzchołkiem, to podformuła związana z tym wierzchołkiem przyjmuje wartość *true* w każdym punkcie od początku weryfikacji do stanu mniejszego o wartość metryczną w stosunku do rozpatrywanego
- wierzchołki typu *AlFuture* (*always-future*  $\varphi$ )
  1. operator  $<$  lub  $\leq$ . Jeżeli wstępna weryfikacja dała wartość *false* (podformuła  $\varphi$  nie jest spełniona), to na podstawie własności 1.8a przypisana zostaje wartość *false* wszystkim stanom od pierwszego nieokreślonego do rozpatrywanego. Gdy wstępna weryfikacja dała wartość *unknown* oraz odległość rozpatrywanego stanu jest większa od pierwszego stanu, w którym formuła  $\varphi$  została nieokreślona, to przypisana jest wartość *true* tej pierwszej nieokreślonej wartości na podstawie własności 1.8b.
  2. operator  $>$  lub  $\geq$ . Jeżeli wstępna weryfikacja dała wartość *false*, określana jest wartość formuły tak jak to wskazuje własność 1.8c. Badana jest odległość punktu rozpatrywanego od początku weryfikacji. Jeżeli odległość ta jest większa od operatora metrycznego związanego z wierzchołkiem, to podformuła związana z tym wierzchołkiem przyjmuje wartość *false* w każdym punkcie od początku weryfikacji do stanu mniejszego o wartość metryczną w stosunku do rozpatrywanego
- wierzchołki typu *Until* ( $\varphi_1$  until  $\varphi_2$ )
  3. końcowa weryfikacja korzysta z własności 1.6b oraz 1.6d.

Dlatego weryfikacja jest złożeniem prostego wnioskowania na grafie (wnioskowanie wstępne - procedura *MTLMarkNode*) oraz wnioskowania końcowego (algorytm *FTLMetricVerification*).

## Opis funkcji *Endmark*

Funkcja *Endmark* służy do określenia wartości podformuły FTL w ostatnim stanie przedziału weryfikacji. Jeżeli w tym stanie okazuje się, że istnieją stany niezwerifikowane (o



wartości *unknown*), wówczas należy w jakiś sposób określić ich wartość. W przypadku wierzchołka *AlFuture* bez operatora metrycznego wykorzystywane jest końcowe markowanie z pracy [Zhou 1994]. We wszystkich stanach, w których podformuła przyjmowała wartości *unknown*, nadawana jest wartość *true*. Jest to jak najbardziej zgodne z semantyką *always-future* bez operatora metrycznego. W przypadku wierzchołka *AlFuture* z operatorem metrycznym wykorzystywana jest definicja 1.7 4a oraz 1.7.4d. Przypisywane są wartości *false* dla punktów oddalonych od punktu końcowego weryfikacji o wartość mniejszą niż wartość operatora metrycznego. Dodatkowo, gdy przy operatorze metrycznym występuje znak  $>$  lub  $\geq$  oraz wartość podformuły w punktach oddalonych od punktu końcowego weryfikacji bardziej niż wynosi wartość operatora metrycznego jest *unknown*, wówczas przypisywane są wartości *true*. W przypadku pozostałych wierzchołków wartość *unknown* zamieniana jest na *false*.

### Opis procedury PTLMarkNodeInMTL

Procedura *PTLMarkNodeInMTL* podobna jest do procedury *PTLMarkNode* metody *PTLVerification* opisanej w podrozdziale 3.1. Różnica polega na tym, że procedura *PTLMarkNodeInMTL* wykorzystywana jest do weryfikacji formuł MTL. W związku z tym do markowania wierzchołków wykorzystywana jest tutaj logika trójwartościowa, a wartości zweryfikowanych wierzchołków są zapisywane w bazie, gdyż mogą okazać się potrzebne do dalszej weryfikacji.

Procedura przyjmuje argument *subver*. Jego wartość jest przekazywana do procedury *MTLMarkNode* i określa, czy ta procedura ma standardowo prowadzić proces weryfikacji po napotkaniu wierzchołka typu  $V_{Past}$  lub  $V_{Future}$ , czy też ma jedynie spróbować pobrać wartość w tym wierzchołku z bazy.

### 3.2.2.5 Formuły zależne

Obecność podformuł zależnych w formule typu MTL bardzo komplikuje proces weryfikacji. Występują cztery przypadki takiej zależności: FTL - FTL, FTL -PTL, PTL - FTL, PTL-PTL.

#### Przypadek 1

**Podformuła FTL jest zależna od podformuły FTL (FTL-FTL)**

#### Przykład 2.12

Niech  $\phi$  n-zależna formuła FTL przy czym :

formuła  $\phi$  jest postaci  $op_1 < (\mathcal{A} \ c1 \ \phi_1$

formuła  $\phi_1$  jest postaci  $op_2 < (\mathcal{A} \ c2 \ \phi_2$

formuła  $\phi_2$  jest postaci  $op_3 < (\mathcal{A} \ c3 \ \phi_3$

....

formuła  $\phi_{n-1}$  jest postaci  $op_n < (\mathcal{A} \ cn \ \phi_n$

formuła  $\phi_n$  jest postaci  $\psi$

gdzie:  $op_1, op_2, op_3 \dots op_n$  - operatory FTL,  $\phi_1, \phi_2, \phi_3 \dots \phi_n$  - podformuły FTL,  $\psi$  - formuła atomowa. Oczywiście z każdą podformułą  $\phi_i$  związany jest wierzchołek  $v_i \in V_{Future}$

Aby zweryfikować formułę  $\varphi$  w stanie  $k$ , musi być zweryfikowana podformuła  $\varphi_1$  w stanach  $\langle k, k+c_1 \rangle$ . Do tego celu jednak musi być zweryfikowana podformuła  $\varphi_2$  w stanach  $\langle k+c_1, k+c_1+c_2 \rangle$  itd. Jak pokazują rozważania z początku rozdziału warunkiem zweryfikowania formuły jest zweryfikowana podformuła. Może się zdarzyć, że przejście z jednego stanu do kolejnego spowoduje, że podformuła np.  $\varphi_{n-1}$  zostanie zweryfikowana w więcej niż jednym punkcie np. w przedziale  $\langle k, k+n \rangle$ . Powstaje wtedy porcja danych o długości  $n$ . Obliczone wartości w tych punktach zapisane są do tymczasowej bazy. W takim przypadku weryfikacja następnej w hierarchii podformuły ( $\varphi_{n-2}$ ) musi odbyć się  $n$  razy, czyli tyle razy, jak długa jest ta porcja danych. W związku z tym musi zostać uruchomiona dodatkowa weryfikacja w wierzchołku  $v_{n-2}$  (pętla while procedura *FTLAddVerification*). Ta dodatkowa weryfikacja korzysta tylko z danych zapisanych w bazie dla wierzchołka  $v_{n-1}$  nie wgłębiając się w podgraf, którego korzeniem jest ten wierzchołek. Jeżeli w wyniku takiego postępowania zweryfikowana zostanie podformuła  $\varphi_{n-2}$  także w kilku punktach, to postępowanie takie jest powtarzane w kolejnym wierzchołku  $v_{n-3}$ . ■

### Definicja 2.9

#### Porcja danych

Porcja danych to zbiór zweryfikowanych stanów w jednym procesie weryfikacji. ■

### Definicja 2.10

#### Długość porcji danych

Długość porcji danych to ilość zweryfikowanych stanów w jednym procesie weryfikacji. ■

Zarówno czas jak i długość porcji danych zależą od operatorów metrycznych, rodzajów podformuł oraz waluacji formuł atomowych. Występowanie porcji danych powoduje drastyczne zwiększenie złożoności obliczeniowej w niektórych stanach. Jednak jak wykaże kolejny rozdział, praktycznie wykorzystywany stopień zagnieżdżenia formuł wynosi 2. Tak więc formuła  $\varphi$  z przykładu 2.12 na pewno nie pojawi się w praktyce.

Przykład 2.13 ukazuje przykładowe tworzenie się porcji danych w formule złożonej.

### Przykład 2.13

Dana niech będzie formuła:

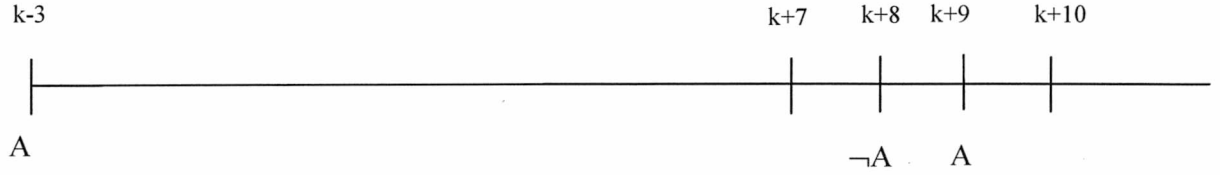
$$\varphi = \text{always-future}_{\leq 4} \text{always-future}_{\leq 6} A$$

Formuła  $\varphi$  może być zapisana w sposób następujący:

$$\varphi_1 = \text{always-future}_{\leq 6} A,$$

$$\varphi = \text{always-future}_{\leq 4} \varphi_1$$

Rysunek 2.15 ukazuje waluację  $\theta$  formuły  $A$ . Tak więc w przedziale  $\langle k-3, k+7 \rangle$  formuła  $A$  przyjmuje wartość *true*, w punkcie  $k+8$  zaszła transakcja, która zmieniła wartość formuły  $A$  na *false*, a w punkcie  $k+9$  formuła  $A$  przyjęła znów wartość *true*. Chcąc obliczyć wartość formuły  $\varphi$  w punkcie  $k$ , należy znać wartość podformuły  $\varphi_1$  w punkcie  $k+4$ . Z kolei, by móc zweryfikować formułę  $\varphi_1$  w punkcie  $k+4$ , należałoby znać wartość formuły atomowej  $A$  w punkcie  $k+10$ . Wobec tego może zaistnieć sytuacja, że stan weryfikacji formuły  $\varphi$  jest równy  $k$ , wtedy gdy stan zaawansowania podformuły  $\varphi_1$  wynosi  $k+10$ . Możliwa jest także sytuacja, kiedy wartość formuły  $\varphi$  w punkcie  $k$  zostanie obliczona znacznie wcześniej.



Rys. 2.11. Przykładowa waluacja formuły A

Załóżmy, że proces weryfikacji posunął się do stanu  $k+7$ . Wówczas stan zaawansowania podformuły  $\varphi_1$  przy waluacji  $\theta$  wynosi  $k+7$  ( $prog\_state(\varphi_1, \theta) = k+7$ ), a stan weryfikacji tej podformuły wynosi  $k+1$  ( $ver\_state(\varphi_1, \theta) = k+1$ ), gdyż formuła atomowa  $A$  przyjmuje wartość *true* w przedziale  $\langle k-3, k+7 \rangle$ , a więc w punkcie  $k+7$  można wnioskować, że wartość  $\varphi_1$  w punkcie  $k+1$  jest prawdziwa. Stan  $k+1$  jest jednocześnie stanem zaawansowania formuły  $\varphi$  ( $prog\_state(\varphi, \theta) = k+1$ ), a wartość podformuły  $\varphi_1$  wynosi *true* w każdym punkcie przedziału  $\langle k-3, k+1 \rangle$ . Na tej podstawie można wnioskować, że formuła  $\varphi$  przyjmuje wartość *true* w stanie  $k-3$ . W związku z tym stan weryfikacji formuły  $\varphi$  jest równy  $k-3$  ( $ver\_state(\varphi, \theta) = k+3$ ).

W stanie  $k+8$  sytuacja ulega radykalnej zmianie. Z uwagi na fakt, że wartość  $A$  jest *false* następuje natychmiastowe wnioskowanie, że podformuła  $\varphi_1$  przyjmuje wartość *false* we wszystkich punktach przedziału  $\langle k+2, k+8 \rangle$ . Punkty te tworzą porcję danych o długości 7. W takim przypadku stan zaawansowania podformuły  $\varphi_1$  jest równy stanowi weryfikacji tej podformuły i wynosi  $k+8$  ( $prog\_state(\varphi_1, \theta) = ver\_state(\varphi_1, \theta) = k+8$ ). W tym momencie nic nie stoi na przeszkodzie, by posunąć się dalej z weryfikacją formuły  $\varphi$  (do stanu  $k+8$ ). Istnieją wszelkie dane do tego potrzebne, gdyż jest już zweryfikowana podformuła  $\varphi_1$  w punktach  $\langle k+2, k+8 \rangle$ . Należy zatem kilkakrotnie uruchomić weryfikację formuły  $\varphi$  (co sugeruje metoda *FTLAddVerification*). ■

## Przypadek 2

### Podformuła FTL jest zależna od podformuły PTL (PTL-FTL)

Niech  $\varphi$  n-zależna formuła PTL przy czym :

formuła  $\varphi$  jest postaci  $op_1 \langle \varphi_1 \rangle$

formuła  $\varphi_1$  jest postaci  $op_2 \langle \varphi_2 \rangle$

gdzie:  $op_1$  - operator PTL,  $op_2$  - operator FTL,  $\varphi_1$  - podformuła FTL. Oczywiście z formułą  $\varphi$  związany jest wierzchołek  $v \in V_{Past}$ , a z formułą  $\varphi_1$  wierzchołek  $v_1 \in V_{Future}$

Taka sytuacja jest analogiczna do tej opisanej przy okazji rozpatrywania zależności podformuł FTL. Różnica polega na tym, że w tym przypadku  $\varphi$  jest formułą PTL, a  $\varphi_1$  formułą FTL. Weryfikacja podformuły  $\varphi_1$  może dać w rezultacie porcję danych o długości  $n$ , co pociągnie za sobą konieczność uruchomienia  $n$ -razy procedury *PTLAddVerification*, która sprawdza jak daleko może pójść naprzód weryfikacja podformuły  $\varphi$ , która sprawdza jak daleko może pójść naprzód weryfikacja podformuły PTL na podstawie tej porcji danych.

### Przypadek 3

#### Podformuła PTL jest zależna od podformuły FTL (FTL-PTL)

Jak wykazuje przypadek 2 możliwe jest uruchomienie weryfikacji podformuły typu PTL w  $n$  stanach na podstawie powstałej porcji danych. Po weryfikacji podformuły PTL w  $n$  stanach może powstać także porcja danych. Wówczas podformuła FTL musi być zweryfikowana także tyle razy, jak długa jest powstała porcja danych. Weryfikacja odbywa się za pomocą procedury *FTLAddVerification*.

### Przypadek 4

#### Podformuła PTL jest zależna od podformuły PTL (PTL-PTL)

Rozpatrywanie tego przypadku ma jedynie wtedy sens, kiedy występuje zależność: PTL-PTL-FTL. Tylko w takim przypadku może powstać porcja danych (na podformule FTL), która spowoduje uruchomienie procedury *FTLAddVerification* dla podformuły PTL nadrzędnej w stosunku do podformuły FTL, a to z kolei spowoduje uruchomienie procedury *PTLAddVerification* dla kolejnej w hierarchii zależności podformuły PTL.

### 3.2.3 Reprezentacja i manipulowanie dodatkowymi danymi

Przedstawione w tym rozdziale metody weryfikacji korzystają z pewnych dodatkowych danych. Dane te muszą być tak przechowywane, by ich złożoność pamięciowa była jak najmniejsza. Jest rzeczą oczywistą, że w środowisku relacyjnej bazy danych, dane te powinny być umieszczone w relacjach, a w przypadku modelu obiektowego w obiektach.

#### 3.2.3.1 Algorytm PTLVerification

Najpierw rozpatrzmy algorytm weryfikacji formuł PTL (*PTLVerification*). W tym przypadku, gdy występują jedynie formuły PTL oraz gdy weryfikacja w każdym punkcie jest kończona w tym punkcie, nie ma potrzeby zachowywania wartości podformuł w punktach wcześniejszych. Jedyne co musi być zachowywane to wartości wierzchołków typu *Previous* oraz *threshold(v)*.

W związku z tym powstaje klasa (relacja) o następujących atrybutach:

```
class ax = { formula, val, node, value, threshold }  
R-ax = < formula, val, node, value, threshold >
```

przy czym:

*formula* - weryfikowana formuła, *val* - waluacja, *node* - wierzchołek typu *Previous*, *value* - wartość tego wierzchołka, a *threshold* - wykorzystywany podczas weryfikacji dodatkowy punkt progowy.

I to jest cała reprezentacja danych dodatkowych potrzebnych do algorytmu.

#### 3.2.3.2 Algorytm MTLVerification

Algorytm weryfikacji formuł FTL i MTL (*MTLVerification*) jest bardziej skomplikowany niż algorytm *PTLVerification* ze względu na swoją złożoność pamięciową. Fakt ten wynika z potrzeby utrzymywania wartości zweryfikowanych podformuł tych stanów,

w których nie została określona jeszcze wartość formuły głównej. Do tego zbioru należą podformuły wszystkich typów, a więc podformuły atomowe, PTL i FTL.

### **Dane związane z grafem**

Z uwagi na bardziej skomplikowane zależności przy połączeniu operatorów FTL i PTL w jedną formułę, dodatkowa klasa (relacja) dla formuł typu *Past* ma jeden atrybut więcej niż w przypadku algorytmu *PTLVerification*.

```
class ax_past = { formula, valuation, node, value, threshold, prog_state}
R-ax_Past = < formula, valuation, node, value, threshold,
prog_state>
```

Dodatkowy atrybut, *prog\_state*, jest stanem zaawansowania weryfikacji podformuły. Jak się można spodziewać, istnieje typ relacji będący odpowiednikiem typu opisanego powyżej także dla formuł FTL. Schemat relacji ma następującą postać:

```
class ax_future = {formula, val, node, ver_state, prog_state}
R-ax_Future = <formula, valuation, node, ver_state, prog_state>
```

przy czym:

*formula* - weryfikowana formuła, *valuation* - waluacja, *node* - rozpatrywany wierzchołek typu *Past* lub *Future*, *ver\_state* - stan rozpatrywania formuły związanej z wierzchołkiem *node*, *prog\_state* - stan zaawansowania weryfikacji formuły;

### **Dane związane z wartościami podformuł we wcześniejszych stanach**

Oprócz tego potrzebny jest jeszcze drugi rodzaj klasy (relacji). Klasa (relacja) ta zawiera wartości podformuł PTL, FTL oraz formuł atomowych we wcześniejszych zweryfikowanych stanach, które nie zostały jeszcze usunięte. Każda relacja z tej klasy relacji składa się z następujących atrybutów:

```
class data = { formula, node, valuation, s_begin, value}
R_Data = < formula,node,valuation,s_begin, value>
```

przy czym:

*formula* - weryfikowana formuła, *node* - wierzchołek typu *Past* lub *Future*, *valuation* - waluacja, *s\_begin* - pierwszy stan sekwencji tych samych wartości *value*, *value* - wartość stanów od stanu *s\_begin*.

Do pełnej weryfikacji potrzebna jest także dodatkowa klasa (relacja) dla wierzchołka  $v_0$ . Zakładamy, że relacja przechowuje te stany, które nie zostały jeszcze zweryfikowane. Wobec tego posiada dwa atrybuty: *s\_begin* i *s\_end* oznaczające odpowiednio początek i koniec przedziału niezwyfikowanych stanów.

```
class node_v0 = { formula, valuation, s_begin, s_end}
R_v0 = <formula, valuation, s_begin, s_end >.
```

Klasa ta (relacja) nie może być zdefiniowana następująco  $\text{class node\_v0} = \langle \text{formula}, \text{valuation}, s\_begin \rangle$ , co oznaczałoby, że wartość w wierzchołku  $v_0$  od stanu  $s\_begin$  ma wartość nieokreśloną, gdyż może się zdarzyć, że istnieją stany określone większe od  $s\_begin$  po sekwencji stanów nieokreślonych.

### 3.2.3.3 Manipulowanie danymi przez algorytm

Mając tak zdefiniowane relacje pomocnicze, można przystąpić do opisu funkcji manipulowania danymi algorytmu *MTLVerification*. Obiekty (relacje) danych związanych z grafem nie wymagają przedstawiania dodatkowych procedur modyfikacji. W momencie utworzenia grafu weryfikacji dla każdego wierzchołka typu *Past* oraz *Future* powstaje nowy obiekt (krotka relacji), która w czasie procesu weryfikacji rejestruje dane.

Inaczej jest z danymi związanymi z wartościami podformuł we wcześniejszych stanach. W tym przypadku kluczem głównym nie jest wierzchołek  $\langle \text{node} \rangle$ , lecz para  $\langle \text{formula}, \text{node}, s\_begin \rangle$ . Wynika to z faktu, że z wierzchołkiem może być związana jedna lub więcej wartości tego wierzchołka z przeszłości.

Działanie procedury wprowadzania wartości wierzchołka (*insert-data*) polega na znalezieniu ostatniej wpisanej wartości dla tego wierzchołka i sprawdzeniu czy wartość tej wartości jest taka sama jak wartość wpisywana. Jeżeli tak - procedura nie robi nic, a w przeciwnym razie tworzony jest nowy obiekt (krotka) związany z wierzchołkiem *node* oznaczający, że od stanu *state* wierzchołek ten ma wartość *value*.

**procedure insert\_data(node v, integer state, boolean value, valuation val)**

begin

var: b boolean;

//  $\varphi$  - rozpatrywana formuła;

if  $b \neq$  (select R\_Data.value  
from R\_Data  
where R\_Data.valuation = val and R\_Data.node = v)  
then insert into R\_Data ( $\varphi$ , val, v, state, value);  
endif;

endproc;

Procedura *delete\_data* usuwa niepotrzebne obiekty (krotki). Najpierw dla danego wierzchołka szukany jest obiekt (krotka), którego wartość *s\_begin* jest maksymalna, ale jednocześnie mniejsza od parametru *state*. Następnie usuwane są wszystkie obiekty (krotki) o wartości *s\_begin* mniejszej od maksymalnej.

**procedure delete\_data (node v , integer state, valuation val)**

begin

var: integer maximal;

//  $\varphi$  - rozpatrywana formuła;

maximal = (select max (R\_Data.s\_begin) from R\_Data  
where R\_Data.valuation = val and R\_Data.node = v and R\_Data.formula =  $\varphi$ )

```

delete from R_Data
where R_Data.valuation = val and R_Data.node = v and R_Data.s_begin < maximal
and R_Data.formula =  $\varphi$ 
endproc;

```

Dwie poniższe procedury służą do rejestrowania (*insert\_unknown*) oraz usuwania (*delete\_unknown*) tych stanów przeszłych, dla których nie została jeszcze zweryfikowana rozpatrywana formuła (nieznana wartość wierzchołka  $v_0$ ).

Procedura *insert\_unknown* dołącza niezweryfikowany stan do obiektu (krotki)  $\langle formula, valuation, s\_begin, s\_end \rangle$  w ten sposób, że powiększa przedział niezweryfikowanych stanów o stan *state*, jeżeli  $state = s\_end + 1$  lub, w przeciwnym razie, tworzy nowy obiekt (krotkę)  $\langle formula, valuation, state, state \rangle$ .

```

procedure insert_unknown (node v, integer state, valuation val)
begin

```

```

    var: integer maximal;

```

```

    //  $\varphi$  - rozpatrywana formuła;

```

```

    maximal = (select max (R_v0.s_end) from R_v0
    where R_v0.valuation = val and R_v0.formula =  $\varphi$ )

```

```

    if maximal + 1 = state then

```

```

        update R_v0
        set R_v0.s_end = maximal + 1
        where R_v0.s_end = maximal

```

```

    else

```

```

        insert into R_v0 ( $\varphi$ , val, state, state)

```

```

    endif;

```

```

endproc;

```

Również usuwanie stanów niezweryfikowanych odbywa się na podobnej zasadzie. Jeżeli usuwany stan znajduje się na brzegu przedziału  $\langle s\_begin, s\_end \rangle$ , wówczas przedział ten zostaje odpowiednio zmniejszony. Jeżeli natomiast usuwany stan znajduje się w środku przedziału  $\langle s\_begin, s\_end \rangle$ , przedział ten musi być podzielony na dwa podprzedziały  $\langle s\_begin, state - 1 \rangle$  oraz  $\langle state + 1, s\_end \rangle$ .

```

procedure delete_unknown (node v, integer state, valuation val)
begin

```

```

    //  $\varphi$  - rozpatrywana formuła;

```

```

    if exists (select * from R_v0

```

```

    where R_v0.s_begin = state and R_v0.s_end = state and R_v0.valuation = val and
    R_v0.formula =  $\varphi$ )

```

```

    then

```

```

        delete from R_v0

```



```

where R_v0.s_begin = state and R_v0.s_end= state and R_v0.valuation= val and
R_v0.formula =  $\varphi$ 

else if exists (select * from R_v0
where R_v0.s_begin = state and R_v0.valuation= val and R_v0.formula =  $\varphi$ )
then update R_v0
set R_v0.s_begin = R_v0.s_begin +1
where R_v0.s_begin = state and R_v0.valuation= val and R_v0.formula =  $\varphi$ 

else if exists (select * from R_v0
where R_v0.s_end = state and R_v0.valuation= val and R_v0.formula =  $\varphi$ )
then update R_v0
set R_v0.s_end = R_v0.s_end -1
where R_v0.s_end = state and R_v0.valuation= val and R_v0.formula =  $\varphi$ )

else
select s_begin as beg ,s_end as end from R_v0
where R_v0.valuation= val and R_v0. s_begin < state and R_v0. s_end > state and
R_v0.formula =  $\varphi$ 

insert into R_v0 values ( $\varphi$ , val , beg, state-1)
insert into R_v0 values ( $\varphi$ , val , state+1,end)
delete from R_v0
where R_v0.s_begin = beg and R_v0.s_end= end and R_v0.valuation= val and
R_v0.formula =  $\varphi$ 
endif;
endproc;

```

Procedury *insert\_unknown* oraz *delete\_unknown* wykorzystywane są tylko w przypadku, gdy muszą zostać zapisane dane niezwerfikowane. Może się tak zdarzyć, że istnieje potrzeba zapisywania i przechowywania wartości zweryfikowanych w wierzchołku  $v_0$ . Wówczas, zamiast  $R_{v0}$ , można wykorzystać obiekty klasy *data* (relację  $R_{Data}$ ) i korzystać z metody *insert\_data*.

### 3.2.4 Metody weryfikacji formuł temporalnych w pojedynczym punkcie

Metoda *MTLVerification* jest najbardziej pełną wersją obsługującą formuły temporalne złożone z wszelkich kombinacji operatorów FTL i PTL. Często jednak, szczególnie w praktycznym zastosowaniu, nie jest wymagana aż tak skomplikowana metoda. W zależności od konkretnego przypadku oraz rodzaju formuł jakie muszą być weryfikowane, metoda może zostać znacznie uproszczona.

Poniższa wersja algorytmu może z powodzeniem służyć do weryfikacji dowolnej prostej formuły MTL w jakimś określonym punkcie czasu  $t$ . Warto przypomnieć, że algorytmy *PTLVerification* oraz *MTLVerification* weryfikowały formuły w sekwencji punktów czasowych, a nie w pojedynczych punktach. Oczywiście metodę *StateVerification* można stosować także w sekwencji punktów czasowych, ale jest to nieoptyczne z uwagi na redundancję obliczeń. W rozdziale III do weryfikacji formuł FTL wykorzystywana jest



okrojona wersja metody *StateVerification*, a mianowicie metoda *FTL StateVerification*, dopuszczająca weryfikację jedynie formuł prostych FTL.

### Przykład 2.12

Dane niech będą formuły:

$$\begin{aligned}\varphi_1 &\equiv A \Rightarrow B \text{ until } <_4 C, \\ \varphi_2 &\equiv \text{always\_future}_{<7} C \wedge \text{sometime\_future}_{>5} D\end{aligned}$$

Jak już to zostało wspomniane, stosowanie metody *StateVerification* ma sens wtedy, gdy weryfikacja odbywa się nie w każdym punkcie jakiegoś przedziału, ale w jednym z tych punktów lub periodycznie. W takich bowiem przypadkach poniższy algorytm jest efektywny.

I tak w przypadku formuły  $\varphi_1$  punkt weryfikacji wyznaczony jest przez punkt, w którym wartość podformuły  $A$  jest *true*. Tylko wtedy bowiem formuła  $\varphi_1$  ma szansę być spełniona.

W przypadku formuły  $\varphi_2$  należy najpierw ustalić punkt weryfikacji, a następnie rozpocząć proces weryfikacji.



Do weryfikacji formuły w pojedynczym punkcie należy podać pewne założenia wstępne:

- jeżeli formuła  $\varphi$  jest formułą PTL z operatorem metrycznym  $s$  oraz musi być zweryfikowana wybiórczo w stanie  $k$  to jej weryfikacja musi się rozpocząć w punkcie  $k-s$  (jeżeli występuje znak  $\leq$  przy operatorze  $s$ ) lub w punkcie  $k-s+1$  (jeżeli występuje znak  $<$  przy operatorze  $s$ ). W przypadku znaku  $\geq$  lub  $>$  przy operatorze metrycznym  $s$  weryfikacja rozpocząć się musi już w stanie na początku przedziału rozpatrywania.
- jeżeli formuła  $\varphi$  jest formułą FTL z operatorem metrycznym  $s$  oraz musi być zweryfikowana wybiórczo w stanie  $k$  to jej weryfikacja musi się zakończyć w punkcie  $k+s$  (jeżeli występuje znak  $\leq$  przy operatorze  $s$ ) lub w punkcie  $k+s-1$  (jeżeli występuje znak  $<$  przy operatorze  $s$ ). W przypadku znaku  $\geq$  lub  $>$  przy operatorze metrycznym  $s$  możliwe jest, że weryfikacja skończy się dopiero na końcu przedziału rozpatrywania.
- jeżeli formuła temporalna nie zawiera operatora metrycznego, jej weryfikacja musi być prowadzona od początku do końca przedziału na początku przedziału rozpatrywania nawet wtedy, gdy wymagane jest zweryfikowanie tej formuły jedynie w wybiórczych punktach przedziału.

Chcąc zbadać wartość formuły w stanie  $t$ , stan ten musi zostać zapisany. Chwila  $t$  jest wówczas punktem odniesienia pozwalającym na końcową weryfikację operatorów metrycznych. I tak w przypadku formuły  $\varphi_1$  z przykładu 2.12 punkt  $t$  jest punktem zajścia zdarzenia  $A$ , a w przypadku formuły  $\varphi_2$  narzucony jest punkt weryfikacji.

#### 3.2.4.1 Metoda StateVerification

Metoda *StateVerification* składa się z procedur *MarkNodeStateVerification* oraz *FTLMetricStateVerification*. Procedura *MarkNodeStateVerification* służy do markowania wierzchołków grafu weryfikowanej formuły, a procedura *FTLMetricStateVerification* jest

wykonuje końcową weryfikację podformuł FTL wynikającą z występowania operatorów metrycznych.

**procedure MarkNodeStateVerification(node v, integer k, integer t, valuation  $\theta$ )**  
**begin**

case v of

$v \in V_{\text{True}} : m'_{v,\theta}(k) := \text{true};$

$v \in V_{\text{False}} : m'_{v,\theta}(k) := \text{false};$

$v \in V_{\text{Atom}} : \text{if } \gamma(v) = \text{true} \text{ then}$   
 $\quad m'_{v,\theta}(k) := \text{true};$   
 $\quad \text{else}$   
 $\quad \quad m'_{v,\theta}(k) := \text{false};$   
 $\quad \text{endif};$

$v \in V_{\neg} :$       $\text{if } \langle v', v \rangle \in E \text{ then}$   
 $\quad \text{MarkNodeStateVerification}(v', k, t, \theta);$   
 $\quad m'_{v,\theta}(k) := \neg m'_{v',\theta}(k);$   
 $\quad \text{endif};$

$v \in V_{\wedge} :$       $\text{if } \langle v', v \rangle \in E \text{ and } \langle v'', v \rangle \in E \text{ then}$   
 $\quad \text{MarkNodeStateVerification}(v', k, t, \theta);$   
 $\quad \text{MarkNodeStateVerification}(v'', k, t, \theta);$   
 $\quad m'_{v,\theta}(k) := m'_{v',\theta}(k) \wedge m'_{v'',\theta}(k);$

$v \in V_{\vee} \text{ or } V_{\text{SMid}} :$   
 $\quad \text{if } \langle v', v \rangle \in E \text{ and } \langle v'', v \rangle \in E \text{ then}$   
 $\quad \quad \text{MarkNodeStateVerification}(v', k, t, \theta);$   
 $\quad \quad \text{MarkNodeStateVerification}(v'', k, t, \theta);$   
 $\quad \quad m'_{v,\theta}(k) := m'_{v',\theta}(k) \vee m'_{v'',\theta}(k);$

$v \in V_{\text{Since}} :$       $\text{if } \langle v', v \rangle \in E \text{ and } \langle v'', v \rangle \in E \text{ then}$   
 $\quad \text{MarkNodeStateVerification}(v', k, t, \theta);$   
 $\quad \text{MarkNodeStateVerification}(v'', k, t, \theta);$   
 $\quad m'_{v,\theta}(k) := m'_{v',\theta}(k) \wedge m'_{v'',\theta}(k);$

$\text{if } \text{sign} = '<' \text{ or } '\leq' \text{ then}$   
 $\quad \text{if } k=t \text{ then } m'_{v',\theta}(t) := m'_{v,\theta}(k);$   
 $\quad \text{endif};$   
 $\text{if } \text{sign} = '>' \text{ [}'\geq'\text{]} \text{ then}$   
 $\quad \text{if } k = t - \text{met}(v) - 1 \text{ [} k = t - \text{met}(v) \text{]} \text{ and } m'_{v,\theta}(k) = \text{false}$   
 $\quad \text{then } m'_{v',\theta}(t) := \text{false};$   
 $\quad \text{else if } k=t \text{ then } m'_{v',\theta}(t) := m'_{v,\theta}(k);$   
 $\quad \text{endif};$   
 $\text{endif};$

```

endif;

v ∈ VAlPast: if <v', v> ∈ E and <v'', v> ∈ E then
    MarkNodeStateVerification (v', k, t, θ);
    MarkNodeStateVerification (v'', k, t, θ);
    m'v, θ(k) := m'v', θ(k) ∧ m'v'', θ(k);

    if sign = '<' or '≤' then
        if k=t then m'v', θ(t) := m'v, θ(k);
    endif;
    if sign='>' ['≥'] then
        if k = t - met(v)-1 [k = t - met(v)]
            then m'v', θ(t) := m'v, θ(k);
        endif;

v ∈ VStPast: if <v', v> ∈ E and <v'', v> ∈ E then
    MarkNodeStateVerification (v', k, t, θ);
    MarkNodeStateVerification (v'', k, t, θ);
    m'v, θ(k) := m'v', θ(k) ∨ m'v'', θ(k);

    if sign='<' (≤) then
        if k=t then m'v', θ(t) := m'v, θ(k);
    endif;
    if sign='>' [≥] then
        if k = t - met(v)-1 [t - met(v)] then m'v', θ(t) := m'v, θ(k);
    endif;

v ∈ VFuture: if m'v, θ(k)=unknown then FTLMetricStateVerification(v, k, t, θ);

```

**endproc;**

**procedure FTLMetricStateVerification(node v, integer k, integer t, valuation θ)**

**begin**

for each <v, v', v''> ∈ K

    m'<sub>v, θ</sub>(k) := unknown;

    if m'<sub>v', θ</sub>(k) = true then

        m'<sub>v, θ</sub>(k) := m'<sub>v', θ</sub>(k);

    if sign(v)=NULL and m'<sub>v, θ</sub>(k) <> unknown then m'<sub>v, θ</sub>(t) := m'<sub>v, θ</sub>(k);

    else

        if v ∈ V<sub>AlFuture</sub>

            if sign(v)='<' (≤) then

                m'<sub>v, θ</sub>(k) := m'<sub>v, θ</sub>(k) or k - t ≥ (>) met(v);

                if m'<sub>v, θ</sub>(k) <> unknown then m'<sub>v, θ</sub>(t) := m'<sub>v, θ</sub>(k);

            else if sign(v)='>' (≥) then

                if (m'<sub>v, θ</sub>(k)=false and k - t ≥ (>) met(v) then

                    m'<sub>v, θ</sub>(t) := false;

            endif;

        else //pozostałe wierzchołki FTL

            if sign(v)='<' (≤) then

```

        m'_{v, \theta}(k) : = m'_{v, \theta}(k) \wedge k - t < (\leq) met(v);
        if m'_{v, \theta}(k) <> unknown then m'_{v, \theta}(t) := m'_{v, \theta}(k);
    else if sign(v) = '>' ( $\geq$ ) then
        if (m'_{v, \theta}(k)=true and k - t > ( $\geq$ ) met(v) then
            m'_{v, \theta}(t) := true;
        else if m'_{v, \theta}(k)=false then m'_{v, \theta}(t) := false;
        endif;
    endif;
    if k = end_state( $\phi_v, \theta$ ) and m'_{v, \theta}(k) = unknown then m'_{v, \theta}(t) := StateEndmark(v, k, t, \theta);
endif;
endfor;
endproc

```

*/\*Funkcja służy do określenia wartości podformuły FTL w ostatnim stanie przedziału weryfikacji. Funkcja jest podobna do funkcji Endmark wykorzystywane w algorytmie MTLVerification\*/*

```

function StateEndmark (node v, integer k, integer t , valuation \theta) : boolean
begin
case v of
    v \in V_{AlFuture} :
        if sign(v) = '<' (' $\leq$ ') then return false;
        else if sign(v) = '>' (' $\geq$ ') then
            if k - t  $\geq$  (>) met(v) then return true;
            else return false;
        else return true;
        endif;
    // Pozostałe wierzchołki
    default:
        return false
endcase;
endfunction;

```

*/\* Procedura wpisuje znaczniki czasowe przed rozpoczęciem weryfikacji do tabeli TimeMarksTable \*/*

```

procedure InsertTimeMarks (formula  $\phi$ , integer t, valuation \theta)
begin
    // Zapamiętanie wartości początkowej i końcowej weryfikacji

    insert into TimeMarksTable values ( $\phi$ , begin_state( $\phi, \theta$ ), \theta);
    insert into TimeMarksTable values ( $\phi$ , end_state( $\phi, \theta$ ), \theta);
    insert into TimeMarksTable values ( $\phi$ , t, \theta);

```

```

for each  $v \in V_{\text{Past}}$  do
if  $\text{sign}(v) = '<'$  then insert into TimeMarksTable values  $(\varphi, t - \text{met}(v) + 1, \theta)$ ;
if  $\text{sign}(v) = '\leq'$  then insert into TimeMarksTable values  $(\varphi, t - \text{met}(v), \theta)$ ;
endfor;

for each  $v \in V_{\text{Future}}$  do
if  $\text{sign}(v) = '<'$  then insert into TimeMarksTable values  $(\varphi, t + \text{met}(v) - 1, \theta)$ ;
if  $\text{sign}(v) = '\leq'$  then insert into TimeMarksTable values  $(\varphi, t + \text{met}(v), \theta)$ ;
endfor;
endproc;

// Zasadnicza weryfikacja formuły  $\varphi$  przy waluacji  $\theta$ 
procedure StateVerification(formula  $\varphi$ , integer  $k$ , integer  $t$ , valuation  $\theta$ )
begin

var: node  $v_0$ ;

/* przypisanie zmiennej początkowego wirzecholka  $v_0$  grafu związanego z formułą  $\varphi$  */
 $v_0 := \varphi(v_0)$ ;

if  $m'_{v_0, \theta}(t) = \text{unknown}$  then
    if  $k$  in TimeMarksTable( $\varphi, \theta$ ) then MarkNodeStateVerification( $v_0, k, t, \theta$ );
    else if  $\text{trans}(\varphi, \theta)$  then MarkNodeStateVerification( $v_0, k, t, \theta$ );
    endif;
endif;
endproc;

// Procedura monitorująca
procedure StateMonitor(integer  $k$ )
begin
for each  $\varphi$  do
    for each  $\theta$  do
        begin
            if  $k = \text{begin\_state}(\varphi, \theta)$  then
                InsertTimeMarks( $\varphi, k, t, \theta$ );
            endif;
            StateVerification( $\varphi, k, t, \theta$ );
        endfor;
    endfor;
endfor;
endproc;

```

### 3.2.4.2 Opis metody StateVerification

Powyższy algorytm różni się zasadniczo od algorytmu wprowadzonego w pierwszej części tego rozdziału. Jest on nie tylko uproszczeniem algorytmu głównego służącym do weryfikacji formuł logiki MTL bez możliwości weryfikacji formuł złożonych, ale oprócz tego

wprowadza koncepcję weryfikacji formuły nie w ciągu punktów ale w jednym ustalonym punkcie. Dzięki temu proces weryfikacji jest bardzo szybki, gdyż kończy się po zweryfikowaniu wartości w tym ustalonym punkcie.

Metoda *StateVerification* korzysta z tablicy znaczników czasowych. Tablica ta zawiera numery tych stanów, które mogą coś zmienić w weryfikacji nawet jeżeli nie zajdzie w nich żadna relewantna transakcja. Te znaczniki czasowe mogą być zapisywane w tabeli (w przypadku modelu relacyjnego) lub jako obiekt (model obiektowy). W przypadku poniższego algorytmu znaczniki czasowe są zapisywane do tabeli *TimeMarksTable* o schemacie  $\langle \text{formula}, \text{timemark}, \text{valuation} \rangle$ .

Przed rozpoczęciem weryfikacji, dla każdej walucji uruchamiana jest procedura *InsertTimeMarks* służąca do zapisywania znaczników czasowych. Metoda *MarkNodeStateVerification* w metodzie *StateVerification* jest uruchamiana w przypadku zajścia relewantnej transakcji bądź jeżeli weryfikacja znajduje się w jakimś punkcie z tablicy znaczników czasowych.

Tak więc, mimo zastosowania operatorów metrycznych, koszt obliczeniowy wzrasta bardzo nieznacznie w stosunku do algorytmów w pracy [Zhou 1994], które weryfikują formuły temporalne bez operatorów metrycznych.

Poniższy przykład uzmysławia metodę weryfikacji.

### Przykład 2.13

Zweryfikujemy formułę  $\varphi_3$  w ustalonym punkcie  $t$ .

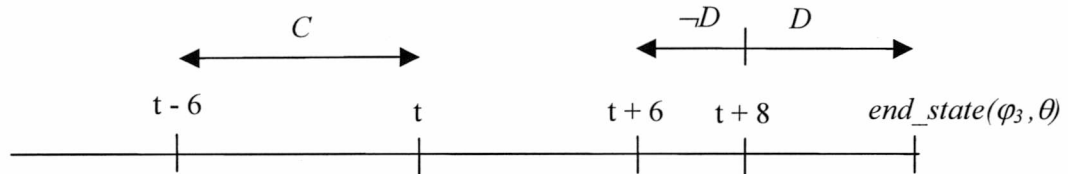
$$\varphi_3 \equiv \text{always\_past}_{<7} C \wedge \text{sometime\_future}_{>5} D$$

Niech:

$$\varphi_{31} \equiv \text{always\_past}_{<7} C,$$

$$\varphi_{32} \equiv \text{sometime\_future}_{>5} D,$$

$\theta$  - walucja podformuł atomowych  $C$  i  $D$ .



Rys. 2.25. Zakresy weryfikacji formuł  $\varphi_{31}$  i  $\varphi_{32}$

Na rysunku 2.25 przedstawiona jest walucja  $\theta$  formuł  $C$  i  $D$ . I tak w przedziale  $<t-6, \text{end\_state}(\varphi, \theta)>$  formuła  $C$  przyjmuje wartość *true*, w przedziale  $<t-6, t+8)$  formuła  $D$  przyjmuje wartość *false*, a w przedziale  $<t+8, \text{end\_state}(\varphi, \theta)>$  formuła  $D$  przyjmuje wartość *true*.

Weryfikacja formuły  $\varphi_3$  odbywa się w sposób następujący:

Najpierw uruchamiana jest procedura *InsertTimeMarks*, która zapisuje znaczniki czasowe utworzone na podstawie ograniczeń metrycznych. Są to cztery elementy:  $t-6$ ,  $t$ ,  $t+6$  oraz  $\text{end\_state}(\varphi, \theta)$ . Procedura *MarkNodeStateVerification* musi być w tych punktach uruchomiona, chyba, że wcześniej zostanie jednoznacznie określona wartość formuły w punkcie  $t$ . Jedynie w tych punktach upływ czasu ma wpływ na wartość formuły temporalnej. Z przedstawionej na rysunku 2.25 walucji  $\theta$  wynika, że jedyna transakcja jaka zaszła w

punkcie  $t+8$  zmieniając wartość podformuły  $D$  z *false* na *true*. W związku z tym algorytm musi być uruchomiony najwyżej w pięciu punktach:  $t-6$ ,  $t$ ,  $t+6$ ,  $t+8$  oraz  $end\_state(\varphi, \theta)$ . W przypadku walucji  $\theta$  wartość poformuły  $\varphi_{32}$  w punkcie  $t$  zostanie określona już w stanie  $t+8$ , a wartość  $\varphi_{32}$  już w stanie  $t$ . W związku z tym algorytm nie musi być uruchamiany w stanie  $end\_state(\varphi, \theta)$ .

### 3.3 Złożoność pamięciowa i obliczeniowa wszystkich metod weryfikacji

#### 3.3.1 Złożoność pamięciowa

Złożoność pamięciowa formuły weryfikowanej za pomocą którego z powyższych algorytmów weryfikacji jest sumą wymagań pamięciowych grafu związanego z tą formułą, wymagań zmiennych pomocniczych (związanych z wierzchołkami *Previous* oraz zmienną *threshold*) oraz wartościami związanymi z historią. Jako, że powyższe algorytmy różnią się siłą ekspresji logiki temporalnej, na której są oparte, najlepiej rozpatrywać je osobno.

Aby obliczyć pesymistyczną złożoność pamięciową formuły  $\varphi$  należy przyjąć następujące założenia. Niech  $m$  - liczba operatorów temporalnych,  $n$  - liczba formuł atomowych niezwiązanych z żadnym operatorem temporalnym,  $k$  - długość rozpatrywanej historii,  $v$  - liczba walucji.

W przypadku formuł PTL bez operatorów metrycznych złożoność pamięciowa formuły jest sumą wartości wierzchołków typu *Previous* (to z kolei jest równe ilości operatorów temporalnych) oraz złożoności grafu formuły. Podgraf operatora temporalnego typu PTL składa się z czterech węzłów, a formuły atomowej maksymalnie z dwóch węzłów. W związku z tym złożoność grafu wynosi  $O(g_\varphi) = 4*m + 2*n$ , a całkowita złożoność pamięciowa wynosi:

$$O(\varphi) = v*m + 4m + 2n.$$

Sytuacja jest oczywiście bardziej skomplikowana, gdy występują operatory metryczne. Wówczas w przypadku algorytmu PTLVerification złożoność pamięciowa wynosi:  $O(\varphi) = 3*v*m + O(g_\varphi)$ , gdyż z każdym operatorem temporalnym związane są maksymalnie trzy zmienne pomocnicze.

W przypadku formuł typu MTL złożoność ta zwiększa się drastycznie o pewne wartości związane z historią obiektów. Jak wspomniano w podrozdziale 2.2.3 do przechowywania danych historycznych służą obiekty klasy *class-data*.

W związku z tym całkowita złożoność obliczeniowa jest sumą złożoności obliczeniowych grafu, zmiennych pomocniczych oraz danych historycznych.

Maksymalna ilość danych historycznych dla każdego operatora temporalnego oraz formuły atomowej niezwiązanej z żadnym operatorem wynosi  $2k$  (w przypadku, gdy musi być przechowywana cała historia), złożoność grafu wynosi:  $O(g_\varphi) = 8*m + 2*n + 2*op$ , gdzie  $op$  - liczba operatorów temporalnych z ograniczeniami metrycznymi.

Oprócz tego występują co najmniej trzy zmienne pomocnicze dla każdego operatora temporalnego. Wobec tego całkowita złożoność pamięciowa formuły  $\varphi$  wynosi:  $O(\varphi) = O(g_\varphi) + 2k*v*(m+n) + 3m$ .

Złożoność pamięciowa metody *StateVerification* jest sumą złożoności grafu i znaczników czasowych zebranych w *TimeMarksTable*. W przypadku braku operatorów metrycznych w tablicy *TimeMarksTable* zapisany jest stan początkowy i stan końcowy obliczeń oraz ewentualny stan i wartość zweryfikowanej formuły. W związku z tym złożoność pamięciowa takiej formuły wynosi:  $O(\varphi) = 3m*v + O(g)$ . Jeżeli występują operatory

metryczne, tablica *TimeMarksTable* zawiera dodatkowo znaczniki związane z operatorami metrycznymi. Złożoność pamięciowa zwiększa się o  $m*v$ .

| Metoda            | PTLVerification         |                          | MTLVerification                         | StateVerification         |                                 |
|-------------------|-------------------------|--------------------------|-----------------------------------------|---------------------------|---------------------------------|
|                   | brak met.               | op. op. met.             |                                         | brak op. met.             | op. met.                        |
| złożoność grafu   | $O(g) = 4m + 2n$        | $O(g) = 4m + 2n + 2op$   | $O(g) = 8m + 2n + 2op$                  | $O(g) = 8m + 2n$          | $O(g) = 8m + 2n + 2op$          |
| formuła $\varphi$ | $O(\varphi) = m + O(g)$ | $O(\varphi) = 3m + O(g)$ | $O(\varphi) = O(g) + 2k*v*(m+n) + 3m*v$ | $O(\varphi) = 3*v + O(g)$ | $O(\varphi) = 3*v + m*v + O(g)$ |

Tabela 2.1. Złożoność pamięciowa metod

### 3.3.2 Złożoność obliczeniowa

Tak jak w przypadku złożoności pamięciowej należy przyjąć następujące założenia. Niech  $m$  - liczba operatorów temporalnych,  $n$  - liczba formuł atomowych niezwiązanych z żadnym operatorem temporalnym,  $k$  - długość rozpatrywanej historii,  $v$  - waluacja.

Problem ten nie wydaje się zbyt trudny gdy rozpatrywane są formuły typu PTL bez ograniczeń metrycznych. Wtedy, praktycznie z każdym węzłem grafu związana jest jedna operacja. Wobec tego złożoność obliczeniowa formuły PTL w takim przypadku jest w przybliżeniu równa złożoności pamięciowej grafu. Wobec tego złożoność obliczeniowa formuły  $\varphi$  wynosi:  $C(\varphi) = v * (4*m + 2*n)$

Sytuacja jest oczywiście bardziej skomplikowana, jeżeli rozpatrywany jest algorytm *PTLVerification* z ograniczeniami metrycznymi. Złożoność obliczeniowa zwiększa się w stosunku do złożoności obliczeniowej bez operatorów metrycznych o końcową weryfikację na wierzchołkach typu *Past* oraz w przypadku, gdy w formule występuje wierzchołek *Since* na wierzchołkach typu *SMid*. Wobec tego maksymalna liczba obliczeń wynosi dla operatora metrycznego wynosi 10, co daje złożoność obliczeniową:  $C(\varphi) = v * 14m + 2n$ .

W powyższych przypadkach obliczenia wykonywane są rekurencyjnie od wierzchołka początkowego grafu do liści grafu. Ta sekwencja obliczeń robiona jest tylko raz w każdym stanie. Natomiast w przypadku formuły FTL lub MTL taka sekwencja obliczeń może być wykonywana kilka razy w pojedynczym stanie. Gdy w wyniku takiego postępowania nie ma jednoznacznie określonej wartości wierzchołka początkowego w tym stanie, to w następnym stanie powtarzana jest próba obliczenia wartości tego wierzchołka. Jeżeli długość historii wynosi  $k$ , to może być maksymalnie  $k$  takich prób. Jednak prędzej czy później wartości dla wierzchołka początkowego grafu dla każdego stanu zostaną określone. W związku z tym znów jest  $k$  sekwencji obliczeń. Złożoność obliczeniowa wynosi więc:  $C(\varphi) = v * (14m * k + 2k * (m+n))$ .

W przypadku metody *StateVerification* złożoność obliczeniowa jest tego samego rzędu co złożoność obliczeniowa metody *PTLVerification*.

|        |                 |                 |                   |
|--------|-----------------|-----------------|-------------------|
| Metoda | PTLVerification | MTLVerification | StateVerification |
|--------|-----------------|-----------------|-------------------|



|                        |                              |                                |                                            |                              |                               |
|------------------------|------------------------------|--------------------------------|--------------------------------------------|------------------------------|-------------------------------|
|                        |                              |                                |                                            |                              |                               |
|                        | brak op. met.                | op. met.                       |                                            | brak op. met.                | op. met.                      |
| Złożoność obliczeniowa | $C(\varphi) = v * (4m + 2n)$ | $C(\varphi) = v * (14m + 2n).$ | $C(\varphi) = v * (14m * k + 2k * (m+n)).$ | $C(\varphi) = v * (8m + 2n)$ | $C(\varphi) = v * (14m + 2n)$ |

Tabela 2.2. Złożoność pamięciowa metod

## 4. ZASTOSOWANIE METOD WERYFIKACJI FORMUŁ LOGIKI TEMPORALNEJ W BAZACH DANYCH

Rozdział III opisuje zastosowanie wprowadzonych w poprzednim rozdziale metod weryfikacji formuł logiki temporalnej w bazach danych. Rozważane są dwa modele baz danych: model nietemporalny (klasyczny) i model temporalny. Oczywiście istnieje jeszcze głębszy podział modeli baz danych (relacyjny, obiektowy, itd.). Przedstawiona w tym rozdziale baza danych jak również nałożone na nią więzy integralności oparte są wprawdzie na najbardziej podstawowym obiektowym modelu danych, lecz nie jest to założenie obligatoryjne, gdyż przedstawione w rozdziale II metody weryfikacji są niezależne od sposobu modelowania danych.

W teorii baz danych modelem temporalnym nazywany jest taki model, który umożliwia rejestrowanie i dostęp do danych z przeszłości, teraźniejszości oraz przyszłości. Jego przeciwieństwem jest model nietemporalny, który zakłada dostęp jedynie do informacji w chwili obecnej.

Koncepcja tego rozdziału jest następująca: W części pierwszej przedstawiono najbardziej podstawowe założenia obiektowego modelu danych, wprowadzono przykładową bazę danych oraz zdefiniowano przykładowe więzy integralności na bazie formuł temporalnych. Część druga rozdziału dotyczy temporalnych baz danych. Przedstawiono w niej ogólne założenia modelu temporalnej bazy danych opartego o czas rzeczywisty (*valid time*). Udowodniono jak przydatna może być weryfikacja oparta o logikę FTL.

### 4.1 Dynamiczne więzy integralności w nietemporalnych bazach danych

#### 4.1.1 Podstawowe definicje bazy danych

W niniejszej pracy przyjmowana jest koncepcja zakładająca, że baza danych składa się z migawek (ang. *snapshots*):

##### **Definicja 3.1**

##### Historia bazy

Historia bazy ma postać nieskończonej sekwencji stanów :

$$\underline{\sigma} = \langle \sigma_0, \sigma_1, \dots, \sigma_{act}, \dots \rangle$$

Stan  $\sigma_i$  oznacza stan bazy w punkcie  $i$ , stan  $\sigma_{act}$  - aktualny stan bazy. Przyjmuje się, że temporalna baza danych ma dostęp do każdego stanu w historii stanów a baza nietemporalna tylko do stanu aktualnego  $\sigma_{act}$ .



#### **Obiektowy model danych**

Jako że zastosowanie przedstawionych w niniejszej rozprawie metod weryfikacji nie jest ograniczone do określonego modelu danych, poniższe założenia obiektowego modelu danych są najbardziej podstawowe. Więcej na ten temat można znaleźć w [Kim 1996] oraz w [Subieta 1998b], które to prace opisują najnowsze trendy rozwoju obiektowych baz danych. Rozważania na temat najbardziej optymalnego modelu danych dla poniżej opisanej

przykładowej bazy są także poza ramami tej pracy. Problemy te są dość dobrze opisane w pracy [Snodgras 1997].

Główną cechą obiektowych baz danych jest występowanie obiektów.

### **Definicja 3.2**

#### Typy danych

Zakłada się, że baza posiada  $S = \{s_1, s_2, \dots\}$  typów obiektów. Każdy obiekt posiada identyfikator obiektu (*oid*) i należy do jakiegoś typu obiektu. Typ obiektu jest zapisywany *type(oid)*. Obiekty są tego samego typu, jeżeli mają takie same atrybuty. Tak więc każdy obiekt typu  $s_i$  posiada zbiór atrybutów:  $attribute = \{a_{i1}, \dots, a_{in}\}$  przy czym każdemu atrybutowi  $a_{ij}$  jest przypisana dziedzina wartości atrybutu oznaczona  $dom(a_{ij})$ . Wartość atrybutu  $a_{ij}$  obiektu  $o$  co jest oznaczone jako  $a_{ij}(o)$  określa funkcja  $a_{ij}(o) \in dom(a_{ij})$ . Oczywiście oprócz typu obiektów zdefiniowane są także podstawowe typy proste: boolean, integer, itd.

### **Definicja 3.3**

#### Obiekty

Obiekt ( $o$ ) jest parą składającą się z identyfikatora obiektu (*oid*) oraz zbioru par: atrybut, wartość  $\langle attribute, value \rangle$  oznaczonych jako *attribute\_values*. Definicja obiektu zapisywana jest w sposób następujący:

$o = \langle oid, attribute\_values \rangle$ , gdzie:  $attribute\_values = \{ \langle a_1, a_1(o) \rangle, \dots, \langle a_n, a_n(o) \rangle \}$  ■

Mając zdefiniowaną bazę jako uporządkowaną kolekcję migawek, można zdefiniować rodzaje transakcji związanych z obiektem w bazie. Do tego celu służy zdefiniowany poniżej predykat *exist*.

### **Definicja 3.4a**

#### Definicja exist

$exist(oid, k): \{ \exists s_i \in S, type(oid) \in s_i, \langle oid, attribute\_values \rangle \in \sigma_k \}$

Obiekt o identyfikatorze *oid* istnieje w stanie bazy o indeksie  $k$ , jeżeli określone są jego wartości atrybutów w tym stanie. ■

### **Definicja 3.4b**

#### Definicja insert

$insert(oid, k): (k = 0 \wedge exist(oid, \sigma_k)) \vee (k > 0 \wedge exist(oid, \sigma_k) \wedge \sim exist(oid, \sigma_{k-1}))$

Obiekt o identyfikatorze *oid* został wprowadzony do stanu bazy o indeksie  $k$ , jeżeli istnieje w tym stanie a nie istniał w stanie poprzednim. ■

### **Definicja 3.4c**

#### Definicja modify

$modify(oid, k): (\{ \langle attribute \rangle := \langle value \rangle \})$

Obiekt o identyfikatorze *oid* został zmodyfikowany w stanie o indeksie  $k$ , jeżeli wartość co najmniej jednego z jego atrybutów się zmieniła.

### Definicja 3.4d

#### Definicja delete

$delete(oid, k) : (k > 0 \wedge \sim exist(oid, k) \wedge exist(oid, k-1))$

Obiekt o identyfikatorze  $oid$  został usunięty ze stanu bazy o indeksie  $k$ , jeżeli nie istnieje w tym stanie a istniał w stanie poprzednim.

Wobec tego można zdefiniować przedział egzystencji obiektu  $lifespan(oid)$ :

### Definicja 3.5

#### Przedział egzystencji obiektu

$lifespan(oid) = \{ \langle \sigma_i, \sigma_j \rangle : insert(oid, i) \wedge delete(oid, j) \wedge \forall k \in \langle i+1, j-1 \rangle exist(oid, k) \}$

Tak więc przedział egzystencji obiektu o identyfikatorze  $oid$  wyznacza taka sekwencja stanów, że stan pierwszy jest stanem, w którym obiekt ten został wprowadzony do bazy, a stan ostatni jest stanem, w którym obiekt został usunięty z bazy, przy czym w stanach pośrednich obiekt zawsze istniał.

#### 4.1.2 Formuły temporalne - założenia wstępne

Przedstawione w rozdziale I formuły temporalne są instrumentem do opisu dynamicznych więzów integralności w bazach danych. Aby formuły te były weryfikowalne, muszą być podjęte pewne kroki służące sprzężeniu formuł temporalnych z bazą danych.

Zgodnie z założeniami wprowadzonymi w rozdziale I, formuły temporalne operują na stanach. Baza danych jest także sekwencją stanów. Wobec tego nic nie stoi na przeszkodzie, aby formuły temporalne były interpretowane na sekwencji stanów bazy.

### Definicja 3.6

#### Waluacja

Niech  $X(\varphi) = \{x_1, x_2, \dots, x_n\}$  oznacza zbiór zmiennych wolnych formuły  $\varphi$ . Z formułą  $\varphi$  związany jest zbiór typów obiektów:  $S_\varphi = \{s_1, s_2, \dots\}$ , gdyż wartościami zmiennych wolnych formuły  $\varphi$  mogą być wartości atrybutów obiektów o typach z  $S_\varphi$ . Waluacja  $\theta$  formuły  $\varphi$  to przyporządkowanie zmiennym wolnym z  $X$  wartości atrybutów obiektów o typach z  $S_\varphi$  na przestrzeni stanów z bazy. W związku z tym  $\theta_i$  jest przyporządkowaniem wartości zmiennym wolnym z  $X$  w stanie o indeksie  $i$ .

Jeżeli formuła  $\varphi$  jest spełniona w stanie o indeksie  $i$  przy waluacji  $\theta$ , to fakt ten jest zapisywany w sposób następujący:

$\varphi[\theta, i] = true$

Analogicznie jak dla pojedynczego obiektu można zdefiniować także operacje  $insert$ ,  $modify$  i  $delete$  dla zestawu obiektów biorących udział w waluacji  $\theta$ . Wówczas:  $insert(\theta, k)$  oznacza, że w stanie  $k$  został wstawiony ostatni obiekt, którego wartość wykorzystywana jest w waluacji  $\theta$ .  $modify(\theta, k)$  oznacza, że wartość któregoś z obiektów waluacji  $\theta$  została

zmieniona w stanie bazy o indeksie  $k$ , a  $delete(\theta, k)$ , że został usunięty pierwszy obiekt  $z$ , którego wartość wykorzystywana jest w waluacji  $\theta$ . W związku z tym przedział egzystencji waluacji jest zdefiniowany jako:

### Definicja 3.7

Przedział egzystencji waluacji

$$lifespan(\theta) = \{ \langle \sigma_i, \sigma_j \rangle : insert(\theta, i) \wedge delete(\theta, j) \wedge \forall k \in \langle i+1, j-1 \rangle exist(\theta, k) \}$$

Przedział egzystencji waluacji  $\theta$  jest ograniczony do takiej sekwencji stanów, że stan pierwszy jest stanem, w którym ostatni obiekt wykorzystywana w waluacji  $\theta$  został wprowadzony do bazy, a stan ostatni jest stanem, w którym pierwszy obiekt waluacji  $\theta$  został usunięty z bazy.

Weryfikacja formuły temporalnej  $\phi$  działającej na waluacji  $\theta$  ma sens jedynie w przedziale egzystencji waluacji  $lifespan(\theta)$ .

Poniższy przykład przedstawia historię dwóch obiektów składającą się z sześciu stanów. Na tę historię nałożona jest formuła PTL.

### Przykład 3.1

Dana jest historia bazy:

$$\underline{\sigma} = \langle \sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6 \rangle$$

Baza składa się z trzech obiektów:  $o_1(A,B)$ ,  $o_2(C,D,E,F)$ ,  $o_3(G,H)$  oraz  $o(o_1, o_2, o_3)$ . Obiekt  $o_1$  ma dwa atrybuty  $(A, B)$ , obiekt  $o_2$  cztery atrybuty:  $(C,D,E,F)$ , obiekt  $o_3$  dwa atrybuty:  $(G,H)$ , a obiekt  $o$  trzy atrybuty, których wartościami są obiekty  $o_1, o_2, o_3$ .

Poniższa tabela przedstawia wartości atrybutów obiektów w stanach  $\underline{\sigma}$  (waluację  $\theta$ ).

Tab. 3.1. Wartości obiektów  $o_1, o_2, o_3$  w całej historii bazy

|            | $o_1$ |    | $o_2$ |    |   |   | $o_3$ |    |
|------------|-------|----|-------|----|---|---|-------|----|
| Atr.       | A     | B  | C     | D  | E | F | G     | H  |
| $\sigma_0$ | 5     | 10 | 23    | 11 | 2 | 5 | 10    | 9  |
| $\sigma_1$ | 6     | 11 | 24    | 12 | 7 | 8 | 11    | 12 |
| $\sigma_2$ | 4     | 14 | 21    | 14 | 4 | 3 | 34    | 35 |
| $\sigma_3$ | 6     | 18 | 22    | 15 | 6 | 1 | 37    | 40 |
| $\sigma_4$ | 8     | 15 | 23    | 18 | 2 | 5 | 12    | 13 |
| $\sigma_5$ | 5     | 13 | 27    | 13 | 5 | 6 | 15    | 17 |
| $\sigma_6$ | 4     | 12 | 21    | 16 | 9 | 9 | 12    | 17 |

Zdarza się, że nie jest potrzebne zachowywanie całej historii obiektów, ale narzucony jest z góry warunek, który musi być zweryfikowany w każdym stanie bazy. Niech takim warunkiem będzie formuła:

$$\varphi = ((o_1(A) > 4 \wedge o_2(C) < 30) \vee (o_1(B) \geq 10 \wedge o_2(D) > 10)) \text{ since } (o_3(G) \geq o_3(H))$$

Niech:  $\varphi_1 = (o_1(A) > 4 \wedge o_2(C) < 30) \vee (o_1(B) \geq 10 \wedge o_2(D) > 10)$  oraz  $\varphi_2 = (o_3(G) \geq o_3(H))$

Aby zweryfikować formułę  $\varphi$  za pomocą dowolnego języka zapytań, należałoby w każdym stanie uruchomić zapytanie, które weryfikowałoby podformuły  $\varphi_1$  oraz  $\varphi_2$  w każdym stanie historii. Pociągałoby to za sobą konieczność przechowywania całej historii oraz całych obiektów. Metoda przedstawiona w tej pracy nie musi przechowywać całej historii bazy. Z wcześniejszych rozważań wiadomo, że zachodzi tautologia:

$$\varphi_1 \text{ since } \varphi_2 \Leftrightarrow \varphi_1 \wedge (\varphi_2 \vee \text{previous}(\varphi_1 \text{ since } \varphi_2))$$

W związku z tym aby zweryfikować formułę:  $\varphi = \varphi_1 \text{ since } \varphi_2$  w stanie  $\sigma_i$ , należy zweryfikować podformuły  $\varphi_1$  i  $\varphi_2$  wartości w tym stanie oraz mieć dostęp do wyniku weryfikacji z poprzedniego stanu. Wobec tego w stanie nie trzeba rozpatrywać całej historii tylko stan aktualny, co znacznie zmniejsza złożoność obliczeniową takiej weryfikacji. Co więcej, nie trzeba przechowywać całej historii bazy, tylko pojedynczą wartość (*true* lub *false*) dla każdej walucji. Wartość ta zapisana jest w kolumnie *Add*.

Tak więc wartości obiektów bazy w stanie  $\sigma_i$  wyglądają w sposób następujący:

Tab. 3.2. Wartości obiektów i zmiennych pomocniczych w stanie:  $\sigma_5$ .

|            | o <sub>1</sub> |    | o <sub>2</sub> |    |   |   | o <sub>3</sub> |    | Add      |
|------------|----------------|----|----------------|----|---|---|----------------|----|----------|
| Atrybut    | A              | B  | C              | D  | E | F | G              | H  | previous |
| $\sigma_5$ | 5              | 13 | 27             | 13 | 5 | 6 | 15             | 17 | True     |

Na podstawie tych wartości można obliczyć wartość formuły  $\varphi$  w stanie  $\sigma_5$ . Na podstawie można stwierdzić, że formuła w tym stanie przyjmuje wartość *true*.

Sytuacja jest nieznacznie bardziej skomplikowana, gdy użyty zostanie operator metryczny przy formule  $\varphi$ . Niech  $\varphi' = \varphi_1 \text{ since}_{\leq 3} \varphi_2$ . Wówczas weryfikując taką formułę należy zapamiętać oprócz wartości z poprzedniego stanu, także zmienną *threshold* oznaczającą największy indeks stanu, w którym była spełniona podformuła  $\varphi_2$ . Wobec tego, zamiast pamiętać całą historię bazy, należy rejestrować jedynie liczbę całkowitą *threshold* związaną z operatorem temporalnym oraz zmienną *previous*. W przypadku operatora *since* liczba w zmiennej *threshold* jest czasem związanym z ostatnim stanem, w którym podformuła  $\varphi_2$  przyjmuje wartość *true*.

Tab. 3.3. Wartości obiektów i zmiennych pomocniczych w stanie:  $\sigma_5$

|            | o <sub>1</sub> |    | o <sub>2</sub> |    |   |   | o <sub>3</sub> |    | Add      |           |
|------------|----------------|----|----------------|----|---|---|----------------|----|----------|-----------|
| Atrybut    | A              | B  | C              | D  | E | F | G              | H  | previous | threshold |
| $\sigma_5$ | 5              | 13 | 27             | 13 | 5 | 6 | 15             | 17 | true     | 0         |

W tym przypadku weryfikacja polega na zrealizowaniu dwóch kroków:

- wykorzystanie tautologii
- końcowa weryfikacja polegająca na odjęciu czasu związanego z rozpatrywaniem

stanem bazy od czasu zapisanego w *threshold* i porównanie wyniku z operatorem metrycznym przy *since*.

O ile pierwszy krok daje w efekcie wartość *true*, to w kroku drugim otrzymuje się, że:  $5 - 0 \leq 3$ , co nie jest prawdą. W związku z tym wartość formuły  $\varphi'$  w tym stanie jest *false*.

#### 4.1.3 Przykładowy schemat bazy

Rozważmy przykładową bazę opisującą działalność zakładu usługowego. W bazie tej zbierane są informacje o oddziałach firmy, ich pracownikach, o klientach, świadczonych przez firmę usługach oraz o projektach prowadzonych przez poszczególne oddziały firmy. Ponieważ przykład ten ma uzmysłowić jedynie specyfikację dynamicznych więzów integralności, podanych zostanie jedynie kilka struktur danych potrzebnych do tego celu. Należy zaznaczyć, że modelowanie poniższych danych oparte jest o model obiektowy, chociaż nie jest to założenie obligatoryjne.

##### Składniki bazy:

Typy obiektów:

$S = \{\text{Oddział, Pracownik, Klient, Usługa, Projekt, Stanowisko}\}$

Zmienne obiektowe:

$Val = \{o \text{ Oddział, } p \text{ Pracownik, } k \text{ Klient, } u \text{ Usługa, } pr \text{ Projekt, } s \text{ Stanowisko}\}$

Obiekty:

Oddział:      oid\_od1, oid\_od2, ....  
Pracownik:    oid\_pr1, oid\_pr2, ...  
Klient:        oid\_kl1, oid\_kl2, ...  
Usługa:        oid\_usl1, oid\_usl2, ...  
Projekt:        oid\_proj1, oid\_proj2, ...  
Stanowisko:    oid\_stan1, oid\_stan2, ....

Typy danych:

Pracownik:    {zarobki: integer;  
                 data\_urodzenia: data;  
                 adres: string;  
                 stanowisko: string;  
                 dział: string;  
                 .....  
                 }

Usługa:        {zapłata: integer;  
                 zamówienie: boolean;  
                 cena: integer;  
                 realizacja: boolean;  
                 }

Wykonanie\_uslugi:  
                 {klient: type(Klient);  
                 usługa: type(Usługa);

pracownik : set{type(Pracownik)}  
}

#### 4.1.4 Przykłady dynamicznych więzów integralności

Na tak zdefiniowany schemat bazy danych można nałożyć więzy integralności będące formułami temporalnymi. W przykładzie 3.2 podano kilka formuł temporalnych wraz z ich znaczeniem

##### Przykład 3.2

- (1)  $\forall u \in \text{type}(\text{Usługa}), u.\text{zamówienie} = \text{true} \Rightarrow \text{sometime-future}_{<10} (u.\text{zapłata} = \text{true} \wedge \text{sometime-future}_{<5} u.\text{realizacja} = \text{true})$

Jeżeli klient złożył zamówienie, to musi zapłacić w ciągu 10 dni, a firma realizuje zamówienie w ciągu 5 dni po zapłacie.

- (2)  $\forall p \in \text{type}(\text{Pracownik}), \neg \text{sometime-past}_{<40} (\text{always-future}_4 p.\text{stanowisko} = \text{'szef'} \wedge p.\text{dział} = \text{'od1'}) \Rightarrow \neg \text{sometime-future} (p.\text{stanowisko} = \text{'szef'} \wedge p.\text{dział} = \text{'od1'})$

Jeżeli pracownik w ostatnich 40 miesiącach nie był szefem oddziału od1 przez co najmniej 4 miesiące to nie może w przyszłości być szefem oddziału od2.

- (3)  $u.\text{zamowienie} \wedge u.\text{cena} = s \Rightarrow \neg (u.\text{cena} > s) \text{ until } u.\text{zapłata}$

Jeżeli klient złożył zamówienie na usługę, to cena tej usługi nie może się zwiększyć do momentu jej realizacji.



Przykład 3.3 ukazuje korzyści jakie przynosi weryfikacja formuły PTL za pomocą metody *PTLVerification* w stosunku do użycia zapytania ad-hoc.

##### Przykład 3.3

Dana niech będzie formuła  $\varphi$ , która jest więzem integralności nałożonym na bazę:

$$\varphi = \neg(\neg(u.\text{cena} < 200 \text{ PLN}) \text{ since }_{<6} (u.\text{begin\_state}))$$

Formuła oznacza, że cena usługi  $u$  nie może być większa od 200 PLN w przeciągu pierwszych 6 miesięcy od powstania usługi. Załóżmy, że cena usługi zmieniała się co miesiąc. Aby zastosować zapytanie ad-hoc, należałoby:

- 1) wykonywać to zapytanie co miesiąc;
- 2) posiadać temporalną bazę danych przechowującą cenę usługi, jaka była każdego miesiąca.

Stosowanie algorytmu PTL niesie za sobą następujące korzyści:

- 1) zamiast wykonywać całe zapytanie o spełnienie formuły, weryfikowana jest jedynie co miesiąc jej statyczna część:  $(u.\text{cena} < 200 \text{ PLN})$
- 2) ceny usług w każdym miesiącu nie muszą być przechowywane, ale jedynie czas powstania usługi oraz wartość wierzchołka typu previous.



3) po 6 miesiącach wartość formuły jest na pewno znana, a więc nie musi być weryfikowana. ■

Przykład 3.4 wprowadza przykładową formułę FTL oraz rozważa i porównuje dwa sposoby weryfikacji tej formuły.

#### Przykład 3.4

Dana niech będzie formuła:

$\forall u \in \text{type}(\text{Usługa}), u.\text{zamówienie} = \text{true} \Rightarrow \text{sometime-future}_{<10} (u.\text{zapłata} = \text{true} \wedge \text{sometime-future}_{<5} u.\text{realizacja} = \text{true})$

Formuła ta ma następujące znaczenie:

*Jeżeli klient złożył zamówienie, to musi zapłacić w ciągu 10 dni, a firma realizuje zamówienie w ciągu 5 dni po zapłacie.*

Weryfikację tej formuły można przeprowadzić na dwa sposoby. Sposób pierwszy polega na zastąpieniu formuły przez formułę:

$\varphi' \equiv \neg (u.\text{zamówienie} = \text{true}) \vee \text{sometime-future}_{<10} (u.\text{zapłata} = \text{true} \wedge \text{sometime-future}_{<5} u.\text{realizacja} = \text{true})$

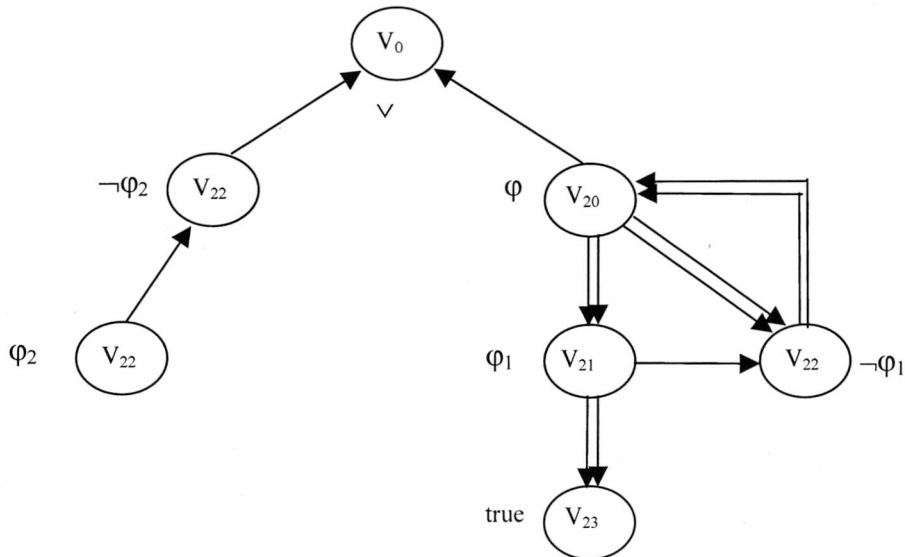
Niech:

$\varphi_1 \equiv u.\text{zapłata} = \text{true} \wedge \text{sometime-future}_{<5} u.\text{realizacja} = \text{true}$

$\varphi_2 \equiv u.\text{zamówienie} = \text{true}$

$\varphi_3 \equiv \text{sometime-future}_{<10} \varphi_1$

Wówczas do weryfikacji może być wykorzystany algorytm *FTLVerification*, a graf FTL tej formuły wygląda w sposób następujący:



Rys. 3.4 . Graf formuły  $\varphi'$

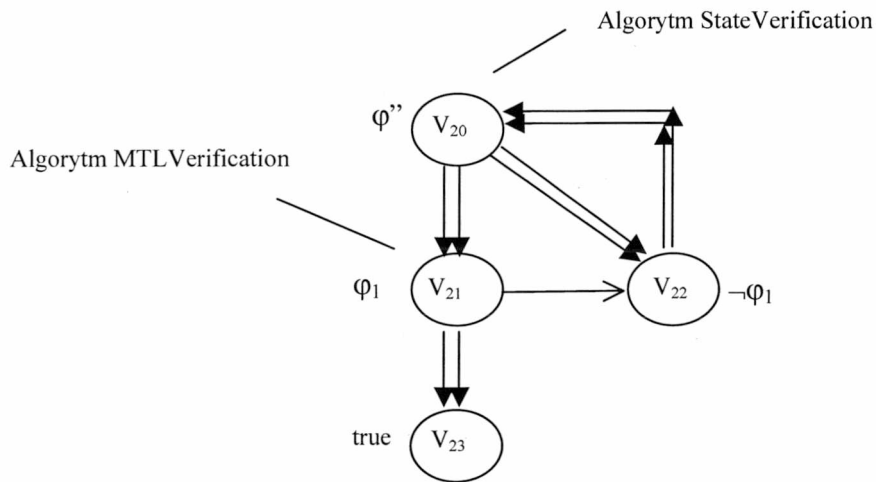
Drugi sposób polega na zastosowaniu dwóch metod weryfikacji jednocześnie. Skoro formuła  $\varphi$  ma w swoim ciele implikację, to wystarczy zweryfikować podformułę:

$$\varphi'' \equiv \text{sometime-future}_{<10} (u.\text{zapłata} = \text{true} \wedge \text{sometime-future}_{<5} u.\text{realizacja} = \text{true})$$

w chwili czasu określonej przez zajście zdarzenia:  $u.\text{zamówienie} = \text{true}$

Tak więc algorytm rozpocznie swoje działanie w chwili  $t$  oznaczającym czas zajścia zdarzenia  $u.\text{zamówienie} = \text{true}$ . Wobec tego nasuwa się przypuszczenie, że można zastosować algorytm uproszczony: *VerificationState*. Niestety, by algorytm ten może być użyty wtedy, gdy rozpatrywana formuła nie jest formułą złożoną. W naszym przypadku tak jest. Aby mogła być obliczona wartość formuły głównej w punkcie  $t$ , wartość podformuły  $\varphi_1$  musi zostać rozpatrzona w każdym punkcie przedziału  $\langle t, t+10 \rangle$ . ■

Wobec tego najlepszym wyjściem jest złożenie dwóch algorytmów. Obrazuje to rysunek 3.5:



Rys. 3.5 . Graf formuły  $\varphi''$

Algorytm *StateVerification* jest uruchamiany dla wierzchołka związanego z formułą  $\varphi''$ , gdyż weryfikacja wymaga stwierdzenia prawdziwości formuły jedynie w stanie  $t$ . Aby to jednak stwierdzić, należy zweryfikować podformułę  $\varphi_1$  w każdym punkcie  $\langle t, t+10 \rangle$ . Metoda jest lepsza, gdyż graf FTL jest znacznie mniej skomplikowany przez co złożoność obliczeniowa jest mniejsza. Oprócz tego w metodzie tej jest jasno zdefiniowany początek i koniec obliczeń.

### Przykład 3.6

Przykładem formuły typu MTL może być formuła zapisana w sposób następujący:

$$\forall p \in \text{type}(\text{Pracownik}), (\text{modify}(p.\text{oddzial}) \Rightarrow \sim(p.\text{stanowisko} = \text{'programista'}) \text{ until}_{<5} p.\text{stanowisko} = \text{'projektant'}) \vee \text{always\_past}_{<6} (p.\text{stanowisko} = \text{'projektant'})$$

Znaczenie tej formuły jest następujące:

*Jeżeli pracownik zmienił oddział, to nie mógł być programistą przez pięć miesięcy przed objęciem stanowiska „projektant” lub przez ostatnie co najmniej 6 miesięcy zajmował stanowisko projektanta.*

Weryfikacja tej formuły odbywa się z wykorzystaniem grafu MTG. ■

#### 4.1.5 Weryfikacja temporalnych więzów integralności

W związku z występowaniem różnych rodzajów więzów integralności w bazach danych, rozważania nad weryfikacją więzów integralności wygodnie jest podzielić na dwie części. W pierwszej rozważane będą formuły logiki temporalnej czasu przeszłego (PTL), a w drugiej formuły logiki mieszanej (MTL).

##### Weryfikacja formuł PTL

Weryfikację formuł PTL wygodnie jest podzielić na dwa przypadki. Pierwszy przypadek rozpatruje brak operatorów metrycznych, drugi natomiast obecność co najmniej jednego operatora temporalnego z operatorem metrycznym. W tym przypadku algorytm weryfikacji formuły  $\varphi$ , gdzie  $\theta$  waluacja, wygląda w sposób następujący:

```

procedure PTLIntegrityMonitor(integer k)
for each  $\varphi$  do
for each  $\theta$  do
begin
    // Początek weryfikacji. Nadawane są wartości początkowe wierzchołków Prev
    if insert( $\theta$ , k) then
        InitValues ( $\varphi$ ,  $\theta$ , k);
        PTLVerification( $\varphi$ ,  $\theta$ , k);
    endif;

    // Koniec weryfikacji.
    if delete( $\theta$ , k) then PTLVerification ( $\varphi$ ,  $\theta$ , k);
    endif;

    if modify( $\theta$ , k) then
        unlock ( $\varphi$ ,  $\theta$ ) ;
        PTLVerification( $\varphi$ ,  $\theta$ , k);
    else if change_state(k-1, k) and verified( $\varphi$ ,  $\theta$ ) = unknown
    then PTLVerification ( $\varphi$ ,  $\theta$ , k) ;
    endif;
    /*Jeżeli formuła nie jest spełniona w punkcie k, a nałożony jest na bazę warunek o
    spełnieniu formuły w tym punkcie, wówczas powinna nastąpić jakaś akcja
    zapobiegająca załamaniu więzów integralności np. rollback*/

    if  $\varphi(\theta$ , k) = false then Action
    endif
endfor;

```

```

endfor;
endproc;

```

Wprowadzona w rozdziale II metoda *PTLVerification* jest uruchamiana jako odpowiedź na zdarzenia *delete*, *insert*, *modify* oraz *change\_state*. Zdarzenie *change\_state* zachodzi jedynie wtedy, gdy w weryfikowanej formule występuje operator PTL z ograniczeniem metrycznym. W przypadku zdarzenia *insert* proces weryfikacji rozpoczyna się dla danej waluacji. W związku z tym wywoływana jest procedura *InitValues*, w której nadawane są wartości początkowe wierzchołków *Prev*.

Komentarza wymaga jeszcze komenda *Action*. Komenda ta jest reakcją na fakt niespełnienia formuły w rozpatrywanym punkcie bazy. To, jaka ta reakcja będzie zależy w dużym stopniu od założeń stawianych bazie danych. Jeżeli zakłada się, że formuła powinna być spełniona w każdym punkcie bazy czyli, że jej niespełnienie narusza integralność bazy, wówczas są dwie przyczyny naruszenia tej integralności: zachodząca transakcja bądź upływ czasu. Te dwa czynniki mają bowiem wpływ na wartość formuły temporalnej. Jeżeli załamanie więzów integralności zostało spowodowane transakcją, wówczas poprzez cofnięcie transakcji (*rollback*) można „naprawić” integralność bazy. Sytuacja wygląda gorzej, jeżeli niespełnienie rozpatrywanej formuły spowodowane jest przez upływ czasu. W tym przypadku baza jest „nienaprawialna”. Wówczas jedyne co można zrobić, to ostrzec, że np. integralność może być załamana w następnym stanie bazy.

Action może być także reakcją na spełnienie formuły. Wówczas oczywiście linia kodu:  
*if  $\varphi(\theta, k) = false$  then Action* ; powinna być zastąpiona przez:

*if  $\varphi(\theta, k) = true$  then Action*.

Może być to zastosowane przez wykorzystaniu temporalnych triggerów. Wtedy spełnienie weryfikowanej formuły może spowodować „odpalenie” triggera.

Tak więc sposób reakcji na spełnienie, bądź niespełnienie weryfikowanej formuły zależy od założenia, jak powinna się zachowywać baza.

## Weryfikacja formuł FTL

Weryfikacja formuł FTL wygląda w sposób następujący:

```

procedure FTLIntegrityMonitor(integer k)
for each  $\varphi$  do
for each  $\theta$  do
begin
    // Początek weryfikacji. Nadawane są wartości początkowe wierzchołków Prev
    if insert( $\theta, k$ ) then
        InitValues ( $\varphi, \theta, k$ );
        MTLVerification( $\varphi, \theta, k$ );
    endif;

    // Koniec weryfikacji
    if delete( $\theta$ ),  $k$  then
        end_state( $\varphi, \theta$ ) :=  $k$ ;
        MTLVerification ( $\varphi, \theta, k$ );
    endif;

```

```

if modify( $\theta, k$ ) then
    unlock( $\varphi, \theta$ );
    MTLVerification( $\varphi, \theta, k$ );
else if change_state( $k-1, k$ ) and verified( $\varphi, \theta$ ) = unknown
then MTLVerification( $\varphi, \theta, k$ );
endif;
/*Jeżeli formuła nie jest spełniona w punkcie k, a nałożony jest na bazę warunek o
spełnieniu formuły w tym punkcie, wówczas powinna nastąpić jakaś akcja
zapobiegająca załamaniu więzów integralności np. rollback*/

if  $\varphi(\theta, k) = \text{false}$  then Action
endif;

endfor;
endfor;
endproc;

```

Weryfikacja formuł FTL jest podobna do wyspecyfikowanej wcześniej weryfikacji formuł PTL. Jediną różnicą jest przypisanie  $k$  jako końca weryfikacji (*end\_state*) w przypadku zajścia zdarzenia *delete*, gdyż wartość *end\_state* jest wymagana podczas weryfikacji formuł MTL.

## 4.2 Więzy integralności w temporalnych bazach danych

Podrozdział ten rozpatruje weryfikację dynamicznych więzów integralności w temporalnych bazach danych opartych o czas rzeczywisty (*valid time*). Warto przypomnieć, że baza taka charakteryzuje się tym, że zachowuje nie tylko wartości atrybutów obiektów w chwili aktualnej, ale także wartości przeszłe, a nawet przypuszczalne wartości przyszłe tych atrybutów. Co więcej, bazę taką można uaktualniać. Uaktualnianie bazy temporalnej opartej o czas rzeczywisty polega nie tylko na uaktualnianiu wartości w stanie aktualnym, ale również w przeszłym i przyszłym.

Przy tak przedstawionych założeniach nasuwa się pytanie, czy warto weryfikować dynamiczne więzy integralności w temporalnych bazach danych. Istnieje bowiem kilka temporalnych języków zapytań takich : TSQL [Sarda 1990], TSQL2 [Snodgrass 1996], TQuel [Snodgrass 1987], które mogłyby być zastosowane do tego celu. Okazuje się jednak, że formułowanie więzów integralności jako zapytań w którymś z powyższych języków może okazać się skomplikowane a ich weryfikacja bardzo kosztowna. Zastosowanie więzów integralności opartych na logice temporalnej znacznie zmniejsza złożoność obliczeniową w stosunku do kierowania bezpośrednich zapytań przez wykorzenienie redundantnych obliczeń.

Praca [Gertz i Lipect 1995] analizuje wykorzystanie grafów przejść utworzonych z formuł czasu przyszłego bez operatorów metrycznych do badania dynamicznych więzów integralności w temporalnych bazach danych.

W niniejszej pracy wykorzystano wprowadzony w rozdziale II algorytm *FTLStateVerification* dopuszczający stosowanie logiki FTL z operatorami metrycznymi. Nie ma bowiem potrzeby badać wykorzystania formuł typu MTL z uwagi na większą złożoność obliczeniową algorytmu *MTLVerification* oraz z powodu potrzeby przechowywania dodatkowych danych weryfikacji (rozdział II) w bazie, która zachowuje dane z przeszłości.

Tak więc stosowanie algorytmu *FTLStateVerification* ze względu na jego niską złożoność obliczeniową jest zupełnie wystarczające.

Jak wiadomo temporalna baza danych charakteryzuje się tym, że transakcje zachodzące na tej bazie mogą oddziaływać na wszystkie stany historii bazy (nie tylko na stan aktualny). W związku z tym, przy każdej zmianie obiektu w przeszłości, zmianie może ulec także wartość już wcześniej zweryfikowanej formuły. Przedstawiona w tej pracy weryfikacja więzów integralności w temporalnej bazie danych polega na odnajdywaniu i omijaniu tych stanów bazy w procesie weryfikacji, które nie mają wpływu na wartość weryfikowanej formuły.

#### 4.2.1 Podstawowe założenia temporalnego modelu danych

Podstawowe założenia dotyczące rozpatrywanego modelu danych są takie jak w pracy [Gertz i Lipeck 1995].

Model czasu w tym modelu jest dyskretny izomorficzny do liczb naturalnych. W związku z tym przedział czasowy, oznaczony przez  $[s, e]$ , jest zbiorem kolejnych punktów czasu określonych przez punkty graniczne  $s$  i  $e$ , przy czym  $s$  jest wartością początkowego punktu czasu a  $e$  wartością punktu końcowego. Przyjmujemy oznaczenie  $t_0$  dla punktu czasu, w którym temporalna baza rozpoczyna swoją egzystencję, a  $t_{now}$  jest punktem czasu chwili obecnej.

Stan bazy  $\sigma_t$  oznacza migawkę bazy danych w punkcie  $t$ . Migawka jest złożeniem wszystkich atrybutów obiektów wraz z przypisanymi do nich wartościami prawdziwymi w tym punkcie.

Jeżeli chodzi o modelowanie danych, to praca ta nie ogranicza się do jakiegoś konkretnego modelu. Opis różnych modeli temporalnych można znaleźć w pracy: [Snodgras 1997]. Przyjmuje się natomiast najprostszą reprezentację obiektową, zakładającą, że obiekt składa się z atrybutów, do których przypisane są wartości z różnych przedziałów czasowych. Tak więc każdy atrybut obiektu ma na przestrzeni życia obiektu kilka różnych wartości, do których przypisany jest przedział czasu, w którym te wartości obowiązują.

Zakłada się, że baza posiada  $S$  typów obiektów. Obiekty są tego samego typu, jeżeli mają takie same atrybuty. Tak więc każdy obiekt typu  $s_i$  posiada zbiór atrybutów  $\{a_{i1}, \dots, a_{im}\}$  przy czym każdemu atrybutowi  $a_{ij}$  jest przypisana dziedzina wartości atrybutu oznaczona  $dom(a_{ij})$ . Każdemu obiektowi przypisany jest czas życia obiektu oznaczony  $t(o) \subseteq [t_0, t]$ , gdzie  $t$  oznacza sztuczny punkt czasu końca bazy. Wartość atrybutu  $a_j$  obiektu  $o$  co jest oznaczone jako  $a_j(o)$  określa funkcja  $a_j(o): t(o) \rightarrow dom(a_j)$ . Dodatkowo każdy obiekt zaopatrzony jest w identyfikator *oid*.

Temporalna modyfikacja obiektów rozpatrywanego modelu danych różni się od tej wprowadzonej w podrozdziale 4.1 Jest zdefiniowana w sposób następujący:

##### Definicja 3.10a

##### Definicja insert

- *insert* <object\_type>(<attribute\_values>[ $s, e$ ])

Nowy obiekt typu *object\_type* jest wstawiany do bazy z wartościami atrybutów *attribute\_values*, które są spełnione w przedziale  $[s, e]$ . Gdy przedział ten nie jest wyspecyfikowany przyjmuje się  $[t_{now}, t]$ .

##### Definicja 3.10b

### Definicja modify

- $modify \langle oid \rangle (\{ \langle attribute \rangle := \langle value \rangle \} [s, e])$

Nowe wartości są przypisane do atrybutów obiektu. Każdej starej wartości atrybutu przypisywany jest czas końcowy  $s-1$  (wartość nie jest usuwana). Gdy przedział  $[s, e]$  nie jest wyspecyfikowany przyjmuje się  $[t_{now}, t]$ .

### **Definicja 3.10c**

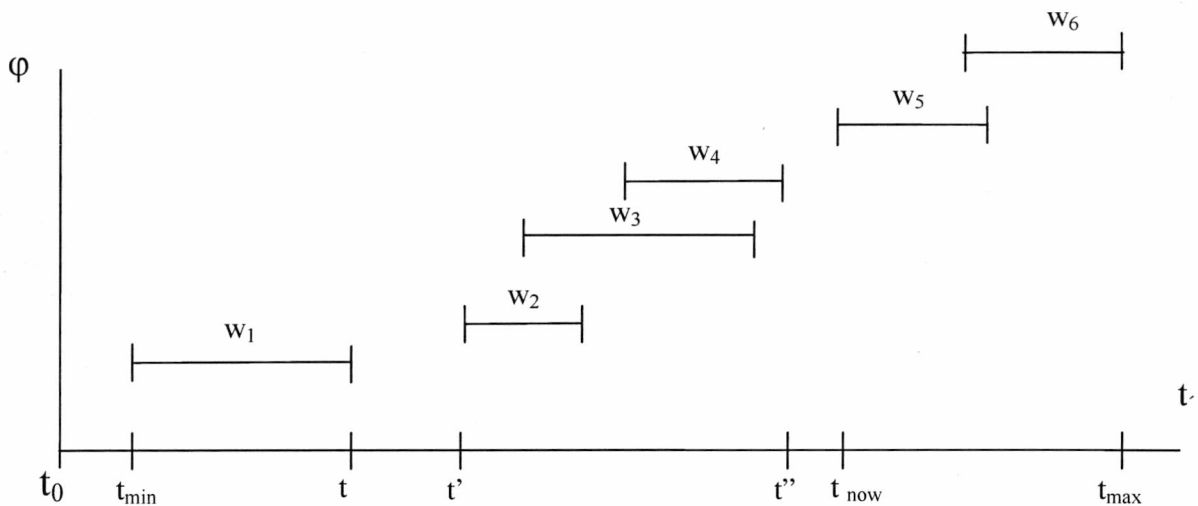
#### Definicja delete

- $delete \langle oid \rangle [(s, e)]$

Obiekt jest modyfikowany w ten sposób, że jego atrybuty są ustawiane jako *null* w wyspecyfikowanym przedziale. Jeżeli czas życia obiektu  $t(o) = [s', e']$  oraz  $s < s' < e$  lub  $s < e' < e$  to czas życia obiektu jest skrócony do  $[e+1, e']$  lub  $[s', s-1]$ . Gdy przedział  $[s, e]$  nie jest wyspecyfikowany przyjmuje się, że czas życia obiektu kończy się w  $t_{now}-1$ .

### **Definicja 3.11**

Transakcją na temporalnej bazie danych jest skończona sekwencja  $\Delta$  temporalnych modyfikacji  $\{w_1, \dots, w_k\}$ . W wyniku transakcji  $\Delta$  powstaje skończona liczba przedziałów modyfikacji, które zawierają te punkty, w których wartości atrybutów obiektów zostały zmienione w wyniku działania transakcji. Sytuację tę może zobrazować rysunek 3.4.



Rys. 3.4. Przedziały działania transakcji

W wyniku działania sekwencji transakcji  $\Delta_1 = \{w_1, \dots, w_6\}$  powstają trzy przedziały modyfikacji:  $\langle t_{min}, t \rangle$ ,  $\langle t', t'' \rangle$ ,  $\langle t_{now}, t_{max} \rangle$ . Punkty czasu  $t_{min}$  oznacza początek pierwszego przedziału modyfikacji, a punkt  $t_{max}$  koniec ostatniego przedziału modyfikacji. Przedziały te tworzą zakres modyfikacji.

#### 4.2.2 Weryfikacja więzów integralności w temporalnych bazach danych

Mając tak zdefiniowany model danych można zastanawiać się nad weryfikacją dynamicznych więzów integralności w temporalnych bazach danych. Jak wspomniano w poprzednim podrozdziale jako formuły więzów integralności rozpatrzone zostaną formuły logiki FTL. Zaproponowano dwie metody weryfikujące dynamiczne więzy integralności w temporalnym modelu bazy opartym o czas rzeczywisty (*valid time*): *SimpleTemporalVerification* oraz *ComplexTemporalVerification*. W pierwszej z nich można stosować formuły proste typu FTL, a druga umożliwia stosowanie także formuł złożonych. Zdecydowano się na rozdzielanie tych dwóch metod, gdyż metoda *SimpleTemporalVerification* jest metodą znacznie mniej skomplikowaną, a oprócz tego występowanie formuł prostych w praktycznych zastosowaniach jest znacznie częstsze niż występowanie formuł złożonych.

##### 4.2.2.1 Weryfikacja formuł prostych FTL

Ideę weryfikacji formuł FTL w temporalnych bazach danych najlepiej przedstawić na przykładzie.

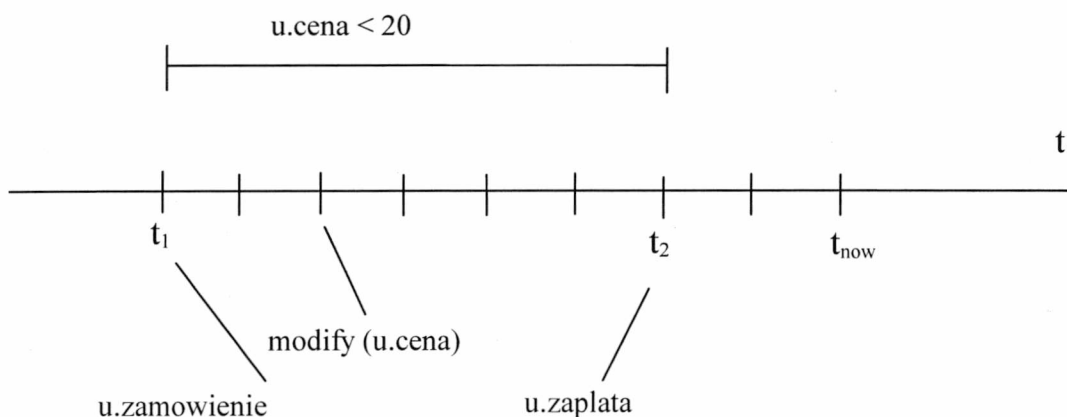
##### Przykład 3.7

$\varphi \equiv u.\text{zamowienie} \Rightarrow u.\text{cena} < 20 \text{ until } u.\text{zapłata}$

Formuła ma następujące znaczenie:

Od momentu zamowienia usługi *u* przez klienta *k* do momentu zapłaty cena usługi musi być mniejsza od 20.

Do weryfikacji można użyć algorytmu *FTLStateVerification*. Algorytm ten rozpocznie swoje działanie w momencie wprowadzenia nowego zamówienia. Rysunek 3.5 ilustruje zachodzące w temporalnej bazie zdarzenia.



Rys. 3.5. Zdarzenia na obiektach w bazie



W punkcie  $t_1$  zaszło zdarzenie zamówienia usługi. Weryfikacja formuły  $\varphi$  dała wartość *true*, gdyż *cena* była mniejsza od 20 od momentu zamówienia ( $t_1$ ) do momentu zapłaty ( $t_2$ ). Algorytm zakłada zapisanie tego stanu wraz z wartością weryfikacji (w tym wypadku *true*). Może się jednak okazać, że w wyniku błędu w zapisie transakcji cena w punkcie  $t_1+2$  musi zostać na nowo zapisana (np.  $u.cena=15$  zamiast  $u.cena=10$ ). W związku z tym poprawność formuły musi być na nowo zweryfikowana. Używając któregoś z klasycznych języków zapytań należałoby postawić pytanie w punkcie  $t_1$ , co pociągałoby za sobą potrzebę analizy całej sekwencji stanów. W rozpatrywanym przypadku należałoby wywołać algorytm tylko w stanie zmodyfikowanym, a więc  $t_1+2$ . Stan ten jest mniejszy od stanu zapisanego, a więc wiadomo, że wartość formuły w tym stanie przed transakcją wynosiła *unknown*. Po transakcji wartość nie zmienia się (wynosi *unknown*), co spowoduje, że wartość całej formuły w punkcie  $t_1$  pozostanie niezmieniona. W ten sposób zamiast weryfikacji formuły we wszystkich stanach, istnieje potrzeba zweryfikowania tejże formuły tylko w jednym stanie.

Warto zauważyć, że w przypadku zmiany ceny w stanie większym niż  $t_2$  weryfikacja nie musi być uruchamiana, gdyż nie zmieni to i tak wartości formuły. Jedynym skomplikowanym przypadkiem jest sytuacja, gdy przesunięty zostanie czas zdarzenia *u.zaplata*.

Zmiana czasu zdarzenia *u.zaplata* np. z  $t_2$  na  $t_2+2$  wymaga podwójnej modyfikacji bazy. Trzeba bowiem usunąć (*delete*) zdarzenie *u.zaplata* z punktu czasu  $t_2$ , oraz wstawić (*insert*) do punktu  $t_2+2$ . Powstanie w ten sposób zakres weryfikacji  $\langle t_2, t_2+2 \rangle$ , w którym formuła musi zostać zweryfikowana. Jeżeli weryfikacja w tym przedziale nie da rezultatów (pozostanie *unknown*), wówczas musi odbywać się dalej aż do zweryfikowania lub do momentu, w którym dane będą nieokreślone. Oczywiście w tym przypadku zapis o zweryfikowaniu formuły w punkcie  $t_2$  musi zostać usunięty.

Warto podkreślić, że w tym przypadku jedynym kosztem w sensie pamięciowym jest zapamiętanie punktu  $t_1$  oraz punktu zweryfikowania  $t_2$  wraz z wartością weryfikacji. Koszt weryfikacji polega jedynie na uruchomieniu algorytmu *FTLStateVerification* w kilku punktach.

■

Powyższy przykład ukazuje, że w przypadku formuł  $\psi \Rightarrow \varphi$ , gdzie  $\psi$  formuła nietemporalna, a  $\varphi$  formuła typu FTL ich weryfikacja zależy od dwóch wymiarów. Pierwszym z nich jest zakres weryfikacji a drugim zakres zależności formuły.

### Definicja 3.9

#### Zakres zależności

Zakres zależności formuły  $\varphi$  i waluacji  $\theta$  to taki przedział, w którym zmiana waluacji  $\theta$  może mieć wpływ na wartość formuły w punkcie zajścia zdarzenia  $\psi$ .

■

### Definicja 3.10

#### Maksymalny zakres zależności

Maksymalny zakres zależności formuły  $\varphi$  to przedział wynikający z występowania operatora metrycznego.

### Definicja 3.11

#### Zakres weryfikacji

Zakres weryfikacji to przedział czasu, w którym formuła musi być zweryfikowana za pomocą algorytmu *FTLStateVerification*. Zakres weryfikacji jest częścią wspólną zakresu zależności i zakresu modyfikacji.



W przykładzie 3.7 zakresem modyfikacji jest punkt  $t_1+2$ , gdyż tylko w tym punkcie należy zaszła transakcja. uruchomić algorytm *FTLStateVerification*. Natomiast zakresem zależności jest przedział  $\langle t_1, t_2 \rangle$ . Punkt  $t_1$  jest punktem rozpoczęcia weryfikacji, a punkt  $t_2$  - punktem, w którym weryfikacja przyjmuje wartość *true* lub *false*. Gdyby punkt  $t_2$  nie był określony, przyjąłby wartość nieskończoności. Zakresem weryfikacji w tym przypadku jest część wspólna zakresu zależności i zakresu modyfikacji, a więc punkt  $t_1+2$ . Jedynie w tym punkcie powinien być uruchomiony algorytm *FTLStateVerification*.

Kolejny przykład rozważa formułę FTL z operatorami metrycznymi.

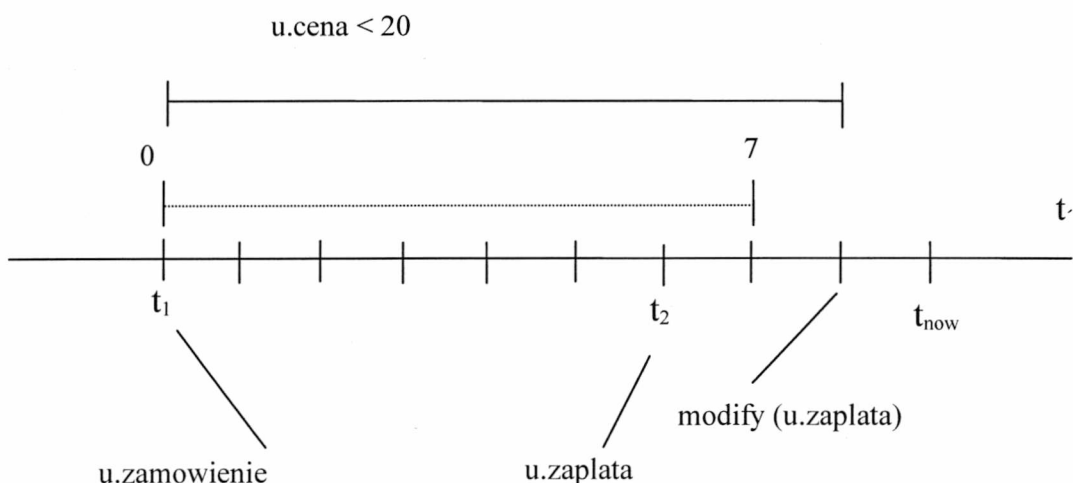
### Przykład 3.8

$u.zamowienie \Rightarrow (u.cena < 20 \text{ until } \neg u.zaplata) \vee \text{sometime\_future}_{<4} u.zamowienieodm$

Znaczenie tej formuły jest następujące:

*Od momentu zamówienia usługi jej cena jest mniejsza od 20, przy czym usługa ta musi być opłacona w ciągu 7 dni od zamówienia lub w ciągu 4 dni musi być wystawione zamówienie odmowne.*

Sytuacja jest bardzo podobna do rozpatrywanej wcześniej. Różnica polega jedynie na zakresie w jakim transakcja może wpływać na wartość formuły. Największym ograniczeniem metrycznym jest 7, a więc wiadomo, że po 7 dniach wartość formuły jest określona (nie jest *unknown*). Wobec tego przedział  $\langle t_1, t_1 + 7 \rangle$  jest maksymalnym zakresem zależności formuły. Tak jak w przypadku formuły poprzedniej, zapisywany jest czas w którym wartość formuły stała się określona (zakres zależności  $\langle t_1, t_2 \rangle$ ). Gdyby taki punkt nie istniał, wówczas zakres zależności byłby jednocześnie maksymalnym zakresem zależności, a więc przedziałem  $\langle t_1, t_1 + 7 \rangle$  (rys. 3.6).



Rys. 3.6. Zdarzenia na obiektach w bazie

W sytuacji pokazanej na rysunku 3.6 linią przerywaną zaznaczono maksymalny zakres zależności. Jeżeli zapłata nastąpiła w punkcie  $t_2$ , to wiadomo, że rozpatrywana formuła jest

spełniona. Wtedy zakres zależności formuły określony jest jako przedział  $\langle t_1, t_1+6 \rangle$ , gdyż w punkcie  $t_1+6$  wiadomo, że wartość formuły jest *true*.

Załóżmy jednak, że w temporalnej bazie danych błędnie zapisano zdarzenie *zapłata*. Zostaje więc uruchomiona transakcja, która usuwa zdarzenie *zapłata* z punktu  $t_2$  i zapisuje je do punktu  $t_2+2$ . Transakcja ta tworzy zakres modyfikacji:  $\langle t_2, t_2+2 \rangle$ , który nie jest rozłączny z określonym powyżej zakresem zależności formuły. Wobec tego w punkcie  $t_2$  zostaje uruchomiony algorytm *FTLStateVerification*, w wyniku którego formuła w tym punkcie przybiera wartość *unknown*. Następnie weryfikacja zostaje uruchomiona w punkcie następnym i tutaj formuła przybiera wartość *false*, gdyż zbadana jest różnica czasu pomiędzy czasem zdarzenia zamówienia, który jest już wcześniej zapisany, a czasem weryfikowanym ( $t_1$ ). W związku z tym zmienia się zakres zależności formuły i wynosi  $\langle t_1, t_1+7 \rangle$ . ■

Alternatywną strategią weryfikacji mogłoby być cofnięcie się do zdarzenia *zamówienie* i uruchamianie procesu weryfikacji od tego czasu w każdym punkcie. Jest to jednak metoda nieefektywna, gdyż wtedy algorytm *FTLStateVerification* musiałby być uruchomiony w każdym stanie. Sytuacja zmienia się nieznacznie w przypadku, gdy przy operatorze pojawia się znak  $\geq$ . Wówczas maksymalny zakres zależności jest przedziałem  $\langle t, \infty \rangle$ .

### Algorytm weryfikacji dynamicznych więzów integralności w temporalnych bazach danych

W celu utworzenia algorytmu weryfikacji należy przygotować najpierw zakres weryfikacji oraz zakresy zależności. Zakres weryfikacji jest definiowany jako część wspólna zakresu modyfikacji i zakresu zależności, który jest charakterystyczny dla danej formuły  $\varphi$  i waluacji  $\theta$ .

Zakres zależności formuły  $\varphi$  przy waluacji  $\theta$  oznaczany jest  $T_{Dep\ \varphi(\theta)} = \langle t_{DepBeg\ \varphi(\theta)}, t_{DepEnd\ \varphi(\theta)} \rangle$ . W przypadku formuł typu  $\psi \Rightarrow \varphi$ , (gdzie  $\psi$  formuła nietemporalna a  $\varphi$  formuła typu FTL) punkty te są łatwo obliczalne. Punkt  $t_{DepBeg\ \varphi(\theta)}$  jest równy chwili zajścia zdarzenia  $\psi$ , a punkt  $t_{DepEnd\ \varphi(\theta)}$  jest z własności ograniczony do punktu końca życia waluacji ( $lifespan(\theta).end$ ), punktu końcowego maksymalnego zakresu zależności formuły  $\varphi$  lub punktu  $t_{Compl_{\varphi}(\theta)}$  oznaczającego punkt, w którym wartość formuły jest określona jednoznacznie.

### Procedure SimpleTemporalVerification()

Niech zakres modyfikacji składa się z  $k$  przedziałów modyfikacji, utworzonych przez działanie transakcji:  $\langle t_1', t_1'' \rangle, \langle t_2', t_2'' \rangle, \dots, \langle t_i', t_i'' \rangle, \dots, \langle t_k', t_k'' \rangle$ , a  $t_e$  indeks stanu w którym obliczana jest wartość formuły  $\varphi$  dla waluacji  $\theta$

```

begin
for each  $\varphi$  do
for each  $\theta$  do
begin
// Jeżeli nowy zestaw obiektów, to zapisanie punktów progowych
if insert ( $\varphi, \theta$ ) then InsertTimeMarks( $\varphi, \theta, t_e$ )
endif;

// Obliczanie pierwszego punktu weryfikacji

```

```

t := ComputeFirstPoint( $\varphi$ ,  $\theta$ );

//Właściwa weryfikacja
while  $j \leq k$  and  $t < t_{DepEnd_{\varphi}(\theta)} + 1$ 

    /* Jedynie w przypadku formuły zależnej uruchamiany jest algorytm
    FTLVerification dla każdego wierzchołka typu Future formuły złożonej
    */
    for each  $\varphi_i \in Dependent(\varphi_i)$ 
        MTLVerification( $\varphi_i$ ,  $t$ ,  $\theta$ );
    endfor;

    if  $t = \text{lifespan}(\theta).end$  then  $end\_state(\varphi, \theta) := t$ ;
    endif;

    StateVerification( $\varphi$ ,  $t$ ,  $t_e$ ,  $\theta$ );

    // Jeżeli formuła jest określona to koniec weryfikacji
    if  $\varphi(\theta, t_e) \diamond \text{unknown}$ 
        then  $t_{Compl_{\varphi}(\theta)} := t$ ;
        exit;
    endif;

    // Jeżeli w punkcie, gdzie wcześniej była weryfikacja zakończona jest
    //wartość unknown, to wpisywana jest wartość null
    if  $t = t_{Compl_{\varphi}(\theta)}$  and  $\varphi(\theta, t_e) = \text{unknown}$  then  $t_{Compl_{\varphi}(\theta)} := \text{null}$ ;
    endif;

     $t := t + 1$ ;
    if  $t = t_i'' + 1$  then  $j := j + 1$ ;  $t := t_{j+1}'$ ;
endwhile;

/*Gdy weryfikacja nie zakończyła się sukcesem oraz został stracony punkt w
którym formuła została poprzednio zweryfikowana, to weryfikacja musi
posuwać się dalej (niezależnie od przedziałów modyfikacji) dopóki nie będzie
końca weryfikacji */

if  $t_{compl(\varphi(\theta))} = \text{null}$  then      VerificationToEnd( $\varphi$ ,  $\theta$ ,  $t$ );
endif;

endfor;
endfor;
endproc;

// Funkcja oblicza punkt początkowy weryfikacji
function ComputeFirstPoint(formuła  $\varphi$ , valuation  $\theta$ ) : integer
begin
var :  $t$  integer;
// Określenie pierwszego punktu oraz pierwszego przedziału weryfikacji

```

```

for i:=0 to k do
// jeżeli punkt początkowy weryfikacji jest w którymś z przedziałów
//modyfikacji
    if  $t_{DepBeg_{\varphi}(\theta)} \in \langle t_i', t_i'' \rangle$  then
         $t := t_{DepBeg_{\varphi}(\theta)}$ ;
         $j := i$ ;
//jeżeli jest poza przedziałami to początek weryfikacji ustawiany jest na
//pierwszy punkt przedziału

    else
         $t := t_{i+1}'$ ;
         $j := i + 1$ ;
    endif;
endfor;

return t;
endfunction;

/* Procedura jest uruchamiana jeżeli weryfikacja musi być prowadzona do momentu
obliczenia wartości formuły niezależnie od zakresu weryfikacji */
procedure VerificationToEnd(formula  $\varphi$ , valuation  $\theta$ , time t)
begin
    while  $t < t_{DepEnd_{\varphi}(\theta)}$  do

        /* Jedynie w przypadku formuły zależnej uruchamiany jest algorytm
        FTLVerification dla każdego wierzchołka typu Future formuły złożonej */

        for each  $\varphi_i \in \text{Dependent}(\varphi_i)$ 
            MTLVerification( $\varphi_i, t, \theta$ );
        endfor;

        if  $t = \text{lifespan}(\theta).end$  then  $\text{end\_state}(\varphi, \theta) := t$ ;
        endif;

        StateVerification( $\varphi, t, t_e, \theta$ );

        if  $\varphi(\theta, t_e) \diamond \text{unknown}$ 
            then  $t_{\text{Compl}_{\varphi}(\theta)} := t$ ;
            exit;
        endif;

         $t := t+1$ ;
    endwhile;
endproc;

```

Opis procedury *SimpleTemporalVerification*:

Weryfikacja odbywa się z wykorzystaniem algorytmu *FTLStateVerification*.

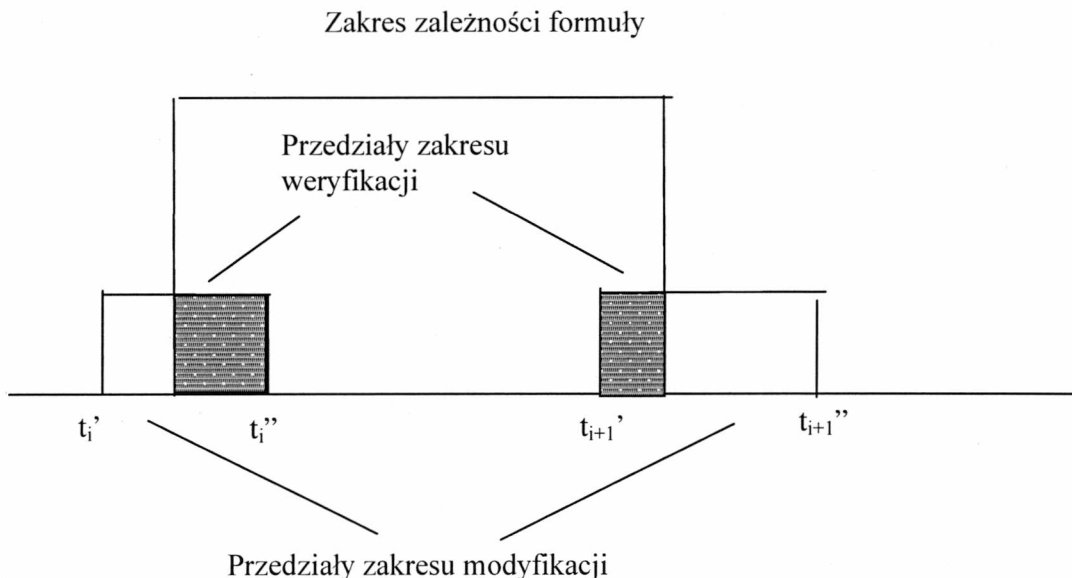
Rysunek 3.7 ukazuje przedziały zakresu weryfikacji. Jest to część wspólna przedziałów zakresu zależności i zakresu modyfikacji. Oba te zakresy są ustalane przed rozpoczęciem weryfikacji. Zakres modyfikacji jest charakterystyczny dla danej transakcji a zakres zależności dla danej formuły i waluacji. Weryfikacja przebiega w sposób następujący: Najpierw ustalany jest punkt początkowy obliczeń dla każdej formuły  $\varphi$  oraz każdej waluacji  $\theta$ . Następnie rozpatrywane są wszystkie przedziały weryfikacji, chyba, że formuła zostanie zweryfikowana lub skończy się czas życia waluacji  $\theta$ . Metoda korzysta z algorytmu *FTLStateVerification*, a więc przedstawiony poniżej zakres uruchamiania algorytmu, nie oznacza wcale, że sam algorytm *FTLStateVerification* jest uruchamiany w każdym punkcie przedziału składającego się na ten zakres. Uruchomienie następuje albo w punkcie, który jest zapisany w *TimeMarksTable* (rozdział II), albo gdy zmieni się któraś ze zmiennych waluacji  $\theta$ . Wreszcie nie ma sensu weryfikować w punktach większych od punktu ( $t_{\text{Compl}(\varphi(\theta))}$ ), w którym weryfikacja dała wynik określony (różny od *unknown*). Może się jednak zdarzyć, że w wyniku transakcji w punkcie  $t_{\text{Compl}(\varphi(\theta))}$  weryfikacja dała w wyniku *unknown*. Wówczas weryfikacja musi być prowadzona dalej, dopóki formuła nie zostanie jednoznacznie określona. Opisuje to ostatnia funkcja *VerificationToEnd*. Linie specyfikacji:

```

for each  $\varphi_i \in \text{Dependent}(\varphi_i)$ 
    MTLVerification( $\varphi_i, t, \theta$ );
endfor;

```

pisane czcionką Courier oznaczają, że są one zastosowane jedynie w przypadku formuł zależnych. Wynika to ze złożenia algorytmów *FTLVerification* i *FTLStateVerification* (rozdział II).



Rys. 3.7. Przedziały uruchamiania algorytmu

#### 4.2.2.2 Weryfikacja formuł złożonych FTL

Weryfikację formuł złożonych FTL można rozpatrywać jako złożenie weryfikacji za pomocą algorytmu *FTLStateVerification* oraz *FTLVerification*. Mimo, że w tym przypadku komplikuje się sposób określania zakresu weryfikacji i zakresu zależności, to jednak sama zasada weryfikacji pozostaje bez zmian.

W przypadku formuł prostych zakres modyfikacji jest ustalony przez punkty, w których działała transakcja. W przypadku formuł złożonych sytuacja wygląda inaczej.

##### Przykład 3.10

Dana niech będzie formuła złożona:

$$\varphi \equiv A \Rightarrow \text{always\_future} \prec_3 \text{sometime\_future} \preceq_5 B$$

Warto zauważyć, że modyfikacja podformuły B w punkcie k może mieć wpływ na wartość podformuły *sometime\_future*  $\preceq_5 B$  w punktach z przedziału  $\langle k-5, k \rangle$ . Algorytm musi więc zostać uruchomiony w tych punktach. W związku z tym zakres modyfikacji nie będzie (tak jak w przypadku formuł prostych) punktem k, lecz przedziałem  $\langle k-5, k \rangle$ .

Tak więc jeżeli transakcja wyprodukowała przedział modyfikacji  $\langle t', t'' \rangle$  (w tym przedziale zaszły zmiany, które mogły mieć wpływ na wartość formuły A lub B), to na podstawie tego przedziału musi być utworzony nowy przedział modyfikacji  $\langle t' - \Delta, t'' \rangle$ , gdzie  $\Delta$  jest przesunięciem zależnym od operatora metrycznego formuły zależnej.

Problemy też powstają z maksymalnym zakresem zależności formuły  $\varphi(t_{Dep \varphi(\theta)})$ . Dla formuł prostych z operatorami metrycznymi  $< (\leq)$  punkt  $t_{DepEnd \varphi(\theta)}$  jest przesunięty w stosunku do  $t_{DepBeg \varphi(\theta)}$  o największy operator metryczny, a dla formuł złożonych o sumę operatorów metrycznych. Również wtedy przesunięcie  $\Delta$  jest sumą operatorów metrycznych w skład której nie wchodzi operator metryczny formuły FTL, która nie jest zależna od innej formuły FTL. Więcej na ten temat w następnym podrozdziale.

W przypadku formuły złożonej, w której jedna z podformuł ograniczona jest operatorem metrycznym ze znakiem  $> (\geq)$  zarówno punkt  $t_{DepEnd \varphi(\theta)}$  jak i przesunięcie  $\Delta$  przyjmują wartość nieokreśloną. Tak więc w tym przypadku zakres modyfikacji jest zdefiniowany jako  $\langle -\infty, t'' \rangle$ , a zakres zależności  $\langle t'', \infty \rangle$ .

Przykład 3.11 ukazuje problemy związane z weryfikacją formuł złożonych FTL w temporalnych bazach danych.

##### Przykład 3.11

Niech:

$$\varphi \equiv A \Rightarrow \text{always\_future} \prec_3 \text{sometime\_future} \succ_5 B$$

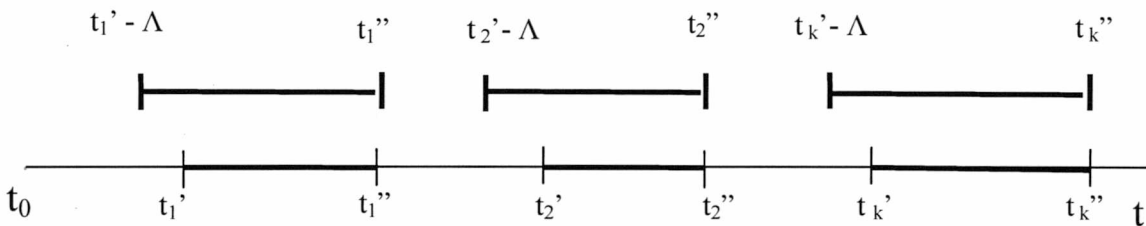
Jak wykazał to rozdział I, wartość podformuły  $\varphi_1 \equiv \text{sometime\_future} \succ_5 B$  w punkcie  $t_1$  zależy od wartości podformuły B w każdym punkcie większym od  $t_1 + 5$ . Wobec tego, jeżeli transakcja wygeneruje zakres modyfikacji składający się z jednego przedziału  $\langle t_b, t_e \rangle$ , to może mieć to wpływ na wartość podformuły  $\varphi_1$  w każdym punkcie mniejszym niż  $t_b - 5$ . Wobec tego zakres modyfikacji musi być rozpatrywany jako przedział  $\langle t_A, t_e \rangle$ , przy czym  $t_A$  jest czasem zajścia zdarzenia A.

Przy weryfikacji formuły  $\varphi$ , najmniejszym punktem, w którym musi być określona wartość podformuły  $\varphi_1$  jest punkt  $t$  zajście zdarzenia A. Skoro wartość podformuły  $\varphi_1$  zależy od wartości podformuły B we wszystkich punktach większych od  $t_A$ , wówczas maksymalnym zakresem zależności formuły  $\varphi$  będzie przedział  $\langle t_A, \infty \rangle$ . Oczywiście jest to maksymalny zakres zależności (wynikający z ograniczenia metrycznego). Może się zdarzyć, że ten zakres zależności będzie mniejszy. ■

### Algorytm weryfikacji formuł złożonych typu FTL w temporalnych bazach danych

Na podstawie powyższych rozważań można pokusić się o stworzenie algorytmu weryfikacji złożonych formuł FTL. Jak to już zostało wcześniej wspomniane algorytm ten jest w pewnym sensie złożeniem metody *FTLVerification* i *FTLStateVerification*. Stopień skomplikowania pogłębia jeszcze fakt, że zakres modyfikacji oraz zakres zależności formuł nie są już tak niewielkie jak to było w przypadku formuł prostych.

Poniższy rysunek ukazuje zależność pomiędzy przedziałami modyfikacji utworzonymi przez transakcję (te na dole) oraz przedziałami modyfikacji powstałymi w wyniku występowania formuł złożonych.



Rys. 3.9. Relacja pomiędzy przedziałami modyfikacji

Przedziały modyfikacji :  $\langle t_1', t_1'' \rangle$ ,  $\langle t_2', t_2'' \rangle$ , ...,  $\langle t_i', t_i'' \rangle$ , ...,  $\langle t_k', t_k'' \rangle$  powstałe w wyniku działania transakcji T muszą być przetworzone na nowe przedziały modyfikacji. Niech  $\Lambda$  oznacza różnicę pomiędzy punktami początkowymi tych dwóch różnych rodzajów przedziałów modyfikacji. Wówczas zakres modyfikacji jest sumą przedziałów :  $\langle t_1' - \Lambda, t_1'' \rangle$ ,  $\langle t_2' - \Lambda, t_2'' \rangle$ , ...,  $\langle t_i' - \Lambda, t_i'' \rangle$ , ...,  $\langle t_k' - \Lambda, t_k'' \rangle$ . Oczywiście przyjęte jest założenie, że tak utworzone przedziały są rozłączne. Jeżeli zastosowanie wartości  $\Lambda$  spowoduje nałożenie się przedziałów na siebie, są one oczywiście łączone w jeden.

Poniżej zdefiniowana jest metoda obliczająca przesunięcie  $\Lambda$ .

Niech formuła  $\varphi$  jest postaci  $\psi \Rightarrow \varphi_1 \oplus \dots \oplus \varphi_i \oplus \dots \oplus \varphi_n$ , gdzie  $\oplus \in \{\wedge, \vee\}$ , a  $\varphi_i$  formuła FTL.

Formuła  $\varphi_i = op_{i1} \sim ci1 \ op_{i2} \sim ci2 \ \dots \ op_{im} \sim cim \ \psi$ , gdzie:

$op_{i1}, op_{i2}, \dots, op_{im}$  - operatory FTL,  $\sim \in \{\leq, <, \geq, >\}$ ,  $\psi$  - formuła atomowa.

// Funkcja obliczająca przesunięcie  $\Lambda$  dla danej formuły  $\varphi$ .

**function Shift ( $\varphi$ ): integer**  
**begin**



```

var: i, max; j, op_max : integer;
i:=1; max:=0;

while i <= n do
    j=2; op_max=0;
    while (j <= m)
        if sign (opij) = '>' or '≥' then
            return Shift:= ∞;
        else if sign (opij) = '<' then
            op_max:= op_max + cij - 1;
        else if sign (opij) = '≤' then
            op_max:= op_max + cij ;
        endif;
    endwhile;
    if max < op_max then
        max:= op_max;
    endif;
endwhile;
return Shift:= max;
endfunction;

```

**procedure ComplexTemporalVerification()**  
**begin**

Zakłada się, że transakcja  $\Delta$  tworzy następujące zakresy modyfikacji:  $\langle t_1', t_1'' \rangle$ ,  $\langle t_2', t_2'' \rangle$ , ...,  $\langle t_i', t_i'' \rangle$  ...,  $\langle t_k', t_k'' \rangle$ .

```

// Szukany jest największe przesunięcie  $\Delta$ 
for each  $\varphi$  do
    if  $\Delta > \text{Shift}(\varphi)$  then
         $\Delta := \text{Shift}(\varphi)$ ;
    endif;
endfor;

```

// Poniższa pętla uaktualnia przedziały modyfikacji.

```

for i:=1 to k do
     $\langle t_i', t_i'' \rangle := \langle t_i' - \Delta, t_i'' \rangle$ ;
endfor;

```

SimpleTemporalVerification();

**endproc;**

Algorytm **ComplexTemporalVerification** różni się od algorytmu **SimpleTemporalVerification** zasadniczo w dwóch aspektach:

- 1) zakresy weryfikacji mogą być znacznie większe: pociąga to za sobą konieczność uruchomienia algorytmu w większej liczbie punktów, co zwiększa złożoność obliczeniową;
- 2) wspomnianą już w opisie procedury **SimpleTemporalVerification** koniecznością uruchomienia procedury FTLVerification.

## ZAKOŃCZENIE

Rozprawa doktorska wprowadza nowe metody weryfikacji dynamicznych więzów integralności z użyciem logiki temporalnej. Rozważany rodzaj logiki to temporalny rachunek zdań z operatorami metrycznymi.

Aby można było stworzyć algorytmy weryfikacji w oparciu tę logikę temporalną, należy ją najpierw zdefiniować, a następnie wprowadzić i udowodnić metody wnioskowania przydatne w procesie weryfikacji. Tak więc w rozdziale I oprócz podstawowych pojęć związanych z logiką temporalną, zdefiniowano składnię i semantykę logiki używanej w dalszej części pracy. Ponadto udowodniono reguły wynikające z definicji operatorów temporalnych ograniczonych metrycznie przydatne do procesu weryfikacji formuł temporalnych.

W rozdziale II zmodyfikowano grafy przejść zaczerpnięte z pracy [Zhou 1994]. Okazało się bowiem, że grafy zdefiniowane w [Zhou 1994] są niewystarczające do weryfikacji formuł logiki temporalnej z ograniczeniami metrycznymi. Zarówno grafy przejść jak i metody wnioskowania są podstawą następujących metod weryfikacji (rozdział II):

- metoda *PTLVerification* - umożliwia weryfikację formuł logiki temporalnej czasu przeszłego z możliwością dodania ograniczeń metrycznych. Z metody tej można właściwie wyodrębnić dwie metody w zależności od aktualnych potrzeb. Jeżeli operatory metryczne nie są potrzebne, pewne linie w specyfikacji algorytmów mogą być usunięte. W związku z tym algorytm się bardzo upraszcza.
- metoda *FTLVerification* - umożliwia weryfikację formuł logiki temporalnej czasu przeszłego z możliwością dodania ograniczeń metrycznych. Tak jak w przypadku metody *PTLVerification* pewne linie specyfikacji można ominąć w zależności od tego, czy stosowane są operatory metryczne oraz czy weryfikowane są formuły złożone.
- metoda *MTLVerification* - umożliwia weryfikację formuł, w których cele mogą występować zarówno operatory logiki czasu przyszłego jak i przeszłego. Oczywiście można dodać ograniczenia metryczne do tych operatorów. W tym przypadku także dozwolone są przeróżne mutacje metod.
- metody *StateVerification* - umożliwia weryfikację formuł prostych logiki temporalnej w pojedynczym stanie lub w odstępach (co kilka stanów). Tutaj można wyróżnić metody: *FTLStateVerification*, *PTLStateVerification* oraz *MTLStateVerification* w zależności od rodzaju stosowanej logiki.

Przy tej okazji warto nadmienić, że algorytmy mogące weryfikować formuły logiki FTL i MTL korzystają z logiki trójwartościowej. Tak więc do standardowych wartości logicznych: *true*, *false*, dodana jest wartość *unknown*. Jest to odejście od wykorzystywania potencjalnej spełnialności, której stosowanie nie ma większego sensu w formułach typu MTL z ograniczeniami metrycznymi.

W powyższych metodach zastosowano także optymalizację polegającą na wprowadzeniu znacznika *verified*. Zauważono bowiem, że na wartość formuły ma wpływ zmiana waluacji formuł atomowych oraz upływający czas. W niektórych przypadkach jednak pomimo występowania operatora metrycznego) czas nie ma znaczenia, a wpływ na wartość rozpatrywanej formuły ma jedynie fakt, czy w danym stanie została zmieniona wartość którejś z formuł atomowych. Wówczas stany te mogą być opuszczane w procesie weryfikacji.

Do weryfikacji powyższych metod potrzebne są pewne informacje będące wynikiem weryfikacji z poprzednich stanów. Rozdział II wprowadza więc podstawowy sposób

modelowania oraz pewne procedury dotyczące manipulacji na tych danych. Dane te mogą być przedstawione jako obiekty bądź relacje. Nie ma to większego znaczenia w procesach weryfikacji, a prostota ich modelowania powoduje uniezależnienie proponowanych metod od modelu danych. Przechowywanie dodatkowych danych jest oczywiście także optymalizowane.

Dla każdej z przedstawionych powyżej metod obliczono złożoność pamięciową i obliczeniową.

Rozdział III opisuje wykorzystanie metod weryfikacji w bazach nietemporalnych i temporalnych opartych o czas rzeczywisty. Najlepszą metodą do zastosowania w przypadku baz nietemporalnych jest *PTLVerification*. Jej podstawową zaletą jest bardzo niska złożoność obliczeniowa i pamięciowa niezależnie od stopnia skomplikowania formuł. W przypadku algorytmu *FTLVerification* złożoność pamięciowa jest większa a złożoność obliczeniowa drastycznie wzrasta przy weryfikacji formuł złożonych.

Temporalny model danych zachowuje całą historię obiektów wraz z ich wartościami z przyszłości. Jako, że nie jest rozpatrywany probabilistyczny model danych, ewentualne wartości z przyszłości muszą być pewne, aby zostały umieszczone w temporalnej bazie danych.

Przy tak zdefiniowanym modelu zmienia się koncepcja transakcji. Rozdział III definiuje więc *insert*, *update* i *delete* do zastosowania w temporalnym modelu danych. Te trzy operacje mogą operować na wartościach obiektów nie tylko w stanie teraźniejszym, ale również w stanach różnych od teraźniejszego. Tak więc transakcja może zajść nie tylko w obecnym stanie bazy ale w dowolnym punkcie historii. Aby zweryfikować więzy integralności po zajściu takiej transakcji przy użyciu któregoś z temporalnych języków zapytań, należałoby sprawdzić całą historię bazy. Dzięki algorytmowi *FTLStateVerification* uruchomionemu tylko w tym punkcie unika się przeglądania całej historii. Można z tego wyciągnąć wniosek, że w wielu przypadkach wystarczy uruchomić ten algorytm jedynie w tych punktach, w których działała transakcja i to jeszcze przy założeniu, że wartości obiektu w tych punktach mogą coś zmienić, jeżeli chodzi o weryfikację formuły. Zysk na złożoności obliczeniowej jest więc ogromny.

W celu przetestowania poprawności metod zaproponowanych w rozprawie doktorskiej, zostały one zaimplementowane w środowisku Visual C++ 6.0. Program sprawdza poprawność formuły temporalnej, umożliwia utworzenie grafu takiej formuły, proponuje wprowadzenie danych jej podformuł atomowych w dowolnej ilości stanów oraz przeprowadza weryfikację tej formuły na wprowadzanych wcześniej danych.

#### Największe osiągnięcia pracy:

- Zbadanie logiki temporalnej z ograniczeniami metrycznymi oraz wprowadzenie i udowodnienie nowych metod wnioskowania dla logiki temporalnej z operatorami metrycznymi.
- Stworzenie kilku metod weryfikacji formuł logiki temporalnej. Szczególnie metody *PTLVerification* oraz *StateVerification* okazały się metodami bardzo wydajnymi ze względu na złożoność obliczeniową i pamięciową. Metoda *MTLVerification* nie jest tak efektywna, ale może mieć szerokie zastosowanie ze względu na swoją siłę ekspresji. Metody weryfikacji są niezależne od modelu danych na którym operują.
- Opracowanie efektywnej metody przechowywania dodatkowych informacji niezbędnych w procesie weryfikacji formuł temporalnych.
- Wykorzystanie logiki trójwartościowej do weryfikacji formuł typu FTL i MTL - odejście od koncepcji potencjalnej spełnialności formuł temporalnych.

- Zapoczątkowanie prac nad problemem weryfikacji dynamicznych więzów integralności w temporalnych bazach danych.
- Znalezienie efektywnej metody weryfikacji dynamicznych więzów integralności w temporalnym modelu baz danych opartym na czasie rzeczywistym polegającej na omijaniu sekwencji stanów nie mających wpływu na wartość weryfikowanej formuły.

### **Porównanie osiągniętych wyników z innymi pracami oraz możliwości przyszłych badań**

W niniejszej pracy skoncentrowano się zasadniczo na badaniu dwóch obszarów wiedzy. Pierwszym z nich jest logika temporalna, a drugim dynamiczne więzy integralności w bazach danych. Poniżej opisano możliwości zastosowania wyników badań przeprowadzonych w rozprawie w innych dziedzinach badawczych. Jest to jednocześnie analiza porównawcza z innymi pracami oraz nakreślenie nowych kierunków badań.

#### Weryfikacja formuł temporalnych

Wyniki niniejszej pracy są zasadniczym krokiem naprzód w stosunku do prac prowadzonych w Hannoverze ([Lipeck 1987], [Gertz i Lipeck 1995]). Podniesiona została siła ekspresji logiki temporalnej poprzez zastosowanie operatorów metrycznych. Używane są wprowadzanie grafy przejść, ale okazuje się, że one same nie wystarczają do poprawnego wnioskowania. Wobec tego zdefiniowane zostały własności wnioskowania, będące podstawą matematyczną do weryfikacji formuł temporalnych z operatorami metrycznymi.

W [Gertz i Lipeck 1996] przedstawiona jest metoda automatycznego reagowania na załamanie więzów integralności. W pracy tej weryfikacja formuł FTL oparte jest o grafy przejść, na podstawie których tworzone są triggery jako reakcje na ewentualne niespełnienie formuły. W naszym przypadku sytuacja jest bardziej skomplikowana. Załamanie więzów integralności może być spowodowane nie tylko poprzez transakcje, ale także poprzez upływający czas. Jeżeli transakcja może być cofnięta, to „cofnięcie” upływającego czasu jest niemożliwe. W związku z tym takie więzy integralności są „nienaprawialne”. Tak przedstawia się ten problem w nietemporalnych bazach danych. Natomiast w modelu temporalnym można pokusić się o modyfikację historii i w ten sposób naprawić więzy integralności załamane upływającym czasem. Problem ten można pozostawić jako przedmiot przyszłych badań.

Metody weryfikacji formuł temporalnych rozpatrzono także w pracy [Sistla i Wolfson 1995]. Podstawą matematyczną tej pracy jest czasowy temporalny rachunek zdań (Timed Propositional Temporal Logic) opisany w [Alur i Henzinger 1994]. Cechą charakterystyczną tej logiki jest występowanie operatora freeze, który umożliwia przypisanie wartości dowolnej zmiennej (*database variable*), a następnie wykorzystanie tej zmiennej w formule. Metody przedstawione w niniejszej rozprawie doktorskiej także pozwalają na wykorzystanie takiej zmiennej.

Niech:

$$\varphi_1 = (Q > 2x) \text{ since } (x = Q)$$

Jeżeli Q dowolne zapytanie zwracające w wyniku liczbę rzeczywistą, to by zweryfikować formułę  $\varphi_1$ , należałoby zapamiętać zmienną x na tej samej zasadzie, na jakiej zapamiętywane są dodatkowe dane do weryfikacji dla metody *MTLVerification*.

Można także weryfikować formuły temporalne w różnych stanach bazy.

Niech:

$$\varphi_2 \equiv [t]A \Rightarrow [t+3] (C \text{ until } <_5 B) \wedge [t+5] \text{sometime\_future } <_3 D$$

Formuła  $\varphi_2$  przyjmie wartość true, jeżeli formuła  $A$  jest spełniona w stanie o punkcie czasu  $t$  oraz podformuła  $C \text{ until } <_5 B$  jest spełniona w punkcie  $t+3$  a podformuła  $\text{sometime\_future } <_3 D$  w punkcie  $t+5$ . Formuła taka może być weryfikowana za pomocą metody *MTLVerification*, gdyż metoda ta zachowuje zweryfikowane wartości podformuł temporalnych w różnych punktach czasu. Również metoda *FTLStateVerification* jest w stanie zweryfikować formułę  $\varphi_2$ . Wystarczy bowiem zmienić punkt odniesienia  $t$  podformuł temporalnych formuły  $\varphi_2$ .

Praca [Sistla i Wolfson 1995] rozpatruje weryfikację formuł logiki FTL i PTL, ale nie umożliwia weryfikacji formuł MTL. Wynika to z faktu, że praca skupia się głównie na weryfikacji temporalnych triggerów a nie dynamicznych więzów integralności.

Chomicki w [Chomicki 1995] przedstawia metody weryfikacji dynamicznych więzów integralności w oparciu logikę temporalną pierwszego rzędu z wykorzystaniem przeszłych operatorów temporalnych. Jak to już zostało wspomniane w rozdziale I logika pierwszego rzędu daje możliwość stosowania kwantyfikatorów: uniwersalnego i szczegółowego. Jednak głównym ograniczeniem pracy [Chomicki 1995] jest takie, że wszystkie podformuły weryfikowanej formuły muszą być niezależne od dziedziny.

### Prewarunki temporalne

Praca [Schwiderski i inni 1994] rozpatruje tzw. temporalne prewarunki zdarzeń (temporal preconditions). Prewarunki temporalne zdarzeń, to formuły logiki PTL, które warunkują zajście jakiegoś zdarzenia w bazie. Innymi słowy: zdarzenie lub sekwencja zdarzeń  $\psi$  może zajść jedynie wtedy, gdy spełniona jest formuła  $\varphi$  ( $\varphi$  - formuła PTL). Można to zapisać w sposób następujący:

$$\psi \rightarrow \varphi.$$

Warto zauważyć, że zdarzenie  $\psi$  nie musi zajść nawet przy spełnionej formule  $\varphi$ . Warunek  $\varphi$  nie powoduje więc żadnej akcji, lecz jest jedynie warunkiem początkowym zajścia zdarzenia  $\psi$ . Prewarunki temporalne są alternatywą do wprowadzonych w rozdziale II formuł postaci  $\psi \rightarrow \varphi$ , gdzie  $\psi$  - zdarzenie, a  $\varphi$  - formuła FTL. Wtedy, jeżeli spełniona było  $\psi$ , to musiała być spełniona formuła  $\varphi$ . Wobec tego najpierw była określona prawdziwość  $\psi$ , a potem  $\varphi$ .

W przypadku prewarunków temporalnych najpierw sprawdzana jest wartość  $\psi$  (formuła typu PTL), a następnie spełnienie bądź niespełnienie tej formuły określa czy może zajść zdarzenie określone w  $\varphi$ .

Wprowadzone w rozdziale II metody weryfikacji logiki PTL z operatorami metrycznymi doskonale nadają się do sprawdzania prawdziwości prewarunków temporalnych.

### Temporalne wyzwalacze (temporal triggers)

W teorii baz danych wyzwalaczami (ang. triggers) nazywane są procedury uruchamiane jako reakcja na zdarzenie zachodzące w bazie. Bazy umożliwiające stosowanie wyzwalaczy nazywane są bazami aktywnymi. Wyzwalacz taki ma postać:

*if Condition then Action*

*Condition* jest z własności formułą logiczną, której częściami składowymi są zdarzenia np. UPDATE, INSERT, DELETE itd., a *Action* jest jakąś dowolną reakcją na warunek. Może to być na przykład usunięcie bądź uaktualnienie jakiegoś obiektu z bazy, wysłanie wiadomości powiadamiającej użytkownika o zaistniałym zdarzeniu itd.

Dzięki logice temporalnej pojęcie wyzwalacza można znacznie rozszerzyć ([Sistla i Wolfson 1995]). Warunkiem może być dowolna formuła temporalna opisująca warunki na przestrzeni sekwencji stanów.

Przykładem wyzwalacza może być:

*if  $\phi$  then Action*, gdzie  $\phi$  jest formułą temporalną, a *Action* dowolną reakcją na zdarzenie.

Jeżeli  $\phi$  jest formułą typu PTL to zgodnie z rozważaniami z rozdziału II w punkcie rozpatrywania (t) znana jest już jej wartość. Wobec tego wyzwalacz odpalany jest w tym punkcie.

W przypadku formuł typu MTL lub FTL wartość formuły w stanie k nie jest jeszcze znana w tym stanie, wobec tego *trigger* może być odpalony z pewnym opóźnieniem.

Logiki stosowane w tej pracy można z powodzeniem wykorzystać jako warunków wstępnych temporalnych wyzwalaczy. Jest to uzyskiwane przede wszystkim dzięki zastosowaniu logiki z operatorami metrycznymi. Jedynie wtedy ma sens stosowanie logiki FTL. Brak możliwości stosowania operatorów metrycznych przy formułach powoduje, że takie formuły są czasami określone dopiero na końcu przedziału rozpatrywania. Wobec tego formuły takie nie bardzo nadają się jako warunki odpalania triggerów.

Kolejnym problemem pojawiającym się przy okazji odpalania triggerów jest problem ich reaktywacji. Istnieją dwie możliwości. Pierwsza zakłada odpalanie triggerów przy każdej wartości prawdziwej formuły  $\phi$ . W przypadku drugim następny stan po stanie, w którym odpalenie triggera miało miejsce, traktowany jest jako początek historii. Oczywiście wybranie któregoś z tych podejść zależy od konkretnego zastosowania. Algorytmy umożliwiają stosowanie zarówno pierwszego jak i drugiego podejścia.

### Systemy czasu rzeczywistego

Jako systemy czasu rzeczywistego rozumie się takie systemy, których cechą krytyczną jest czas wykonywanych operacji, które przekroczenie jest niedozwolone i może doprowadzić do katastrofy systemu. Jak opisują prace [Vrbsky 1993] oraz [Vrbsky 1997], przykładem takich operacji mogą być zapytania stawiane bazie systemu czasu rzeczywistego. Czasem lepiej jest znaleźć przybliżoną odpowiedź na zapytanie nie przekraczając przy tym czasu krytycznego, niż szukać dokładnej odpowiedzi zbyt długo.

Urządzenia czasu rzeczywistego dysponują niekiedy dość małymi zasobami pamięci stałej. Wobec tego stosowanie danych temporalnych może łatwo doprowadzić do przepełnienia dostępnej pamięci.

Te wymienione powody mogą skłaniać do zastosowania logiki temporalnej jako najbardziej odpowiedniego języka temporalnych zapytań. Korzyścią płynącą z takiego postępowania byłoby mniejsze prawdopodobieństwo przekroczenia zasobów pamięciowych jak również mniejsze prawdopodobieństwo przekroczenia czasu krytycznego wykonywania zapytań.

### Temporalne bazy danych



Druga część rozdziału III przedstawia metodę wykorzystania formuł FTL do weryfikacji dynamicznych więzów integralności w temporalnych bazach danych opartych o czas rzeczywisty. Kierunek ten jest właściwie niezbadany. Tylko praca [Gertz i Lipeck 1995] jest próbą rozwiązania tego zagadnienia. Gertz i Lipeck korzystają jednak z logiki FTL bez operatorów metrycznych. W związku z tym zastosowanie w niniejszej rozprawie logiki z ograniczeniami metrycznymi jest pewnym krokiem naprzód na tym polu.

Interesujące byłoby także rozważenie zastosowania weryfikacji opartej o logikę przedziałową (ang. *interval - based logic*) w temporalnych bazach danych opartych o czas rzeczywisty, co byłoby bardziej zgodne z modelem danych takiej bazy. Również ciekawe byłoby zastosowanie rozgałęzionej logiki temporalnej (ang. *branching - time temporal logic*). Logika taka zakłada występowanie rozgałęzień opisujących różne przewidywane przyszłości. Byłoby to semantycznie zgodne z probabilistycznym temporalnym modelem danych, zakładającym występowanie kilku przewidywalnych ścieżek rozwoju temporalnej bazy danych.

Problemy te są jednak na tyle obszerne, że powinny być kontynuowane w dalszych badaniach naukowych.



## LITERATURA

- [Abiteboul 1995] Abiteboul S., Hull R., Vianu V., *Foundations of Databases*. Addison - Wesley Reading 1995.
- [Abbot i inn 1992] Abbot R., K., Garcia-Molina H., *Scheduling Real-Time Transactions: A Performance Evaluation*, ACM Transactions on Database Systems, vol. 17, no.3, September 1992.
- [Alur i Henzinger 1994] Alur T., Henzinger T. A. *A Really Tetmporal Logic*, *Journal of the ACM*, 1994, vol.41, no.1
- [Bellini i inn 2000] Bellini P., Mattolini R., *Temporal Logics for Real-Time System Specyfication*, ACM Computing Surveys, Vol. 32, No. 1, March 2000
- [Błażewicz 1988] Błażewicz J., *Złożoność obliczeniowa problemów kombinatorycznych*, Wydawnictwo Naukowo - Techniczne, Warszawa 1988.
- [Caruso i inn. 1990] Caruso, M., Sciore, E. *The Vision Object - Oriented Database Management Systems*. Advances in Database Programming Languages. ACM Press, New York. 1990.
- [Ceri s. i inn. 1994] Ceri S., Fraternali P., Paraboschi S., *Automatic Generetion of Production Rules for Integrity Maintenance*, ACM Transactions on Database Systems, vol. 19, no.3, September 1994.
- [Chakravarthy i Kim 1994] Chakravarthy S., Kim S-K, *Resolution of Time Concepts in Temporal Databases*, Information Sciences 80, 1994
- [Chomicki 1995] Chomicki J. *Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding* ACM Transactions on Database Systems, vol. 20, No.1, June 1995.
- [Chomicki i Toman 1996 ] Chomicki J., Toman D., *Temporal Logic in Information Systems*, <http://www.brics.dk/LS/97/1/BRICS-LS-97-1.pdf>
- [Chomicki i Toman 1996 ] Chomicki J., Toman D., *Implementing Temporal Integrity Constraints Using an Active DBMS*, IEEE Transaction on Knowledge and Data Engineering, vol. 7, no. 8, August 1995
- [Clifford i inn. 1994] Clofford J., Crocker A., *On Completeness of Historical Relational Query Languages*, ACM Transactions on Database Systems, vol. 19, No.1, March 1994.
- [Dyreson i Snodgrass 1998] Dyreson C., E., Snodgrass R., T., *Supporting Valid-Time Indeterminacy*, ACM Transactions on Database Systems, vol. 23, No.1, March 1998.
- [Gertz 1993] Gertz M., *On Specifying the Reactive Behavior on Constraints Violation*, Informatik - Berichte, No. 2/93, Universitaet Hannover, Dezember 1993

[Gertz 1994] Gertz M., *Specifying Reactive Integrity Control in Active Databases*, Informatik - Berichte, Universitaet Hannover, Dezember 1994

[Gertz 1996] Gertz M. *Diagnosis and Repair of Constraint Violations in Database Systems*. Universität Hannover, Ph. D. Dissertation , Mai 1996 , <http://www-b.informatik.uni-hannover.de/schriften/dissertationen.php>

[Gertz i Lipeck1994] Gertz M. Lipeck U., *ICE An Environment for Integrity Centered Database Design*, Informatik - Berichte No. 2/94, Universitaet Hannover, Marz 1994

[Gertz i Lipeck 1996] Gertz M., Lipeck U. W., *Deriving Optimized Integrity Monitoring Triggers from Dynamic Integrity Constraints*. Data & Knowledge Engineering 20 (1996).

[Gertz i Lipeck 1995] Gertz M., Lipeck U. W., „Temporal” Integrity Constraints in Temporal Databases. Informatik Berichte Institut fuer Informatik, Universitaet Hannover.

[Grefen i Apers 1993] Grefen P., Apers P. *Integrity Control in Relational Database Systems - An Overview*. Data & Knowledge Engineering 10 (1993).

[Hajnicz 1996] Hajnicz E. *Reprezentacja logiczna wiedzy zmieniającej się w czasie*. Akademicka Oficyna wydawnicza PLJ Warszawa 1996

[Hyun i inn. 1997] Hyun S., J., Su S., Y. *Semantics - Based Time-Alignment Operations in Temporal Processing and Optimization*, Information Sciences vol. 103, December 1997

[Huzar i Magott 1994] Huzar Z., Magott J., *Systemy czasu rzeczywistego - metody specyfikacji*. Informatyka, 1994, nr 8

[Kaefer i Schoning 1992] Kaefer W., Schoning H., *Mapping a Version Model to a Complex - Object Data Model*. Proceedings of the International Conference on Data Engineering. Tempe, Arizona. 1992.

[Kania i Gołuchowski 1996a] Kania K., Gołuchowski J., *Reprezentacja czasu w modelu konceptualnym SI*, Informatyka 8/1996

[Kania i Gołuchowski 1996b] Kania K., Gałuchowski J., *Zagadnienia reprezentacji czasu w klasycznych systemach baz danych*, „Informatyka” nr 7, 1996.

[Kim 1996] Kim W. *Wprowadzenie do obiektowych baz danych*. Wydawnictwo Naukowo Techniczne Warszawa 1996.

[Klimek 1999] Klimek R., *Wprowadzenie do logiki temporalnej*, Wydawnictwa AGH, Kraków 1999

[Królikowska i Marcinkiewicz 2000] Królikowska B., Marcinkiewicz J., *Czas jako atrybut zdarzenia w relacyjnych i obiektowych bazach danych*, Informatyka 9/2000

[Lipeck i Saake 1987] Lipeck U. W., Saake G. *Monitoring Dynamic Integrity Constraints Based on Temporal Logic*. Information Systems 12 :3 (1987).

[Lipeck 1988] Lipeck, U.W.: *Zur dynamischen Integritaet von Datenbanken: Grund- lagen der Spezifikation und ~ Überwachung (Dynamic Database Integrity: Foundations of Specification and Monitoring)*. Informatik-Fachbericht 209, Springer-Verlag, Berlin 1989 (original version: Habilitation Thesis, Techn. Univ. Braunschweig 1988).

[Lipeck 1990] Lipeck, U.W.: *Transformation of Dynamic Integrity Constraints into Transaction Specifications*. Theoretical Computer Science, Vol. 76 (1990), 115-142

[Lorentzos N.A. i inn. 1999] Lorentzos N.A., *Time-evolving rule-based knowledge bases*, Data & Knowlegde Engineering 29 (1999).

[Macyna i Mazur 2000] Macyna W., Mazur Z., *Integralność i języki zapytań w systemach czasu rzeczywistego*, VII Konferencja: Systemy Czasu Rzeczywistego 2002

[Malinowski 1990] Malinowski G., *Logiki wielowartościowe*, PWN Warszawa 1990

[MaP81] Manna, Z. and Pnueli, A.: *Verification of Concurrent Programs: The Temporal Framework*. The Correctness Problem in Computer Science (R.S.Boyer, J.S.Moore, eds.), Academic Press

[Mazur i Majkowski 1998] Mazur Z., Majkowski A., *Wybrane zagadnienia dotyczące systemów temporalnych baz danych*, Informatyka 12/1998

[Mirkowska i Salwicki 1992] Mirkowska G., Salwicki A. *Logika algorytmiczna dla programistów*, Wydawnictwa Naukowo-Techniczne, Warszawa 1992

[Moerkotte i Lockemann 1991] Moerkotte G., Lockemann P.C., *Reactive Consistency Control in Deductive Databases*, ACM Transactions on Database Systems, vol. 16, No.4,December 1991.

[Orman 1998a] Orman L. , *Differential Relational Calculus for Intrgity Maintenance*, IEEE Transactions on Knowledge and Data Engineering, vol. 10, no.2, March/April 1998

[Orman 1998a] Orman L., *Storage and Retrieval of Database Constraints*, Information Systems, vol. 23, No. 6, 1998

[Ozsoyogulu i Snodgrass 1995] Ozsoyogulu G., Snodgrass R., *Temporal and real-time databases: A survey*. IEEE Transactions on Knowledge and Data Engineering, 1995, vol. 7, no.4

[Pankowski 1995] Pankowski T., *Active Objects in Temporal ECA Rules in Intelligent Database Systems*, Proceeding „Decentralized Intelligent Multi Agent Systems” Kraków, November 1995

[Pankowski 1997] Pankowski T. *Więzy integralności a zarządzanie transakcjami w SQLowych systemach baz danych czasu rzeczywistego*, SYSTEMY CZASU RZECZYWISTEGO '97 Materiały konferencyjne, s.143 – 152.

[P<sup>3</sup>odzień i Kraken 2000] P<sup>3</sup>odzien J., Kraken A., *Object Query Optimization Through Detecting Independent Subqueries*, Information Systems, vol. 25, no. 8, 2000.

[Roddick J., F., Patrick J., D. 1992] Roddick J., F., Patrick J., D., *Temporal Semantics in Information Systems - A Survey*, Information Systems Vol.17, No. 3

[Rodrigues i inn. 1999] Rodriguez L., Ogata H., Yano Y., *TVOO: A Temporal Versioned Object-Oriented Data Model*, Information Science 114 (1999)

[Rose i Segev 1993a] Rose E., Segev A. *TOOSQL - A Temporal Object - Oriented Query Language*. Proceedings of the Tenth International Conference on the Entity Relationship Approach. 1993.

[Rose i Segev 1993b] Rose E., Segev A. *TOOA - A Temporal Object - Oriented Algebra*. Proceedings of the European Conference on Object - Oriented Programming. 1993.

[Sarda 1990] Sarda N., L., *Extensions to SQL for Historical Databases*, IEEE Transactions on Knowledge and Data Engineering, vol. 2, no.8, June 1990

[Schwidorski i inn 1994] Schwiderski S., Hartmann T. i Saake G., *Monitoring Temporal Preconditions in a Behaviour Oriented Object Model*. Data & Knowledge Engineering 14 (1994).

[Sistla i Wolfson 1995] Sistla A., Wolfson O. *Temporal Triggers in Active Databases*. IEEE Transactions on Knowledge and Data Engineering, vol. 7, No. 3 June 1995

[Snodgrass 1987] Snodgrass R. *The Temporal Query Language TQuel*, ACM Transactions on Database Systems, vol. 12, no.2, June 1987.

[Snodgras 1996] Snodgras R., Boehlen M., Jensen Ch. S., Steiner A. *Adding Transaction Time to SQL\Temporal* International Organisation For Standardization

[Snodgras 1997] Snodgras R., *Temporal Object - Oriented Databases: A Critical Comparition*, Modern Database Systems, Ch. 19, 1995

[Soljee i Swart 1999] Soljee R.,R, Swart H, C., M, *Three Types of Redundancy in Integrity Checking: An Optimal Solution*, Data & Knowledge Engineering 30, 1999

[Storey V., C. 1996] Storey V., C., Yang H., Goldstein R., C., *Semantic Integrity Constraints in Knowledge-based Database Design System*, Data & Knowledge Engineering 20, 1996

[Subieta 1997] Subieta K. *Obiektywność w bazach danych - koncepcje, nadzieje i fakty (1)*. „Informatyka” listopad 1997.

[Subieta 1998a] Subieta K. *Obiektywność w bazach danych - koncepcje, nadzieje i fakty (2)*. „Informatyka” styczeń 1998.

[Subieta 1998b] Subieta K. *Obiektywność w programowaniu i bazach danych*. Akademicka Oficyna wydawnicza PLJ Warszawa 1998

[Vrbsky 1997] Vrbsky S.V., *A Data Model for Approximate Query Processing of Real – Time*, Databases Data & Knowledge Engineering 21 (1997).

[Vrbsky 1993] Vrbsky S.V. and Liu J.W.S., *APPROXIMATE: A query processor that produces monotonically improving approximate answers*, IEEE Trans. on Knowledge and Data Engineering 5(6) (Dec. 1993), p. 1056-1068.

[Wang i inn. 1997] Wang X., S., Bettini C., Brodsky A., *Logical Design for Temporal Databases with Multiple Granularities*, ACM Transactions on Database Systems, vol. 22, No.2, June 1997.

[Werner 2000] Werner A., *Znacznikowanie czasu ważności danych temporalnej bazy danych*, Zeszyty Naukowe Politechniki Śląskiej 2000

[Zhou 1994a] Zhou H. Überwachung dynamischer Integritätsbedingungen in vergangenheitsbezogener und gemischter temporaler Logik. *Technical Report 01/94, Institut für Informatik, Universität Hannover, February 1994.*

[Zhou 1994b] Zhou H. Anwendung temporaler Logiken zur Überwachung dynamischer Integritätsbedingungen von Datenbanken, Ph. D. Dissertation, Fachbereich Mathematik, Universität Hannover

## DODATEK A. WYBRANE PRAWA LOGIKI TEMPORALNEJ

Poniższy dodatek przedstawia podstawowe prawa i aksjomaty logiki temporalnej, zaczerpnięte z pozycji [1]:

### 1. Prawa logiki FTL

#### 1.1. Prawa dualności:

- a) always- future  $\neg\varphi \Leftrightarrow \neg$  sometime-future  $\varphi$
- b) sometime- future  $\neg\varphi \Leftrightarrow \neg$  always-future  $\varphi$
- c) next  $\neg\varphi \Leftrightarrow \neg$  next  $\varphi$
- d) always-future sometime- future  $\neg\varphi \Leftrightarrow \neg$  sometime-future always-future  $\varphi$
- e) sometime-future always- future  $\neg\varphi \Leftrightarrow \neg$  always-future sometime-future  $\varphi$

#### 1.2. Prawa introspektywności

- a) always- future  $\varphi \Rightarrow \varphi$
- b)  $\varphi \Rightarrow$  sometime- future  $\varphi$
- c)  $\varphi_1$  until  $\varphi_2 \Rightarrow \varphi_1 \vee \varphi_2$
- d)  $\varphi_2 \Rightarrow \varphi_1$  until  $\varphi_2$

#### 1.3. Prawa idempotencji

- a) always-future always- future  $\varphi \Leftrightarrow$  always-future  $\varphi$
- b) sometime-future sometime- future  $\varphi \Leftrightarrow$  sometime-future  $\varphi$
- c) always-future sometime- future always-future sometime- future  $\varphi \Leftrightarrow$  always-future sometime-future  $\varphi$
- d) sometime-future always- future sometime-future always- future  $\varphi \Leftrightarrow$  sometime -future always-future  $\varphi$
- e)  $\varphi_1$  until  $(\varphi_1$  until  $\varphi_2) \Leftrightarrow \varphi_1$  until  $\varphi_2$
- f)  $(\varphi_1$  until  $\varphi_1)$  until  $\varphi_2 \Leftrightarrow \varphi_1$  until  $\varphi_2$

#### 1.4 Prawa absorbcji

- a) always-future sometime - future  $\varphi \Leftrightarrow$  sometime - future always-future sometime - future  $\varphi \Leftrightarrow$  always- future always-future sometime - future  $\varphi \Leftrightarrow$  always-future sometime - future always-future sometime - future  $\varphi \Leftrightarrow$  sometime-future always-future always-future sometime - future  $\varphi$

b) sometime-future always-future  $\varphi \Leftrightarrow$  sometime-future sometime-future always-future  $\varphi \Leftrightarrow$  always-future sometime-future always-future  $\varphi \Leftrightarrow$  always-future sometime-future sometime-future always-future  $\varphi \Leftrightarrow$  sometime-future always-future sometime-future always-future  $\varphi$

### 1.5 Prawa rozdzielczości

- a) always-future  $(\varphi_1 \wedge \varphi_2) \Leftrightarrow$  always-future  $\varphi_1 \wedge$  always-future  $\varphi_2$
- b) sometime-future  $(\varphi_1 \vee \varphi_2) \Leftrightarrow$  sometime-future  $\varphi_1 \vee$  sometime-future  $\varphi_2$
- c) sometime-future always-future  $(\varphi_1 \wedge \varphi_2) \Leftrightarrow$  sometime-future always-future  $\varphi_1 \wedge$  sometime-future always-future  $\varphi_2$
- d) always-future sometime-future  $(\varphi_1 \vee \varphi_2) \Leftrightarrow$  always-future sometime-future  $\varphi_1 \vee$  always-future sometime-future  $\varphi_2$
- e)  $\text{next}(\varphi_1 \wedge \varphi_2) \Leftrightarrow \text{next } \varphi_1 \wedge \text{next } \varphi_2$
- f)  $\text{next}(\varphi_1 \vee \varphi_2) \Leftrightarrow \text{next } \varphi_1 \vee \text{next } \varphi_2$
- g)  $\text{next}(\varphi_1 \Rightarrow \varphi_2) \Leftrightarrow (\text{next } \varphi_1 \Rightarrow \text{next } \varphi_2)$
- h)  $\text{next}(\varphi_1 \Leftrightarrow \varphi_2) \Leftrightarrow (\text{next } \varphi_1 \Leftrightarrow \text{next } \varphi_2)$
- i)  $(\varphi_1 \wedge \varphi_2) \text{ until } \varphi_3 \Leftrightarrow \varphi_1 \text{ until } \varphi_3 \wedge \varphi_2 \text{ until } \varphi_3$
- j)  $\varphi_1 \text{ until } (\varphi_2 \vee \varphi_3) \Leftrightarrow \varphi_1 \text{ until } \varphi_2 \vee \varphi_1 \text{ until } \varphi_3$

### 1.6 Prawa rekursywnej równoważności

Prawa rekursywnej równoważności używane są do tworzenia grafów temporalnych w rozdziale II.

- a) always-future  $\varphi \Leftrightarrow \varphi \wedge \text{next always-future } \varphi_2$
- b) sometime-future  $\varphi \Leftrightarrow \varphi \vee \text{next sometime-future } \varphi_2$
- c)  $\varphi_1 \text{ until } \varphi_2 \Leftrightarrow \varphi_2 \vee (\varphi_1 \wedge \text{next } (\varphi_1 \text{ until } \varphi_2))$

## 2. Prawa logiki MTL

- a) always-future  $\varphi_1 \vee$  always-future  $\varphi_2 \Leftrightarrow$  always-future  $(\text{always-past } \varphi_1 \vee \text{always-past } \varphi_2)$   
 sometime-future  $\varphi_1 \wedge$  sometime-future  $\varphi_2 \Leftrightarrow$  sometime-future  $(\text{sometime-past } \varphi_1 \wedge \text{sometime-past } \varphi_2)$
- b) sometime-future always-future  $\varphi_1 \vee$  sometime-future always-future  $\varphi_2 \Leftrightarrow$  sometime-future always-future  $(\text{previous } (\varphi_1 \text{ since } (\varphi_1 \wedge \neg \varphi_2)) \vee \varphi_2)$
- c) always-future sometime-future  $\varphi_1 \wedge$  always-future sometime-future  $\varphi_2 \Leftrightarrow$  always-future sometime-future  $(\text{previous } (\neg \varphi_1 \text{ since } \varphi_2)) \wedge \varphi_1$
- d) always-future  $\varphi \Leftrightarrow$  always-future sometime-future  $(\text{always-past } \varphi)$
- e) always-future  $\varphi \Leftrightarrow$  sometime-future always-future  $(\text{always-past } \varphi)$
- f) sometime-future  $\varphi \Leftrightarrow$  always-future sometime-future  $(\text{sometime-past } \varphi)$

- g) sometime-future  $\varphi \Leftrightarrow$  sometime-future always-future (sometime-past  $\varphi$ )
- h)  $\varphi_1$  until  $\varphi_2 \Leftrightarrow$  sometime-future ( $\varphi_1 \wedge$  previous always-past  $\varphi_2$ )



## DODATEK B. LOGIKA TRÓJWARTOŚCIOWA

Przedstawiony dodatek przedstawia „tabele prawd” podstawowych operatorów logicznych :  
 $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Leftrightarrow$ ,  $\Rightarrow$  dla logiki trójwartościowej wykorzystywanej przy weryfikacji formuł FTL.

| $\varphi$      | $\neg \varphi$ |
|----------------|----------------|
| <b>true</b>    | false          |
| <b>unknown</b> | unknown        |
| <b>false</b>   | true           |

a) negacja

| $\Rightarrow$  | <b>true</b> | <b>unknown</b> | <b>false</b> |
|----------------|-------------|----------------|--------------|
| <b>true</b>    | true        | unknown        | false        |
| <b>unknown</b> | true        | true           | unknown      |
| <b>false</b>   | true        | true           | true         |

b) implikacja

| $\vee$         | <b>true</b> | <b>unknown</b> | <b>false</b> |
|----------------|-------------|----------------|--------------|
| <b>true</b>    | true        | true           | true         |
| <b>unknown</b> | true        | unknown        | unknown      |
| <b>false</b>   | true        | unknown        | false        |

c) alternatywa

| $\wedge$       | <b>true</b> | <b>unknown</b> | <b>false</b> |
|----------------|-------------|----------------|--------------|
| <b>true</b>    | true        | unknown        | false        |
| <b>unknown</b> | unknown     | unknown        | false        |
| <b>false</b>   | false       | false          | false        |

d) koniunkcja

| $\Leftrightarrow$ | <b>true</b> | <b>unknown</b> | <b>false</b> |
|-------------------|-------------|----------------|--------------|
| <b>true</b>       | true        | unknown        | false        |
| <b>unknown</b>    | unknown     | true           | unknown      |
| <b>false</b>      | false       | unknown        | true         |

e) równoważność

