Zachodniopomorski Uniwersytet Technologiczny
w Szczecinie

Algorytmy obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności z ograniczeniami afinicznymi

Rozprawa doktorska

Wydział Informatyki

mgr inż. Tomasz Klimek

Promotor
prof. dr hab. inż. Włodzimierz Bielecki

Szczecin, 2011

Podziękowania

Składam serdeczne podziękowania wszystkim, którzy służąc mi swoją pomocą przyczynili się do powstania niniejszej pracy, a w szczególności:

- prof. dr hab. inż. Włodzimierzowi Bieleckiemu za opiekę naukową w trakcie moich studiów doktoranckich oraz cenną pomoc merytoryczną podczas realizacji pracy naukowej,
- dr inż. Markowi Pałkowskiemu za pomoc w trakcie przeprowadzanych badań eksperymentalnych, z wykorzystaniem opracowanego przez niego narzędzia,
- moim Rodzicom za wszelką pomoc w trakcie pisania pracy, cierpliwość i motywację.

To czego szukamy, znajduje się zawsze w zasięgu naszego wzroku, wystarczy tylko rozejrzeć się wokół uważnie i w skupieniu, aby odkryć, dokąd chce nas zaprowadzić los i czy warto nam w tym kierunku pójść. Albowiem wszystko się ze sobą łączy i ma określony sens.

— Paulo Coelho
Zahir

Spis treści

1. Wstęp

Niniejszy rozdział, stanowi wprowadzenie do problematyki poruszanej w rozprawie doktorskiej. W podrozdziale 1.1, zaprezentowano pokrótce opis stanu wiedzy w zakresie poruszanego tematu, oraz prowadzonych badań. Przedstawiono uzasadnienie podjętego zagadnienia, lukę poznawczą, w której zawiera się praca, oraz dziedzinę nauk do jakiej praca została zakwalifikowana. W kolejnym podrozdziale, zdefiniowano cel pracy, wraz z tezą badawczą pracy. Przedstawiono przesłanki przemawiające za wybranym kierunkiem badań, oraz korzyści wynikające z tego wyboru. Podrozdział 1.3, obejmuje strukturę pracy, wraz z krótką charakterystyką poszczególnych rozdziałów.

1.1 Stan problemu

Wyznaczenie dokładnego tranzytywnego domknięcia grafu, w przypadku wielu algorytmów, uznawane jest powszechnie, jako podstawowy czynnik, determinujący prawidłowość dokonywanych przez nich obliczeń. Dotyczy to przede wszystkim problemów takich jak:

1. **Analiza hierarchii** (systemy baz danych : ***BOM*** ang. ***B****ill-****O****f-****M****aterial*, który działa na hierarchii części pewnego wyrobu np. *„Podaj jaka jest całkowita masa silnika”*) [3, 48, 55, 79, 80, 101]

2. **Analiza przepływów** (sieci : transportowe, kolejowe, gazowe, wodociągowe, kanalizacyjne, kablowe, komputerowe i telekomunikacyjne np. *„Podaj*

*połączenie pomiędzy miastem **A** a miastem **B** o minimalnym czasie podróży"*) [16, 37, 41, 42, 111]

3. **Analiza zależności czasowych** (systemy równoległe i rozproszone : ekstrakcja drobno- i gruboziarnistej równoległości np. *„Wyznacz niezależne fragmenty kodu do zrównoleglenia"*) [10, 12, 14, 15, 77, 84, 99].

Jeżeli zagadnienie dopuszcza możliwość ponowienia procesu obliczeniowego, w przypadku zmiany parametrów wejściowych (co może wystąpić, w sytuacji dodania lub usunięcia kolejnych wierzchołków), wówczas za prawidłowe uznaje się zastosowanie jednego z dobrze znanych algorytmów przeszukiwania w głąb (ang. depth-first search) lub przeszukiwania wszerz (ang. breadth-first search) [66]. Przez przeszukiwanie grafu, rozumie się systematyczne przechodzenie wzdłuż jego krawędzi, w celu odwiedzenia wszystkich wierzchołków. Algorytm przeszukiwania grafu, w takim ujęciu, może służyć do zbierania informacji o strukturze grafu. Dotychczas zaproponowano kilka sposobów rozwiązania tego problemu, które zaprezentowano w licznych publikacjach [1, 36, 39, 41, 42, 52, 53, 61, 62, 105]. Większość z owych algorytmów, rozpoczyna się od przeszukania wejściowego grafu, w celu zdobycia takiej strukturalnej informacji. Inne z kolei, są modyfikacjami podstawowych algorytmów przeszukiwania [4, 37, 55, 56, 58, 63, 68, 80, 96, 100, 102, 103, 109] . Niestety tylko niewielka część z powyższych metod, dotyczy bezpośrednio grafów sparametryzowanych [9, 15, 17, 18, 20, 21, 22, 30, 31, 72, 107], czyli takich których liczbę wierzchołków określa się wyrażeniem zawierającym parametry(zmienne symboliczne), a sam proces obliczania tranzytywnego domknięcia, przeprowadzany jest jednokrotnie, z gwarancją że otrzymany wynik pozostanie prawidłowy, niezależnie od zmieniających się wartości parametrów wejściowych. Przyczyna takiego stanu rzeczy, uwarunkowana jest bowiem specyfiką danego zadania, która w sposób bezpośredni determinuje konieczność doboru, zarówno algorytmu jak i odpowiedniej formy reprezentacji grafu [30, 31, 38, 48, 51, 57, 60, 65, 83].

Postęp technologiczny w dziedzinie sprzętu komputerowego, zaobserwowany na przełomie ostatnich lat, niewątpliwie przyczynił się do gwałtownego zapotrzebowania na zwiększenie mocy obliczeniowej jednostek wykonawczych. Tendencja ta, ma charakter stały i w najbliższej przyszłości będzie ulegać nasileniu. Niestety aktualne ograniczenia technologiczne, nie pozwalają przyspieszać taktowania procesorów w takim tempie, jakie zakłada prawo Moore`a, dlatego też sensowne staje się wykorzystanie systemów wieloprocesorowych lub procesorów wielordzeniowych.

Początki prac nad wykorzystaniem wielu procesorów do współbieżnego wykonywania obliczeń, sięgają lat 60-tych ubiegłego wieku. Pojawienie się komputerów wieloprocesorowych, stało się zatem naturalnym wyjściem naprzeciw tym potrzebom i tym samym zapoczątkowało rozważania, zmierzające ku efektywnemu wykorzystaniu ich zasobów, zarówno w środowiskach akademickich jak i biznesowych [84, 99].

Statystycznie najwięcej czasu, jaki procesor potrzebuje do realizacji zadania, zajmuje wykonanie instrukcji, zawartych w pętlach programowych, w szczególności w pętlach składających się z wielu iteracji. Ze względu na czas wykonania, transformacja tych obszarów programu, może przynieść największe korzyści. Jak dotąd zaproponowano wiele rozwiązań umożliwiających wykonanie automatycznej transformacji pętli [14, 40, 49, 50, 76, 77, 78], jednak żadne z nich, nie pozwala na wyszukanie pełnej równoległości zawartej w pętli programowej. Proces jej poszukiwania kontynuowano zatem, w pracach [14, 15, 84, 99], w których to przedstawiono algorytmy pozwalające na wyszukanie maksymalnej liczby niezależnych fragmentów kodu dla pętli, poprzez dokonanie podziału, jej przestrzeni iteracji w pewien usystematyzowany sposób. **Fragmentem kodu** nazywamy wówczas, zbiór wszystkich iteracji powiązanych zależnościami między iteracjami lub instrukcjami pętli. Fragment kodu jest niezależny, jeżeli nie istnieje żadna zależność pomiędzy jego iteracjami i iteracjami należącymi do innego fragmentu kodu [84]. W celu znalezienia niezależnych fragmentów kodu, konieczne jest przeprowadzenie dokładnej analizy zależności, której zadaniem jest określenie, w jaki sposób iteracje, bądź instrukcje pętli, zależą od siebie w trakcie wykonania. Jeśli za formę reprezentacji zależności przyjmiemy graf, w którym wierzchołki, symbolizują kolejne iteracje lub instrukcje pętli, a krawędzie występujące pomiędzy nimi zależności, to skuteczność wspomnianych algorytmów, w znacznym stopniu uzależniona jest od możliwości obliczenia tranzytywnego domknięcia takiego grafu, czyli dostarczenia informacji dotyczącej bezpośrednich i pośrednich połączeń pomiędzy jego wierzchołkami. Autorzy prac [14, 15, 84, 99], we wnioskach z przeprowadzonych badań, niejednokrotnie zwracali uwagę, na konieczność opracowania nowych, bądź usprawnienia obecnych algorytmów obliczania tranzytywnego domknięcia relacji zależności, w celu zwiększenia liczby pętli, dla których zastosowanie proponowanych przez nich rozwiązań byłoby możliwe.

W niniejszej dysertacji zaprezentowano autorskie algorytmy, pozwalające na obliczenie dokładnego tranzytywnego domknięcia relacji zależności lub jego przybliżenia górnego, w przypadkach gdy, inne obecnie znane tego typu rozwiązania zawodzą [9, 48, 72, 107]. Algorytmy te, mogą być stosowane z powodzeniem dla relacji

zależności, w przypadku których ograniczenia zawierają zmienne symboliczne (parametry).

W pracy tej, szczególny nacisk, położony został na obliczaniu tranzytywnego domknięcia sparametryzowanych relacji zależności w sposób iteracyjny. W przypadku, porażki, tzn. gdy brak jest możliwości osiągnięcia warunku zbieżności rozwiązań iteracyjnych, wówczas następuje próba połączenia technik iteracyjnych i nieiteracyjnych, poprzez dokonanie podziału przestrzeni iteracji pętli na podprzestrzenie. Tranzytywne domknięcie relacji zależności obliczane jest wówczas w sposób nieiteracyjny, indywidualnie w ramach danej podprzestrzeni i iteracyjnie na całej przestrzeni iteracji pętli, w celu połączenia otrzymanych wcześniej wyników, uzyskanych na poszczególnych podprzestrzeniach.

Celem prowadzonych badań, było wypełnienie luki poznawczej, poprzez opracowanie algorytmów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności z ograniczeniami afinicznymi, z zamiarem zastosowania otrzymanych rozwiązań, do ekstrakcji zarówno drobno- i gruboziarnistej równoległości, dla szerszego spektrum pętli programowych w porównaniu z istniejącymi metodami.

Niniejsza praca, leży w dziedzinie nauk technicznych i dotyczy dyscypliny informatyka. Problematyka w niej poruszana, mieści się w zakresie teorii grafów i przetwarzania równoległego. Wyniki badań, zostały przedstawione w autorskich publikacjach w czasopismach i na konferencjach polskich [16, 17, 20, 22, 24], oraz międzynarodowych [18, 19, 21]. Opracowane algorytmy, stanowią rozwinięcie zagadnień związanych z obliczaniem tranzytywnego domknięcia grafów sparametryzowanych.

1.2 Cel i teza badawcza pracy

Aktualnie dostępne algorytmy obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności [9, 48, 72, 79, 106], pozwalają na uzyskanie dokładnego wyniku, w pewnym ograniczonym stopniu i zakresie. Najczęściej ograniczenia te, dotyczą liczby koniunkcji składających się na relację zależności, rodzaju zawartych w niej ograniczeń, jak i również sposobu podziału jej przestrzeni iteracji. W odniesieniu do liczby koniunkcji wchodzących w skład relacji zależności, dziedzina problemu może zostać zawężona, do obliczania tranzytywnego domknięcia prostych (składających się z pojedynczej koniunkcji), lub złożonych (składających się z wielu koniunkcji) relacji zależności. W drugim przypadku, zawężenie dziedziny

problemu, dotyczy bezpośrednio rodzaju występujących ograniczeń, w ramach relacji zależności tzn. czy są to ograniczenia afiniczne, czy też nieafiniczne. Ostatni typ ograniczeń, dotyczy sytuacji, w której relacja nie opisuje wszystkich zależności na całej przestrzeni iteracji pętli, tylko dokonuje jej podziału, na podprzestrzenie, bądź też łańcuch jej zależności został przerwany, na skutek występowania wierzchołków, które zostały wykluczone ze zbioru, definiującego jej dziedzinę.

Zgodnie z bieżącym stanem wiedzy, nie istnieje sformalizowane podejście, umożliwiające obliczenie dokładnego tranzytywnego domknięcia relacji, w ogólnym przypadku [9, 30, 31, 72]. Jednak, nałożenie jednego lub kombinacji powyższych ograniczeń, pozwala na zawężenie problemu, do wybranych klas relacji zależności, o określonych własnościach, dla których uzyskanie dokładnego wyniku, w skończonym czasie jest wykonalne.

Celem niniejszej pracy jest opracowanie algorytmów umożliwiających obliczenie dokładnego tranzytywnego domknięcia sparametryzowanych relacji zależności z ograniczeniami afinicznymi lub jego przybliżenia górnego wraz z zastosowaniem opracowanych algorytmów do wyeksponowania drobno- i gruboziarnistej równoległości w pętlach programowych.

Badania prowadzone w ramach pracy, służą do wykazania prawdziwości następującej tezy badawczej.

Możliwe jest opracowanie algorytmów obliczania dokładnego tranzytywnego domknięcia sparametryzowanych relacji zależności z ograniczeniami afinicznymi dla większego spektrum pętli programowych w porównaniu z istniejącymi metodami.

Ograniczeniem klasycznych algorytmów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności, w sposób iteracyjny [48, 51, 52, 55] jest brak możliwości uzyskania dokładnego rozwiązania, dla relacji zależności reprezentujących graf o topologii łańcucha. Powodem jest niespełnienie warunku zbieżności, czyli rozpoznania wszystkich zależności przechodnich. Postępowanie kończy się wówczas z wynikiem w postaci przybliżenia dolnego tranzytywnego domknięcia relacji (niepełny zestaw zależności przechodnich). Z drugiej strony, podejście polegające na nieiteracyjnym obliczaniu tranzytywnego domknięcia sparametryzowanych relacji zależności, zaproponowane w [9, 72], w przypadku wielu relacji, może doprowadzić do powstania nieistniejących, tzw. fałszywych zależności przechodnich. O nieistniejącej (fałszywej) zależności przechodniej mówimy wtedy, gdy nie wynika ona z przebiegu zależności bezpośrednich. Uzyskany w ten sposób wynik, przyjmuje wtedy postać przybliżenia górnego tranzytywnego domknięcia relacji zależności (wprowadzenie

zależności przechodnich nie wynikających z przebiegu zależności podstawowych), co w konsekwencji uniemożliwia uzyskanie pełnej równoległości dla pętli programowych, w ramach zastosowań w zakresie przetwarzania równoległego i rozproszonego. Celem opracowania algorytmów zawartych w niniejszej pracy jest powiększenie zbioru pętli, dla których obliczenie dokładnego tranzytywnego domknięcie ich relacji zależności jest możliwe. Przekłada się to bowiem bezpośrednio, na wzrost stopnia ich równoległości.

Wzrost stopnia równoległości rozumiany jest jako:

- zrównoleglenie pętli programowej, dla której inne znane autorowi rozwiązania zawodzą, tj. nie pozwalają na uzyskanie kodu równoległego,
- uzyskanie większej liczby równoległych fragmentów kodu.

Uzyskanie kodu równoległego dla pętli, w której inne metody zawodzą i odnotowanie przyspieszenia obliczeń, pozwala stwierdzić jednoznacznie, że zostało dokonane zwiększenie jej stopnia równoległości. Zwiększenie stopnia równoległości, to również możliwość uzyskania większej liczby fragmentów kodu. Uzyskany kod równoległy, cechuje się wówczas lepszym wskaźnikiem skalowalności, a możliwość wykonania większej liczby iteracji pętli równolegle, wiąże się ze zmniejszeniem części sekwencyjnej.

Zagadnienia poruszane w niniejszej rozprawie doktorskiej, stanowią kontynuację badań zawartych w pracach [3, 4, 9, 48, 51, 62, 72, 83, 106, 107, 109], w których to zaproponowano algorytmy obliczania tranzytywnego domknięcia relacji zależności, zarówno w sposób iteracyjny [3, 48, 51, 62, 72, 83] jak i nieiteracyjny [9, 72. 106, 107, 109]. Przeanalizowano mocne i słabe strony tych rozwiązań. Wyciągnięte wnioski, posłużyły do opracowania algorytmów hybrydowych, o większym zakresie zastosowania, łączących techniki iteracyjnego i nieiteracyjnego obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności. Uzyskane przez nie wyniki, stanowią newralgiczny element wielu algorytmów, dokonujących ekstrakcji równoległości drobno- i gruboziarnistej [84, 87, 99].

W zakresie przetwarzania równoległego, ziarnistość obliczeń (ang. *granularity*) wyznacza stosunek czasu obliczeń, do czasu potrzebnego na synchronizację i komunikację. W przypadku, gdy czas obliczeń jest mniejszy od czasu zarządzania wątkami (stosunek czasów jest mniejszy lub zbliżony do 1), mówimy o drobnoziarnistej równoległości (ang. *fine-grained parallelism*). Drobnoziarnistość, przy prawidłowym mapowaniu zadań do poszczególnych procesorów, umożliwia maksymalną redukcję bezczynności jednostek przetwarzających, oraz efektywne zarządzanie obciążeniem systemu wieloprocesorowego. W związku z tym, podział kodu na drobne ziarna, preferowany jest w systemach, w których koszt zarządzania wątkami jest niewielki, a

balansowanie obciążeniem ma zasadnicze znaczenie, dla wydajności całego systemu. Jeżeli natomiast czas obliczeń jest większy od czasu potrzebnego na zarządzanie wątkami, to mamy do czynienia z gruboziarnistą równoległością (ang. *coarse-grained parallelism*). Zaletą takiego podziału, jest możliwość zwiększenia wydajności algorytmu, poprzez: redukcję czasu potrzebnego na synchronizację i komunikację, zwiększenie lokalności kodu, oraz zmniejszenie zapotrzebowania na pamięć. Ze względu na duże czasy przestojów i różną wielkość ziaren, główną niedogodnością jest trudność w balansowaniu obciążeniem na poszczególnych jednostkach przetwarzających. Gruboziarnisty podział, preferowany jest w systemach, w których koszt komunikacji między jednostkami przetwarzającymi jest znaczny (np. systemy rozproszone) [99].

Za podjęciem zagadnienia będącego tematem niniejszej pracy, przemawiają przesłanki naukowe i ekonomiczne. Biorąc pod uwagę przesłanki naukowe, ważnym wydaje się udzielenie odpowiedzi na pytanie, jaka jest maksymalna liczba niezależnych fragmentów kodu, dostępna w zadanej pętli programowej. Niestety, ograniczenia obecnie znanych algorytmów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności [72], znacząco utrudniły odpowiedź na powyższe pytanie autorom prac [84, 99], którzy dla wielu przykładów badanych pętli, zdecydowali się na przebieranie iteracji fragmentów kodu w czasie wykonywania programu, co znacząco zmniejszyło uzyskane przyspieszenie.

Przesłanki ekonomiczne poruszonego zagadnienia, to możliwość zastosowania uzyskanych rozwiązań w procesie całkowitej automatyzacji transformacji kodu sekwencyjnego na odpowiednik równoległy. Wiąże się z tym zmniejszenie kosztów oraz redukcja błędów ludzkich, związanych z ręcznym przekształcaniem kodu. W ostatnich latach producenci oprogramowania ogólnodostępnego i komercyjnego, coraz częściej wykorzystują w swoich aplikacjach kod równoległy w celu przyspieszenia obliczeń. Przetwarzanie równoległe, znajduje zastosowanie w popularnych aplikacjach graficznych, programach do przetwarzania dokumentów, czy pakietach biurowych [84].

Dane wejściowe algorytmów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności stanowi kod źródłowy zapisany w języku Petit [70]. W celu umożliwienia konwersji pętli programowych zapisanych w języku Fortran, wykorzystany został translator Fortran2Petit. Wyjściem algorytmów, po uprzednim wykryciu i redukcji zależności [99] jest tranzytywne domknięcie relacji zależności.

Do wyznaczenia zależności wykorzystano narzędzie Petit [70]. Operacje na zbiorach i relacjach (wyjście programu Petit) przeprowadzono w ramach biblioteki Omega [71] . Stanowi ona integralną część projektu Omega, opracowanego na na

Uniwersytecie Maryland (University of Maryland). Badania przeprowadzono na maszynie z procesorem wielordzeniowym Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00 GHz.

Zaproponowane algorytmy, zweryfikowano pod względem dokładności uzyskiwanych wyników.

1.3 Struktura pracy

Praca podzielona została na osiem rozdziałów. Zawartość merytoryczna każdego z nich opisana została poniżej.

Pierwszy rozdział stanowi wprowadzenie do tematyki pracy. Przedstawiono w nim uzasadnienie wyboru tematu, tło omawianych zagadnień, cel, tezę badawczą oraz strukturę pracy.

W rozdziale 2 przedstawiono wprowadzenie z zakresu teorii grafów i przetwarzania równoległego. Omówione zostały podstawowe pojęcia związane z rodzajem grafów i sposobem ich reprezentacji. Podano definicję ścieżki w grafie, tranzytywnego domknięcia grafu, aproksymacji tranzytywnego domknięcia grafu i definicję tranzytywnej redukcji grafu. Omówione zostały również, podstawowe pojęcia związane z analizą zależności i ziarnistością obliczeń. Wyjaśniono podstawy arytmetyki Presburgera, która jest niezbędna do zrozumienia reprezentacji zależności i tym samym reprezentcji grafów sparametryzowanych, oraz mechanizmu działania przedstawionych algorytmów.

Rozdział 3 poświęcony został klasom relacji zależności i ich rozpoznaniu. Przedstawiono topologię grafu zależności, jak i również zaprezentowano obecnie znaną klasyfikację relacji zależności. Zaprezentowano także autorskie algorytmy umożliwiające ich rozpoznanie w sposób formalny.

Kolejny rozdział zawiera opis zaproponowanych algorytmów obliczania tranzytywnego domknięcia relacji zależności, wraz z przykładami obrazującymi ich działanie. Dodatkowo, opis ten poprzedzono niezbędną wiedzą umożliwiającą zrozumienie proponowanych rozwiązań.

Rozdział 5, zawiera prezentację prac pokrewnych dotyczących tematyki poruszanej w niniejszej dysertacji. Dokonano w nim porównania zakresu stosowalności obecnie znanych tego typu rozwiązań.

W rozdziale 6 zaprezentowano badania eksperymentalne przeprowadzone przy pomocy autorskiego modułu, realizującego algorytmy opisane w poprzednim rozdziale,

w ramach narzędzia autorstwa dr inż. Marka Pałkowskiego opisanego w pracy [84]. Rozdział kończy podsumowanie wraz z wnioskami wynikającymi z pracy.

W rozdziale 7 zaprezentowano przykłady zastosowania relacji tranzytywnego domknięcia, do ekstrakcji drobno- i gruboziarnistej równoległości zawartej w pętlach programowych.

W ostatnim rozdziale zawarto wnioski końcowe, dotyczące realizacji podjętych w pracy zagadnień.

2. Podstawowe definicje i pojęcia

Na przełomie ostatnich lat teoria grafów (ang. *graph theory*) ugruntowała swoją silną pozycję, jako istotny aparat matematyczny w wielu różnych dziedzinach nauki, począwszy od rachunku operacyjnego, zagadnień z zakresu chemii, genetyki i lingwistyki po elektronikę, geografię i architekturę [34, 36, 41, 42, 82, 104, 110, 111]. Jednocześnie okazała się dyscypliną matematyczną wartą samodzielnych badań. Najważniejsze jej osiągnięcia były rezultatem prób rozwiązania konkretnych zadań praktycznych. Nie sposób nie docenić istoty rozwiązania problemów takich jak: rozwiązanie Eulera problemu mostów królewieckich [42], prace dotyczące zliczania cząstek chemicznych [111] i prace Kirchhoffa nad obwodami elektrycznymi [41]. Dzisiejsze zainteresowanie tą tematyką w znacznym stopniu spowodowane jest tym, że jest ona nie tylko elegancką teorią matematyczną, ale ma rozliczne zastosowania w innych dziedzinach.

Najprościej rzecz ujmując grafy przedstawiają pewien zbiór punktów i pokazują w jaki sposób są one między sobą połączone, dzięki czemu mogą one posłużyć do poglądowego przedstawienia złożonych sytuacji i relacji z którymi spotykamy się w codziennym życiu [34]. Jest to bardzo wygodny i stosunkowo prosty w implementacji sposób przedstawienia często skomplikowanych zależności. Z punktu widzenia informatyki, znajdują one wszechstronne zastosowanie do reprezentowania i przetwarzania danych, oraz do modelowania obiektów rzeczywistych, np. układów komunikacyjnych w miastach [95], sieci połączeń telefonicznych [82], układów elektronicznych [104], algorytmów itd.

Z uwagi na ciągły wzrost ilości danych koniecznych do przetworzenia, wiążący się nierozerwalnie ze wzrostem zapotrzebowania na moc obliczeniową systemów informatycznych, szczególnie interesujące stało się wykorzystanie teorii grafów w

zagadnieniach dotyczących przetwarzania równoległego i rozproszonego [1, 24, 26, 27, 43, 54, 72, 75, 84, 87, 99]. Nie trudno dziś zauważyć, iż moce obliczeniowe pojedynczych jednostek przetwarzających w wielu dziedzinach nauki, przemysłu, jak również w codziennym życiu, stały się niewystarczające. Jednym z bardziej popularnych rozwiązań problemu braku mocy obliczeniowej jest udostępnienie użytkownikowi systemów posiadających więcej niż jedną jednostkę przetwarzającą [6, 47, 75]. Podstawową różnicę pomiędzy programami dedykowanymi na maszyny wieloprocesorowe, a programami sekwencyjnymi stanowi liczba wątków odpowiedzialna za rozwiązanie określonego problemu. W programach sekwencyjnych, obliczenia realizowane są przez pojedynczy wątek, natomiast w programach dedykowanych na maszyny równoległe problem rozwiązywany jest przez wiele wątków, mapowanych na więcej niż jedną jednostkę przetwarzającą [26, 27, 40, 77, 78]. Podział problemu na mniejsze zadania i przydzielanie ich do niezależnych wątków [29], wiąże się z koniecznością uwzględnienia wielu dodatkowych czynników [49, 50]. Istotna staje się zatem, możliwość określenia zbioru wierzchołków osiągalnych z dowolnego wierzchołka początkowego, w celu wyznaczenia fragmentów kodu do zrównoleglenia [10, 11, 12, 13, 14, 15]. Wymaganie to dotyczy bezpośrednio problemu wyznaczenia domknięcia przechodniego w grafie skierowanym [1, 5, 9, 16, 17, 20, 21, 30, 31, 39, 52, 58, 68, 69, 83, 96, 100].

W niniejszym rozdziale, przedstawione zostały zagadnienia związane z przetwarzaniem równoległym i podstawami teorii grafów. Omówione zostały rodzaje pętli programowych, zależności w pętlach i ziarnistość kodu. Zaprezentowano rodzaje grafów, sposoby ich reprezentowania, wraz z definicją tranzytywnego domknięcia grafu. W celu zrozumienia proponowanych w niniejszej dysertacji rozwiązań przytoczona została arytmetyka Presburgera, za pomocą której reprezentowane są zależności w pętlach. Operacje arytmetyki Presburgera są podstawą aparatu matematycznego zaproponowanych algorytmów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności.

2.1 Wprowadzenie do teorii grafów

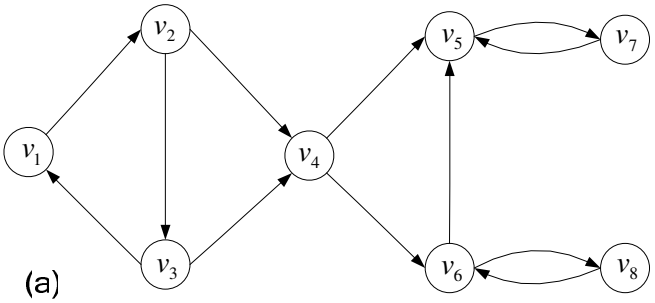
Matematyka autorowi tej pracy, zawsze kojarzyła się przede wszystkim z arytmetyką, ale współcześni matematycy rzadko się nią zajmują. Wynika to stąd, że arytmetyka pozwala spojrzeć na rzeczywistość zaledwie z jednego, bardzo wąskiego punktu widzenia, a współczesna rzeczywistość jest o wiele bardziej skomplikowana i wymaga rozważenia różnych punktów widzenia. Takim a nie innym punktem widzenia

jest właśnie **teoria grafów**, która najprościej rzecz ujmując zajmuje się opisywaniem połączeń między rzeczami [110]. Z teoretycznego punktu widzenia obojętne jest czy tymi rzeczami są domy łączące się siecią ulic, czy komputery połączone kablami lub satelitami w sieć komputerową. Podobnie jak i obojętne jest czy liczymy domy lub odległości pomiędzy domami, czy robimy to samo z komputerami w przypadku arytmetyki.

Od strony formalnej za **teorię grafów** uważa się dział matematyki zajmujący się badaniem i rozwijaniem algorytmów wyznaczających pewne właściwości grafów [95]. Powszechnie uznawana jest ona za jedno ze znaczących pól działania informatyki, a opracowane w ramach niej algorytmy stosuje się do rozwiązywania wielu zadań praktycznych, często w dziedzinach na pozór nie związanych z grafami.

2.1.1 Definicja grafu skierowanego

Graf skierowany (lub **digraf)** *G* jest opisany parą (*V*, *E*), gdzie *V* jest zbiorem skończonym, a *E* jest relacją binarną w *V*. Zbiór *V* jest nazywany **zbiorem wierzchołków** *G*, a jego elementy są nazywane **wierzchołkami** [36].



	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0
3	1	0	0	1	0	0	0	0
4	1	0	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0
6	0	0	0	0	1	0	0	1
7	0	0	0	0	1	0	0	0
8	0	0	0	0	0	1	0	0

(b)

$Adj(v_1)=\{v_2\}$
$Adj(v_2)=\{v_3, v_4\}$
$Adj(v_3)=\{v_1, v_4\}$
$Adj(v_4)=\{v_5, v_6\}$
$Adj(v_5)=\{v_7\}$
$Adj(v_6)=\{v_5, v_8\}$
$Adj(v_7)=\{v_5\}$
$Adj(v_8)=\{v_6\}$

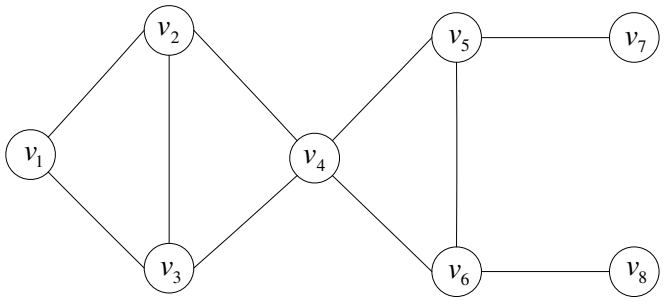
(c)

Rys. 2.1 Przykład grafu skierowanego (a) wraz z formą jego reprezentacji (b) macierz sąsiedztwa (c) lista sąsiedztwa [83]

Zbiór E jest nazywany **zbiorem krawędzi** G, a jego elementy nazywamy **krawędziami**. Na rysunku 2.1 przedstawiono graf skierowany o zbiorze wierzchołków $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. (wierzchołki są przedstawione jako kółka, a krawędzie jako strzałki).

2.1.2 Definicja grafu nieskierowanego

W **grafie nieskierowanym** $G = (V, E)$, zbiór krawędzi E to zbiór *nieuporządkowanych* par wierzchołków. Oznacza to, że krawędź jest zbiorem $\{u, v\}$, gdzie $u, v \in V$ i $u \neq v$ [36]. Do oznaczenia krawędzi będziemy używać zapisu (u, v) zamiast oznaczenia w postaci zbioru $\{u, v\}$; zapisy (u, v) i (v, u) oznaczają tę samą krawędź. Na rysunku 2.2 przedstawiono graf nieskierowany o zbiorze wierzchołków $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$.



Rys. 2.2 Graf niekierowany [83]

2.1.3 Formy reprezentacji grafu

Mimo iż wygodnie jest przedstawiać graf w postaci rysunku, na którym punkty są połączone liniami, to taka reprezentacja może być nieodpowiednia, jeśli chcemy przechować duży graf w pamięci komputera [34, 36, 42].

Istnieją dwa standardowe sposoby reprezentowania grafu $G = (V, E)$: za pomocą **list sąsiedztwa** lub **macierzy sąsiedztwa** [45].

W reprezentacji grafu $G = (V, E)$ za pomocą **list sąsiedztwa** dana jest tablica Adj zawierająca $|V|$ list, po jednej dla każdego wierzchołka z V. Dla każdego $u \in V$ elementami listy sąsiedztwa $Adj[u]$ są wszystkie wierzchołki v takie, że krawędź

$(u,v)\in E$; tzn. *Adj*[*u*] składa się ze wszystkich wierzchołków sąsiadujących z *u* w *G*. Każda krawędź jest reprezentowana dwukrotnie. Jeśli jest $|V|$ węzłów, to do reprezentowania węzła potrzeba $\lceil \log_2 |V| \rceil$ bitów. Reprezentacja ta wymaga zatem w sumie $2 \cdot \lceil \log_2 |V| \rceil \cdot |E|$ bitów, gdzie $|E|$ jest liczbą krawędzi. Na rysunku 2.1c widzimy przykład takiej reprezentacji. Zazwyczaj preferuje się reprezentację listową, ponieważ umożliwia ona przedstawienie w zwarty sposób grafów rzadkich – to jest takich, dla których $|E|$ jest dużo mniejsze od $|V|^2$.

Inny przydatny sposób reprezentacji wykorzystuje macierze. W reprezentacji grafu *G* = (*V*, *E*) za pomocą **macierzy sąsiedztwa** zakładamy, że wierzchołki są ponumerowane od 1, 2, …, $|V|$ w pewien dowolny sposób. Reprezentacja macierzowa grafu *G* składa się wtedy z macierzy *A* = (a_{ij}) wymiaru $|V| \times |V|$, takiej, że

$$a_{ij} = \begin{cases} 1, & jeżeli\ (i,\ j) \in E \\ 0, & w\ przeciwnym\ razie \end{cases} \qquad (2.1)$$

i oznacza to, że wyraz o indeksach *i*, *j* jest równy 1, jeśli wierzchołek *i* jest incydenty z wierzchołkiem *j*, i równy 0 w przeciwnym razie. Na rysunku 2.1 przedstawiono graf *G* wraz z możliwą formą jego reprezentacji w postaci: listy sąsiednich wierzchołków (b) i macierzy sąsiednich wierzchołków (c).

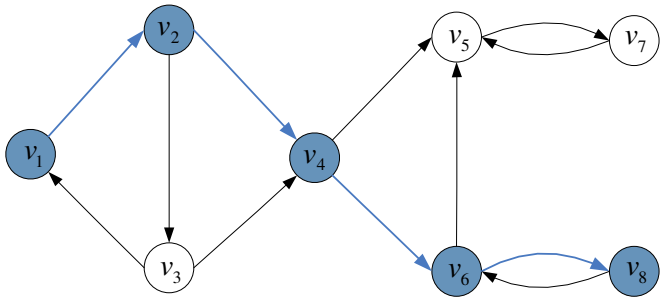
2.1.4 Definicja ścieżki w grafie

Ścieżka (droga) długości *k* z wierzchołka *u* do wierzchołka *u'* w grafie *G* = (*V*, *E*) jest ciągiem wierzchołków $\langle v_0, v_1, v_2, ..., v_k \rangle$ takich, że $u = v_0$, $u' = v_k$ i $(v_{i-1}, v_i) \in E$ dla $i = 1,2,...,k$ [36]. Ścieżka zawiera wierzchołki $v_0, v_1, ..., v_k$ i krawędzie (v_0, v_1), (v_1, v_2), …, (v_{k-1}, v_k).

Długość ścieżki jest liczbą krawędzi ścieżki. Na rysunku 2.3 przedstawiono ścieżke z wierzchołka v_1 do v_8 [36].

Niepustą ścieżką z wierzchołka *v* do wierzchołka *u* co zapisujemy $v \overset{+}{\rightarrow} u$ nazywamy ścieżkę o długości co najmniej jeden. **Ścieżkę** nazywamy **prostą** jeżeli wszystkie jej krawędzie i wierzchołki z wyjątkiem możliwie pierwszego i ostatniego są różne [36].

W grafie skierowanym ścieżka $\langle v_0, v_1, v_2, ..., v_k \rangle$ tworzy **cykl**, jeśli $v_0 = v_k$ oraz zawiera co najmniej jedną krawędź. Innymi słowy **cyklem** w grafie nazywamy niepustą prosta ścieżkę, która zaczyna się i kończy w tym samym wierzchołku. Graf, który nie zawiera cykli nazywamy **grafem acyklicznym** [36].

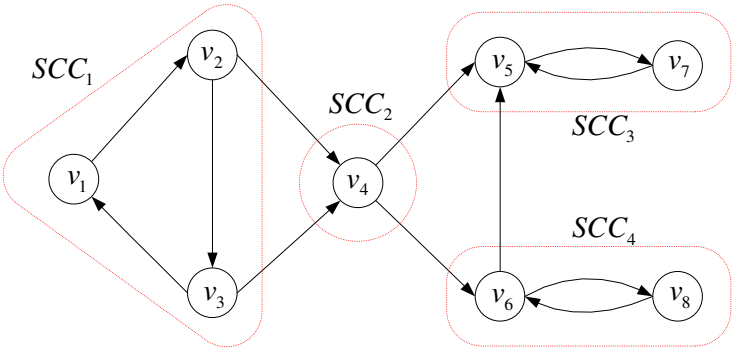


Rys. 2.3 Ścieżka z wierzchołka v_1 do v_8 [op. własne]

Cykl nazywamy **prostym**, jeśli, dodatkowo, $v_0, v_1, v_2, ..., v_k$ są różne. **Pętla** jest cyklem o długości 1. Jeśli istnieje ścieżka *p* z *u* do *u'* , to mówimy, że *u'* jest **osiągalny** z *u* po ścieżce *p,* co zapisujemy : $u \xrightarrow{p} u'$ [36].

2.1.5 Definicja silnie spójnej składowej

Spójna składowa grafu nieskierowanego to dowolny maksymalny podgraf, w którym każde dwa wierzchołki są połączone ścieżką [36].



Rys. 2.4 Silnie spójne składowe grafu *G* [83]

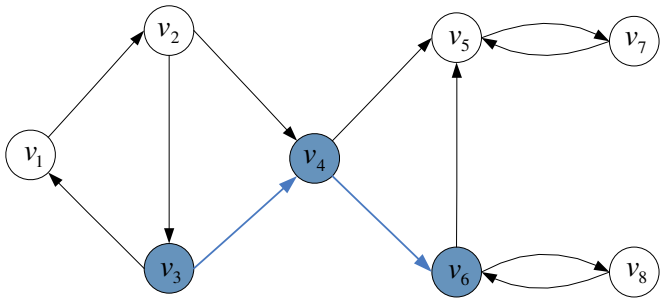
Graf nieskierowany jest spójny, jeśli ma dokładnie jedną spójną składową, to znaczy, jeśli każdy wierzchołek jest dostępny z każdego innego wierzchołka.

Spójna składowa grafu skierowanego to dowolny maksymalny podgraf, w którym każde dwa wierzchołki są osiągalne jeden z drugiego [36].

Silnie spójną składową skierowanego grafu $G = (V, E)$ nazywamy maksymalny zbiór wierzchołków $U \subseteq V$ taki, że dla każdej pary wierzchołków u i v z U istnieją ścieżki $u \to v$ oraz $v \to u$; tzn. wierzchołki u i v są osiągalne jeden z drugiego. Inaczej mówiąc, spójna składowa jest maksymalnym zbiorem węzłów takim, że między każdą parą węzłów z tego zbioru istnieje ścieżka [36]. Graf na rysunku 2.4 ma cztery spójne składowe $\{v_1, v_2, v_3\}$, $\{v_4\}$, $\{v_5, v_7\}$ i $\{v_6, v_8\}$.

2.1.6 Tranzytywne domknięcie grafu

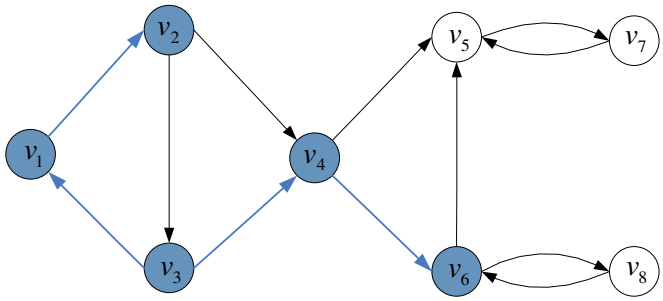
Jedną z pierwszych rzeczy, jakie zwykle chcielibyśmy wiedzieć o danym grafie, jest to, z których węzłów można dotrzeć do których. Jeśli potrzebujemy właśnie takiej informacji, to konieczna jest możliwość przeszukania grafu w pewien określony sposób [66, 100]. Często polega to na sprawdzeniu, każdej jego części, zanim zostanie odnaleziony interesujący nas wierzchołek. Większość algorytmów umożliwia przechodzenie wszystkich węzłów grafu, w pewien systematyczny sposób. Znane są dwa podstawowe tego typu warianty. Jeden z nich polega na rozpoczęciu przechodzenia od odwiedzenia węzła, a następnie rekurencyjnym odwiedzeniu wszystkich nie odwiedzonych dotychczas sąsiadów tego węzła. Istotne z tego punktu widzenia jest to, iż postępując w taki sposób możemy nie odwiedzić wszystkich węzłów w grafie, a jedynie te, które są osiągalne z pierwszego węzła.



Rys. 2.5 Przeszukiwanie w głąb (pierwsze dwa kroki) grafu G (v_3 - wierzchołek startowy) [op. własne]

Po odwiedzeniu wszystkich węzłów osiągalnych z węzła startowego, musimy wybrać kolejny węzeł. Jeżeli nie był dotychczas odwiedzony, to odwiedzamy go, a potem rekurencyjnie odwiedzamy jego sąsiadów i tak dalej, dopóki nie odwiedzimy wszystkich węzłów. Ponieważ odchodzimy od węzła tak daleko, jak można przed powrotem i wyborem kolejnego nie odwiedzonego węzła, metoda ta nosi nazwę **przeszukiwania w głąb** (**DFS**, ang. *depth-first serach*) [66, 102]. Rysunek 2.5 przedstawia przykład wykonania dwóch kroków procedury przeszukiwania w głąb grafu *G* począwszy od wierzchołka v_3.

Innym sposobem przechodzenia grafu jest metoda nazywana **przeszukiwaniem wszerz** (**BFS**, ang. *breadth-first serach*) [66, 102], polega ona na tym, że zamiast wędrować tak daleko, jak to możliwe z danego węzła, proces przeszukiwania dotrze najpierw do wszystkich węzłów leżących o jedną krawędź od początkowego, następnie do wszystkich węzłów odległych od niego o dwie krawędzie i tak dalej. Rysunek 2.6 przedstawia przykład wykonania dwóch kroków procedury przeszukiwania wszerz grafu *G* począwszy od wierzchołka v_3.



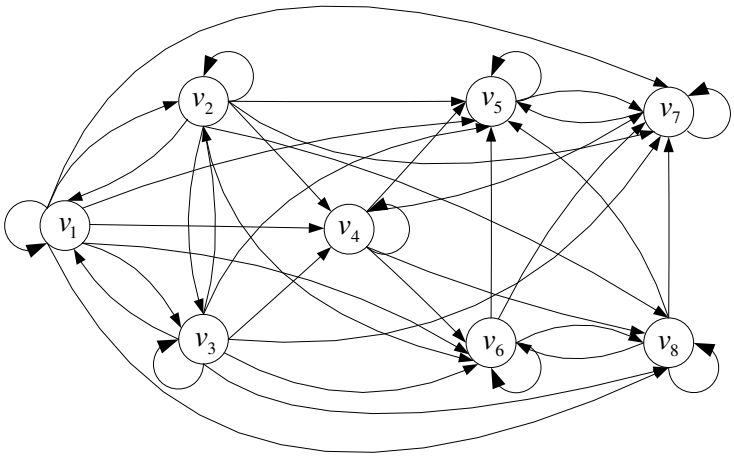
Rys. 2.6 Przeszukiwanie wszerz (pierwsze dwa kroki) grafu *G* (v_3 - wierzchołek startowy) [op. własne]

Niezależnie od przyjętej metody odwiedzania wierzchołków, jej efekt końcowy umożliwia nam skonstruowanie grafu, który zawiera krawędź od wierzchołka *U* do wierzchołka *V*, jeśli tylko graf wyjściowy zawiera taką ścieżkę pomiędzy zadaną parą wierzchołków [102]. Własność taka nosi nazwę domknięcia przechodniego grafu.

Graf skierowany $G^+ = (V, E)$ nazywamy **domknięciem przechodnim grafu** *G*, gdy E^+ jest zbiorem wszystkich takich par (*v, w*) wierzchołków ze zbioru *V*, że w grafie *G* istnieje droga z *v* do *w* [36]. Rysunek 2.7 przedstawia wyznaczone tranzytywne domknięcie wejściowego grafu *G* z rysunku 2.1a.

Wiele różnych problemów można rozwiązać, wykorzystując informacje dotyczące połączeń między wierzchołkami. Zostały one sklasyfikowane na następujące warianty:

- **połączenia z jednym wierzchołkiem docelowym** – należy zidentyfikować połączenia z każdego wierzchołka źródłowego *s* do danego wierzchołka docelowego *t*.
- **połączenia pomiędzy parą wierzchołków –** należy znaleźć ścieżkę z wierzchołka *u* do *v* dla danej pary wierzchołków *u* i *v*.
- **połączenia pomiędzy wszystkimi parami wierzchołków –** dla każdej pary wierzchołków *u*, *v* należy znaleźć ścieżkę z *u* do *v*.



Rys. 2.7 Tranzytywne domknięcie G^+ grafu *G z rysunku 2.1a* [83]

Za najbardziej popularny algorytm obliczania **tranzytywnego domknięcia grafu** *G*, reprezentowanego za pomocą macierzy sąsiedztwa uznaje się rozwiązanie zaproponowane przez Floyd-Warshall [34, 36, 41, 42]. Rysunek 2.8 zawiera pełny kod tego algorytmu. Dokonuje on przekształcenia macierzy sąsiedztwa danego grafu *G* w macierz sąsiedztwa jego domknięcia przechodniego.

Myślą przewodnią algorytmu Floyd-Warshall jest to, że szukając ścieżek we właściwej kolejności, możemy znajdować je szybko. Zakładamy, że węzły grafu są uporządkowane; dla rysunku 2.7 będzie to następujący porządek: $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8 \rangle$. Postępowanie rozpoczynamy najpierw od poszukiwania ścieżek pomiędzy każdą parą węzłów, które nie przechodzą przez żadne węzły pośrednie. Potem znajdujemy ścieżki nie przechodzące przez żaden węzeł późniejszy

niż v_1; węzły pośrednie muszą być wcześniejsze lub równe v_1. Następnie znajdujemy wszystkie ścieżki, które nie przechodzą przez żaden węzeł pośredni późniejszy niż v_2, potem – przez żaden późniejszy niż v_3. Postępowanie kończymy na ścieżkach przechodzących przez węzły nie późniejsze niż v_8, z wynikiem w postaci wszystkich ścieżek między każdą parą węzłów. Pozostaje pytanie jak odszukać te ścieżki ? Jedyne ścieżki nie przechodzące przez żadne węzły pośrednie to ścieżka pusta, prowadząca od każdego węzła do niego samego, oraz ścieżki składające się z jednej krawędzi.

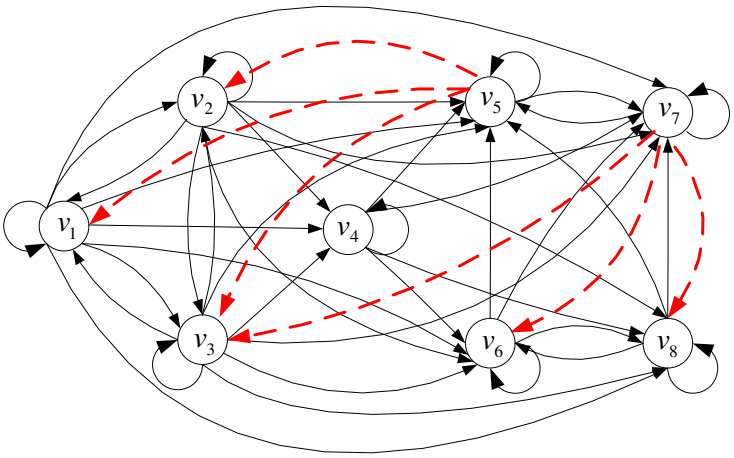
```
    Floyd-Warshall ( G )
(1) for j ← 1 to n
(2)     for i ← 1 to n
(3)         if Adj[i, j] = 1
(4)             for k ← 1 to n
(5)                 if Adj[j, k] = 1
(6)                     Adj[i, k] ← 1
```

Rys. 2.8 Algorytm Floyd-Warshall obliczania tranzytywnego domknięcia grafu [36]

Ścieżki odnajdujemy w następujący sposób: przypuśćmy, że odnaleźliśmy wszystkie ścieżki bez węzłów pośrednich późniejszych niż v_4, a chcemy sprawdzić, czy istnieje ścieżka od *X* do *Y* bez węzłów pośrednich późniejszych niż v_5 [45]. Jeśli istnieje ścieżka od *X* do *Y*, nie zawierająca węzłów późniejszych niż v_4, to na ścieżce takiej nie ma węzłów późniejszych niż v_5. W przeciwnym razie, musimy sprawdzić, czy istnieje ścieżka przechodząca przez v_5, ale nie przez żaden późniejszy węzeł. Taką ścieżkę można jednak podzielić na dwie: musi składać się ze ścieżki do *X* do v_5 nie przechodzącej przez węzły późniejsze niż v_4 (jeśli przechodziłaby przez v_5, zawierałaby zamkniętą pętlę), a za nią ścieżki do v_5 do *Y*, również nie zawierającej węzłów późniejszych niż v_4. W drugim przypadku, obydwie ścieżki nie zawierają węzłów późniejszych niż v_4. Dokonujemy sprawdzenia, czy odszukaliśmy już ścieżkę od *X* do v_5 i kolejną od v_5 do *Y*; jeśli tak, to odznaczamy, że znaleźliśmy ścieżkę od *X* do *Y*. Obliczenia te są wykonywane dla każdej pary węzłów *X* i *Y*. Czas działania algorytmu wynosi $O\left(|N|^3\right)$.

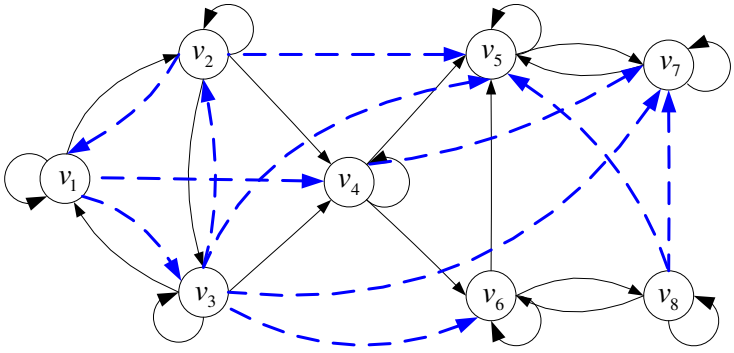
2.1.7 Aproksymacja tranzytywnego domknięcia grafu

Obliczenie dokładnego tranzytywnego domknięcia grafu *G*, z wielu powodów może być trudne, albo wręcz niemożliwe w przypadku ogólnym [9, 72]. Najczęściej spowodowane jest to ogromną liczbą wierzchołków wejściowych [83], albo wręcz dotyczy przypadku, w którym liczba ta nie jest znana, czyli przedstawiona w postaci zmiennej symbolicznej [30, 31, 72].



Rys. 2.9 Przybliżenie górne tranzytywnego domknięcia G^{+}_{UB} grafu *G* [op. własne]

W takich sytuacjach powszechną praktyką staje się obliczenie przybliżenia górnego G^{*}_{UB} tranzytywnego domknięcia grafu G takiego, że. $G^{*} \subset G^{*}_{UB}$.



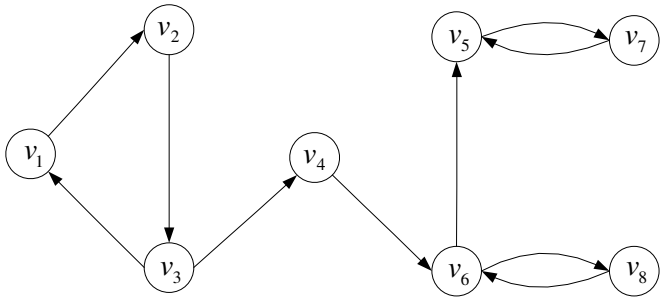
Rys. 2.10 Przybliżenie dolne tranzytywnego domknięcia G^{+}_{LB} grafu *G* [op. własne]

Rozwiązanie takie, zawiera wówczas nieistniejące (fałszywe) zależności przechodnie (rysunek 2.9, oznaczone kolorem czerwonym), nie wynikające z przebiegu zależności bezpośrednich.

Alternatywą dla przybliżenia górnego G^*_{UB} tranzytywnego domknięcia grafu G, jest jego przybliżenie dolne G^*_{LB}, takie, że $G^*_{LB} \subset G^*$. Wówczas uzyskane rozwiązanie zawiera niepełny zestaw poszukiwanych zależności przechodnich (rysunek 2.10, oznaczone kolorem niebieskim).

2.1.8 Definicja tranzytywnej redukcji

Tranzytywną redukcją $G_r = (V, E_r)$ grafu $G = (V, E)$ nazywamy graf, który posiada minimalną liczbę krawędzi taką, że tranzytywne domknięcie takiego grafu jest tożsame z tranzytywnym domknięciem grafu *G* [61, 73, 105]. Rysunek 2.11 przedstawia jeden z wariantów grafu tranzytywnej redukcji dla wejściowego grafu *G* z rysunku 2.1a.



Rys. 2.11 Tranzytywna redukcja grafu *G* z rysunku 2.1a [83]

2.2 Zrównoleglenie pętli programowej

Przetwarzaniem równoległym (ang. *parallel computing*), określa się współdziałanie wielu autonomicznych jednostek przetwarzających tak, aby współpracowały w rozwiązywaniu wspólnego problemu [6, 47]. W celu rozwiązania problemu w sposób równoległy, należy dokonać jego podziału na mniejsze zadania. Niestety jednym z podstawowych ograniczeń związanych z równoległym przetwarzaniem danych jest to, że dane potrzebne do przeprowadzenia jednej operacji mogą być uzyskiwane w wyniku przeprowadzenia innej operacji i dlatego tych dwóch

operacji nie można przeprowadzić w tym samym czasie [6, 54]. Dostęp do wspólnych danych i ich modyfikacja przez wiele równoległych wątków, prowadzi w ogólnym przypadku do występowania **zależności danych.** Ich uwzględnienie w trakcie procesu zrównoleglenia jest niezbędne dla uzyskania poprawności kodu równoległego [90]. Kod równoległy jest poprawny, jeżeli wynik jego wykonania z wykorzystaniem dowolnej liczby procesorów jest równy wynikowi wykonania kodu sekwencyjnego, dla tego samego zbioru danych wejściowych.

Proces **zrównoleglenia pętli programowej,** powinien być poprzedzony dokonaniem analizy zależności [88, 90]. Cenna z tego punktu widzenia jest również wiedza, odnośnie właściwości pętli dla zastosowania dalej odpowiednich transformacji, pozwalających na uzyskanie kodu równoległego. W podrozdziale 2.2.1 przedstawiono definicje i rodzaje zależności, w podrozdziale 2.2.2 z kolei zaprezentowano rodzaje pętli programowych.

2.2.1 Zależności w pętlach programowych

W przypadku występowania zależności, niemożliwe jest bezpośrednie zrównoleglenie wybranego fragmentu kodu. Ze względu na rodzaj występujących zależności, wyróżniamy dwa typy ograniczeń uniemożliwiających bezpośrednie zrównoleglenie fragmentu kodu:

- zależność sterowania
- zależność danych

Zależność sterowania (ang. *control dependence*) pomiędzy instrukcją **S1** i **S2** ma miejsce wówczas, gdy instrukcja **S1** decyduje, czy instrukcja **S2** będzie wykonywana, zgodnie z poniższym przykładem [6] :

S1 : **if** ($x = 0$)
S2 : $y \leftarrow 1$;

Między instrukcjami **S1** i **S2** występuje **zależność danych** (instrukcja **S2** zależy od instrukcji **S1**) wtedy i tylko wtedy, gdy [6, 90]:

- obydwie instrukcje odwołują się do tej samej komórki pamięci i przynajmniej jedno z odwołań jest zapisem do pamięci.

- istnieje ścieżka odwołań do tej samej komórki pamięci od instrukcji **S1** do instrukcji **S2** w trakcie wykonania

W obrębie zależności danych istnieją trzy podstawowe typy zależności [6, 90]:

1. Zależność przepływu danych, zwana również **zależnością prostą** (ang. *Data-Flow Dependence, True Dependence*) [6] – występuje między instrukcjami **S1** i **S2** (**S1** $\prec$ **S2**), jeżeli zapis danych następuje w instrukcji poprzedzającej ich odczyt:

   S1 : $X \leftarrow \ldots$
   S2 : $\ldots \leftarrow X$

   Zależność ta oznacza, że instrukcja **S2** pobiera wartość obliczoną za pomocą instrukcji **S1** i oznaczana jest **S1** δ **S2** (odczyt **S2** uzależniony jest od **S1**)

2. **Zależność odwrotna** (ang. *Antidependence*) [6] – występuje między instrukcjami **S1** i **S2** (**S1** $\prec$ **S2**), jeżeli odczyt danych następuje w instrukcji poprzedzającej ich zapis:

   S1 : $\ldots \leftarrow X$
   S2 : $X \leftarrow \ldots$

   Zależność ta zapobiega zamianie **S1** z **S2**, która to mogłaby doprowadzić do wadliwego użycia wartości obliczonej za pomocą instrukcji **S2**. Zależność odwrotna (zwana też anty-zależnością) oznaczana jest **S1** δ^{-1} **S2** (lub **S1** δ^{-} **S2**).

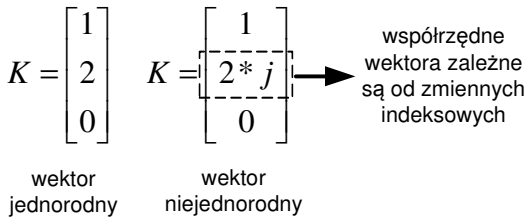
3. Zależność po wyjściu (ang. *Output Dependence*) [6] – występuje między instrukcjami **S1** i **S2** (**S1** $\prec$ **S2**), jeżeli zapis danych wykonywany jest w obydwu instrukcjach:

   S1 : $X \leftarrow \ldots$
   S2 : $X \leftarrow \ldots$

   Zależność ta zapobiega zamianie, w wyniku której kolejna instrukcja mogłaby odczytywać nieprawidłową wartość i oznaczana jest: **S1** δ^{0} **S2**.

Rozpatrywane powyżej zależności danych, mogą występować w ramach pojedynczej iteracji pętli, albo pomiędzy kolejnymi jej iteracjami. W pierwszym przypadku mamy do czynienia z **zależnością niezależną od pętli** (ang. *loop-independent dependence*), natomiast w drugim z **zależnością przenoszoną pętlą** (ang. *loop-carried dependence*) [6].

Zależność jednorodna (ang. *uniform dependence*) – jest to zależność charakteryzująca się stałym wektorem dystansu tzn. składowe takiego wektora, reprezentowane są za pomocą stałych wartości należących do zbioru liczb całkowitych. Jeżeli współrzędne wektora dystansu są zależne od zmiennych indeksujących pętli to mówimy o **zależności niejednorodnej** (ang. *non-uniform dependence*) – rysunek 2.12.



Rys. 2.12 Rodzaje wektorów dystansu [99]

2.2.2 Klasyfikacje pętli programowych

Algorytmy zrównoleglenia pętli programowych ukierunkowane są na określone klasy pętli programowych. Oznacza to, że mogą istnieć przypadki pętli, dla których dany algorytm nie dokona zrównoleglenia. Klasyfikacja pętli, pozwala na porównanie rozwiązań i ocenę ich zakresu stosowalności. Charakterystyka pętli określona jest przez stopień i typ jej zagnieżdżenia, obecność sparametryzowanych granic oraz rodzaje zależności.

Pętla idealnie zagnieżdżona (ang. *perfectly nested loop*) jest to pętla *n*-krotnie zagnieżdżona ($n>0$), której wszystkie instrukcje znajdują się w ciele pętli najbardziej zagnieżdżonej. Jeśli nie wszystkie instrukcje tworzą ciało najbardziej wewnętrznej pętli, to pętle nazywamy **nieidealnie zagnieżdżoną** (ang. *non-perfectly nested loop*) [6]. Poniżej przedstawiono przykłady takich pętli.

Pętla idealnie zagnieżdżona

```
for i = 1 to M do
  for j = 1 to N do
    instrukcja 1
    instrukcja 2
  endo
endo
```

Pętla nieidealnie zagnieżdżona

```
for i = 1 to M do
  instrukcja 1
  for j = 1 to N do
    instrukcja 2
  endo
endo
```

Pętla sparametryzowana (ang. *parameterized loop*) jest to pętla, której granice określone są przez parametr. Innymi słowy wartość granic w trakcie kompilacji nie jest znana. Jeśli wszystkie granice określone są za pomocą liczb, to jest to pętla **niesparametryzowana** (ang. *unparameterized loop*). Poniżej pokazano przykłady takich pętli.

Pętla niesparametryzowana

```
for i = 1 to 100 do
  for j = 1 to 100 do
    instrukcja 1
  endo
endo
```

Pętla sparametryzowana

```
for i = 1 to M do
  for j = 1 to N do
    instrukcja 1
  endo
endo
```

Pętlę zawierającą wyłącznie jednorodne zależności, nazywamy **pętlą jednorodną** (ang. *uniform loop*); jeśli występują zależności niejednorodne, to jest to przypadek **pętli niejednorodnej** (ang. *non-uniform loop*). W poniższych przykładach, pierwsza z pętli charakteryzuje się zależnością o stałym wektorem dystansu [0, 1]. W drugim przypadku zależność danych posiada niejednorodny wektor dystansu [0, i], gdzie $1\text{-}N \leq i \leq N$. Łatwiejszym zadaniem w ogólnym przypadku jest zrównoleglenie pętli idealnie zagnieżdżonej, niż nieidealnie zagnieżdżonej. Wszystkie instrukcje pętli tej samej iteracji można potraktować jako jedną i dokonać próby dodatkowej redukcji zależności. Uzyskanie kodu równoległego pętli z zależnościami jednorodnymi jest zazwyczaj mniej skomplikowane, niż w przypadku obecności zależności niejednorodnych [6].

Pętla jednorodna

```
for i = 1 to N do
  for j = 1 to N do
    a[i][j] = a[i][j+1]
  endo
endo
```

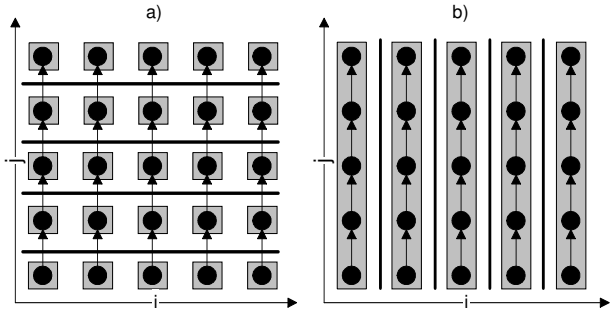
Pętla niejednorodna

```
for i = 1 to N do
  for j = 1 to N do
    a[i][j] = a[i][j+i]
  endo
endo
```

Zrównoleglenie pętli sparametryzowanej jest trudniejsze, niż pętli o stałych, znanych granicach.

2.2.3 Ziarnistość kodu

W przetwarzaniu równoległym **ziarnistość kodu** (ang. *granularity*), określa ilość obliczeń programu pomiędzy zdarzeniami synchronizacji lub komunikacji i uważana jest za kluczowy wskaźnik, mający zasadnicze znaczenie w określaniu w jakim stopniu obciążenie programu może być zbalansowane na maszynie równoległej. Zadania mogą być zdefiniowane na różnych poziomach ziarnistości. Z jednej strony, pojedynczy program w zbiorze programów może być traktowany jako pojedyncze zadanie. Z drugie strony, pojedyncze instrukcje w programie mogą być postrzegane jako zadania równoległe. Między tymi dwoma ekstremami, znajduje się zbiór modeli określających struktury kontrolujące wykonanie programu, oraz wspierające je architektury [47]. Liczba oraz wielkość zadań, na jakie problem został podzielony określają **ziarnistość dekompozycji**. Ze względu na wielkość ziarna obliczeń w przetwarzaniu równoległym wyróżnia się podział grubo- i drobnoziarnisty [6, 40, 43, 54, 75].



Rys. 2.13 Drobnoziarnista (a) i gruboziarnista
(b) równoległość – rys. poglądowy [99]

Jeśli stosunek czasu obliczeń aplikacji do czasu potrzebnego na synchronizację i komunikację jest mniejszy, oraz ilość obliczeń pomiędzy punktami synchronizacji jest stosunkowo niewielka, to mówimy o **drobnoziarnistej równoległości** (ang. *fine-grained parallelism*) – rysunek 2.13a. Drobnoziarnistość umożliwia maksymalną redukcję czasów przestoju i efektywnego planowania obciążenia. Jednak narzut czasowy potrzebny do tworzenia, komunikacji, synchronizacji i zakończenia działania

wątków może być zbyt kosztowny i całkowicie obniżyć wydajność przetwarzania. W związku z tym podział kodu na drobne ziarna, preferowany jest w systemach, gdzie koszt komunikacji między jednostkami przetwarzającymi jest relatywnie mały oraz planowanie obciążenia jest bardzo ważne [6, 47].

Gruboziarnista równoległość (ang. coarse-grained) – rysunek 2.13b pozwala na redukcję kosztów zarządzania wieloma zadaniami, oraz zmniejszenie zapotrzebowania na pamięć poprzez zwiększenie lokalności kodu [6, 47]. Należy jednak pamiętać, że gruboziarnisty podział, bardzo często prowadzi do nierównomiernego rozłożenia obliczeń między ziarnami, co z kolei uniemożliwia prawidłowe balansowanie obciążeniem. Gruboziarnistość preferowana jest w systemach gdzie koszt komunikacji między jednostkami przetwarzającymi jest stosunkowo duży (np. dla systemów rozproszonych). Wybór właściwego stopnia ziarnistości kodu równoległego, uzależnione jest architekturą programową i sprzętową. Warunkiem koniecznym jest uzyskanie krótszego czasu wykonania kodu równoległego, niż czas wykonania jego sekwencyjnego odpowiednika:

$$t_{równ} < t_{sekw}, \tag{2.2}$$

gdzie: t_{sekw} to czas wykonania kodu sekwencyjnego, $t_{równ}$ - czas wykonania kodu równoległego. Uwzględniając dodatkowy czas niezbędny do obsługi kodu równoległego t_{dod}, tj. czas synchronizacji, komunikacji czy zarządzania wątkami, oraz wykonanie kodu równoległego na *P* procesorach, uzyskuje się następującą nierówność:

$$\frac{t_{sekw}}{P} + t_{dod} < t_{sekw}. \tag{2.3}$$

Z powyższego wzoru (2.3) wynika, iż przy określaniu ziarnistości kodu ważne jest oszacowanie dodatkowego czasu, przeznaczonego na obsługę równoległości i sprawdzenie czy jest prawdziwa poniższa nierówność:

$$t_{dod} < (P-1) \cdot t_{sekw}\ . \tag{2.4}$$

Spełnienie powyższego warunku, gwarantuje uzyskanie wartości przyspieszenia obliczeń większej niż 1. Odpowiedni typ użytej transformacji pętli umożliwia uzyskanie pożądanej ziarnistości kodu dla wybranej architektury [47]. Transformacje pozwalające na uzyskanie drobnoziarnistej równoległości to: rozszerzenie skalaru (ang. *scalar expansion*), zmianę nazw zmiennych skalarnych (ang. *scalar renaming*), zmianę nazw

zmiennych tablicowych (ang. *array renaming*), podział węzłów (ang. *node splitting*), redukcję (ang. *reduction*), podział zmiennych indeksowych (ang. *index-set splitting*). Z kolei jako transformacje pozwalające uzyskać kod gruboziarnisty wyróżnia się: prywatyzację (ang. *privatization*), podział pętli (ang. *loop distribution*), wyrównanie (ang. *alignment*), replikację kodu (ang. *code replication*), łączenie pętli (ang. *loop fusion*), odwrócenie wykonania pętli (ang. *loop reversal*), wielowymiarowe łączenie petli (ang. *multilevel loop fusion*). Wynikiem transformacji zmiany kolejności wykonania pętli (ang. *loop interchange*) i przekoszenia pętli (ang. *loop skewing*) może być zarówno kod drobno- jak i gruboziarnisty [6].

W ogólnym przypadku, ziarnistość kodu jest pojęciem względnym, ściśle związanym z określoną architekturą systemu wieloprocesorowego [47]. Kod wykonany na systemie z pamięcią dzieloną, sklasyfikowany jako gruboziarnisty, może zostać uznany za drobnoziarnisty w systemie z pamięcią rozproszoną.

2.3 Arytmetyka Presburgera

Do opisu zależności oraz implementacji algorytmów wybrana została analiza zależności zaproponowana przez Pugha oraz Wonncotta [81, 91]. Wynikiem przeprowadzonej w ten sposób analizy są relacje zależności zbudowane z formuł Presburgera, będące podstawą arytmetyki Presburgera [89, 93]. Formuły te składają się z liniowych ograniczeń nad zmiennymi całkowitymi przy zastosowaniu operatorów logicznych negacji, koniunkcji i alternatywy; $\neg$, $\wedge$, $\vee$, kwantyfikatora ogólnego (ang. *universal quantifier*) $\forall$ i kwantyfikatora szczegółowego (ang. *existantial quantifier*) $\exists$ [81]. W arytmetyce Presburgera wyróżnia się także zestaw operacji binarnych i unarnych, wykonywanych na zbiorach i relacjach.

2.3.1 Relacje i zbiory

W celu zrozumienia dalszej części pracy, niezbędne jest wyjaśnienie podstawowych pojęć związanych z formułami Presburgera, wraz z wybranymi narzędziami na niej opartych [70, 71] :

- ***k*-krotka** (ang. t*uple*) jest to punkt w przestrzeni Z^k o wartościach całkowitych o wymiarze *k.*
- **zbiór** (ang. s*et*) zbudowany jest z *k*-krotek, gdzie *k* jest liczbą całkowitą.

- **relacja** (ang. r*elation*) jest to mapowanie *n* wymiarowych krotek na *m* wymiarowe krotki.

Postać ogólna relacji wygląda następująco:

$$\{[s_1, s_2, ..., s_k] \rightarrow [t_1, t_2, ..., t_{k'}] \mid \vee_{i=1}^{n} \exists \alpha_{i1}, \alpha_{i2}, ..., \alpha_{im_i} \; s.t. \; F_i \}, \tag{2.5}$$

gdzie F_i to ograniczenia w postaci afinicznych równań $\left(\sum_{i=1}^{n} x_i \cdot c_i + c_0 = 0\right)$ i nierówności $\left(\sum_{i=1}^{n} x_i \cdot a_i + a_0 \geq 0\right)$ nałożonych na składowe krotki wejściowej $s_1, s_2, ..., s_k$, wyjściowej $t_1, t_2, ..., t_{k'}$, zmienne egzystencjalne $\alpha_{i1}, \alpha_{i2}, ..., \alpha_{im_i}$ i stałe symboliczne. Ze względu na liczbę koniunkcji relacje dzielimy na :

- **proste** (ang. *simple/single conjunct*) – relacja składająca się z pojedynczej koniunkcji

$$\{[s_1, s_2, ..., s_k] \rightarrow [t_1, t_2, ..., t_k] \mid \exists \alpha_1, \alpha_2, ..., \alpha_m \; s.t. \; F\}, \tag{2.6}$$

- **złożone** (ang. multiple *conjunct*) –relacje składające się ze skończonej sumy relacji prostych

$$\bigcup_{i=1}^{n} \{[s_1, s_2, ..., s_k] \rightarrow [t_1, t_2, ..., t_k] \mid \exists \alpha_{i1}, \alpha_{i2}, ..., \alpha_{im} \; s.t. \; F_i\}. \tag{2.7}$$

Przewagą w reprezentowaniu zależności w postaci relacji krotek, w porównaniu z tradycyjnym sposobem ich reprezentacji przy pomocy wektora zależności lub odległości jest bardziej zwięzły i dokładny opis występowania zależności. Natomiast za pomocą zbioru, można opisać iteracje pętli, pomiędzy którymi występują zależności.

Przykład:

for *i* = 1 **to** *N* **do**
  for *j* = 1 **to** *M* **do**
    a(*i*,*j*)=*a*(*i*,*j*-1)
  endor
endfor

Relacja zależności

$$R = \{[i, j] \rightarrow [i, j+1] \mid 1 \leq i \leq n \wedge 1 \leq j < n\}$$

Zależne iteracje pętli

$$IS = \{[i, j] \mid 1 \leq i \leq n \wedge 1 \leq j < n\}$$

W dalszej części tego rozdziału opisano operacje arytmetyki Presburgera, wraz z przykładami, które są podstawą prezentowanych w niniejszej pracy algorytmów.

Operacje te przedstawione są za pomocą notacji matematycznej, natomiast w przykładach użyty jest zapis relacji i zbiorów charakterystyczny dla narzędzi Omega Calculator i Petit [70, 71]. W podrozdziale 2.4, przedstawiono krótki opis tych narzędzi.

2.3.2 Operacje binarne

Operacje binarne wymagają dwóch argumentów. Argumentami są relacje lub zbiory.

Suma (ang. *Union*)

Operację sumy można wykonać zarówno na zbiorach i relacjach. Jeżeli $R_1 = \{[x_1] \rightarrow [y_1] \mid f_1(x_1, y_1)\}$ oraz $R_2 = \{[x_2] \rightarrow [y_2] \mid f_2(x_2, y_2)\}$, wówczas sumą R_1 i R_2 jest relacja $R = \{[x] \rightarrow [y] \mid f_1(x, y) \vee f_2(x, y)\}$. Wymiary argumentów muszą być jednakowe.

Przykład 2.1

Niech będą dane dwa zbiory:

$$S_1 = \{[x] \mid 1 \le x \le 20\},$$
$$S_2 = \{[x] \mid 15 \le x \le 30\}.$$

Wynikiem operacji sumy zbiorów $S_1 \cup S_2$ jest zbiór $S = \{[x] \mid 1 \le x \le 30\}$.

Przykład 2.2

Niech dane będą dwie relacje:

$$R_1 = \{[x_1, x_2] \rightarrow [x_1, x_2 + 1] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\},$$
$$R_2 = \{[x_1, x_2] \rightarrow [x_1 + 1, x_2] \mid 1 \le x_1 < n \wedge 1 \le x_2 \le n\}.$$

Wynikiem sumy relacji $R_1 \cup R_2$ jest nowa relacja:

$$R = \{[x_1, x_2] \rightarrow [x_1, x_2 + 1] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\} \cup$$
$$\{[x_1, x_2] \rightarrow [x_1 + 1, x_2] \mid 1 \le x_1 < n \wedge 1 \le x_2 \le n\}.$$

Iloczyn (ang. *Intersection*)

Operację iloczynu można wykonać zarówno na zbiorach i relacjach. Jeżeli $R_1 = \{[x_1] \to [y_1] \mid f_1(x_1, y_1)\}$ oraz $R_2 = \{[x_2] \to [y_2] \mid f_2(x_2, y_2)\}$,wówczas iloczynem (częścią wspólną) R_1 i R_2 jest relacja $R = \{[x] \to [y] \mid f_1(x, y) \wedge f_2(x, y)\}$. Wymiary argumentów muszą być jednakowe.

Przykład 2.3

Niech będą dane dwa zbiory:

$$S_1 = \{[x] \mid 1 \le x \le 20\},$$
$$S_2 = \{[x] \mid 15 \le x \le 30\}.$$

Wynikiem operacji iloczynu zbiorów $S_1 \cap S_2$ jest zbiór $S = \{[x] \mid 15 \le x \le 20\}$.

Przykład 2.4

Niech dane będą dwie relacje:

$$R_1 = \{[x_1, x_2] \to [x_1, x_2 + 1] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\},$$
$$R_2 = \{[x_1, x_2] \to [x_1 + 1, x_2] \mid 1 \le x_1 < n \wedge 1 \le x_2 \le n\}.$$

Wynikiem iloczynu relacji $R_1 \cap R_2$ jest pusta relacja $R = \varnothing$. W zapisie relacji i zbiorów Omega Calculator koniunkcję ograniczeń oznacza się symbolem *&&*, alternatywę symbolem *OR*, *union* lub *||*.

Różnica (ang. *difference*)

Operację różnicy można wykonać zarówno na zbiorach i relacjach. Jeżeli $R_1 = \{[x_1] \to [y_1] \mid f_1(x_1, y_1)\}$ oraz $R_2 = \{[x_2] \to [y_2] \mid f_2(x_2, y_2)\}$, wówczas różnicą R_1 i R_2 jest relacja $R = \{[x] \to [y] \mid f_1(x, y) \wedge \neg f_2(x, y)\}$. Wymiary argumentów muszą być jednakowe.

Przykład 2.5

Niech będą dane dwa zbiory:

$$S_1 = \{[\![x]\!] | 1 \le x \le 20\},$$
$$S_2 = \{[\![x]\!] | 15 \le x \le 30\}.$$

Wynikiem operacji różnicy zbiorów $S_1 - S_2$ jest zbiór $S = \{[\![x]\!] | 1 \le x \le 15\}$.

Przykład 2.6

Niech dane będą dwie relacje:

$$R_1 = \{[\![x_1, x_2]\!] \to [x_1, x_2 + 1] | 1 \le x_1 \le n \wedge 1 \le x_2 < n\},$$
$$R_2 = \{[\![x_1, x_2]\!] \to [x_1 + 1, x_2] | 1 \le x_1 < n \wedge 1 \le x_2 \le n\}.$$

Wynikiem różnicy relacji $R_1 - R_2$ jest nowa relacja:

$$R_1 = \{[\![x_1, x_2]\!] \to [x_1, x_2 + 1] | 1 \le x_1 \le n \wedge 1 \le x_2 < n\}.$$

Ograniczenia dziedziny

Operacja ograniczenia dziedziny (ang. *restrict domain*) posiada dwa argumenty; pierwszy z nich to relacja, drugi zbiór. Jeżeli $R = \{[\![x_1]\!] \to [y_1] | f_1(x_1, y_1)\}$ oraz $S = \{[\![x_2]\!] | f_2(x_2)\}$, wówczas ograniczeniem dziedziny relacji *R* na zbiór *S* jest relacja. $R = \{[\![x]\!] \to [y] | f_1(x, y) \wedge f_2(x)\}$. Wymiar zmiennych wejściowych relacji musi być równy wymiarowi zbioru.

Przykład 2.7

$$R = \{[\![x_1, x_2]\!] \to [x_1, x_2 + 1] | 1 \le x_1 \le 100 \wedge 1 \le x_2 < 100\},$$
$$S = \{[\![x_1, x_2]\!] | 1 \le x_1 \le 10 \wedge 1 \le x_2 \le 10\}.$$

Wynikiem operacji ograniczenia dziedziny relacji *R* do zbioru *S*, $R \setminus S$ jest relacja:

$$R = \{[\![x_1, x_2]\!] \to [x_1, x_2 + 1] | 1 \le x_1 \le 10 \wedge 1 \le x_2 \le 10\}.$$

Ograniczenia zakresu

Operacja ograniczenia zakresu (ang. *restrict range*) posiada dwa argumenty; pierwszy z nich to relacja, drugi zbiór. Jeżeli $R = \{[\![x_1]\!] \to [y_1] | f_1(x_1, y_1)\}$, oraz

$S = \{[x_2] \mid f_2(x_2)\}$, wówczas ograniczeniem zakresu relacji R na zbiór S jest relacja $R = \{[x] \to [y] \mid f_1(x, y) \wedge f_2(y)\}$. Wymiar zmiennych wyjściowych relacji, musi być równy wymiarowi zbioru.

Przykład 2.8

$$R = \{[x_1, x_2] \to [x_1, x_2 + 1] \mid 1 \le x_1 \le 100 \wedge 1 \le x_2 < 100\},$$
$$S = \{[x_1, x_2] \mid 1 \le x_1 \le 10 \wedge 1 \le x_2 \le 10\}.$$

Wynikiem operacji ograniczenia zakresu relacji R do zbioru S, R / S jest relacja:

$$R = \{[x_1, x_2] \to [x_1, x_2 + 1] \mid 1 \le x_1 \le 10 \wedge 1 \le x_2 \le 9\}.$$

Aplikacja relacji na zbiorze

W proponowanych algorytmach użyto operacji $R(S)$, czyli aplikacji relacji na zbiorze. Operacja ta wymaga dwóch argumentów, relacji R oraz zbioru S i składa się z dwóch operacji z arytmetyki Presburgera; ograniczenia dziedziny i zakresu. Najpierw wykorzystując argumenty, obliczana jest relacja na ograniczonym zbiorze dziedziny S (operacja opisana w punkcie 2.3.2). Następnie, należy obliczyć zakres otrzymanej relacji, aby uzyskać wynik operacji R(S), (punkt 2.3.3).

Jeżeli $R = \{[x_1] \to [y_1] \mid f_1(x_1, y_1)\}$ oraz $S = \{[x_2] \mid f_2(x_2)\}$, wówczas aplikacją relacji na zbiorze $R(S)$, jest zbiór $S = \{[y] \mid \exists\, x : \{x \to y \mid f_1(x, y) \wedge f_2(x)\}\}$.

Przykład 2.9

$$R = \{[x_1, x_2] \to [x_1, x_2 + 1] \mid 1 \le x_1 \le 100 \wedge 1 \le x_2 < 100\},$$
$$S = \{[x_1, x_2] \mid 1 \le x_1 \le 10 \wedge 1 \le x_2 \le 10\}.$$

Wynikiem operacji aplikacji relacji R na zbiorze S, $R(S)$ jest zbiór:

$$S = \{[x_1, x_2] \mid 1 \le x_1 \le 10 \wedge 2 \le x_2 \le 11\}.$$

2.3.3 Operacje unarne

Operacje unarne wymagają tylko jednego argumentu – zbioru lub relacji.

Operacja dziedziny (ang. *domain*)

Operację dziedziny (ang. *domain*) można zastosować tylko w stosunku do relacji. Wynikiem tej operacji dla relacji $R = \{[x] \rightarrow [y] \mid f(x, y)\}$ jest zbiór $S = \{[x] \mid \exists\, y : f(x, y)\}$.

Przykład 2.10

$$R = \{[x_1, x_2] \rightarrow [x_1, x_2 + 1] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\}$$

Wynikiem operacji dziedziny dla relacji *R*, domain(*R*) jest zbiór:

$$S = \{[x_1, x_2] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\}.$$

Operacja zakresu (ang. *range*)

Operacja zakresu (ang. *range*) może być stosowana tylko w przypadku relacji. Wynikiem tej operacji dla relacji $R = \{[x] \rightarrow [y] \mid f(x, y)\}$ jest zbiór $S = \{[y] \mid \exists\, x : f(x, y)\}$.

Przykład 2.11

$$R = \{[x_1, x_2] \rightarrow [x_1, x_2 + 1] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\}$$

Wynikiem operacji zakresu dla relacji *R*, range(*R*) jest zbiór:

$$S = \{[x_1, x_2] \mid 1 \le x_1 \le n \wedge 2 \le x_2 \le n\}.$$

Negacja (ang. *negation*)

Argumentem tej operacji może być zarówno relacja jaki i zbiór. Wynikiem tej operacji dla $R = \{[x] \rightarrow [y] \mid f(x, y)\}$ jest relacja $R = \{[x] \rightarrow [y] \mid \neg f(x, y)\}$.

Przykład 2.12

$$S = \{[x_1] \mid 1 \le x_1 \le 10\}$$

Wynikiem operacji negacji zbioru *S* jest zbiór $\neg S = \{[x_1] \mid i < 1 \wedge OR\ i > 10\}$.

Przykład 2.13

$$R = \{[x_1, x_2] \rightarrow [x_1, x_2 + 1] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\}$$

Wynikiem operacji negacji relacji R jest relacja:

$$\begin{aligned}\neg R = {} & \{[x_1, x_2] \rightarrow [y_1, y_2] \mid x_2 \le 0\} \cup \{[x_1, x_2] \rightarrow [y_1, y_2] \mid n \le x_2 \wedge 1 \le x_2\} \cup \\ & \{[x_1, x_2] \rightarrow [y_1, y_2] \mid 1 \le x_2 < n \wedge x_1 \le 0\} \cup \{[x_1, x_2] \rightarrow [y_1, y_2] \mid 1 \le x_2 < n < x_1\} \cup \\ & \{[x_1, x_2] \rightarrow [y_1, y_2] \mid 1 \le x_2 \le n-1,\ y_2 - 2 \wedge 1 \le x_1 \le n\} \cup \\ & \{[x_1, x_2] \rightarrow [y_1, y_2] \mid 1,\ y_2 \le x_2 < n \wedge 1 \le x_1 \le n\} \cup \\ & \{[x_1, x_2] \rightarrow [y_1, x_2 + 1] \mid 1 \le x_1 \le n,\ y_1 - 1 \wedge 1 \le x_2 < n\} \cup \\ & \{[x_1, x_2] \rightarrow [y_1, x_2 + 1] \mid 1,\ y_1 + 1 \le x_1 \le n \wedge 1 \le x_2 < n\}.\end{aligned}$$

Inwersja (ang. *inversion*)

Argumentem tej operacji może być tylko relacja. Wynikiem również jest relacja. Dla relacji $R = \{[x] \rightarrow [y] \mid f(x, y)\}$ inwersją jest relacja $R = \{[y] \rightarrow [x] \mid f(x, y)\}$.

Przykład 2.14

$$R = \{[x_1, x_2] \rightarrow [x_1, x_2 + 1] \mid 1 \le x_1 \le n \wedge 1 \le x_2 < n\}$$

Wynikiem inwersji relacji *R* jest relacja:

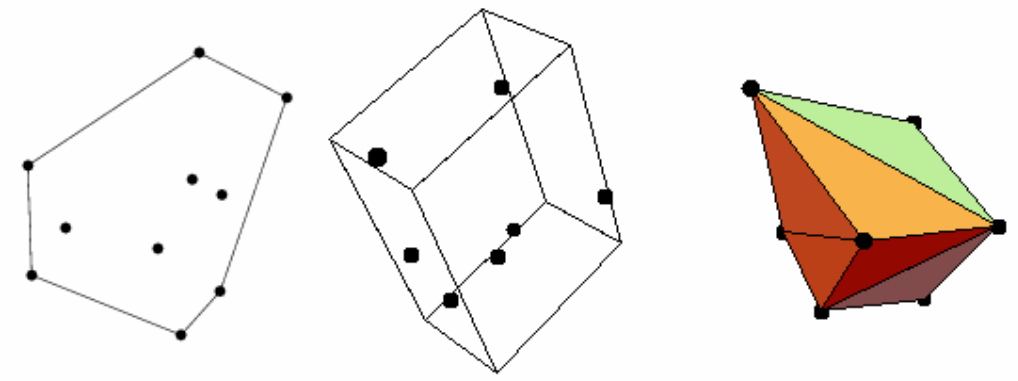
$$R = \{[x_1, x_2] \rightarrow [x_1, x_2 - 1] \mid 1 \le x_1 \le n \wedge 2 \le x_2 \le n\}.$$

Region wypukły (ang. *convex hull*)

Regionem wypukłym zbioru *S*, w *n* – wymiarowej przestrzeni nazywamy część wspólną wszystkich wypukłych zbiorów zawierających *S*. Dla *N* punktów $p_1, p_2, \ldots p_n$ region wypukły *C*, określony jest następującym wzorem [85, 112]:

$$C \equiv \left\{ \sum_{j=1}^{N} \lambda_j p_j : \lambda_j \ge 0 \wedge \sum_{j=1}^{N} \lambda_j = 1 \right\}. \tag{2.7}$$

Interpretację graficzną operacji wyznaczenia regionu wypukłego, dla *N* punktów p_1, p_2, … p_n, przedstawia rysunek 2.14.



Rys. 2.14. Przykłady regionu wypukłego, źródło [112]

Argumentem tej operacji, może być zarówno relacja jaki i zbiór. Poniżej przedstawiono przykład znajdowania regionu wypukłego dla zbioru *S* i relacji *R*.

Przykład 2.15

$$S = \{[x_1] | 1 \le x_1 \le 4 \vee 6 \le x_1 \le 10\},$$
$$R = \{[x_1] \rightarrow [x_1 + 1] | 1 \le x_1 \le 4 \vee 6 \le x_1 \le 10\}.$$

Wynikiem operacji wyznaczenia regionu wypukłego dla zbioru *S*, jest zbiór ConvexHull(S) $= \{[x_1] | 1 \le x_1 \le 10\}$.

Wynikiem operacji wyznaczenia regionu wypukłego dla relacji *R*, jest relacja ConvexHull(R) = $= \{[x_1] \rightarrow [x_1 + 1] | 1 \le x_1 \le 10\}$.

Dodatkowo Omega Caclulator pozwala na następujące obliczenia [71]:

- region liniowy $X = \{ \sum_{i_1}^{t} lambda_i x_i \mid x_i \in X \}$,
- region afiniczny $X = \{ \sum_{i_1}^{t} lambda_i x_i \mid x_i \in X \wedge 1 = \sum_{i=1}^{t} lambda_i \}$,
- region stożkowy $X = \{ \sum_{i_1}^{t} lambda_i x_i \mid x_i \in X \wedge \forall_{i=1}^{t} \lambda_i \ge 0 \}$.

Biblioteka Omega, udostępnia kilka rodzajów funkcji [71] obliczania regionu, które różnią się dokładnością i narzutem obliczeniowym. W niektórych przypadkach,

obliczenia mogą być zbyt kosztowne i dokonywana jest aproksymacja wyniku. Funkcja *Hull* biblioteki Omega jest połączeniem metod *Quick Hull* i *FastTightHull,* sprawdza ona czy jakiekolwiek ograniczenie danej koniunkcji, graniczy z wszystkimi koniunkcjami [85].

2.4 Zastosowanie biblioteki Omega w oparciu o arytmetykę Presburgera

Na potrzeby niniejszej pracy wybrane zostały narzędzia Petit i Omega Calculator. Petit, narzędzie akademickie [70], pozwala na identyfikację zależności występujących w pętlach programowych, zgodnie z analizą zależności zaproponowaną przez Pugha oraz Wonnacotta [88]. Analizowane w ten sposób programy, zapisane są w języku Petit, a uzyskane relacje zależności stanowią główne dane wejściowe dla opisanych w pracy algorytmów. Ponadto, omawiane narzędzie dokonuje rozpoznania rodzaju zależności (prosta, odwrotna, po wyjściu) jak i również odpowiada na pytanie pomiędzy którymi instrukcjami pętli dana zależność zachodzi.

Omega Calculator [71] z kolei, umożliwia manipulacje na zbiorach i relacjach krotek. Przy jego pomocy możliwe jest wykonanie dowolnej operacji, opisanej w podrozdziałach 2.3.2 i 2.3.3, na relacjach zależności uzyskanych w wyniku działania narzędzia Petit. Omega Calculator dostarcza aplikację konsolową oraz bibliotekę dla programisty. Za pomocą konsoli można interaktywnie wykonywać operacje na zbiorach i relacjach. W implementacji algorytmów w niniejszej pracy, skorzystano z biblioteki Omega Calculator napisanej w języku C++. Dokumentacja biblioteki [70, 71] opisuje jak tworzyć relacje oraz zbiory i wykonywać na nich operacje, zgodne z arytmetyką Presburgera w kodzie aplikacji. W przypadku niektórych operacji w ramach biblioteki Omega, w otrzymanym wyniku można uzyskać ograniczenia typu **UNKNOWN**. Oznaczają one niedokładny opis relacji lub zbioru. Są wynikiem operacji arytmetyki Presburgera, w której nie zdołano uzyskać dokładnego wyniku (z powodu ograniczeń algorytmów zaimplementowanych w narzędziu Omega [71]).

2.5 Podsumowanie

W niniejszym rozdziale, przedstawiono wprowadzenie do teorii grafów i przetwarzania równoległego. Zaprezentowano podstawowe definicje i pojęcia tj. definicję grafu skierowanego, nieskierowanego, ścieżki w grafie, silnie spójnej

składowej, tranzytywnego domknięcia grafu, tranzytywnej redukcji i aproksymacji tranzytywnego domknięcia grafu. Przybliżono również najpopularniejsze formy reprezentacji grafu. Szczegółowo omówiono zależności występujące w kodzie, ich rodzaje i sposoby reprezentacji z naciskiem na reprezentację zaproponowaną przez Pugh i Wonnacott [88]. Omówiona arytmetyka Presburgera posłuży do łatwiejszego zrozumienia opisu proponowanych algorytmów zawartych w kolejnych rozdziałach.

3. Klasy relacji zależności i ich rozpoznanie

Relacje krotek pozwalają w sposób zwięzły symbolizować wiele istotnych informacji, zebranych podczas analizy kodu programu. Przykładowo, umożliwiają przedstawienie dokładnych zależności występujących w pętlach programowych, czyli innymi słowy dokonują opisu które instancje instrukcji i w jaki sposób są zależne względem siebie. Nie ulega wątpliwości, iż stanowi to kluczowy czynnik pozwalający na zastosowanie odpowiedniej transformacji pętli programowej w celu zwiększenia ekstrakcji drobno- i gruboziarnistej równoległości. Relacje zależności, można przyporządkować do różnych klas i niezwykle ważne jest aby dysponować mechanizmami umożliwiającymi rozpoznanie danej klasy relacji [19]. W rozdziale tym, przedstawiono klasy sparametryzowanych relacji zależności i techniki umożliwiające ich rozpoznanie.

3.1 Topologie grafu zależności

Poprzez topologię grafu zależności autor rozumie topologię grafu, utworzonego przez wszystkie zależności pętli, opisanych za pomocą relacji, w taki sposób, że początek i koniec każdej zależności reprezentuje węzły grafu, natomiast istniejące zależności symbolizują przechodzące pomiędzy nimi krawędzie.

Aby określić rodzaj topologii grafu zależności, w dalszej części tego paragrafu wprowadzone zostały dwa dodatkowe pojęcia, ułatwiające przebieg tego procesu. Należą do nich **zbiór wspólnych początków i końców relacji zależności**. Niech relacja R będzie unią dowolnej liczby n relacji:

$$R = R_1 \cup R_2 \cup R_3 \cup ... \cup R_n . \tag{3.1}$$

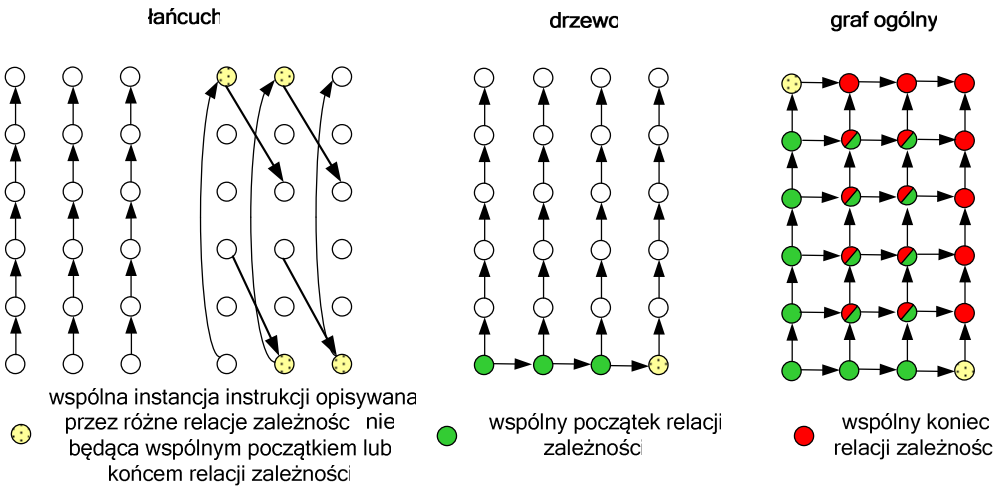
Zbiór wspólnych początków relacji zależności (3.1) (ang. ***C**ommon **D**ependence **S**ources*) można określić następującym wzorem:

$$CDS = \left\{ [e] \mid e = R^{-1}\left(e^{'}\right) = R^{-1}\left(e^{''}\right) \wedge e^{'}, e^{''} \in range(R) \wedge e^{'} \neq e^{''} \right\}. \tag{3.2}$$

Zbiór wspólnych końców relacji zależności (3.1) (ang. ***C**ommon **D**ependence **D**estinations*) można określić analogicznym wzorem:

$$CDD = \left\{ [e] \mid e = R\left(e^{'}\right) = R\left(e^{''}\right) \wedge e^{'}, e^{''} \in domain(R) \wedge e^{'} \neq e^{''} \right\}. \tag{3.3}$$

Warto zauważyć, że powyższe wzory (3.2) i (3.3) umożliwiają także wyznaczenie wspólnych początków i końców relacji zależności składających się z pojedynczej koniunkcji (nie będących unią wielu relacji).



Rys 3.1 Przykłady topologii grafu zależności [84]

Topologia grafu zależności, opisanych przez relację *R* jest łańcuchem, jeśli spełniony jest następujący warunek:

$$CDS = \varnothing \wedge CDD = \varnothing. \tag{3.4}$$

Topologia grafu zależności, opisanych przez relację *R* jest drzewem, jeśli spełniony jest warunek:

$$CDS \neq \varnothing \wedge CDD = \varnothing. \tag{3.5}$$

Topologia grafu zależności, opisanych przez relacje *R* jest grafem ogólnym (ani łańcuchem, ani drzewem), jeśli spełniony jest warunek:

$$CDD \neq \varnothing . \tag{3.6}$$

Innymi słowy: topologia łańcucha, nie posiada wspólnych początków i końców relacji zależności. W topologii drzewa, pomiędzy zależnościami występują wspólne początki zależności, lecz nie ma wspólnych końców zależności.

Występowanie wspólnych końców relacji zależności, oznacza topologię ogólną grafu. Na rysunku 3.1 zilustrowano na przykładach pojęcie wspólnej instancji instrukcji różnych relacji zależności, wspólnego początku relacji zależności, wspólnego końca relacji zależności oraz rodzaje topologii. Analiza topologii grafu zależności jest również ważna dla rozpoznania klasy relacji zależności i doboru odpowiedniego algorytmu obliczania jej tranzytywnego domknięcia.

3.2 Relacja zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form dependence relation*)

Relację *R* nazywamy relacją zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form relation*) wtedy i tylko wtedy gdy przyjmuje następującą postać [72] :

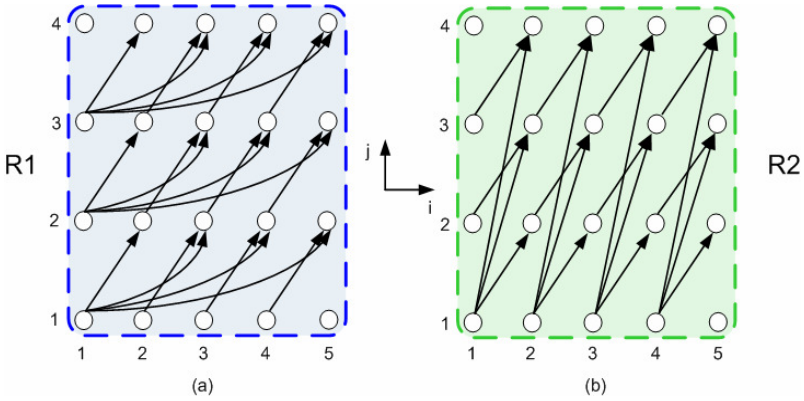
$$R=\left\{\begin{array}{l}[i_1,i_2,...,i_m]\rightarrow[j_1,j_2,...,j_m]:\ \forall p,1\le p\le m,\\ \qquad\qquad\qquad\qquad \exists\,\alpha_p\ s.t.\ L_p\le j_p-i_p\le U_p\wedge j_p-i_p=M_p\alpha_p\end{array}\right\}, \tag{3.7}$$

gdzie L_p, U_p, $M_p\in Z$ i definiują one wartości stałe nakładające ograniczenia na różnicę składowych krotki wyjściowej j_p , wejściowej i_p i zmiennych egzystencjalnych α_p. W przypadku gdy $L_p=-\infty$ lub $U_p=+\infty$ ograniczenie takie jest pomijane przy zapisie relacji.

Rysunek 3.2 przedstawia relacje R_1 (a) i R_2 (b) dla wartości $n = 5$ i $m = 4$, będące relacjami zależności o zmiennym wektorze dystansu klasy delta.

$$\begin{aligned}R_1&=\left\{[i,j]\rightarrow[i',j+1]\mid 1\le i<i'\le n\wedge 1\le j<m\right\},\\ R_2&=\left\{[i,j]\rightarrow[i+1,j']\mid 1\le i<n\wedge 1\le j<j'\le m\right\}.\end{aligned} \tag{3.8}$$

Poniżej przedstawiono algorytm 3.1, którego zadaniem jest określenie czy rozważana relacja *R*, należy do typu relacji o zmiennym wektorze dystansu klasy delta,



Rys. 3.2 Przykłady relacji delta (ang. *d-form relations*) [op. własne]

poprzez utworzenie analogicznej relacji *r*, na bazie zbioru wektorów dystansu *d*, relacji *R*. Jeżeli jest to możliwe, oznacza to że oprócz ograniczeń na dziedzinę i zakres, rozważana relacja *R*, posiada tylko i wyłącznie ograniczenia charakterystyczne dla relacji (3.7), zatem należy ona do klasy relacji delta (ang. *d-form relation*).

Algorytm 3.1 Rozpoznanie relacji zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form relation*).

Wejście: relacja *R*

Wyjście: *prawda* – jeżeli relacja *R* należy do klasy relacji delta

f*ałsz* – jeżeli relacja *R* nie należy do klasy relacji delta

Metoda:

1. Oblicz zbiór wektorów dystansu relacji *R* zgodnie z poniższym wzorem

   $d = \{[d_1,...,d_k] \mid \exists s_j, t_j \; s.t. \; 1 \le j \le k \wedge d_j = t_j - s_j \wedge \exists \alpha_1,...,\alpha_m \; s.t. \; F\}$.

   / Zbiór wektorów dystansu d, obliczamy jako różnicę wartości poszczególnych składowych krotki wyjściowej* t_j *i wejściowej* s_j*, gdzie F to ograniczenia dotyczące dziedziny i zakresu relacji R */*

2. Utwórz relację *r*, na bazie zbioru wektorów dystansu *d*

   $r = \{[i_1, i_2,..., i_k] \rightarrow [j_1, j_2,..., j_k] \mid \forall \, p, 1 \le p \le k, j_p - i_p = d_p\}$.

   / Relację r, utworzono na bazie wektora dystansu d, na nieograniczonej przestrzeni(bez ograniczeń na dziedzinę i zakres relacji)*

3. Nałóż ograniczenia na dziedzinę i zakres relacji *r*, zgodnie z ograniczeniami relacji *R*

   $$r = (r \setminus domain(R)) / range(R).$$

4. Sprawdź czy spełniony jest poniższy warunek

$$r = R\,.$$

5. **Jeżeli** $r = R$ *prawda*

   Jeżeli $r \neq R$ *fałsz*

Przykład 3.1

Rozważmy przebieg poszczególnych kroków algorytmu 3.1, dla przykładowej relacji *R*

Wejście: $R = \{[i, j] \to [i', j+2] \mid 1 \le i < i' \le n \wedge 1 \le j \le m-2\}$

1. $d = \{[d_1, 2] \mid 1 \le d_1 < n \wedge 3 \le m\}$
2. $r = \{[i_1, i_2] \to [j_1, j_2] \mid i_1 < j_1 \wedge j_2 - i_2 = 2 \wedge j_1 < n + i_1 \wedge 3 \le m\}$
3. $r = \{[i, j] \to [i', j+2] \mid 1 \le i < i' \le n \wedge 1 \le j \le m-2\}$
4. $r = R$
5. *prawda* – relacja *R* należy do klasy relacji delta

3.3 Relacja zależności reprezentująca graf o topologii łańcucha o stałym wektorze dystansu (ang. *uniform dependence relation*)

Relacja zależności *R*, należy do klasy relacji reprezentujących graf o topologii łańcucha o stałym wektorze dystansu, jeżeli jej ograniczenia dotyczą bezpośrednio różnicy, pomiędzy wartościami zmiennych krotki wyjściowej i wejściowej i różnice te mają zawsze charakter stały [72]. Postać ogólna relacji zależności *R*, reprezentującej graf o topologii łańcucha o stałym wektorze dystansu wygląda następująco:

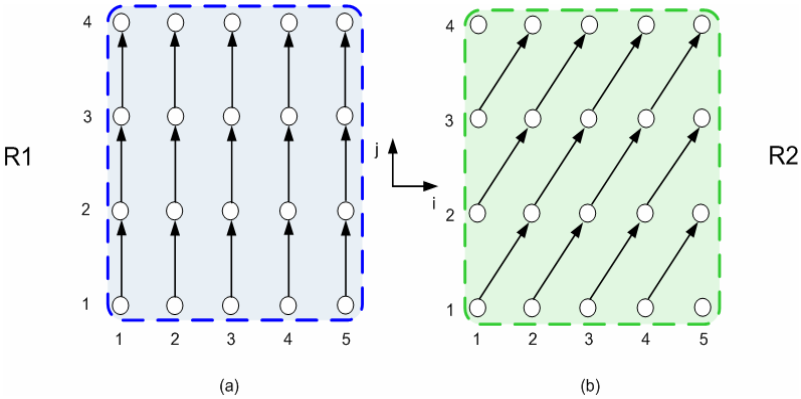
$$R = \{[i_1, i_2, ..., i_m] \to [j_1, j_2, ..., j_m] \mid \forall p, 1 \le p \le m,\ j_p - i_p = M_p\}\,, \tag{3.9}$$

gdzie $M_p \in Z$ i określa stały przyrost odległości.

Rysunek 3.3 przedstawia relacje R_1 (a) i R_2 (b) dla wartości $n = 5$ i $m = 4$, należące do klasy relacji o topologii łańcucha o stałym wektorze dystansu.

$$\begin{aligned} R_1 &= \{[i, j] \to [i, j+1] \mid 1 \le i \le n \wedge 1 \le j < m\}, \\ R_2 &= \{[i, j] \to [i+1, j+1] \mid 1 \le i < n \wedge 1 \le j < m\}. \end{aligned} \tag{3.10}$$

Poniżej przedstawiono algorytm 3.2, którego zadaniem jest określenie, czy rozważana relacja *R*, należy do klasy relacji reprezentujących graf o topologii łańcucha,



Rys 3.3 Przykłady relacji o topologii łańcucha o stałym wektorze dystansu [op. własne]

o stałym wektorze dystansu, poprzez obliczenie jej zbioru wektorów dystansu d i sprawdzenie czy dla minimalnego m i maksymalnego M, wektora dystansu należącego do zbioru d, zachodzi związek $m = M$. Jeżeli tak, oznacza to iż wektor dystansu jest stały, a rozważana relacja należy do relacji reprezentujących graf o topologii łańcucha o stałym wektorze dystansu.

Algorytm 3.2 Rozpoznanie klasy relacji reprezentującej graf o topologii łańcucha o stałym wektorze dystansu

Wejście: relacja R

Wyjście: *prawda* – jeżeli relacja R, należy do klasy relacji reprezentujących graf o topologii łańcucha o stałym wektorze dystansu

f*ałsz* – jeżeli relacja R, nie należy do klasy relacji reprezentujących graf o topologii łańcucha o stałym wektorze dystansu

Metoda:

1. Oblicz zbiór wektorów dystansu relacji R, zgodnie z poniższym wzorem

   $$d = \left\{ [d_1,...,d_k] \mid \exists s_j, t_j \; s.t. \; 1 \le j \le k \wedge d_j = t_j - s_j \wedge \exists \alpha_1,...,\alpha_m \; s.t. \; F \right\}.$$

   /* *Zbiór wektorów dystansu d, obliczamy jako różnicę wartości poszczególnych składowych krotki wyjściowej* t_j *i wejściowej* s_j*, gdzie* F *to ograniczenia dotyczące dziedziny i zakresu relacji R* */

2. Wyznacz minimalny $m = \min(d)$ i maksymalny $M = \max(d)$ wektor dystansu ze zbioru wektorów dystansu d.

/* *Wyznaczenie wektorów m i M, polega na dobraniu takich wartości poszczególnych składowych* $d_j \in d$, $1 \le j \le k$, *gdzie k to liczba składowych wektora dystansu, aby wartość wyrażenia* $\sqrt{d_1^2+d_2^2+...+d_k^2}$, *definiująca długość wektora w k-wymiarowej przestrzeni, była minimalna lub maksymalna* */

3. **Jeżeli** $m = M$ *prawda*
   Jeżeli $m \neq M$ *fałsz*

Przykład 3.2

Rozważmy przebieg poszczególnych kroków algorytmu 3.2 dla przykładowej relacji *R*.

Wejście: $R = \{[\![i, j]\!] \rightarrow [i+1, j+2] \mid 1 \le i < n \wedge 1 \le j \le m-2\}$

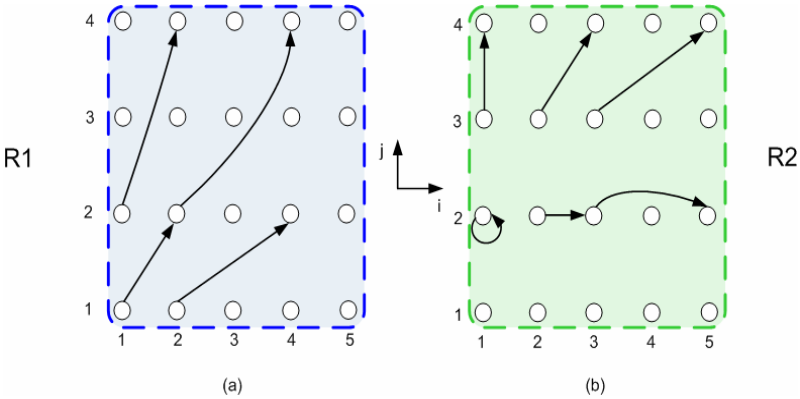
1. $d = \{[\![1, 2]\!]\}$
2. $m = \{[\![1, 2]\!]\}$, $M = \{[\![1, 2]\!]\}$
3. *prawda* – relacja *R* należy do klasy relacji reprezentujących graf o topologii łańcucha o stałym wektorze dystansu.

3.4 Relacja zależności reprezentująca graf o topologii łańcucha o zmiennym wektorze dystansu (ang. *relation describing dependence chains only*)

Relacja zależności *R*, należy do klasy relacji reprezentujących graf o topologii łańcucha o zmiennym wektorze dystansu, jeżeli spełnione są następujące warunki: $\min(d) \neq \max(d)$, gdzie *d*, to zbiór wektorów dystansu relacji *R* i każdy koniec/początek zależności opisywanej przez krotkę relacji posiada dokładnie jeden początek/koniec, tzn. nie istnieją wspólne końce/początki dla dwóch, lub większej liczby krawędzi opisanych przy pomocy danej relacji [19].

Rysunek 3.4 przedstawia relacje R_1(a) i R_2(b) dla wartości $n = 5$ i $m = 4$, należące do klasy relacji zależności reprezentujących graf o topologii łańcucha o zmiennym wektorze dystansu.

$$R_1 = \{[\![i, j]\!] \rightarrow [2i, 2j+1] \mid 1 \le i \wedge 1 \le j \wedge 2i \le n \wedge 2j < m\},$$
$$R_2 = \{[\![i, j]\!] \rightarrow [2i-1, 2j-2] \mid 1 \le i \wedge 1 \le j \wedge 2i \le 1+n \wedge 2j < 2+m\}. \quad (3.11)$$



Rys 3.4 Przykłady relacji reprezentujących graf o topologii łańcucha o zmiennym wektorze dystansu [op. własne]

Poniżej przedstawiono algorytm 3.3, którego zadaniem jest określenie czy rozważana relacja *R* należy do klasy relacji reprezentujących graf o topologii łańcucha, o zmiennym wektorze dystansu, poprzez wykluczenie istnienia stałego wektora dystansu i zweryfikowanie jej łańcuchowej topologii, sprawdzając czy istnieją takie s_i i s_j, że $s_i \neq s_j$ i $R(s_i) = R(s_j)$ (co dyskwalifikuje łańcuchową topologię relacji, ze względu na wspólne końce zależności) i t_i i t_j, że $t_i \neq t_j$ i $R^{-1}(t_i) = R^{-1}(t_j)$ (co dyskwalifikuje łańcuchową topologię relacji, ze względu na wspólne początki zależności).

Algorytm 3.3 Rozpoznanie klasy relacji reprezentującej graf o topologii łańcucha o zmiennym wektorze dystansu.

Wejście: relacja *R*

Wyjście: *prawda* – jeżeli relacja *R*, należy do klasy relacji reprezentujących graf o topologii łańcucha o zmiennym wektorze dystansu

f*ałsz* – jeżeli relacja *R*, nie należy do klasy relacji reprezentujących graf o topologii łańcucha o zmiennym wektorze dystansu

Metoda:

1. Utwórz leksykograficznie dodatnią relację *LF*, o wymiarach krotek wejściowej i wyjściowej, zgodnych z wymiarami krotek relacji *R* taką, że $\forall x \rightarrow y \in LF, 0 \prec y - x$.

2. Wyznacz minimalny $m = \min(d)$ i maksymalny $M = \max(d)$ wektor dystansu ze zbioru wektorów dystansu d.

   /* *Wyznaczenie wektorów m i M, polega na dobraniu takich wartości poszczególnych składowych* $d_j \in d$, $1 \le j \le k$, *gdzie k to liczba składowych wektora dystansu, aby wartość wyrażenia* $\sqrt{d_1^2+d_2^2+...+d_k^2}$, *definiująca długość wektora w k-wymiarowej przestrzeni, była minimalna lub maksymalna* */

3. Sprawdź czy istnieją takie s_i i s_j, że $s_i \neq s_j$ i $R(s_i) = R(s_j)$ korzystając ze wzoru

$$s = R^{-1} \cap \left(LF^{-1} \circ R^{-1}\right).$$

   /* *Wykluczenie wspólnych początków zależności* */

4. Sprawdź czy istnieją takie t_i i t_j, że $t_i \neq t_j$ i $R^{-1}(t_i) = R^{-1}(t_j)$, korzystając ze wzoru

$$t = R \cap \left(LF \circ R\right).$$

   /* *Wykluczenie wspólnych końców zależności* */

5. *Prawda* - **jeżeli** $(m \neq M) \wedge (s = t = \varnothing)$.

   Fałsz - w przeciwnym wypadku.

Przykład 3.3

Rozważmy przebieg poszczególnych kroków algorytmu 3.3 dla przykładowej relacji *R*.

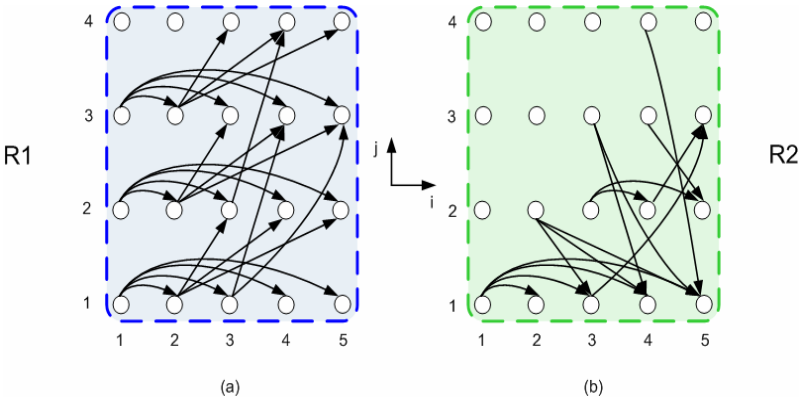
Wejście: $R = \{[\![i, j]\!] \rightarrow [2i, 2j+1] \mid 1 \le i \wedge 1 \le j \wedge 2i \le n \wedge 2j \le m\}$

1. $LF = \{[\![i, j]\!] \rightarrow [i', j'] \mid i < i'\} \cup \{[\![i, j]\!] \rightarrow [i, j'] \mid j < j'\}$
2. $M = \{[\![d_1, d_2]\!] | 2d_1 \le n \le 2d_1 + 1 \wedge 2d_2 - 1 \le m \le 2d_2 \wedge 1 \le d_1 \wedge 2 \le d_2\}$

   $m = \{[\![1, 2]\!] \mid 2 \le n \wedge 3 \le m\}$
3. $s = R^{-1} \cap \left(LF^{-1} \circ R^{-1}\right) = \varnothing$
4. $t = R \cap \left(LF \circ R\right) = \varnothing$
5. $(m \neq M) \wedge (s = t = \varnothing)$, *prawda* – relacja *R*, należy do klasy relacji reprezentujących graf o topologii łańcucha o zmiennym wektorze dystansu.

3.5 Relacja zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej (ang. *dependence relation with coupled index variables*)

Relacja zależności R, należy do klasy relacji o powiązanych elementach składowych krotek, jeżeli co najmniej jedna ze składowych krotki wejściowej lub wyjściowej przyjmuje postać wyrażenia zawierającego dwie lub więcej zmiennych [19].



Rys. 3.5 Przykłady relacji zależności o powiązanych składowych krotek [op. własne]

Rysunek 3.5 przedstawia relacje R_1(a) i R_2(b) dla wartości $n = 5$ i $m = 4$, należące do klasy relacji o powiązanych elementach składowych krotek wyjściowych.

$$\begin{aligned} R_1 &= \{[i,j] \rightarrow [i', i+j-1] \mid 1 \le i < i' \le n \wedge 1 \le j \wedge i+j \le 1+m\}, \\ R_2 &= \{[i,j] \rightarrow [i', i-j+1] \mid 1 \le j \le i < i' \le n \wedge j \le m \wedge i < m+j\}. \end{aligned} \tag{3.12}$$

Poniżej przedstawiono algorytm 3.4, którego zadaniem jest określenie, czy rozważana relacja R należy do klasy relacji o powiązanych elementach składowych krotek wejściowej lub wyjściowej, poprzez utworzenie m relacji zależności, (m to liczba składowych krotek relacji R) dla każdej pary elementów składowej krotki wejściowej i wyjściowej, wraz z odpowiadającymi im ograniczeniami (ograniczenia, które nie dotyczą bezpośrednio danej pary rozważanych elementów są pomijane). Następnie tworzymy relację zależności o m składowych krotki wejściowej i wyjściowej, dodając do niej ograniczenia z wyznaczonych uprzednio m relacji zależności. Jeżeli relacja ta, nie jest tożsama z relacją wejściową, oznacza to, że składowe relacji wejściowej, nie są w pełni niezależne względem pozostałych składowych. Zatem, rozważana relacja

należy do klasy relacji o powiązanych elementach składowych krotek wejściowej lub wyjściowej.

Algorytm 3.4 Rozpoznanie klasy relacji o powiązanych elementach składowych krotek wejściowej lub wyjściowej.

Wejście: relacja *R*

Wyjście: *prawda* – jeżeli relacja *R*, należy do klasy relacji zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej.

f*ałsz* – jeżeli relacja *R*, nie należy do klasy relacji zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej.

Metoda:

1. Utwórz relacje R_i, $1 \le i \le m$ w następujący sposób:

   $R_i = \{ [s_i] \rightarrow [t_i] \mid \exists s_j, t_j : 1 \le j \ne i \le m \wedge$

   $[s_1, s_2, ..., s_m]$: ograniczenia afiniczne z $R \wedge$

   $[t_1, t_2, ..., t_m]$: ograniczenia afiniczne z R $\}$,

   gdzie *m* to liczba składowych krotek relacji *R*.

   / Każda z krotek relacji R_i, $1 \le i \le m$ zawiera tylko i wyłącznie i-tą składową relacji R. Pozostałe składowe są ignorowane, a ich zmienne wyeliminowane poprzez nałożenie na nie kwantyfikatora egzystencjalnego.*/*

2. Utwórz relację R_c, jako połączenie poszczególnych relacji R_i, $1 \le i \le m$, w następujący sposób:

   $R_c = \{ [s_1, s_2, ..., s_m] \rightarrow [t_1, t_2, ..., t_m] \mid \overset{m}{\underset{i=0}{\forall}}$ ograniczenia nałożone na

   składową s_i z R_i $\wedge$ ograniczenia nałożone na składową t_i z R_i $\}$.

   / Tworzymy relację zależności o m składowych krotki wejściowej i wyjściowej, dodając do niej ograniczenia z wyznaczonych uprzednio m relacji zależności. */*

3. *Prawda* - **jeżeli** $R_c \ne R$**.**

   / składowe relacji wejściowej R nie są w pełni niezależne względem pozostałych jej składowych. */*

   Fałsz - w przeciwnym wypadku

Przykład 3.4

Rozważmy przebieg poszczególnych kroków algorytmu 3.4 dla przykładowej relacji *R*

Wejście: $R = \{[l,i,k] \rightarrow [l',i-k-1,k'] \mid 1 \le l < l' \le m \wedge i < n \wedge 0 \le k' \wedge 2+k+k' \le i \wedge 0 \le k\}$

1. $R_1 = \{[l] \rightarrow [l'] \mid \exists i,k,k' : 1 \le l < l' \le m \wedge i < n \wedge 0 \le k' \wedge 2+k+k' \le i \wedge 0 \le k\}$ =

   $\{[l] \rightarrow [l'] \mid 1 \le l < l' \le m \wedge 3 \le n\}$

   $R_2 = \left\{ [i] \rightarrow [i'] \mid \exists l,l',k,k' : i' = i-k-1 \wedge 1 \le l < l' \le m \wedge i < n \wedge 0 \le k' \wedge 2+k+k' \le i \wedge 0 \le k \right\}$ =

   $\{[i] \rightarrow [i'] \mid 1 \le i' < i < n \wedge 2 \le m\}$

   $R_3 = \{[k] \rightarrow [k'] \mid \exists l,l',i : 1 \le l < l' \le m \wedge i < n \wedge 0 \le k' \wedge 2+k+k' \le i \wedge 0 \le k\}$ =

   $\{[k] \rightarrow [k'] \mid 0 \le k \wedge 0 \le k' \wedge 2 \le m \wedge 3+k+k' \le n\}$

2. $R_c = \left\{ [l,\ i,\ k] \rightarrow [l',i',k'] \mid \underbrace{1 \le l < l' \le m \wedge 3 \le n}_{\text{ograniczenia z } R_1} \wedge \underbrace{1 \le i' < i < n \wedge 2 \le m}_{\text{ograniczenia z } R_2} \wedge \underbrace{0 \le k \wedge 0 \le k' \wedge 2 \le m \wedge 3+k+k' \le n}_{\text{ograniczenia z } R_3} \right\}$

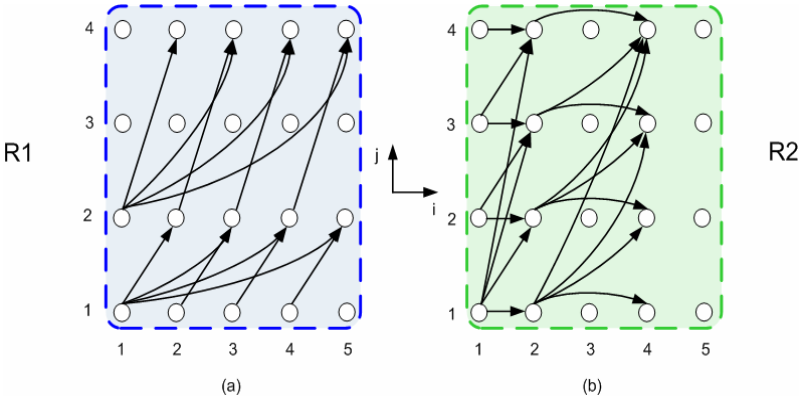
3. $R_c \neq R$, *prawda* – relacja *R* należy do klasy relacji zależności o powiązanych elementach składowych krotek wyjściowej.

3.6 Relacja zależności o niepowiązanych elementach składowych krotek wejściowej i wyjściowej (ang. *dependence relation with non-coupled index variables*)

Relacja zależności *R*, należy do klasy relacji o niepowiązanych elementach składowych krotek, jeżeli każda ze składowych krotki wejściowej i wyjściowej przyjmuje postać wyrażenia zawierającego tylko jedną zmienną [19].

Rysunek 3.6 przedstawia relacje R_1(a) i R_2(b) dla wartości $n = 5$ i $m = 4$, należące do klasy relacji o niepowiązanych elementach składowych krotek wejściowej i wyjściowej.

$$\begin{aligned} R_1 &= \{[i,\ j] \rightarrow [i',\ 2j] \mid 1 \le i < i' \le n \wedge 1 \le\ j \wedge 2j \le m\}, \\ R_2 &= \{[i,\ j] \rightarrow [2i,\ j'] \mid 1 \le\ j,\ j' \le m \wedge 1 \le i \wedge 2i \le n\}. \end{aligned} \tag{3.13}$$



Rys 3.6 Przykłady relacji zależności o niepowiązanych składowych krotek [op. własne]

Algorytm rozpoznania klasy relacji o niepowiązanych elementach składowych krotek wejściowej i wyjściowej jest analogiczny do algorytmu 3.4. Różnica dotyczy tylko i wyłącznie warunku końcowego, który w przypadku relacji zależności o niepowiązanych elementach składowych krotek wejściowej i wyjściowej powinien spełniać następującą własność $R_c = R$, co oznacza że składowe relacji wejściowej są w pełni niezależne względem pozostałych składowych.

Przykład 3.5

Rozważmy przebieg poszczególnych kroków algorytmu 3.4 dla przykładowej relacji *R*.

Wejście: $R = \{[i, j] \rightarrow [i', 2j] \mid 1 \le i < i' \le n \wedge 1 \le j \wedge 2j \le m\}$

1. $R_1 = \{[i] \rightarrow [i'] \mid \exists j : 1 \le i < i' \le n \wedge 1 \le j \wedge 2j \le m\} =$
   $\{[i] \rightarrow [i'] \mid 1 \le i < i' \le n \wedge 2 \le m\}$

   $R_2 = \{[j] \rightarrow [2j] \mid \exists i, i' : 1 \le i < i' \le n \wedge 1 \le j \wedge 2j \le m\} =$
   $\{[j] \rightarrow [2j] \mid 1 \le j \wedge 2j \le m \wedge 2 \le n\}$

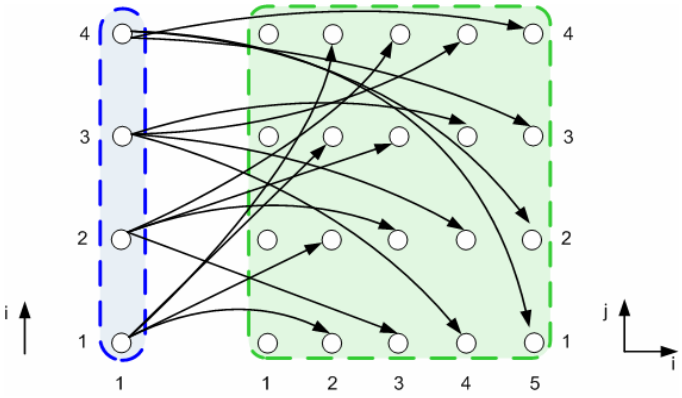
2. $R_c = \left\{ [i, j] \rightarrow [i', 2j] \mid \underbrace{1 \le i < i' \le n \wedge 2 \le m}_{\text{ograniczenia z } R_1} \wedge \underbrace{1 \le j \wedge 2j \le m \wedge 2 \le n}_{\text{ograniczenia z } R_2} \right\}$

3. $R_c = R$, *prawda* – relacja *R*, należy do klasy relacji zależności o niepowiązanych elementach składowych krotek wejściowej i wyjściowej.

3.7 Relacja zależności o różnych wymiarach krotek wejściowej i wyjściowej (ang. *dependence relation with different numbers of index variables of input and output relation tuples*)

Relacja zależności R, należy do klasy relacji o różnych wymiarach krotek wejściowej i wyjściowej, jeżeli liczba składowych krotki wejściowej różni się od liczby składowych krotki wyjściowej [19].

Rysunek 3.7, przedstawia relację $R = \{[i] \rightarrow [i+1, j] \mid 1 \leq i < n \wedge 1 \leq j \leq m\}$, należącą do klasy relacji zależności o różnych wymiarach krotek wejściowej i wyjściowej.



Rys 3.7 Przykład relacji zależności o różnych wymiarach krotek wejściowej i wyjściowej [op. własne]

3.8 Podsumowanie

Pojęcie klas relacji zależności, nie jest zagadnieniem nowym. W wielu pracach [2, 3, 9, 30, 31, 32, 33, 48, 52 i 72] dokonywano podziału relacji zależności na poszczególne grupy. Ich nazwy, nie koniecznie odpowiadają tym zaproponowanym w obecnym rozdziale i artykule [19], gdzie oprócz klasyfikacji relacji zależności, zaproponowano również autorskie algorytmy umożliwiające ich rozpoznanie w sposób formalny.

Dysponowanie mechanizmami, których działanie polega na określeniu klasy, do jakiej należy rozpatrywana relacja prosta (składające się z pojedynczej koniunkcji), jest o tyle ważne, że pozwala na zastosowanie odpowiedniego algorytmu obliczania tranzytywnego domknięcia takiej relacji, co w porównaniu z dotąd stosowaną praktyką, sprowadzało się do próby zastosowania wielu znanych sposobów obliczania tranzytywnego domknięcia sparametryzowanej relacji zależności (przedstawionych w kolejnym rozdziale), weryfikacji otrzymanego wyniku i kontynuowania tego procesu, poprzez rozpatrzenie możliwości wykorzystania kolejnych podejść, w przypadku porażki poprzedniego rozwiązania (uzyskanie wyniku w postaci przybliżenia górnego, bądź dolnego dla tranzytywnego domknięcia sparametryzowanej relacji zależności).

4. Algorytmy obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności

Analiza zagadnień dotyczących **grafów sparametryzowanych**, czyli takich których liczbę węzłów określa się wyrażeniem zawierającym parametry(zmienne symboliczne) należy do aktualnych, ale nie do końca zbadanych pojęć teorii grafów [9, 15, 17, 18, 20, 21, 22, 30, 31, 35, 72]. **Graf sparametryzowany *G***, składa się ze sparametryzowanego zbioru $V(G)$, którego elementy nazywamy wierzchołkami i sparametryzowanej rodziny $E(G)$, par nieuporządkowanych elementów zbioru $V(G)$, nazywanych krawędziami [30, 31, 72]. Wiele wcześniejszych definicji i algorytmów, przedstawionych w rozdziale 2, dotyczącym wprowadzenia do ogólnej teorii grafów, można rozszerzyć na grafy sparametryzowane. Jednak podstawowym ograniczeniem, pojawiającym się szczególnie w zadaniach polegających na określeniu istniejących połączeń pomiędzy wierzchołkami w grafie [1, 5, 45, 52, 59, 60, 62, 65, 66, 68, 100] jest to, że każdy z analizowanych wierzchołków musi być traktowany indywidualnie, co w przypadku grafów sparametryzowanych, nie jest akceptowalne z uwagi na brak możliwości spełnienia warunku stopu wielu algorytmów [18, 39, 48, 52, 62, 65, 72]. Kluczowe zatem staje się zastosowanie odpowiedniej formy ich reprezentacji.

Najbardziej popularne okazały się: drzewa wierzchołków sąsiednich zaproponowane przez Dar i Jagadish [38], łańcuchy wierzchołków sąsiednich Jakobsona [63, 64] i przedziały wierzchołków sąsiednich [79, 80, 83], powstałe w wyniku ich ponumerowania w pewien usystematyzowany sposób. Każda z powyższych form reprezentacji grafu o nieokreślonej, lecz w zamyśle dużej liczbie wierzchołków, bazuje na ich podziale na kolekcje rozłącznych zbiorów, a następnie na poddaniu ich procesowi kompresji, w celu oszczędności rozmiaru pamięci niezbędnej do ich

przechowywania jak i minimalizacji liczby wykonywanych do niej odwołań. Nie pozwoliło to jednak, na całkowite wyeliminowanie konieczności w większości przypadków tylko częściowego ponowienia procesów obliczeniowych, w sytuacji zmiany parametrów wejściowych (dodanie bądź usunięcie wierzchołka lub krawędzi). Problem taki, nie występuje jeśli do reprezentacji grafów sparametryzowanych zdecydujemy się na wykorzystanie relacji krotek [16, 48, 72, 83], które zyskały swoją popularność, dzięki zastosowaniu ich w analizatorach dokonujących dokładnej analizy zależności, zawartych w pętlach programowych [70, 88, 90, 91]. W ich przypadku, składowe krotek wejściowej i wyjściowej symbolizują wierzchołki grafu, a afiniczne ograniczenia, dopuszczalne połączenia pomiędzy tymi wierzchołkami. Pomiędzy zadaną parą wierzchołków, reprezentowanych przez składowe krotki wejściowej i wyjściowej istnieje krawędź, jeżeli ograniczenia w postaci równań i nierówności dla tych składowych są spełnione. Szczegóły dotyczące tej formy reprezentacji grafów, zaprezentowane zostały w podrozdziale 2.3.1.

W niniejszym rozdziale, przedstawiono algorytmy obliczenia tranzytywnego domknięcia w grafach sparametryzowanych, o liczbie wierzchołków przedstawionych w formie zmiennej symbolicznej, w skończonej liczbie kroków.

4.1 Obliczanie tranzytywnego domknięcia relacji zależności składającej się z pojedynczej koniunkcji (ang. *simple/single conjunct relation*)

Tranzytywne domknięcie relacji zależności R, definiuje się w następujący sposób [72] :

$$R^+ = \bigcup_{k=1}^{\infty} R^k \text{ , gdzie } R^k = \underbrace{R \circ R \circ \ldots \circ R}_{k \text{ razy}}. \tag{4.1}$$

Istnieją dwie relacje związane z tranzytywnym domknięciem:

- tranzytywne domknięcie :

$$x \to z \in R^* \Leftrightarrow x = z \vee \exists y \, s.t. \; x \to y \in R \wedge \; y \to z \in R^*, \tag{4.2}$$

- dodatnie tranzytywne domknięcie :

$$x \to z \in R^+ \Leftrightarrow x \to z \in R \vee \exists y \, s.t. \; x \to y \in R \wedge \; y \to z \in R^+. \tag{4.3}$$

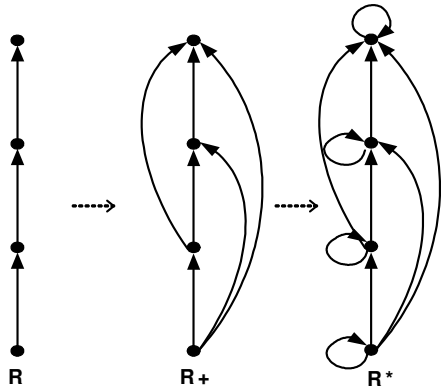
Rysunek 4.1, przedstawia graf reprezentowany przez relację R (4.4), dla wartości $n = 4$,

$$R = \{[i] \rightarrow [i+1] \mid 1 \le i < n\}, \tag{4.4}$$

wraz z odpowiadającymi jej relacjami R^+ i R^* (4.5)

$$\begin{aligned} R^+ &= \{[i] \rightarrow [i'] \mid 1 \le i < i' < n\}, \\ R^* &= \{[i] \rightarrow [i'] \mid 1 \le i \le i' < n\}. \end{aligned} \tag{4.5}$$

Z powyższego wzoru (4.1) wynika, iż proces obliczenia relacji R^+, powinien być poprzedzony wyznaczeniem relacji R^k. Najprostsze rozwiązanie umożliwiające obliczenie relacji R^k, sprowadza się do zastosowania algorytmu iteracyjnego [3, 5, 18, 20, 48, 56, 62, 79, 80, 83, 101].



Rys 4.1 Relacja reprezentująca graf, wraz z jej dodatnim tranzytywnym domknięciem i tranzytywnym domknięciem [99]

Przykład takiego algorytmu, przedstawia rysunek 4.2. W algorytmie [83] z rysunku 4.2, relacja R^i zawiera zależności przechodnie obliczone w *i*-tej iteracji, a R^k sumę tych zależności, uzyskanych w iteracjach od pierwszej do *i*-1. Algorytm ten, poddano licznym modyfikacjom, mającym na celu skrócenie czasu jego działania, poprzez zwiększenie przyrostu zależności przechodnich uzyskanych w danej iteracji, zgodnie ze wzorem (4.4).

$$R^+ = \bigcup_{k=1}^{\infty} R^k = R \circ \prod_{k=1}^{\infty} \left(I \cup R^{2^k}\right) = R \circ (I \cup R) \circ \left(I \cup R^2\right) \circ \left(I \cup R^4\right) \ldots \tag{4.4}$$

Pomimo wyraźnej prostoty, algorytmy iteracyjne nie mogą być wykorzystywane wprost do wyznaczenia dokładnego domknięcia przechodniego relacji sparametryzowanych,

```
1   R^k = ∅
2   R^1 = R
3    i = 2
4   do
5       R^i = R^(i−1) ∘ R
6       if (R^i = ∅) ∨ (R^i − R^(i−1) − R^(i−2) − ... − R^1 = ∅) then
7          begin
8             R^k = (R^1 ∧ k = 1) ∪ (R^2 ∧ k = 2) ∪ ... ∪ (R^(i−1) ∧ k = i − 1)
9             break
10         end
11       i = i + 1
12  while i < N
13  if  R^k = ∅  then
14    begin
15       print „Obliczenie dokładnego  R^k  nie powiodło się”
16    end
```

Rys 4.2 Iteracyjny algorytm obliczania potęgi *k* relacji [op. własne]

ponieważ w ich przypadku warunek stopu (linia nr 6 algorytmu z rysunku 4.2), nie zawsze będzie spełniony. Na przykład, dla następującej relacji:

$$R = \{[m,n] \rightarrow [m+1, m-n+1] : 1 \le n < m \le P-2\}, \tag{4.5}$$

metoda iteracyjna zawodzi. Zgodnie z bieżącym stanem wiedzy, nie istnieje sformalizowane podejście umożliwiające obliczenie relacji R^+ w ogólnym przypadku [30, 31, 72]. Dla relacji składającej się z pojedynczej koniunkcji, postaci $R = \{[x] \rightarrow [Ax+b] \mid Cx \ge d\}$, gdzie *A* i *C* to macierze a *b* i *d* to wektory o elementach należących do zbioru liczb całkowitych, Boigelot w [30] zdefiniował warunek konieczny (twierdzenie 8.53, strona 236), determinujący możliwość obliczenia relacji R^+, w sposób dokładny. Kelly z kolei w [72], zaproponował wyodrębnienie specjalnej klasy relacji zależności, nazywanej klasą relacji ang. *d-form*, w przypadku której obliczenie jej tranzytywnego domknięcia jest zawsze możliwe. Na podobny krok, zdecydował się i autor tej pracy, rozszerzając zbiór relacji prostych o kolejne klasy relacji [19], dla których rozwiązanie rozważanego problemu, wraz z uzyskaniem dokładnego wyniku jest wykonalne.

W podrozdziałach 4.1.1-6, przedstawiono techniki obliczania tranzytywnego domknięcia dla poszczególnych klas relacji zależności po ich uprzednim rozpoznaniu, przy pomocy algorytmów 3.1-4. Należy jednak pamiętać o tym, że do ich poprawnego zastosowania niezbędne jest, aby rozważana relacja *R* składała się tylko i wyłącznie z pojedynczej koniunkcji tzn. nie może być ona unią wielu relacji, których krotki wejściowe i wyjściowe są różne. Jedyne odstępstwo od tej reguły dotyczy unii relacji, których krotki wejściowe i wyjściowe są jednakowe, tak jak dla przykładowej relacji *R*

$$R = \{[i,1] \rightarrow [i+1,1] \mid 1 \le i < 24\} \cup \{[i,1] \rightarrow [i+1,1] \mid 10 \le i < 49\}. \tag{4.6}$$

Wówczas tworzymy jedną relację, która zawiera połączenie ograniczeń, wszystkich relacji za pomocą operatora *OR.* Dla powyższej relacji (4.6), będzie to relacja *R* (4.7)

$$R = \{[i,1] \rightarrow [i+1,1] \mid 1 \le i < 24 \; OR \; 10 \le i < 49\}. \tag{4.7}$$

Przypadek taki, w którym relacja zależności składa się z wielu koniunkcji został rozważony w podrozdziale 4.2.

4.1.1 Obliczanie tranzytywnego domknięcia relacji zależności o różnych wymiarach krotek wejściowej i wyjściowej

Dla relacji zależności *R*, o różnych wymiarach krotek wejściowej i wyjściowej spełniony jest następujący warunek $domain(R) \cap range(R) = \varnothing$. Oznacza to, iż relacja *R* nie tworzy łańcucha połączonych ze sobą krawędzi grafu i tym samym R^+ przyjmuje postać [20] :

$$R^+ = \begin{cases} R & \text{dla} \quad k = 1 \\ \varnothing & \text{dla} \quad k > 1 \end{cases}. \tag{4.8}$$

Przykład 4.1

Rozważmy przykład relacji $R = \{[i] \rightarrow [i', j] : 1 \le i < i' \le n \wedge 1 \le j \le m\}$.

Ponieważ $domain(R) \cap range(R) = \varnothing$, dlatego

$$R^+ = \{[i] \rightarrow [i', j] : 1 \le i < i' \le n \wedge 1 \le j \le m\}. \tag{4.9}$$

4.1.2 Obliczanie tranzytywnego domknięcia relacji zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form dependence relation*).

Tranzytywne domknięcie relacji zależności *R*, o zmiennym wektorze dystansu klasy delta zapisujemy zgodnie z następującym wzorem [72] :

$$R^{+}=\left\{\begin{array}{l}[i_1,i_2,\ldots,i_m]\rightarrow[j_1,j_2,\ldots,j_m] \mid \exists k>0\ s.t.\forall p, 1\le p\le m, \\ \qquad\qquad \exists\alpha_p\ s.t.\ L_p k\le j_p-i_p\le U_p k \wedge j_p-i_p=M_p\alpha_p\end{array}\right\}, \quad (4.10)$$

gdzie L_p, U_p, $M_p\in Z$ i definiują one wartości stałe, nakładające ograniczenia na różnicę składowych krotki wyjściowej j_p , wejściowej i_p i zmiennych egzystencjalnych α_p. W przypadku gdy $L_p=-\infty$ lub $U_p=+\infty$, ograniczenie takie jest pomijane przy zapisie relacji.

Przykład 4.2

Rozważmy przykład relacji $R=\{[i,j]\rightarrow[i',j+2]: 1\le i<i'\le n \wedge 1\le j\le m-2\}$.

Dla powyższej relacji, R^{+} wyznaczone na podstawie wzoru (4.10), wygląda następująco:

$$R^{+}=\left\{\begin{array}{l}[i,j]\rightarrow[i',j'] \mid \exists k:\ k>0 \wedge \underbrace{i'-i\ge k}_{1} \wedge \underbrace{j'-j=2k}_{2} \wedge \\ \qquad \underbrace{1\le i\le n-1 \wedge 1\le j\le m-2}_{3} \wedge \underbrace{2\le i'\le n \wedge 3\le j'\le m}_{4}\end{array}\right\}.$$

Gdzie poszczególne ograniczenia oznaczają kolejno:

(1) ograniczenia nałożone na różnicę składowych krotki wyjściowej *i'* i wejściowej *i*, dla danego *k*.

(2) ograniczenia nałożone na różnicę składowych krotki wyjściowej *j'* i wejściowej *j*, dla danego *k*.

(3) ograniczenia definiujące dziedzinę relacji *R*.

(4) ograniczenia definiujące zakres relacji *R*.

4.1.3 Obliczanie tranzytywnego domknięcia relacji zależności reprezentującej graf o topologii łańcucha o stałym wektorze dystansu (ang. *uniform dependence relation*)

Obliczanie tranzytywnego domknięcia relacji zależności *R*, o stałym wektorze dystansu, tworzącej graf o topologii łańcucha, nie odbiega znacząco od podejścia dotyczącego relacji o zmiennym wektorze dystansu klasy delta, przedstawionego w podrozdziale 4.1.2 i wygląda następująco [72] :

$$R^{+} = \left\{ [i_1, i_2, ..., i_m] \rightarrow [j_1, j_2, ..., j_m] \mid \exists k > 0 \text{ s.t. } \forall p, 1 \le p \le m,\ j_p - i_p = M_p k \right\}, \qquad (4.11)$$

gdzie $M_p \in Z$, definiuje wartości stałe nakładające ograniczenia na różnicę składowych krotki wyjściowej j_p i wejściowej i_p. Z zapisu (4.11) wynika, iż w przypadku relacji zależności o stałym wektorze dystansu, tworzącej graf o topologii łańcucha w odróżnieniu od relacji zależności o zmiennym wektorze dystansu klasy delta, mamy do czynienia ze stałym przyrostem odległości M_p dla poszczególnych wartości zmiennej *k*.

Przykład 4.3

Rozważmy przykład relacji $R = \{[i, j] \rightarrow [i+1, j-1] : 1 \le i < n \wedge 2 \le j \le m\}$.

Dla powyższej relacji R^{+} wyznaczone na podstawie wzoru (4.11) wygląda następująco:

$$R^{+} = \left\{ [i, j] \rightarrow [i', j'] \mid \exists k :\ k > 0 \wedge \underbrace{i' - i = k}_{1} \wedge \underbrace{j' - j = -k}_{2} \wedge \underbrace{1 \le i \le n-1 \wedge 2 \le j \le m}_{3} \wedge \underbrace{2 \le i' \le n \wedge 1 \le j' \le m-1}_{4} \right\}.$$

Gdzie poszczególne ograniczenia oznaczają kolejno:

(1) ograniczenia nałożone na różnicę składowych krotki wyjściowej *i'* i wejściowej *i*, dla danego *k*.

(2) ograniczenia nałożone na różnicę składowych krotki wyjściowej *j'* i wejściowej *j*, dla danego *k*.

(3) ograniczenia definiujące dziedzinę relacji *R*

(4) ograniczenia definiujące zakres relacji *R*

4.1.4 Obliczanie tranzytywnego domknięcia relacji zależności reprezentującej graf o topologii łańcucha o zmiennym wektorze dystansu (ang. *relation describing dependence chains only*)

Przed przystąpieniem do wyznaczenia tranzytywnego domknięcia dla powyższej klasy relacji zależności, niezbędne jest, aby spełniała ona dodatkowe wymaganie dotyczące liczby zmiennych zawartych w opisie relacji. Mianowicie, w celu uproszczenia i zachowania przejrzystości przeprowadzonych obliczeń, powinna ona być mniejsza, lub równa liczbie zmiennych indeksujących pętli. Dla przykładowej relacji zależności $R=\{[i,j]\rightarrow[i',j] \mid i'=i+1 \wedge -j+1=0 \wedge 1\leq i<24\}$, powyższe wymaganie nie jest spełnione, w związku z tym, rozwiązujemy wszystkie równania i nierówności będące elementami ograniczeń tej relacji. W wyniku przeprowadzonej transformacji, otrzymujemy relację $R=\{[i,1]\rightarrow[i+1,1] \mid 1\leq i<24\}$.

Dysponując relacją *R*, spełniającą wszystkie przedstawione wcześniej wymagania [22], przystępujemy do obliczenia tranzytywnego domknięcia relacji zależności, reprezentującej graf o topologii łańcucha o zmiennym wektorze dystansu, w przypadku której, połączone krawędzie symbolizujące istniejące zależności, tworzą łańcuchy o początkach należących do zbioru dziedziny i zakresu relacji. Własność ta sprawia, że możliwe jest wygenerowanie ciągu wyznaczającego poszczególne wierzchołki należące do takich łańcuchów, po uprzednim utworzeniu i rozwiązaniu układu równań rekurencyjnych [2, 8, 15, 22, 56, 62] . Dla relacji *R* postaci (4.12) :

$$\left\{ A\cdot\begin{bmatrix} i_1 \\ i_2 \\ \dots \\ i_n \end{bmatrix}+\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}\rightarrow C\cdot\begin{bmatrix} i_1 \\ i_2 \\ \dots \\ i_n \end{bmatrix}+\begin{bmatrix} d_1 \\ d_2 \\ \dots \\ d_n \end{bmatrix} \mid \text{ograniczenia afiniczne} \right\}, \tag{4.12}$$

gdzie *A* i *C* to macierze współczynników o wymiarach $n\times n$, układ równań rekurencyjnych (4.13) wraz z wartościami początkowymi (4.14) wygląda następująco :

$$Ai^k+b=Ci^{k-1}+d \ , \tag{4.13}$$

$$\left[i_1^0,i_2^0,\dots,i_n^0\right]=\left[t_1,t_2,\dots,t_n\right] . \tag{4.14}$$

Dla przykładowej relacji : $R=\{[i]\rightarrow[i+1]\}$, tworzymy równanie rekurencyjne $i^k=i^{k-1}+1$, którego rozwiązaniem względem wartości początkowej i^0 jest : $i^k=i_0+k$. Układy równań rekurencyjnych (4.13), z ograniczeniami (4.14) mogą być rozwiązywane przez

wiele akademickich i komercyjnych narzędzi takich jak: Maple [117], Mathematica [112], Maxima [118] i MuPAD [119]. Dysponując rozwiązaniem $\left[i_1^{k,t}, i_2^{k,t}, ..., i_n^{k,t}\right]$ układów równań (4.13) i (4.14) możliwe jest utworzenie relacji R^+ postaci (4.15) :

$$R^+ = \left\{ \begin{array}{l} \left[t_1, t_2, ..., t_n\right] \rightarrow \left[x_1^{k,t}, x_2^{k,t}, ..., x_n^{k,t}\right] \mid \exists\, k : k > 0 \wedge \underbrace{t = \left[t_1, t_2, ..., t_n\right] \in domain(R)}_{1} \wedge \\ \underbrace{x_j^{k,t} = i_j^{k,t} \wedge 1 \le j \le n}_{2} \wedge \underbrace{x^{k,t} = \left[x_1^{k,t},\ x_2^{k,t}, ..., x_n^{k,t}\right] \in range(R)}_{3} \end{array} \right\}, \tag{4.15}$$

gdzie $t = \left[t_1, t_2, ..., t_n\right]$ oraz $x = \left[\, x_1^{k,t}, x_2^{k,t}, ..., x_n^{k,t} \,\right]$ określają odpowiednio wejściową oraz wyjściową krotkę relacji R^+, a kolejne ograniczenia oznaczają :

(1) ograniczenia nałożone na składowe krotki wejściowej, dotyczące dziedziny relacji R^+.
(2) ciąg wyznaczający kolejne wartości poszczególnych składowych krotki wyjściowej relacji R^+ dla zadanych wartości *k* oraz *t*.
(3) ograniczenia nałożone na składowe krotki wyjściowej, dotyczące zakresu relacji R^+

Przykład 4.4

Rozważmy przykład tworzenia relacji R^+ wraz z odpowiednimi ograniczeniami. Dla danej pętli :

```
for i = 1 to n do
  for j = 1 to n do
    a(i+1, 4*j - n) = a(i,2*j)
  endfor
endfor
```
(4.16)

Petit [70] znalazł następującą relację zależności (4.17) :

$$R = \{\ [i, j] \rightarrow [i+1, j'] : \mathrm{n} + 2\mathrm{j}' = 4\mathrm{j} \wedge 1 \le j' \wedge 3j' \le 4j \wedge 1 \le i \wedge i + 2j' < 4j\ \}. \tag{4.17}$$

Liczba zmiennych zawartych w opisie relacji zależności (4.17) nie jest mniejsza lub równa ilości zmiennych indeksujących pętli (4.16), w związku z tym rozwiązujemy wszystkie równania i nierówności będące elementami ograniczeń tej relacji, otrzymując relację [22] :

$$R = \{\, [i, 2j] \rightarrow [i+1, 4j-\mathrm{n}] : 1 \le i < n \wedge 4j-2 \wedge 4j \le 3n \,\}. \tag{4.18}$$

1. Dla relacji (4.18), tworzymy układ równań rekurencyjnych :

$$\left\{ \begin{array}{l} x_1^{k+1} = x_1^k + 1 \\ 2 \cdot x_2^{k+1} = 4 \cdot x_2^k - n \end{array} \right\}, \tag{4.19}$$

z wartościami początkowymi :

$$\left\{ \begin{array}{l} x_1^0 = t_1 \\ x_2^0 = t_2 \end{array} \right\}. \tag{4.20}$$

Jego rozwiązaniem jest :

$$\left\{ \begin{array}{l} x_1^{k,t_1} = t_1 + k \\ x_2^{k,t_2} = \dfrac{1}{2} \cdot (n - 2^k \cdot n + 2 \cdot 2^k \cdot t_2) \end{array} \right\}. \tag{4.21}$$

2. Ustalamy zbiór dziedziny relacji R :

$$\{\, [i, j] : \exists\, alpha : 0 = n + 2alpha \wedge 1 \le i < n \le 4j-2 \wedge 4j \le 3n \,\}. \tag{4.22}$$

3. Ustalamy zbiór zakresu relacji R :

$$\{\, [i, j] : \exists\, alpha :\ n = 2j + 4alpha \ \&\&\ 2 \le i \le n \wedge 1 \le j \le n \,\}. \tag{4.23}$$

4. Tworzymy relację R^+, zgodnie ze wzorem (4.15) :

$$R^+ = \left\{ \begin{array}{l} [t_1, t_2] \rightarrow [x_1^{k,t_1}, x_2^{k,t_2}] : \exists\ k :\ k \ge 1 \wedge \\ \qquad \underbrace{\exists\, \alpha : 0 = n + 2\alpha \wedge 1 \le t_1 < n \le 4 \cdot t_2 - 2 \wedge 4 \cdot t_2 \le 3n}_{1} \wedge \\ \qquad \underbrace{x_1^{k,t_1} = t_1 + k \wedge x_2^{k,t_2} = \dfrac{1}{2} \cdot \left(n - 2^k \cdot n + 2 \cdot 2^k \cdot t_2\right)}_{2} \wedge \\ \qquad \underbrace{\exists\, \alpha : n = 2 \cdot x_2^{k,t_2} + 4 \cdot \alpha \wedge 2 \le x_1^{k,t_1} \le n \wedge 1 \le x_2^{k,t_2} \le n}_{3} \end{array} \right\}. \tag{4.24}$$

Gdzie poszczególne ograniczenia oznaczają kolejno:

(1) ograniczenia nałożone na składowe krotki wejściowej, dotyczące dziedziny relacji R^+.

(2) ciąg wyznaczający kolejne wartości poszczególnych składowych krotki wyjściowej relacji R^+, dla zadanych wartości *k* oraz *t*.

(3) ograniczenia nałożone na składowe krotki wyjściowej, dotyczące zakresu relacji R^+.

4.1.5 Obliczanie tranzytywnego domknięcia relacji zależności o niepowiązanych elementach składowych krotek wejściowej i wyjściowej (ang. *dependence relation with non-coupled index variables*)

Relacje zależności o niepowiązanych elementach składowych krotek wejściowej i wyjściowej, to tzw. relacje hybrydowe, czyli takie których część odpowiadających sobie składowych krotki wejściowej i wyjściowej jest charakterystyczna dla relacji zależności o zmiennym wektorze dystansu klasy delta, a pozostałe opisują relacje zależności reprezentujące graf o topologii łańcucha o zmiennym wektorze dystansu 19].

Obliczanie tranzytywnego domknięcia tego typu relacji [20], sprowadza się do zastosowania odpowiednich algorytmów, przedstawionych w podrozdziałach 4.1.2 i 4.1.4, niezależnie względem każdej z par odpowiadających sobie składowych krotek wejściowej i wyjściowej.

Przykład 4.5

Dla przykładowej relacji zależności *R* (4.25), para składowych krotki wejściowej i

$$R = \{[i, j] \rightarrow [i', 2j] : 1 \le i < i' \le n \wedge j \ge 1 \wedge 2j \le m\} \tag{4.25}$$

i wyjściowej i', spełnia wymagania charakterystyczne dla relacji zależności o zmiennym wektorze dystansu klasy delta, z kolei para składowych krotki wejściowej - j i wyjściowej $2j$, odpowiada relacji zależności reprezentującej graf o topologii łańcucha o zmiennym wektorze dystansu. Stosując połączenie technik przedstawionych w podrozdziałach 4.1.2 i 4.1.4, niezależnie względem poszczególnych par składowych krotek wejściowej i wyjściowej, możliwe jest wyznaczenie tranzytywnego domknięcia powyższej relacji, które przyjmuje następującą postać [20] :

$$R^+ = \left\{ [i, j] \rightarrow [i', j'] \mid \exists k : k > 0 \wedge \underbrace{j \ge 1 \wedge 2j \le m \wedge 2 \le j' \le m \wedge j' = j \cdot 2^k \wedge \exists \alpha : 2 \cdot \alpha = j' \wedge}_{1} \underbrace{1 \le i < i' \le n \wedge i' - i \ge k}_{2} \right\}. \tag{4.26}$$

Gdzie poszczególne ograniczenia mają następujące znaczenie:

(1) ograniczenia nałożone na składowe j i j', charakterystyczne dla relacji zależności reprezentującej graf o topologii łańcucha o zmiennym wektorze dystansu.

(2) ograniczenia nałożone na składowe i i i', charakterystyczne dla relacji zależności o zmiennym wektorze dystansu klasy delta.

4.1.6 Obliczanie tranzytywnego domknięcia relacji zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej (ang. *dependence relation with coupled index variables*)

Obliczanie tranzytywnego domknięcia relacji zależności, oparte na utworzeniu i rozwiązaniu układu równań rekurencyjnych (przedstawione w podrozdziale 4.1.4), nie może być zastosowane wprost w przypadku relacji zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej. Relacje takie, bowiem bardzo rzadko opisują graf o topologii łańcucha tzn. ich ograniczenia dotyczące przyrostu odległości pomiędzy poszczególnymi parami wierzchołków należących do zbioru dziedziny i zakresu relacji (Δ), zawierają nie tylko równania, ale też i nierówności [19]. Dlatego w celu wyznaczenia tranzytywnego domknięcia tego typu relacji, na bazie algorytmu przedstawionego w podrozdziale 4.1.4, niezbędne jest dokonanie konwersji wejściowej relacji zależności, polegającej na sprowadzeniu wszystkich nierówności dotyczących bezpośrednio ograniczeń (Δ) do równań, poprzez wprowadzenie dodatkowych zmiennych symbolicznych, zgodnie z poniższym przykładem.

Przykład 4.6

Dla przykładowej relacji R (4.27), para składowych krotek wejściowej k i wyjściowej

$$R = \{[i,k] \rightarrow [i-k-1,k'] : i < n \wedge 0 \leq k' \wedge 2+k+k' \leq i \wedge 0 \leq k\} \tag{4.27}$$

k', wraz z ograniczeniami charakterystycznymi dla relacji zależności o zmiennym wektorze dystansu klasy delta, uniemożliwia obliczenie relacji R^+, zgodnie z algorytmem przedstawionym w podrozdziale 4.1.4 Po wprowadzeniu dodatkowej zmiennej symbolicznej l, dla powyższej pary składowych relacji R, otrzymujemy relację $\overline{R}$, dla której możliwe jest obliczenie jej tranzytywnego domknięcia z wykorzystaniem rozwiązania uzyskanego z układu równań rekurencyjnych.

$$\overline{R} = \{ [i,k] \to [i-k-1,\ k+l] : i < n \wedge 0 \le l+k \wedge 2+l+2k \le i \wedge 0 \le k \} \tag{4.28}$$

1. Dla relacji $\overline{R}$, tworzymy układ równań rekurencyjnych :

$$\left\{ \begin{array}{l} x_1^{\,k+1} = x_1^{\,k} - x_2^{\,k} - 1 \\ x_2^{\,k+1} = x_2^{\,k} + l \end{array} \right\}, \tag{4.29}$$

z wartościami początkowymi :

$$\left\{ \begin{array}{l} x_1^{\,0} = t_1 \\ x_2^{\,0} = t_2 \end{array} \right\}. \tag{4.30}$$

Jego rozwiązaniem jest :

$$\left\{ \begin{array}{l} x_1^{k,t_1} = \dfrac{1}{2}\left(-2\cdot k + k\cdot l - k^2\cdot l + 2\cdot t_1 - 2\cdot k\cdot t_2\right) \\ x_2^{k,t_2} = k\cdot l + t_2 \end{array} \right\}. \tag{4.31}$$

2. Ustalamy zbiór dziedziny relacji $\overline{R}$:

$$\{\ [i,k] : i < n \wedge 0 \le l+k \wedge 2+l+2\cdot k \le i \wedge 0 \le k\ \}. \tag{4.32}$$

3. Ustalamy zbiór zakresu relacji $\overline{R}$:

$$\{\ [i,k] : 0,\ l \le k < i \wedge 2+i+k \le l+n\ \}. \tag{4.33}$$

3. Tworzymy relację R^+, zgodnie ze wzorem (4.15) :

$$R^+ = \left\{ \begin{array}{l} [t_1,t_2] \to [x_1^{k,t_1}, x_2^{k,t_2}] : \exists\ k, l :\ k \ge 1 \wedge \\ \qquad \underbrace{i < n \wedge 0 \le l+k \wedge 2+l+2\cdot k \le i \wedge 0 \le k}_{1} \wedge \\ \underbrace{x_1^{k,t_1} = \dfrac{1}{2}\cdot\left(-2\cdot k + k\cdot l - k^2\cdot l + 2\cdot t_1 - 2\cdot k\cdot t_2\right) \wedge x_2^{k,t_2} = t_2 + k\cdot l \wedge}_{2} \\ \qquad \underbrace{0,\ l \le k < i \wedge 2+i+k \le l+n}_{3} \end{array} \right\}. \tag{4.34}$$

Gdzie poszczególne ograniczenia, mają następujące znaczenie:

(1) ograniczenia nałożone na składowe krotki wejściowej, dotyczące dziedziny relacji R^+.

(2) ciąg wyznaczający kolejne wartości poszczególnych składowych krotki wyjściowej relacji R^+, dla zadanych wartości *k* oraz *t*.

(3) ograniczenia nałożone na składowe krotki wyjściowej, dotyczące zakresu relacji R^+.

Pomimo, iż utworzenie i rozwiązanie układu równań rekurencyjnych (4.29), wraz z wartościami początkowymi (4.30), umożliwia obliczenie dokładnego tranzytywnego domknięcia, dla omawianej klasy relacji zależności, nie zawsze uzyskany w ten sposób wynik jest najlepszym z możliwych, z punktu widzenia zastosowań praktycznych. Nie trudno zauważyć, iż otrzymane rozwiązanie (4.34), zawiera wyrażenia nieliniowe, które de facto stają się ograniczeniami poszukiwanej relacji R^+. Jakiekolwiek późniejsze wykorzystanie obliczonej relacji R^+, do ekstrakcji czy to drobnoziarnistej, czy też gruboziarnistej równoległości [6, 10, 11, 12, 13, 14, 15, 23, 24, 26, 27, 28, 43, 75, 76, 77, 78], staje pod dużym znakiem zapytania, ponieważ nie są znane obecnie na rynku narzędzia, umożliwiające generowanie kodu ze zbioru zawierającego ograniczenia nieliniowe. Dlatego też, zdaniem autora obliczanie tranzytywnego domknięcia relacji zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej, w odróżnieniu od relacji zależności o zmiennym wektorze dystansu tworzącej graf o topologii łańcucha, powinno być poprzedzone próbą obliczenia go w sposób iteracyjny, zgodnie z algorytmem z rysunku 4.2 przedstawionym w podrozdziale 4.1.

Przykład 4.7

Dla tej samej relacji zależności *R* (4.35), dla której wcześniej obliczyliśmy jej tranzytywne domknięcie (4.34), korzystając z rozwiązania układu równań rekurencyjnych (4.31), algorytm iteracyjny z rysunku 4.2, zaproponowany w

$$R = \{ [i,k] \to [i-k-1,k'] : i < n \wedge 0 \le k' \wedge 2+k+k' \le i \wedge 0 \le k \} \tag{4.35}$$

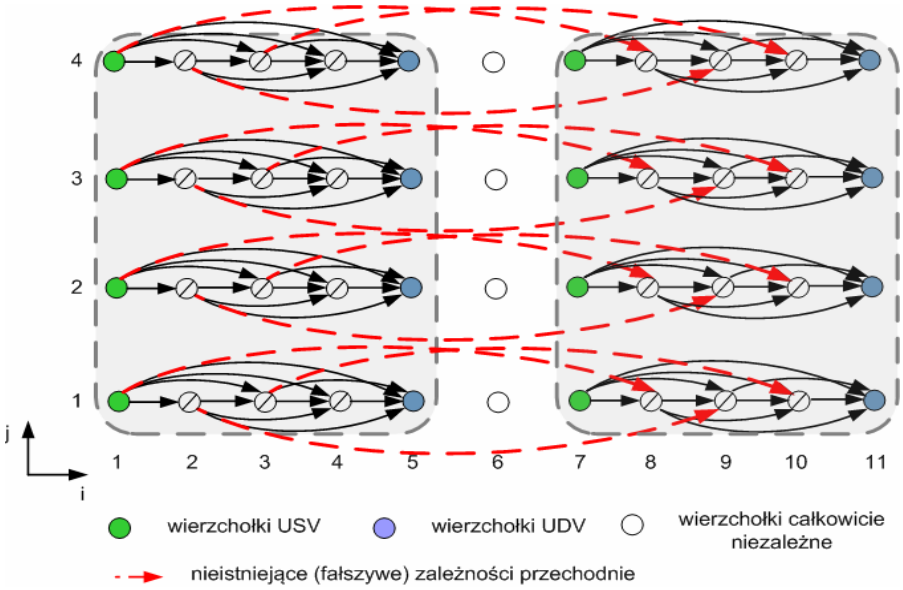
podrozdziale 4.1, pozwala uzyskać rozwiązanie tożsame, składające się co prawda z dwóch koniunkcji, ale nie zawierają one ograniczeń nieliniowych.

$$\begin{aligned} R^+ = \{ [t_1,t_2] \to [x_1,x_2] \mid 0 \le x_2 < x_1 \wedge t_1 < n \wedge 0 \le t_2 \wedge 2+t_2+x_1 \le t_1 \} \cup \\ \{ [t_1,t_2] \to [t_1-t_2-1,x_2] \mid t_1 < n \wedge 0 \le x_2 \wedge 2+t_2+x_2 \le t_1 \wedge 0 \le t_2 \}. \end{aligned} \tag{4.36}$$

4.1.7 Obliczanie tranzytywnego domknięcia relacji o przerwanym łańcuchu zależności

Tranzytywne domknięcie relacji zależności obliczone zostało w sposób dokładny, wtedy i tylko wtedy, gdy generuje zależności przechodnie, wyłącznie dla tworzących je zależności bezpośrednich. Oznacza to, że relacja R^+ obliczona w wyniku zastosowania dowolnego z algorytmów przedstawionych w podrozdziałach 4.1.2-6, nie może zawierać nieistniejących (fałszywych) zależności przechodnich (rysunek 4.3 oznaczone kolorem czerwonym), które nie wynikają z występowania zależności bezpośrednich, reprezentowanych przez relację *R*.

Niebezpieczeństwo występowania fałszywych zależności przechodnich, w przypadku relacji zależności składających się z pojedynczej koniunkcji, dotyczy bezpośrednio relacji o przerwanym łańcuchu zależności, czyli takich w których w ramach ograniczeń wyznaczających dziedzinę relacji, mogą pojawić się wierzchołki, które zostały z niej wykluczone (wierzchołek jest końcem zależności ale nie jest początkiem żadnej innej zależności).



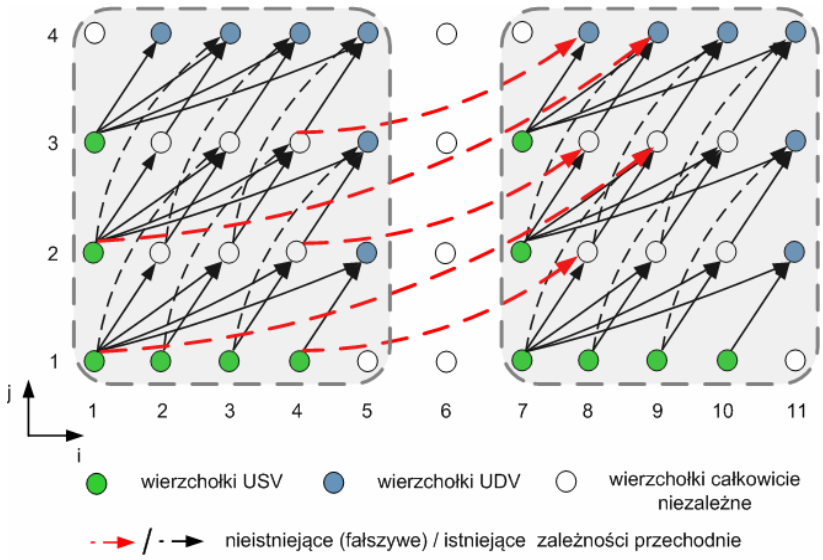
Rys 4.3 Nieistniejące zależności przechodnie dla tranzytywnego domknięcia relacji o przerwanym łańcuchu zależności [op. własne]

Nieistniejące (fałszywe) zależności przechodnie najszybciej usuwa się poprzez wprowadzenie ograniczenia (4.37) postaci:

$$\neg \exists\, k' : 1 \le k' < k \ \wedge x^{k',t} \in UDV\,, \tag{4.37}$$

do relacji R^+, obliczonej zgodnie z algorytmami przedstawionymi w podrozdziałach 4.1.3 i 4.1.4, gdzie zbiór *UDV* (ang. *Ultimate Destination Vertex*) określa wierzchołki, które nie są początkami żadnej zależności (rysunek 4.3, oznaczone kolorem niebieskim). Przeciwieństwem takich wierzchołków jest zbiór *USV* (ang. *Ultimate Source Vertex*), którego wierzchołki z kolei nie są końcami żadnej zależności (rysunek 4.3, oznaczone kolorem zielonym). Symbol $(\neg)$ oznacza negację, $(\exists)$ jest kwantyfikatorem egzystencjalnym. Ograniczenie (4.37), nie dopuszcza fałszywych zależności przechodnich pomiędzy przerwami w łańcuchach, powstałych w wyniku występowania nieciągłości łańcuchów zależności, czyli między wierzchołkami, które są końcami zależności, ale nie są początkami żadnej innej zależności lub są wierzchołkami całkowicie niezależnymi.

Pomimo wyraźnej prostoty, zakres stosowalności ograniczenia (4.37), sprowadza się tylko i wyłącznie do relacji zależności reprezentujących graf o topologii łańcucha. Tylko w przypadku takich relacji, fakt nie istnienia takiego $k' < k$, dla którego osiągnięto już wierzchołek $x^{k',t} \in UDV$ (rysunek 4.3 wierzchołki *UDV* oznaczono kolorem niebieskim), jest wystarczający do usunięcia wszystkich fałszywych zależności, występujących pomiędzy przerwami w łańcuchach zależności.



Rys 4.4 Nieistniejące zależności przechodnie dla tranzytywnego domknięcia relacji klasy delta o przerwanym łańcuchu zależności [op. własne]

Dla pozostałych klas relacji zależności, (podrozdziały 3.2, 3.5 i 3.6) dodanie warunku (4.37) do relacji R^+, obliczonej zgodnie z algorytmami przedstawionymi w podrozdziałach 4.1.2, 4.1.5 i 4.1.6, nie pozwala w pełni na usunięcie wszystkich fałszywych zależności. Dla przykładowej relacji zależności R (4.35) o zmiennym

$$R = \{[i, j] \rightarrow [i', \mathrm{j}+1] \mid \exists \alpha : 1, 6\alpha + 1 \leq i < i' \leq 6\alpha + 5, n \wedge 1 \leq j < m\} \tag{4.38}$$

wektorze dystansu klasy delta, dla której łańcuch zależności został przerwany, wprowadzenie ograniczenia (4.37) do relacji R^+, obliczonej zgodnie ze wzorem 4.10, przedstawionym w podrozdziale 4.1.2, nie wyeliminowało wszystkich fałszywych zależności. (rysunek 4.4, oznaczone kolorem czerwonym). Dlatego też, konieczne stało się zastosowanie bardziej zaawansowanego podejścia.

Zadaniem algorytmu przedstawionego poniżej, jest identyfikacja i usunięcie wszystkich nieistniejących zależności, zawierających na ścieżce (definicję ścieżki w grafie przedstawiono w podrozdziale 2.1.4) zależności występujące w miejscach przerw łańcuchów zależności.

Algorytm 4.1 Usuwanie nieistniejących tranzytywnych zależności z relacji prostej o przerwanym łańcuchu zależności

Wejście: relacja R^+, R

Wyjście: relacja R^+ pozbawiona fałszywych zależności

1. Utwórz relację $\overline{R}$ jako kopię relacji *R*, pozbawioną ograniczeń dotyczących jej dziedziny i zakresu.

   /* *Relację* $\overline{R}$, *tworzymy na nieograniczonej przestrzeni, na bazie wektora dystansu* $d = \{[d_1, ..., d_k] \mid \exists s_j, t_j \; s.t. \; 1 \leq j \leq k \wedge d_j = t_j - s_j \wedge \exists \alpha_1, ..., \alpha_m \; s.t. \; F\}$, *gdzie d, obliczamy jako różnicę wartości poszczególnych składowych krotki wyjściowej* t_j *i wejściowej* s_j, *a F to ograniczenia dotyczące dziedziny i zakresu relacji R* */

2. Oblicz wypukły zbiór *C*, zawierający elementy zbiorów dziedziny i zakresu relacji *R*.

$$C = Hull\left(domain\left(R\right) \cup range\left(R\right)\right).$$

/ Rozważamy wszystkie elementy zawarte wewnątrz wypukłego regionu C, czyli również i te, które zostały wykluczone ze zbioru* $domain(R) \cup range(R)$ *za pomocą kwantyfikatora* $\exists$ **/*

3. Wprowadź ograniczenia na dziedzinę relacji $\overline{R}$, jako elementy zbioru *C*.

$$\overline{R} \setminus C .$$

4. Wprowadź ograniczenia na zakres relacji $\overline{R}$, jako elementy zbioru *C*.

$$\overline{R} / C .$$

5. Oblicz relację $\overline{R}^*$, stosując odpowiedni algorytm dla danej klasy relacji zależności, przedstawiony w podrozdziale 4.1.

6. Wyznacz relację R', zawierającą zależności występujące w miejscach przerw łańcuchów zależności.

$$R' = \overline{R} - R .$$

/ Relacja* R'*, stanowi uzupełnienie relacji R o dodatkowe zależności występujące w miejscach przerw łańcuchów zależności. Przykład takiej relacji, przedstawiono na rysunku 4.5 */*

7. Usuń wszystkie fałszywe zależności, zawierające na ścieżce, zależności występujące w miejscach przerw łańcuchów zależności.

$$R^+ = R^+ - \left(\overline{R}^* \circ R' \circ \overline{R}^*\right).$$

/ Usuń wszystkie zależności przechodnie, które mogły powstać w wyniku złożenia relacji* $\overline{R}^*$*, z relacją* R'*, będącą uzupełnieniem relacji R */*

Ponieważ zarówno wprowadzenie ograniczenia (4.37) do obliczonej uprzednio relacji R^+, jak i wykonanie algorytmu 4.1 w celu usunięcia z niej fałszywych zależności, wiąże się z dodatkowym nakładem obliczeniowym, uzasadnione okazało się opracowanie algorytmu, umożliwiającego rozpoznanie relacji o przerwanym łańcuchu zależności i zastosowanie powyższych procedur, tylko wtedy gdy jest to konieczne. Propozycję takiego algorytmu przedstawiono poniżej.

Algorytm 4.2 Rozpoznanie relacji o przerwanym łańcuchu zależności

Wejście: relacja *R*

Wyjście: *prawda* – jeżeli łańcuch zależności relacji *R* jest przerwany

f*ałsz* – jeżeli łańcuch zależności relacji *R* nie jest przerwany

1. Utwórz relację $\overline{R}$, jako kopię relacji *R*, pozbawioną ograniczeń, dotyczących jej dziedziny i zakresu.

   /* *Relację* $\overline{R}$, *tworzymy na nieograniczonej przestrzeni, na bazie wektora dystansu* $d = \{[d_1,...,d_k] \mid \exists s_j, t_j \; s.t. \; 1 \le j \le k \wedge d_j = t_j - s_j \wedge \exists \alpha_1,...,\alpha_m \; s.t. \; F\}$, *gdzie d obliczamy jako różnicę wartości poszczególnych składowych krotki wyjściowej* t_j *i wejściowej* s_j, *a F to ograniczenia dotyczące dziedziny i zakresu relacji R* */

2. Oblicz wypukły zbiór *C*, zawierający elementy zbiorów dziedziny i zakresu relacji *R*.

$$C = Hull\,(domain\,(R) \cup range\,(R)).$$

   /* *Rozważamy wszystkie elementy zawarte wewnątrz wypukłego regionu C, czyli również i te, które zostały wykluczone ze zbioru* $domain(R) \cup range(R)$, *za pomocą kwantyfikatora* $\exists$ */

3. Wprowadź ograniczenia na dziedzinę relacji $\overline{R}$, jako elementy zbioru *C*.

$$\overline{R} \setminus C.$$

4. Wprowadź ograniczenia na zakres relacji $\overline{R}$, jako elementy zbioru *C*.

$$\overline{R} \,/\, C.$$

5. **Jeżeli** $\overline{R} \neq R$ *prawda*

   /* *Łańcuch zależności relacji R jest przerwany* */

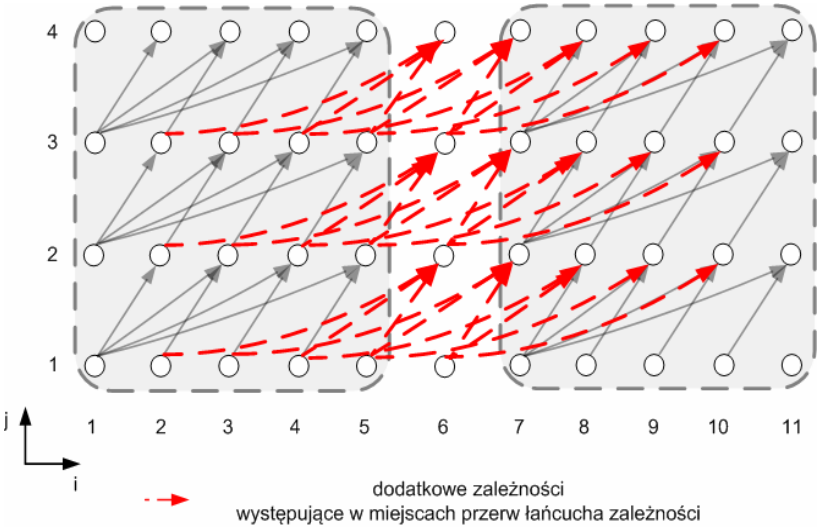
   Jeżeli $\overline{R} = R$ *fałsz*

   /* *Łańcuch zależności relacji R nie jest przerwany* */

Przykład 4.8

Rozważmy przykład relacji zależności *R* (4.38), przedstawionej na rysunku 4.5.

Dokonujemy sprawdzenia, czy łańcuch zależności relacji R (4.38) jest przerwany. W tym celu, postępujemy zgodnie z algorytmem 4.2.



Rys 4.5 Przykład relacji o przerwanym łańcuchu zależności [op. własne]

Rozważmy przebieg poszczególnych kroków algorytmu 4.2, dla przykładowej relacji R.

Algorytm 4.2

Wejście: $R=\{[i,j]\rightarrow[i',j+1] \mid \exists\alpha:1,6\cdot\alpha+1\le i<i'\le 6\cdot\alpha+5, n\wedge 1\le j<m\}$

1. $\overline{R}=\{[i,j]\rightarrow[i',j'] \mid 1\le i'-i\le 4\ \wedge\ j'-j=1\}$
2. $C=\{[i,j] \mid 1\le i\le n\wedge 1\le j\le m\wedge 2+i\le n+j\wedge 2\le n\wedge 2\le m\wedge 2+j\le m\}$
3. $\overline{R}=\{[i,j]\rightarrow[i',j+1] \mid 1\le i\le i'-1, n\wedge 1\le j\le m\wedge 2+i\le n+j\wedge 2\le n\wedge 2\le m\wedge 2+j\le m+i\}$
4. $\overline{R}=\{[i,j]\rightarrow[i',j+1] \mid 1\le i<i'\le n\wedge 1\le j<m\}$
5. $\overline{R}\neq R$, *prawda* - łańcuch zależności relacji R, jest przerwany (rysunek 4.5)

Relacja R^+ (4.39), obliczona zgodnie z podejściem przedstawionym w podrozdziale 4.1.2, zawiera nieistniejące krawędzie. Niektóre z nich, zaznaczono kolorem czerwonym na rysunku 4.5. Aby je usunąć, postępujemy zgodnie z algorytmem 4.1.

$$R^+=\{[i,j]\rightarrow[i',j'] \mid \exists\alpha,\beta:1,6\alpha+1\le i\le 6\alpha+4\wedge 1\le j<j'\le m\wedge 6\beta+2\le i'\le n, 6\beta+5\wedge i+j'\le j+i'\wedge 4j+i'\le i+4j'\}. \quad (4.39)$$

Rozważmy przebieg poszczególnych kroków algorytmu 4.1 dla przykładowej relacji *R*.

Algorytm 4.1

Wejście: $R = \{[i, j] \to [i', j+1] \mid \exists \alpha : 1, 6 \cdot \alpha + 1 \le i < i' \le 6 \cdot \alpha + 5, n \wedge 1 \le j < m\}$

$$R^+ = \{[i, j] \to [i', j'] \mid \exists\, \alpha, \beta : 1, 6\alpha + 1 \le i \le 6\alpha + 4 \wedge 1 \le j < j' \le m \wedge 6\beta + 2 \le i' \le n, 6\beta + 5 \wedge \mathrm{i} + \mathrm{j}' \le \mathrm{j} + \mathrm{i}' \wedge 4\mathrm{j} + \mathrm{i}' \le \mathrm{i} + 4\mathrm{j}'\}$$

1. $\overline{R} = \{[i, j] \to [i', j'] \mid 1 \le i' - i \le 4 \ \wedge \ j' - j = 1\}$
2. $C = \{[i, j] \mid 1 \le i \le n \wedge 1 \le j \le m \wedge 2 + i \le n + j \wedge 2 \le n \wedge 2 \le m \wedge 2 + j \le m\}$
3. $\overline{R} = \{[i, j] \to [i', j+1] \mid 1 \le i \le i' - 1, n \wedge 1 \le j \le m \wedge 2 + i \le n + j \wedge 2 \le n \wedge 2 \le m \wedge 2 + j \le m + i\}$
4. $\overline{R} = \{[i, j] \to [i', j+1] \mid 1 \le i < i' \le n \wedge 1 \le j < m\}$
5. $\overline{R}^* = \{[i, j] \to [i', j'] \mid 1 \le j < j' \le m \wedge i' \le n \wedge i + j' \le j + i' \wedge 1 \le i\}$
6. $R' = \overline{R} - R = \{[i, j] \to [i', j+1] \mid \exists\, \alpha : 1 \le i < i' \le n \wedge 1 \le j \le m \wedge i \le 6\alpha + 6 \wedge 6 + 6\alpha \le i'\}$
7. $R^+ = \{[i, j] \to [i', j+2] \mid \exists \alpha : 6\alpha + 7, 1 \le i \le i' - 2 \wedge 1 \le j \le m - 2 \wedge i' \le n, 6\alpha + 11\} \cup$
   $\{[i, j] \to [i', j'] \mid \exists \alpha : 1 \le j \le j' - 3 \wedge i' \le 6\alpha + 11, n \wedge j' \le m \wedge 7 + 6\alpha \le i \wedge i + j' \le j + i' \wedge 1 \le i\} \cup$
   $\{[i, j] \to [i', j+1] \mid \exists \alpha : 6\alpha + 1, 1 \le i < i' \le 6\alpha + 5, n \wedge 1 \le j < m\}$

4.1.8 Podsumowanie

W niniejszym podrozdziale zaprezentowano powszechnie znane (podrozdziały 4.1.1-3) i autorskie (podrozdziały 4.1.4-6) techniki umożliwiające obliczenie dokładnego tranzytywnego domknięcia relacji zależności, składających się z pojedynczej koniunkcji. Pierwotnie rozwiązanie tego problemu, sprowadzało się do zastosowania algorytmu iteracyjnego (rysunek 4.2), obliczającego tranzytywne domknięcie relacji zgodnie ze wzorem 4.1. Podejście to, okazało się być niewystarczające, w szczególności dla sparametryzowanych relacji zależności reprezentujących graf o topologii łańcucha. Dlatego też, jednym z pomysłów [72] było dokonanie podziału relacji, na relacje o stałym wektorze dystansu (ang. *uniform dependence relations*) i relacje o zmiennym wektorze dystansu (ang. *non-uniform dependence relation*), dla których tranzytywne domknięcie obliczane jest na podstawie wzorów 4.10 i 4.11. Jednak zastosowanie wzoru 4.10, dla wszystkich relacji o zmiennym wektorze dystansu w celu obliczenia ich tranzytywnego domknięcia, nie

zawsze kończyło się sukcesem i uświadomiło konieczność podziału relacji zależności o zmiennym wektorze dystansu na poszczególny klasy, wraz z określeniem metod obliczania ich tranzytywnego domknięcia w sposób formalny.

Podrozdziały 4.1.4-6, zawierają sformalizowane autorskie podejścia, umożliwiające obliczenie dokładnego tranzytywnego domknięcia, dla każdej z owych klas sparametryzowanych relacji zależności, przedstawionych w rozdziale 3, po ich uprzednim rozpoznaniu w sposób formalny. Dodatkowo w odróżnieniu od istniejących rozwiązań, rozważono również możliwość zastosowania autorskich algorytmów, w celu rozpoznania i obliczenia dokładnego tranzytywnego domknięcia relacji o przerwanym łańcuchu zależności.

4.2 Obliczanie tranzytywnego domknięcia relacji zależności składających się z wielu koniunkcji (ang. *multiple conjunct relations*)

W przypadku złożonych relacji zależności, (składających się z unii skończonej liczby relacji prostych) procedura wyznaczania R^+ jest bardziej skomplikowana. Dotychczas nie potwierdzono, ani też nie wykluczono możliwości obliczenia dokładnego tranzytywnego domknięcia, unii dla żadnej z klas relacji zależności. Boigelot [30, 31] i Kelly [72] dla relacji złożonych, proponują rozwiązanie w postaci aproksymacji (przybliżenie górne lub dolne), gdzie złożone relacje zależności aproksymowane są do relacji zależności składających się z pojedynczej koniunkcji. Następnie, tranzytywne domknięcie obliczane jest zgodnie z algorytmami charakterystycznymi dla relacji prostych (podrozdział 4.1).

W dalszej części tego rozdziału, przedstawione zostały trzy sposoby obliczania tranzytywnego domknięcia relacji złożonych : podejście iteracyjne, podejście nieiteracyjne i podejście hybrydowe, będący połączeniem obu technik : iteracyjnej i nieiteracyjnej.

4.2.1 Podejście iteracyjne

W podrozdziale 4.1, przedstawiono algorytm iteracyjnego obliczania relacji R^+, zgodnie z poniższym wzorem:

$$R^+ = R \cup \left(R \circ R^+\right). \tag{4.40}$$

Zwrócono również uwagę, na słabe strony tego typu rozwiązań, dotyczące bezpośrednio relacji zależności reprezentujących graf o topologii łańcucha, w przypadku których nie ma możliwości osiągnięcia zbieżności, czyli rozpoznania wszystkich zależności przechodnich. Dlatego też, rodzinę algorytmów iteracyjnego obliczania relacji R^+, poddawano licznym modyfikacjom [3, 5, 48, 51, 53, 55, 56, 59, 62, 63, 72, 79, 80, 106], mającym na calu zarówno zmniejszenie liczby wykonanych iteracji, niezbędnych do osiągnięcia zbieżności, jak i również zachowanie dokładności otrzymanego rozwiązania.

W odróżnieniu od zaproponowanych dotychczas usprawnień algorytmów iteracyjnych, udoskonalenie przedstawione w tym podrozdziale, polega na rozpoznaniu klasy każdej z relacji zależności wchodzącej w skład unii relacji wejściowych [19], zgodnie z algorytmami przedstawionymi w rozdziale 3, na początku i w każdym kolejnym przebiegu algorytmu. Po rozpoznaniu klas relacji zależności, następuje obliczenie ich tranzytywnego domknięcia zgodnie z algorytmami przedstawionymi w podrozdziale 4.1. Przygotowane w ten sposób relacje, stanowią parametry wejściowe algorytmu 4.3, którego dalsza część to realizacja wzoru (4.40) w ujęciu klasycznym.

Algorytm 4.3. Obliczenie tranzytywnego domknięcia złożonej relacji zależności

Wejście : $R = \bigcup_{i=1}^{m} R_i$, gdzie *m* to liczba prostych relacji

N – maksymalna liczba możliwych do wykonania iteracji algorytmu

mode [*exact*, *upper bound*] – akceptowany wynik końcowy.

/ Dla wartości mode = exact, algorytm dąży do uzyskania dokładnego rozwiązania tranzytywnego domknięcia relacji R, jeśli nie jest to możliwe, to zwraca dolną granice tranzytywnego domknięcia relacji R. Wartość mode = upper bound, oznacza, iż akceptujemy rozwiązanie w postaci przybliżenia górnego tranzytywnego domknięcia relacji R. */*

Wyjście : R^* - dokładne tranzytywne domknięcie relacji R, jej przybliżenie górne R^*_{UB}, takie że $R^* \subset R^*_{UB}$, lub przybliżenie dolne R^*_{LB}, takie że $R^*_{LB} \subset R^*$.

Metoda :

if *mode* **is** *exact* **then**

$R_{out} \leftarrow R' \leftarrow \Delta \leftarrow \text{Algorytm4.4}(R_{in})$ */* Zainicjuj zmienne R_{out}, R' i Δ, wywołując Algorytm 4.4 , z parametrem w postaci zbioru relacji wejściowych R_{in} */*

$i \leftarrow 1$

loop

$R_{new} \leftarrow R' \circ \Delta$ */* Zmienna R_{out} jest wynikiem złożenia relacji R' i Δ */*

$\Delta \leftarrow R_{new} - R_{old}$ */* Zmienna Δ przechowuje różnicę pomiędzy zbiorem relacji R_{new} i R_{old} */*

$i \leftarrow i + 1$

if $\Delta = \varnothing$ OR $i > N$ **then**

break */* Przerwanie działania algorytmu w przypadku zerowego przyrostu zależności przechodnich lub w przypadku osiągnięcia maksymalnej liczby możliwych do wykonania iteracji */*

end if

$\Delta \leftarrow \text{Algorytm4.4}(\Delta)$

$R_{old} \leftarrow R_{old} \cup \Delta$

end loop

if $i \leq N$ **then**

print *„Obliczono dokładną postać relacji R^*”*

$R^* \leftarrow R_{old}$

else

print *„Obliczono przybliżenie dolne relacji R^*”*

$R^* \leftarrow R_{new}$

end if

return R^*

else

$d^* \leftarrow \text{Algorytm}4.5(R_i)$ /* *Aproksymacja relacji* R_{in} *do relacji zależności o zmiennym wektorze dystansu klasy delta i obliczenie unii tranzytywnych domknięć tych relacji.* */

$R^* \leftarrow \prod_{i=1}^{m} d_i^*$ /* *Dokonanie złożenia relacji* d_i^*, *pozbawionych ograniczeń dotyczących dziedziny i zakresu, co skutkuje obliczeniem przybliżenia górnego relacji* R^* */

$R^* \leftarrow R^* \setminus \text{domain}(R_{in})$ /* *Wprowadzenie ograniczeń dotyczących dziedziny relacji* R^* */

$R^* \leftarrow R^* / \text{range}(R_{in})$ /* *Wprowadzenie ograniczeń dotyczących zakresu relacji* R^* */

print „*Obliczono przybliżenie górne relacji* R^*”

return R^*

end if

Koniec

Algorytm 4.4, dokonuje rozpoznania klas relacji zależności, zgodnie z algorytmami przedstawionymi w rozdziale 3 i oblicza unię tranzytywnych domknięć tych relacji.

Algorytm 4.4. Rozpoznanie klas relacji zależności i obliczenie unii tranzytywnych domknięć tych relacji

Wejście : $R_{in} = \bigcup_{i=1}^{m} R_i$, gdzie *m* to liczba prostych relacji

Wyjście : $R_{out} = \bigcup_{i=1}^{m} R_i^*$

Metoda :

$R_{out} \leftarrow \varnothing$

for all $R_i \in R_{in}$ **do**

1. Rozpoznaj klasę relacji zależności stosując algorytmy przedstawione w rozdziale 3 i oblicz relację R_i^* zgodnie z algorytmami przedstawionymi w podrozdziale 4.1.
2. $R_{out} \leftarrow R_{out} \cup R_i^*$

end for

return R_{out}

Koniec

Pomimo wprowadzonych udoskonaleń do algorytmu iteracyjnego obliczania relacji R^+, nie zawsze gwarantuje on osiągnięcie warunku stopu (zbieżności). W takich sytuacjach, autor proponuje rozwiązanie w postaci przybliżenia górnego R_{UB}^+ relacji tranzytywnego domknięcia R^+, takiego że $R^+ \subset R_{UB}^+$. W tym celu, konieczne staje się dokonanie konwersji relacji wejściowych, do relacji zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form relations*) i obliczenie unii tranzytywnych domknięć tych relacji. Zadanie to realizuje algorytm 4.5.

Algorytm 4.5. Aproksymacja relacji wejściowych do relacji zależności o zmiennym wektorze dystansu klasy delta i obliczenie unii tranzytywnych domknięć tych relacji.

Wejście : $R_{in} = \bigcup_{i=1}^{m} R_i$, gdzie *m* to liczba prostych relacji

Wyjście : $R_{out} = \bigcup_{i=1}^{m} d_i^*$, gdzie d_i to aproksymacja relacji R_i do relacji zależności o zmiennym wektorze dystansu klasy delta, pozbawiona ograniczeń dotyczących dziedziny i zakresu.

Metoda :

$R_{out} \leftarrow \mathrm{I}$ */* Zainicjuj zmienną R_{out} relacją tożsamości */*

for all $R_i \in R_{in}$ **do**

1. Aproksymuj relację R_i, do relacji d_i o zmiennym wektorze dystansu klasy delta postaci :

$$d_i \leftarrow \left\{ [s_1,...,s_m] \rightarrow [t_1,...,t_m] \mid \forall p, 1 \le p \le m, \exists \alpha_p \; s.t. \; L_p \le t_p - s_p \le U_p \wedge t_p - s_p = M_p \alpha_p \right\},$$

gdzie d_i pozbawione jest ograniczeń dotyczących dziedziny i zakresu relacji.

2. Oblicz relację d_i^+, zgodnie z algorytmem przedstawionym w podrozdziale 4.1.2
3. $R_{out} \leftarrow R_{out} \cup d_i^+$

end for

return R_{out}

Koniec

Uzasadnienie poprawności algorytmu 4.5

Powszechnie wiadomo, że dla relacji o stałym wektorze dystansu (ang. *uniform relations*) operator złożenia $(\circ)$, jest przemienny. Jeżeli relacje zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form relations*), mogą być przedstawione jako relacje zależności o stałym wektorze dystansu, na nieograniczonej przestrzeni (pozbawione ograniczeń na dziedzinę i zakres relacji), zatem operator złożenia $(\circ)$ jest również przemienny dla tego typu relacji.

Aby wykazać, że $\left(\prod_{i=1}^{m} d_i^* \setminus \text{domain}(R_{in})\right) / \text{range}(R_{in})$, uzyskane w wyniku wykonania algorytmu 4.5, reprezentuje przybliżenie górne tranzytywnego domknięcia skończonej unii prostych relacji zależności, rozważmy następującą własność relacji d_i. Ponieważ, $d = \bigcup_{i=1}^{m} d_i$, gdzie m to liczba prostych relacji zależności tworzących unię relacji zależności, takich, że $R_i \in d_i$, i d stanowi przybliżenie górne relacji $R_{in} = \bigcup_{i=1}^{m} R_i$, w związku z tym prawdą jest, iż $\left(d^* \setminus \text{domain}(R_{in})\right) / \text{range}(R_{in})$ reprezentuje przybliżenie górne relacji R_{in}^*, takie że $R_{in}^* \subseteq d^*$. Zważywszy na fakt, iż w przypadku relacji zależności, dla których operator złożenia $(\circ)$ jest przemienny prawdziwe są następujące własności:

i) operator $(\cup)$ jest łączny i przemienny

$(R_1 \cup R_2 = R_2 \cup R_1,\ R_1 \cup (R_2 \cup R_3) = (R_1 \cup R_2) \cup R_3)$

ii) operator $(\circ)$ jest łączny

$(R_1 \circ (R_2 \circ R_3) = (R_1 \circ R_2) \circ R_3)$

iii) operator $(\circ)$ jest rozdzielny względem $(\cup)$

$(R_1 \circ (R_2 \cup R_3) = R_1 \circ R_2 \cup R_1 \circ R_3)$

iv) $R^n \cup R^n = R^n$, gdzie $n \in Z$,

zatem

$$d^* = \bigcup_{i=0}^{\infty} R^i = \mathrm{I} \cup (d_1 \cup d_2 \cup ... \cup d_m) \cup (d_1 \cup d_2 \cup ... \cup d_m)^2 \cup (d_1 \cup d_2 \cup ... \cup d_m)^3 \cup$$

$$\cup ... = \prod_{i=0}^{m} d_i^* .$$

Przykład 4.9

Aby zademonstrować działanie algorytmu 4.3, rozważmy następujący przykład złożonej relacji zależności.

$$\begin{aligned} R = {} & \{[i,5] \to [i, i+7] \mid 1 \le i \le n-7\} \cup \\ & \{[i,5] \to [i', i+7] \mid 1 \le i < i' \le n \wedge i \le n-7\} \cup \\ & \{[i,j] \to [j-7,5] \mid 1 \le i \le j-8 \wedge j \le n\} \cup \\ & \{[i,j] \to [i', j] \mid 1 \le i < i' \le n \wedge 1 \le j \le n\}. \end{aligned} \tag{4.41}$$

Tabela 4.1, zawiera kolejne zależności przechodnie (Δ), uzyskane w wyniku wykonania poszczególnych *i* - przebiegów algorytmu iteracyjnego.

Tabela 4.1 Przebieg algorytmu iteracyjnego

i	$\Delta = R_{new} - R_{old}$
1	$\{[i,5] \to [i', i'+7] \mid 1 \le i < i' \le n-7\} \cup$ $\{[i,5] \to [i', j'] \mid i+8 \le j' \le i'+6, n \wedge i' \le n\} \cup$ $\{[i,j] \to [i', 5] \mid i+8 \le j \le i'+6, n \wedge i' \le n \wedge 1 \le i\}$
2	$\{[i,j] \to [i', i'+7] \mid 1 \le i \le j-8 \wedge j-6 \le i' \le n-7\} \cup$ $\{[i,j] \to [i', j'] \mid i+8 \le j < j' \le i'+6, n \wedge i' \le n \wedge 1 \le i\}$
3	$\{[i,j] \to [i', j'] \mid FALSE\}$

Algorytm 4.3, osiągnął warunek zbieżności po wykonaniu trzech przebiegów z wynikiem końcowym postaci :

$$
\begin{aligned}
R^{*} = \{&[i,5] \to [i', i'+7] \mid 1 \le i < i' \le n-7\} \cup \\
&\{[i,5] \to [i', j'] \mid i+8 \le j' \le i'+6, n \wedge i' \le n \wedge 1 \le i\} \cup \\
&\{[i, j] \to [i', 5] \mid i+8 \le j' \le j \le i'+6, n \wedge i' \le n \wedge 1 \le i\} \cup \\
&\{[i,5] \to [i, i+7] \mid 1 \le i \le n-7\} \cup \\
&\{[i,5] \to [i', i+7] \mid 1 \le i < i' \le n \wedge i \le n-7\} \cup \\
&\{[i, j] \to [j-7, 5] \mid 1 \le i \le j-8 \wedge j \le n\} \cup \\
&\{[i, j] \to [i', j] \mid 1 \le i < i' \le n \wedge 1 \le j \le n\} \cup \\
&\{[i, j] \to [i', i'+7] \mid 1 \le i \le j-8 \wedge j-6 \le i' \le n-7\} \cup \\
&\{[i, j] \to [i', j'] \mid i+8 \le j < j' \le i'+6, n \wedge i' \le n \wedge 1 \le i\} \cup \\
&\{[i, j] \to [i, j]\}.
\end{aligned}
$$

4.2.2 Podejście nieiteracyjne

Obliczanie tranzytywnego domknięcia złożonej sparametryzowanej relacji zależności w sposób iteracyjny, zgodnie z algorytmem 4.3, gwarantuje uzyskanie dokładnego wyniku w przypadku relacji, których składowe wektora dystansu przyjmują tylko wartości dodatnie bądź tylko wartości ujemne. Dla przykładowej złożonej relacji :

$$
\begin{aligned}
R = \{&[i, j] \to [i+1, j] \mid 1 \le i < n \wedge 1 \le j \le m\} \cup \\
&\{[i, j] \to [i+1, j+1] \mid 3 \le i \le n-3 \wedge 2 \le j \le m-2\},
\end{aligned}
\tag{4.42}
$$

gdzie $\Delta = \{[1,0],[1,1]\}$, uzyskamy rozwiązanie dokładne, podczas gdy dla relacji R :

$$
\begin{aligned}
R = \{&[i, j] \to [i+1, j+1] \mid 1 \le i < n \wedge 1 \le j < m\} \cup \\
&\{[i, j] \to [i+1, j-1] \mid 1 \le i < n \wedge 2 \le j \le m\},
\end{aligned}
\tag{4.43}
$$

gdzie $\Delta = \{[1,1],[1,-1]\}$, rozważane podejście zawiedzie, proponując w zależności od decyzji użytkownika, wynik w postaci przybliżenia R^{*}_{LB} lub R^{*}_{UB}. Przyczyna takiego stanu rzeczy, wynika bezpośrednio z braku możliwości osiągnięcia zbieżności iteracyjnego algorytmu obliczania tranzytywnego domknięcia relacji [18], których składowe wektora dystansu przyjmują wartości o przeciwnych znakach (4.43).

W konsekwencji każdy kolejny krok algorytmu iteracyjnego, to identyfikacja pozostałych krawędzi przechodnich, co w przypadku sparametryzowanych relacji zależności tworzących graf o topologii łańcucha, uniemożliwia uzyskanie dokładnego rozwiązania w skończonym czasie. Problem ten, nie występuje w przypadku nieiteracyjnego obliczania tranzytywnego domknięcia złożonej relacji zależności, dla której ograniczenia dotyczącę dziedziny i zakresu są ignorowane.

W podrozdziale tym, przedstawiono metodę obliczania tranzytywnego domknięcia złożonej, sparametryzowanej relacji zależności, opisaną w pracy [9]. Zgodnie z zaprezentowanym tam podejściem, dysponując zbiorem $n \geq 2$ relacji R_i, $i = 1, 2, \ldots, n$, tworzymy zbiór relacji $\overline{R}_i$, poprzez usunięcie z relacji R_i, ograniczeń dotyczących jej dziedziny i zakresu. Jeśli dla zbioru relacji $\overline{R}_i$, operator złożenia $(\circ)$ spełnia warunek przemienności tzn. $\overline{R}_i \circ \overline{R}_j = \overline{R}_j \circ \overline{R}_i$ dla każdego *i*, *j* takiego, że $i \neq j$ i $1 \leq i, j \leq n$, to relację $R^+ = \bigcup_{k=1}^{\infty} R^k$, obliczamy zgodnie z poniższym wzorem:

$$R^+ = \left\{ \begin{array}{l} [x] \rightarrow [y] \mid x \in domain(R) \wedge y \in range(R) \wedge \\ \exists k, k_i \geq 0,\ i = 1,2,\ldots,n,\ s.t.\ y \in \prod_{i=1}^{n} \overline{R}_i^{k_i}(x) \wedge \sum_{i=1}^{n} k_i = k \wedge k > 0 \end{array} \right\}. \qquad (4.44)$$

Dla następujących klas relacji zależności, operator złożenia $(\circ)$ relacji zawsze spełnia warunek przemienności [97] :

1. Relacje zależności o stałym wektorze dystansu (ang. *uniform relations*), postaci $R = \{[x] \rightarrow [x + b] \mid Cx \geq d\}$, gdzie *C* to macierz, a *b* i *d* to wektory o elementach należących do zbioru liczb całkowitych.
2. Relacje zależności o zmiennym wektorze dystansu (ang. *non-uniform relations*), postaci $R = \{[x] \rightarrow [Ax] \mid Cx \geq d\}$, gdzie *A* jest macierzą diagonalną, a *C* i *d* to kolejno: macierz i wektor o elementach należących do zbioru liczb całkowitych.
3. Relacje zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form relations*), postaci $R = \{[x] \rightarrow [y] \mid b_1 \leq y - x \leq b_2 \wedge Cx \geq d\}$, gdzie *C* to macierz, a b_1, b_2 i *d* to wektory o elementach należących do zbioru liczb całkowitych.
4. Relacje postaci $R = \{[x] \rightarrow [OP(A, x, op)] \mid Cx \geq d\}$, gdzie *A* jest macierzą diagonalną, *C* i *d* to kolejno: macierz i wektor należący do zbioru liczb całkowitych, a *OP* to operator generujący następujące wartości wyjściowe:

$$OP(A,\ x,\ op) = \begin{vmatrix} a_{11} & op_1 & x_1 \\ a_{22} & op_2 & x_2 \\ & \dots & \\ a_{nn} & op_n & x_n \end{vmatrix},$$

gdzie

$$A = \begin{vmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn} \end{vmatrix},\quad x = \begin{vmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{vmatrix},\quad op = \begin{vmatrix} op_1 \\ op_2 \\ \dots \\ op_n \end{vmatrix},\quad op_i \in \{+,\times\},\quad 1 \le i \le n,\ op \text{ jest}$$

jednakowy dla wszystkich relacji R_i.

Dowód :

Aby udowodnić, że dla relacji $\overline{R}_1 = \{[x] \to [OP(A,x,op)]\}$ i $\overline{R}_2 = \{[x] \to [OP(B,x,op)]\}$, operator złożenia jest przemienny, dla danych macierzy *A* i *B*, wektora *x* i operatora *op*, obliczamy:

$$\overline{R}_1 \circ \overline{R}_2 = \left\{ \begin{vmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{vmatrix} \to \begin{vmatrix} a_{11}\, op_1\, b_{11}\, op_1\, x_1 \\ a_{22}\, op_2\, b_{22}\, op_2\, x_2 \\ \dots \\ a_{nn}\, op_n\, b_{22}\, op_n\, x_n \end{vmatrix} \right\},$$

$$\overline{R}_2 \circ \overline{R}_1 = \left\{ \begin{vmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{vmatrix} \to \begin{vmatrix} b_{11}\, op_1\, a_{11}\, op_1\, x_1 \\ b_{22}\, op_2\, a_{22}\, op_2\, x_2 \\ \dots \\ b_{nn}\, op_n\, a_{22}\, op_n\, x_n \end{vmatrix} \right\}.$$

Ponieważ operacja dodawania / mnożenia dla każdego op_i, $1 \le i \le n$, spełnia własność łączności i przemienności, dlatego $\overline{R}_1 \circ \overline{R}_2 = \overline{R}_2 \circ \overline{R}_1$, co pozwala stwierdzić iż operator złożenia $(\circ)$ dla relacji $\overline{R}_1$ i $\overline{R}_2$ jest przemienny.

Dla przykładowych relacji $\overline{R}_1 = \{[i, j] \to [i+1, 2 \cdot j]\}$ i $\overline{R}_2 = \{[i, j] \to [i+3, 5 \cdot j]\}$, operator złożenia relacji $(\circ)$ jest przemienny (należą one do czwartej klasy z $op = [+ \cdot]$), podczas gdy dla relacji $\overline{R}_3 = \{[i, j] \to [2 \cdot i,\ j+1]\}$ i $\overline{R}_4 = \{[i, j] \to [i+3,\ 5 \cdot j]\}$ operator złożenia $(\circ)$ nie jest przemienny.

Przykład 4.10

Rozważmy przykład złożonej relacji zależności $R = \{[i, j] \to [i+1, j+1] \mid 1 \le i < n \wedge 1 \le j < n\} \cup \{[i, j] \to [i+1, j-1] \mid 1 \le i < n \wedge 2 \le j \le n\}$. Po usunięciu ograniczeń dotyczących dziedziny i zakresu z poszczególnych koniunkcji relacji *R* otrzymujemy relacje $\overline{R}_1$ i $\overline{R}_2$ postaci:

$$\begin{aligned} \overline{R}_1 &= \{[i, j] \to [i+1, j+1]\}, \\ \overline{R}_2 &= \{[i, j] \to [i+1, j-1]\}. \end{aligned} \tag{4.45}$$

Operator złożenia $(\circ)$ dla relacji $\overline{R}_1$ i $\overline{R}_2$ jest przemienny ponieważ $\overline{R}_1 \circ \overline{R}_2 - \overline{R}_2 \circ \overline{R}_1 = \varnothing$ i $\overline{R}_2 \circ \overline{R}_1 - \overline{R}_1 \circ \overline{R}_2 = \varnothing$, w związku z tym przystępujemy do obliczenia $\overline{R}_1^{k_1}$ i $\overline{R}_2^{k_2}$ zgodnie ze wzorem 4.11 przedstawionym w podrozdziale 4.1.3. W wyniku jego zastosowania, otrzymujemy poniższe relacje:

$$\begin{aligned} \overline{R}_1^{k_1} &= \{[i, j] \to [i', j'] \mid \exists k_1 \ge 0 : i' - i = k_1 \wedge j' - j = k_1\}, \\ \overline{R}_2^{k_2} &= \{[i, j] \to [i', j'] \mid \exists k_2 \ge 0 : i' - i = k_2 \wedge j' - j = -k_2\}. \end{aligned} \tag{4.46}$$

Złożenie relacji $\overline{R}_1^{k_1}$ i $\overline{R}_2^{k_2}$, przyjmuje następującą postać:

$$\overline{R}_1^{k_1} \circ \overline{R}_2^{k_2} = \{[i, j] \to [i', j'] \mid i' - i = k_1 + k_2 \ \wedge\ j' - j = k_1 - k_2 \ \wedge k_1 \ge 0 \wedge k_2 \ge 0\}. \tag{4.47}$$

Na tym etapie możliwe jest obliczenie relacji R^+, zgodnie ze wzorem (4.44)

$$R^+ = \left\{ \begin{array}{l} [i, j] \to [i', j'] \mid \underbrace{1 \le i < n \wedge 1 \le j \le n}_{\text{domain}(R)} \wedge \underbrace{2 \le i' \le n \wedge 1 \le j' \le n}_{\text{range}(R)} \wedge \\ \exists k, k_1, k_2 : \underbrace{i' - i = k_1 + k_2 \ \wedge\ j' - j = k_1 - k_2}_{[i', j'] \in \prod_{i=1}^{2} \overline{R}^{k_i}([i, j])} \wedge k_1 \ge 0 \wedge k_2 \ge 0 \wedge \underbrace{k_1 + k_2 = k}_{\sum_{i=1}^{2} k_i = k} \wedge k > 0 \end{array} \right\}. \tag{4.48}$$

Powyższy przykład, został wykonany zgodnie z podejściem przedstawionym w artykule [9], w którym autorzy w celu obliczenia tranzytywnego domknięcia złożonej

relacji zależności, zdecydowali się na usunięcie wszystkich ograniczeń, dotyczących dziedziny i zakresu z poszczególnych koniunkcji relacji wejściowej (4.45). Niestety podejście takie, może doprowadzić do powstania nieistniejących zależności przechodnich. Dla przykładowych złożonych relacji zależności R_1 (rysunek 4.6a) i R_2 (rysunek 4.6b)

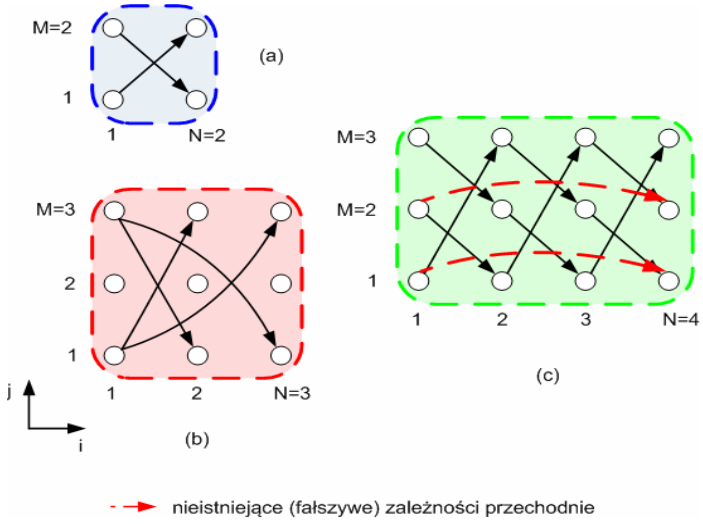
$$\begin{aligned} R_1 = \{[i,j] \rightarrow [i+1,j+1] \mid 1 \le i, i+1 \le N \wedge 1 \le j, j+1 \le M\} \cup \\ \{[i,j] \rightarrow [i+1,j-1] \mid 1 \le i, i+1 \le N \wedge 1 \le j, j-1 \le M\}, \end{aligned} \tag{4.49}$$

$$\begin{aligned} R_2 = \{[i,j] \rightarrow [i',j+2] \mid i'-i \ge 1 \wedge 1 \le i, i' \le N \wedge 1 \le j, j+2 \le M\} \cup \\ \{[i,j] \rightarrow [i'+1,j-2] \mid i'-i \ge 1 \wedge 1 \le i, i' \le N \wedge 1 \le j, j+2 \le M\}, \end{aligned} \tag{4.50}$$

ryzyko takie nie występuje, ponieważ w ich przypadku minimalne wartości parametrów N i M , dla których ograniczenia tych relacji są spełnione nie różnią się. Dla relacji (4.49) to wartości ($N \ge 2$ i $M \ge 2$), a dla relacji (4.50) to ($N \ge 2$ i $M \ge 3$). Inaczej wygląda wpływ wartości parametrów N i M na ograniczenia złożonej relacji zależności

$$\begin{aligned} R_3 = \{[i,j] \rightarrow [i+1,j+2] \mid 1 \le i, i+1 \le N \wedge 1 \le j, j+2 \le M\} \cup \\ \{[i,j] \rightarrow [i+1,j-1] \mid 1 \le i, i+1 \le N \wedge 1 \le j, j-1 \le M\}. \end{aligned} \tag{4.51}$$

Dla wartości ($N \ge 2$ i $M \ge 3$) wszystkie ograniczenia relacji (4.51) są spełnione, podczas gdy dla ($N \ge 2$ i $M = 2$), spełnione są tylko ograniczenia relacji $\{[i,j] \rightarrow [i+1,j-1] \mid 1 \le i, i+1 \le N \wedge 1 \le j, j-1 \le M\}$.



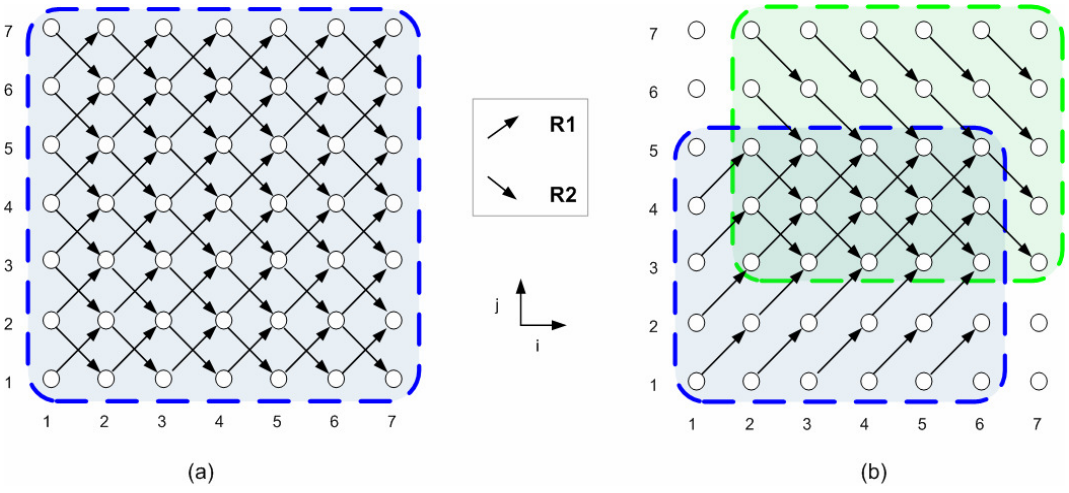
Rys 4.6 Przykłady relacji zależności dla których ryzyko wystąpienia fałszywych zależności nie występuje (a) , (b) i występuje (c) [op. własne]

Obliczając relację R^+, stosując podejście przedstawione w pracy [9], dla wartości ($N \geq 2$ i $M = 2$) wprowadzimy nieistniejące zależności przechodnie (rysunek 4.6c), ponieważ dokonując operacji złożenia poszczególnych relacji $\overline{R}_i^{k_i}$, $1 \leq i \leq n$, gdzie n to liczba koniunkcji relacji wejściowej, nie uwzględniamy minimalnych wartości parametrów N i M, dla których ograniczenia tych relacji są spełnione (4.47). Dlatego też, wartości te należy uwzględnić w trakcie wyznaczania relacji $\overline{R}_i$ (4.37).

$$\begin{aligned} \overline{R}_1 &= \{[i, j] \rightarrow [i+1, j+2] \mid N \geq 2 \wedge M \geq 3\}, \\ \overline{R}_2 &= \{[i, j] \rightarrow [i+1, j-1] \mid N \geq 2 \wedge M \geq 2\}. \end{aligned} \tag{4.52}$$

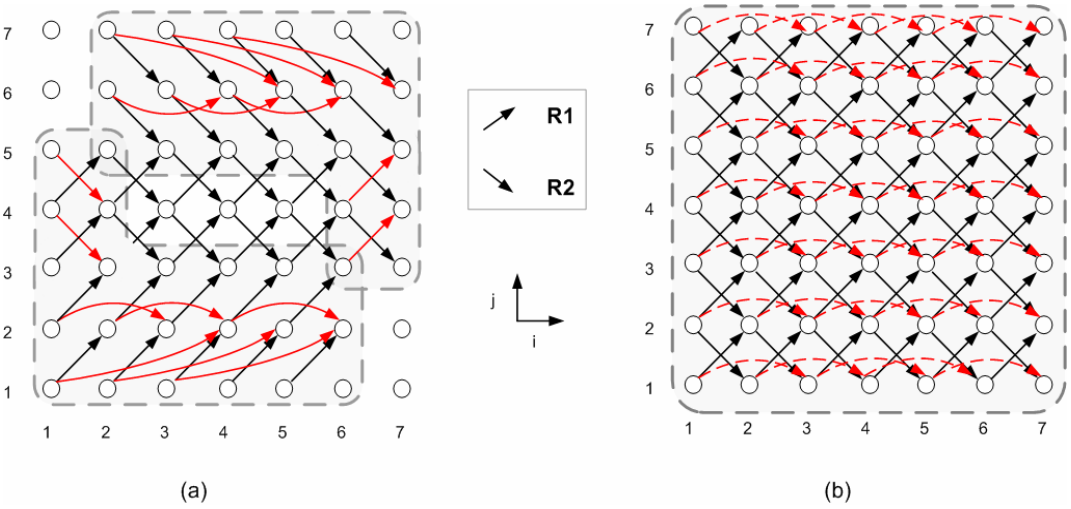
4.2.3 Podejście hybrydowe

Obliczanie relacji R^+, przedstawione w podrozdziale 4.2.2, znajduje zastosowanie, gdy złożona, sparametryzowana relacja zależności R, opisuje zależności na całej przestrzeni iteracji pętli (rysunek 4.7a). Jeśli relacje zależności, dokonują podziału przestrzeni iteracji na podprzestrzenie, co przedstawiono na rysunku 4.7b, wówczas podejście przedstawione w podrozdziale 4.2.2, nie pozwala na uzyskanie dokładnego wyniku.



Rys 4.7. Relacje zależności R_1 i R_2 opisujące zależności na
(a) całej przestrzeni iteracji pętli, (b) podprzestrzeniach przestrzeni iteracji pętli
[op.własne]

W takich sytuacjach, możliwe jest obliczenie przybliżenia górnego $\left(R_{UB}^{*}\right)$ tranzytywnego domknięcia relacji zależności *R*, co w konsekwencji wprowadza do otrzymanego rozwiązania nieistniejące zależności przechodnie (rysunek 4.8a).

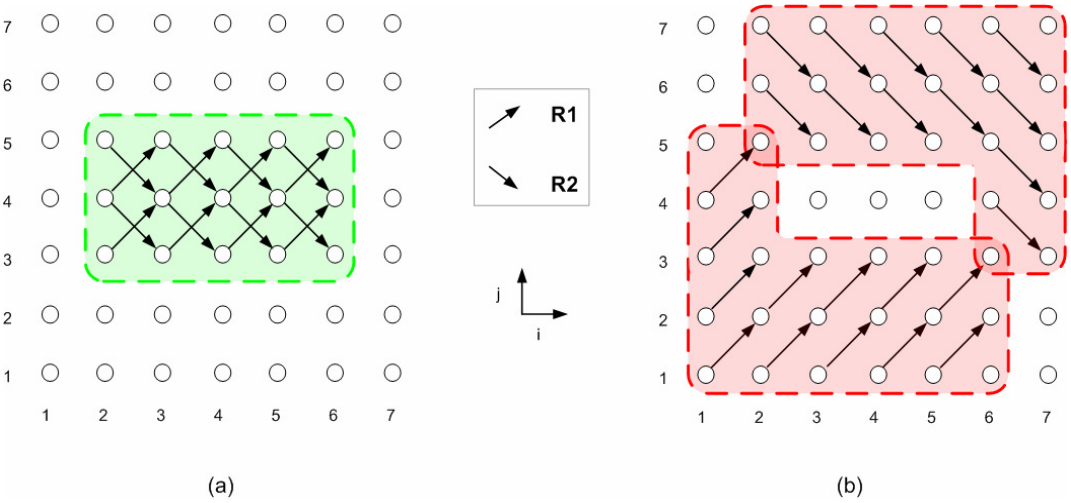


Rys.4.8 Ograniczenia metod [9] (a) i [18] (b) obliczania tranzytywnego domknięcia złożonych sparametryzowanych relacji zależności [op. własne]

Podział przestrzeni iteracji na podprzestrzenie, nie stanowi ograniczenia dla algorytmu iteracyjnego obliczania tranzytywnego domknięcia relacji zależności *R*, przedstawionego w podrozdziale 4.2.1.

Na podstawie przeprowadzonych obserwacji, autor tej pracy zdecydował się na połączenie obu technik: iteracyjnej [18] i nieiteracyjnej [9], biorąc pod uwagę pozytywne aspekty każdego rozwiązania.

Idea proponowanego poniżej algorytmu 4.6, sprowadza się do podziału całej przestrzeni iteracji na podprzestrzenie wspólne dla danej grupy relacji zależności (rysunek 4.9a), na których obliczamy tranzytywne domknięcie relacji złożonych zgodnie z podejściem przedstawionym w podrozdziale 4.2.2. Pozostała część przestrzeni iteracji (rysunek. 4.9b), rozpatrywana jest indywidualnie względem każdej z relacji zależności, gdzie tranzytywne domknięcie obliczane jest zgodnie z podejściem charakterystycznym dla relacji prostych (podrozdział 4.1). Etapem końcowym, staje się połączenie obliczonych wcześniej relacji opisujących tranzytywne domknięcie na poszczególnych podprzestrzeniach przestrzeni iteracji, stosując algorytm iteracyjny, przedstawiony w podrozdziale 4.2.1.



Rys 4.9. Przestrzeń iteracji (a) wspólna i (b) własna relacji zależności R_1 i R_2. [op. własne]

Poniżej przedstawiono algorytm 4.6, wykorzystujący zarówno techniki iteracyjne jak i nieiteracyjne, w celu obliczenia tranzytywnego domknięcia złożonej relacji *R*.

Algorytm 4.6. Obliczanie tranzytywnego domknięcia relacji *R*, z wykorzystaniem technik iteracyjnych i nieiteracyjnych.

Wejście : $R = \bigcup_{i=1}^{m} R_i$, gdzie *m* to liczba prostych relacji.

N – maksymalna liczba możliwych do wykonania iteracji algorytmu.

mode [*exact*, *upper bound*] – akceptowany wynik końcowy.

/ Dla wartości mode = exact, algorytm dąży do uzyskania dokładnego rozwiązania tranzytywnego domknięcia relacji R, jeśli nie jest to możliwe, to zwraca dolną granice tranzytywnego domknięcia relacji R. Wartość mode = upper bound, oznacza, iż akceptujemy rozwiązanie w postaci przybliżenia górnego tranzytywnego domknięcia relacji R. */*

Wyjście : R^* - dokładne tranzytywne domknięcie relacji R, jej przybliżenie górne R^*_{UB}, takie że $R^* \subset R^*_{UB}$, lub przybliżenie dolne R^*_{LB}, takie że $R^*_{LB} \subset R^*$.

Metoda :

1. Oblicz relację R^*, korzystając z podejścia przedstawionego w podrozdziale 4.2.2 i przypisz uzyskane rozwiązanie do zmiennej *TC*.

2. Jeżeli w wyniku obliczeń przeprowadzonych w punkcie 1, uzyskamy dokładne rozwiązanie tranzytywnego domknięcia relacji R (w podrozdziale 4.3, przedstawiono warunek determinujący dokładność otrzymanego rozwiązania) to

   Wypisz „*Obliczono dokładne* R^* ”
   return *TC*
   Koniec
   W przeciwnym wypadku przejdź do punktu 3.

3. Jeżeli *mode = upper bound*
   Wypisz „*Obliczono przybliżenie górne relacji* R^* ”
   return *TC*
   W przeciwnym wypadku przejdź do punktu 4.

4. $TC \leftarrow \varnothing$ */* Zainicjuj zmienną reprezentującą wynik końcowy */*

   / Rozważ punkty wspólne, pomiędzy każdą k - elementową kombinacją bez powtórzeń m-elementowego zbioru relacji wejściowych R. */*

5. Dla każdego k – elementowego, $2 \le k \le m$ podzbioru S_{ki}, $1 \le i \le n \wedge n = C^k_m$, gdzie C^k_m to k – elementowa kombinacja bez powtórzeń m – elementowego zbioru koniunkcji relacji *R*:

   5.1 Wyznacz punkty wspólne S_{common} pomiędzy relacjami $R_j \in S_{ki}$, $1 \le j \le k$ w następujący sposób :

$$S_{common} = \left(domain(R_1) \cup range(R_1)\right) \cap ... \cap \left(domain(R_k) \cup range(R_k)\right)$$

5.2 Oblicz wypukły zbiór C, zawierający elementy zbioru S_{common} , taki że dla M punktów p_1, p_2, … p_M , przestrzeń wypukła C, określona jest następującym wzorem [112] :

$$C \equiv \left\{ \sum_{j=1}^{N} \lambda_j p_j : \lambda_j \geq 0 \wedge \sum_{j=1}^{N} \lambda_j = 1 \right\}.$$

5.3 Utwórz relację R_c i sprowadź jej ograniczenia na dziedzinę i zakres do zbioru S_{common} .

$$R_c = \{[I] \rightarrow [J] \mid I, J \in S_{common}\}.$$

5.4 Dla każdej relacji $R_j \in S_{ki}$, $1 \leq j \leq k$ wykonaj operację przecięcia z relacją R_c ,

$$\overline{R} = (R_1 \cap R_c) \cup (R_2 \cap R_c) \cup \ldots \cup (R_k \cap R_c).$$

5.5 Oblicz relację $\overline{R^*}$ korzystając z podejścia przedstawionego w podrozdziale 4.2.1.

5.6 Jeżeli w wyniku obliczeń przeprowadzonych w punkcie 5.5, nie uzyskano dokładnego $\overline{R^*}$ (w podrozdziale 4.3, przedstawiono warunek determinujący dokładność otrzymanego rozwiązania), to rozpatrz kolejną kombinację relacji wejściowych *R*, kontynuując postępowanie od punktu 5.

5.7 Sprawdź czy wyznaczone $\overline{R^*}$, w punkcie 5.5 spełnia warunek $\overline{R^*} \not\subset TC$.

5.8 Jeśli warunek w punkcie 5.7 jest spełniony, to $TC = TC \cup \overline{R^*}$.

5.9 Rozważ kolejną kombinację relacji wejściowych kontynuując postępowanie od punktu 5, lub przejdź do punktu 6, gdy rozpatrzono już wszystkie możliwe kombinacje.

6. Sprawdź czy $(R_1^* \cup R_2^* \cup \ldots \cup R_m^*) \not\subset TC$.

Jeśli tak to $TC = TC \cup R_1^* \cup R_2^* \cup \ldots \cup R_m^*$.

7. Dla otrzymanego *TC*, zastosuj algorytm iteracyjny, przedstawiony w podrozdziale 4.2.1.

   TC = **Iterative Algorithm** (*TC*)

8. Jeżeli algorytm iteracyjny, nie osiągnął warunku zbieżności [18] to przejdź do punktu 10. W przeciwnym wypadku przejdź do punktu 9.

9. **Wypisz** „*Obliczono dokładne* R^* ”
   return *TC*
   Koniec

10. **Wypisz** „*Obliczono przybliżenie dolne relacji* R^* ”
    return *TC*
    Koniec

Rozważmy przebieg poszczególnych kroków algorytmu 4.6, dla przykładowych relacji R_1 i R_2.

Przykład 4.11

$$R_1 = \{[i,j] \to [i+1, j+1] \mid 1 \le i \le 5 \wedge 1 \le j \le 4\},$$
$$R_2 = \{[i,j] \to [i+1, j-1] \mid 2 \le i \le 6 \wedge 4 \le j \le 7\}.$$

Dane wejściowe algorytmu :

$R = R_1 \cup R_2$

$N = 10$

mode = exact

Wyniki poszczególnych kroków przedstawiono poniżej.

TC $\leftarrow \varnothing$.

1. ***TC*** **= {[i,j]->[i',j'] : Exists(alpha : 2alpha = i+j+i'+j' & 1<=i<=i'<=7,j'+4 & 1<=j<=4 & 2<=j'<=6 & i<=5 & i+j'<=j+i'} union {[i,j]->[i',j'] : Exists(alpha : 2alpha = i+j+i'+j' & 2<=i<i'<=7,j'+4 & 4<=j<=7 & 2<=j'<=6 & i+j<=i'+j'};**

2. Wynik uzyskany w punkcie 1, nie jest rozwiązaniem dokładnym. Przykładową nadmiarową krawędź przechodnią, uzyskaną w wyniku zastosowania algorytmu z podrozdziału 4.2.2, przedstawiono na rysunku 4.8a.

```
TC intersection {[1,1]->[4,2]};
{[1,1] -> [4,2] }
```

4. $TC \leftarrow \varnothing$

5. $S_{21} = \{R_1, R_2\}$

   5.1 S_{common} = `(domain(R1) union range(R1)) intersection (domain(R2) union range(R2)) = {[i,j]: 2 <= i <= 6 && 4 <= j <= 5} union {[i,j]: 3 <= i <= 6 && 3 <= j <= 5}`

   5.2 S_{common} = `{[i,j]: 2 <= i <= 6 && 3 <= j <= 5}`

   5.3 R_c = `{[i,j]->[i',j']};`

   R_c = R_c \ S_{common}

   R_c = R_c / S_{common}

   R_c = `{[i,j] -> [i',j'] : 2 <= i <= 6 && 3 <= j <= 5 && 2 <= i' <= 6 && 3 <= j' <= 5}`

   5.4 $\overline{R} = (R_1 \cap R_c) \cup (R_2 \cap R_c)$ = `{[i,j] -> [i+1,j+1] : 2 <= i <= 5 && 3 <= j <= 4} union {[i,j] -> [i+1,j-1] : 2 <= i <= 5 && 4 <= j <= 5}`

   5.5 $\overline{R^*}$ = `{[i,j] -> [o0,j-i+o0] : 2 <= i < o0 <= 6 && 3 <= j && j+o0 <= 5+i} union {[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && 3 <= j <= 5 && 3 <= o1 <= 5 && o0 <= 6 && 2 <= i && 2+i+o1 <= j+o0 && i+j <= o0+o1)}`

   5.7 $\overline{R^*} \not\subset TC$

   5.8 TC = $TC \cup \overline{R^*}$.

6. $(R_1^* \cup R_2^*) \not\subset TC$ (rysunek 4.9b)

$TC = TC \cup R_1^* \cup R_2^*$ = {[i,j] -> [o0,o1] : Exists (alpha :
2alpha = i+j+o0+o1 && 3 <= j <= 5 && 3 <= o1 <= 5 && o0 <= 6 &&
2 <= i && 2+i+o1 <= j+o0 && i+j <= o0+o1)} union
{[i,j] -> [o0,j-i+o0] : 1 <= i < o0 <= 6 && j+o0 <= 5+i &&
1 <= j} union {[i,j] -> [i,j] } union
{[i,j] -> [o0,i+j-o0] : 2 <= i < o0 <= 7 && j <= 7 &&
3+o0 <= i+j}

7. *TC* = Iterative Algorithm (*TC*) =

```
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && o0 <= 6
&& i+j+o1 <= 6+o0 && 6+i+o1 <= j+o0 && 3 <= o1 && 2 <= i)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && o0 <= 6
&& 4+i+o1 <= j+o0 && 3 <= o1 && i+j+o1 <= 6+o0 && 4 <= j && 2+i+j
<= o0+o1 && 2 <= i)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && j <= 4
&& o0 <= 6 && 4+i+o1 <= j+o0 && 1 <= i && 3 <= o1)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && 4 <= o1
<= 5 && o0 < = 6 && 4+i+o1 <= j+o0 && 2+i+j <= o0+o1 && 2 <= i)}
union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && 4 <= o1
<= 5 && j <= 4 && o0 <= 6 && 4+i+j <= o0+o1 && 1 <= i && 2+i+o1 <=
j+o0)} union {[i,j] -> [o0,o1] : Exists ( alpha : 2alpha =
i+j+o0+o1 && j <= 5 && o <= 7 && 6+i+o1 <= j+o0 && 3 <= o1 && 2 <=
i)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && 3 <= o1
<= 4 && j <= 4 && o0 <= 7 && 1 <= i && 4+i+o1 <= j+o0)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && 3 <= o1
<= 4 && j <= 7 && o0 <= 7 && i+j+o1 <= 6+o0 && 6+i+o1 <= j+o0 && 2
<= i)} union {[i,j] -> [o0,o1] : Exists ( alpha : 2alpha =
i+j+o0+o1 && 3 <= o1 <= 4 && o0 <= 7 && 4+i+o1 <= j+o0 && 4 <= j &&
2+i+j <= o0+o1 && 2 <= i)} union {[i,j] -> [o0,o1] : Exists ( alpha
: 2alpha = i+j+o0+o1 && j <= 5 && o0 <= 6 && 4+i+o1 <= j+o0 && 2 <=
i && 3 <= o1)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && 1 <= j
<= 4 && 3 <=o1 <= 5 && o0 <= 6 && 2+i+j <= o0+o1 && 1 <= i &&
2+i+o1 <= j+o0)} union {[i,j] -> [o0,o1] : Exists ( alpha : 2alpha
= i+j+o0+o1 && 4 <= j <= 7 && 3 <= o1 <= 5 && o0 <= 6 && 2 <= i &&
4+i+o1 <= j+o0 && i+j+o1 <= 8+o0 && i+j <= o0+o1)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && o0 <= 6
&& o1 <= 5 && 2+i+j <= o0+o1 && 2 <= i && 6+o0 <= i+j+o1 && 2+i+o1
<= j+o0)} union {[i,j] -> [o0,o1] : Exists ( alpha : 2alpha =
i+j+o0+o1 && 3 <= j <= 5 && 3 <=o1 <= 4 && o0 <= 7 && 4+i+o1 <=
j+o0 && 2 <= i)} union
{[i,j] -> [o0,o1] : Exists ( alpha : 2alpha = i+j+o0+o1 && o0 <= 7
&& 2+i+o1 <=j+o0 && 1 <= j && 3 <= o1 && j+o0+o1 <= 10+i && 2+i+j
<= o0+o1 && 1 <= i)} union {[i,j] -> [o0,o1] : Exists ( alpha :
2alpha = i+j+o0+o1 && 3 <= j <= 5 && 3 <= o1 <= 5 && o0 <= 6 && 2
<= i && 2+i+o1 <= j+o0 && i+j <= o0+o1)} union
{[i,j] -> [o0,j-i+o0] : 1 <= i < o0 <= 6 && j+o0 <= 5+i && 1 <= j}
union
{[i,j] -> [o0,i+j-o0] : 2 <= i < o0 <= 7 && j <= 7 && 3+o0 <= i+j}
union
{[i,j] -> [i,j] }
```

9 **Wypisz** „*Obliczono dokładne* R^* ”

4.3 Weryfikacja dokładności tranzytywnego domknięcia relacji zależności

Następstwem wykonania algorytmów przedstawionych w podrozdziałach 4.1 i 4.2 jest obliczenie tranzytywnego domknięcia sparametryzowanej relacji zależności, składającej się z pojedynczej lub z wielu koniunkcji. Nie zawsze jednak uzyskany przez nie wynik, reprezentuje rozwiązanie dokładne. W odróżnieniu od algorytmów iteracyjnego obliczania tranzytywnego domknięcia relacji R (podrozdziały 4.2.1 i 4.2.3), gdzie brak możliwości osiągnięcia zbieżności $(\Delta \neq \varnothing)$, oznacza iż uzyskano przybliżenie dolne - R_{LB}^{+} relacji R^{+}, dla algorytmów nieiteracyjnych (podrozdziały 4.1.2-6 i 4.2.2) istnieje ryzyko wprowadzenia fałszywych zależności przechodnich, reprezentujących przybliżenie górne - R_{UB}^{+} relacji R^{+}. Wynika ono z faktu ignorowania ograniczeń dotyczących dziedziny i zakresu relacji, w początkowej fazie działania tych algorytmów. Dlatego, też konieczne stało się zdefiniowanie warunku determinującego dokładność uzyskanej relacji R^{+}.

Poniżej przedstawiono taki warunek, który wraz z dowodem jego poprawności zaczerpnięto z pracy [72].

Dla leksykograficznie dodatnich relacji, istnieje możliwość dokonania sprawdzenia czy uzyskane przybliżenie górne - R_{UB}^{+} relacji R^{+}, stanowi jej rozwiązanie dokładne. Relacja R jest leksykograficznie dodatnia wtedy i tylko wtedy, gdy :

$$\forall\, x \to y \in R,\ 0 \prec y - x\,. \tag{4.53}$$

Twierdzenie 4.1

Dla leksykograficznie dodatnich relacji R_{UB}^{+}, R [72] :

$$R \subseteq R_{UB}^{+} \Rightarrow \left(R_{UB}^{+} = R^{+} \Leftrightarrow R_{UB}^{+} \subseteq \left(R \cup R \circ R_{UB}^{+}\right)\right). \tag{4.54}$$

Uzasadnienie dokładności tranzytywnego domknięcia relacji zależności [72]

Aby wykazać, że $\left(R_{UB}^{+} \subseteq \left(R \cup R \circ R_{UB}^{+}\right)\right) \Rightarrow \left(R_{UB}^{+} \subseteq R^{+}\right)$, przedstawmy to wyrażenie w formie odpowiedniej dla zapisu relacji zależności : $\left(\forall x \to z \in R_{UB}^{+}, x \to z \in R \vee \exists\, y \; s.t. \; x \to y \in R \wedge y \to z \in R_{UB}^{+}\right) \Rightarrow \left(\forall x \to z \in R_{UB}^{+}, x \to z \in R^{+}\right)$. Na podstawie twierdzenia o indukcji matematycznej względem $z - x$, biorąc pod uwagę zachowanie leksykograficznego porządku wierzchołków x i z, dla dowolnego $x \to y \in R_{UB}^{+}$, jeżeli $x \to z \in R$ to $x \to z \in R^{+}$. W pozostałych przypadkach, $\exists\, y \; s.t. \; x \to y \in R \wedge y \to z \in R_{UB}^{+}$. Ponieważ R_{UB}^{+} i R zachowują leksykograficzny porządek, taki że $z - x \succ z - y$, dlatego też zgodnie z hipotezą indukcyjną, $y \to z \in R^{+}$, $x \to z \in R^{+}$ i tym samym $R_{UB}^{+} \subseteq R^{+}$.

Ponieważ $R^{+} \subseteq R_{UB}^{+}$ wynika z założeń początkowych, zatem $R_{UB}^{+} = R^{+}$, jeżeli $R_{UB}^{+} \subseteq R \cup R \circ R_{UB}^{+}$.

Przykład 4.12

Rozważmy przykład relacji R^{+}, obliczonej w wyniku zastosowania algorytmu iteracyjnego z podrozdziału 4.2.2. Ponieważ algorytm ten osiągnął warunek zbieżności $(\Delta = \varnothing)$, po wykonaniu trzech przebiegów, dlatego też na tym etapie możemy stwierdzić, iż obliczono relację R_{UB}^{+} postaci:

$$
\begin{aligned}
R_{UB}^{+} = {} & \{[i,5] \to [i', i'+7] \mid 1 \le i < i' \le n-7\} \cup \\
& \{[i,5] \to [i', j'] \mid i+8 \le j' \le i'+6, n \wedge i' \le n \wedge 1 \le i\} \cup \\
& \{[i, j] \to [i', 5] \mid i+8 \le j' \le j \le i'+6, n \wedge i' \le n \wedge 1 \le i\} \cup \\
& \{[i,5] \to [i, i+7] \mid 1 \le i \le n-7\} \cup \\
& \{[i,5] \to [i', i+7] \mid 1 \le i < i' \le n \wedge i \le n-7\} \cup \\
& \{[i, j] \to [j-7, 5] \mid 1 \le i \le j-8 \wedge j \le n\} \cup \\
& \{[i, j] \to [i', j] \mid 1 \le i < i' \le n \wedge 1 \le j \le n\} \cup \\
& \{[i, j] \to [i', i'+7] \mid 1 \le i \le j-8 \wedge j-6 \le i' \le n-7\} \cup \\
& \{[i, j] \to [i', j'] \mid i+8 \le j < j' \le i'+6, n \wedge i' \le n \wedge 1 \le i\}.
\end{aligned}
$$

Sprawdźmy zatem czy powyższa relacja - R_{UB}^{+}, reprezentuje rozwiązanie dokładne - R^{+}.

$$
\begin{aligned}
R \cup \left(R \circ R_{UB}^{+}\right) = \{ & [i,5] \rightarrow [i, i+7] \mid 1 \le i \le n-7 \} \cup \\
& \{ [i, j] \rightarrow [j-7,5] \mid 1 \le i \le j-8 \wedge j \le n \} \cup \\
& \{ [i, j] \rightarrow [i', j] \mid 1 \le i < i' \le n \wedge 1 \le j \le n \} \cup \\
& \{ [i, j] \rightarrow [i', j'] \mid i+8 \le j < j' \le i'+6, n \wedge i' \le n \wedge 1 \le i \} \cup \\
& \{ [i,5] \rightarrow [i', j'] \mid i+8 \le j' \le i'+6, n \wedge i' \le n \wedge 1 \le n \} \cup \\
& \{ [i, j] \rightarrow [i',5] \mid i+8 \le j \le i'+6, n \wedge i' \le n \wedge 1 \le i \} \cup \\
& \{ [i,5] \rightarrow [i', i+7] \mid 1 \le i < i' \le n \wedge i \le n-7 \} \cup \\
& \{ [i, j] \rightarrow [i', i'+7] \mid 1 \le i \le j-8 \wedge j-6 \le i' \le n-7 \} \cup \\
& \{ [i,5] \rightarrow [i', i'+7] \mid 1 \le i < i' \le n-7 \}.
\end{aligned}
$$

Zgodnie z twierdzeniem $R_{UB}^{+} - \left(R \cup R \circ R_{UB}^{+}\right) = \varnothing$, zatem $R_{UB}^{+} \subseteq \left(R \cup R \circ R_{UB}^{+}\right)$, co pozwala nam stwierdzić, iż $R_{UB}^{+} = R^{+}$. Relacja R^{+}, obliczona w wyniku zastosowania algorytmu iteracyjnego reprezentuje rozwiązanie dokładne.

4.4 Podsumowanie

W niniejszym podrozdziale, przedstawiono algorytmy umożliwiające obliczenie dokładnego tranzytywnego domknięcia sparametryzowanych relacji zależności, składających się z wielu koniunkcji. Pierwszym rozwiązaniem w tym przypadku, zaproponowanym przez Boigelot [30, 31] i Kelly [72], było dokonanie aproksymacji relacji wejściowych, do relacji składającej się z pojedynczej koniunkcji i obliczenie jej tranzytywnego domknięcia za pomocą powszechnie stosowanych metod [3, 8, 48, 51, 62, 101, 105]. Uzyskane w ten sposób rozwiązanie, charakteryzuje się znacznym przybliżeniem (dodatkowe zależności przechodnie), które w przypadku zastosowań w zakresie przetwarzania równoległego, zmniejsza stopień uzyskanej równoległości, zawartej w pętli programowej. Dalsze prace [4, 9, 56, 63, 72, 79, 106], prowadzone były w celu opracowania nowych technik, umożliwiających poprawienie dokładności uzyskanego rozwiązania.

Podrozdziały 4.2.1 i 4.2.3, zawierają autorskie algorytmy 4.3-6 będące kontynuacją tego procesu. Wprowadzone do nich udoskonalenia, polegają między innymi na obliczeniu tranzytywnego domknięcia przy zastosowaniu algorytmów 4.1.1-6 dla każdej z relacji wchodzącej w skład unii relacji i dokonaniu podziału przestrzeni iteracji, na podprzestrzenie, przed właściwym wykonaniem algorytmu iteracyjnego.

W rozdziale 6 zbadano zakres stosowalności zaproponowanych algorytmów z wykorzystaniem dwóch zestawów pętli testowych.

5. Prace pokrewne

W literaturze odnaleźć można wiele rozwiązań, pozwalających na obliczenie tranzytywnego domknięcia grafu. Zostały one sklasyfikowane na: algorytmy grafowe [42, 52], algorytmy bazujące na operacji mnożenia macierzy [51] i algorytmy iteracyjne [48, 72]. Jednak najliczniejszą grupę stanowią algorytmy grafowe, zarówno te które dokonują podziału wierzchołków na silnie spójne składowe [102, 103], jak i te operujące na pełnym zestawie jego elementów [66]. Spora ich część, w niezmienionej formie, bądź po wprowadzonych modyfikacjach, doczekała się realizacji w komercyjnych narzędziach takich jak pakiet *Mathematica* [112]. Niestety z punktu widzenia przetwarzania równoległego, zastosowanie tych algorytmów okazuje się być ograniczone, ponieważ zostały one ukierunkowane na rozwiązaniu problemu dla grafów skończonych, czyli takich których liczba wierzchołków jest wartością znaną, w chwili dokonywania obliczeń. W przypadku grafów sparametryzowanych, dla których liczba wierzchołków wyrażona jest za pomocą wartości symbolicznej, lista publikacji w których zaproponowano potencjalne rozwiązania jest niewielka i sprowadza się do zaledwie kilkunastu pozycji [3, 9, 15, 18, 20, 21, 22, 30, 31, 62, 72, 83, 106, 107]. Analizując każdą z nich, dochodzimy do wniosku, iż poszukiwania dotyczyły nie tylko nowych algorytmów ale i odpowiedniej formy reprezentacji grafu. Pomimo, iż macierz sąsiedztwa okazała się być niewystarczającą dla tego typu zadania, pozwoliła dostrzec analogię, pomiędzy operacją mnożenia macierzy, a obliczaniem tranzytywnego domknięcia [51]. Jeśli macierzą sąsiedztwa grafu $G=(V,E)$ będzie macierz A o wymiarach $n\times n$, wówczas A^k będzie macierzą wartości logicznych, takich że $A^k[i,j]=1$, wtedy i tylko wtedy gdy istnieje ścieżka $v_i \xrightarrow{*} v_j$ o długości k. Zatem wzór (5.1) wraz z wprowadzeniem relacji krotek jako formy reprezentacji grafu

doprowadził do rozwoju algorytmów obliczania ich tranzytywnego domknięcia w sposób iteracyjny.

$$A^+ = A \vee A^2 \vee ... \vee A^n, \tag{5.1}$$

gdzie $\vee$ jest operatorem alternatywy logicznej. Algorytm 5.1, stanowi przykład takiego algorytmu przedstawionego w pracy [83].

Algorytm 5.1 Naiwny algorytm iteracyjny [83]

(1) $R^+ = R$
(2) $\Delta = R$
(3) **repeat**
(4) $\Delta = \Delta \circ R - R^+$
(5) $R^+ = R^+ \cup \Delta$
(6) u**ntil** $\Delta = \varnothing$

Ze względu na swoją prostotę, nie pozwala on na osiągnięcie zbieżności, dla relacji reprezentujących graf o topologii łańcucha. Dla przykładowej relacji $R = \{[i] \to [i+1]\}$, warunek stopu $(\Delta = \varnothing)$ algorytmu 5.1, nigdy nie będzie spełniony.

Pomimo wyraźnych ograniczeń algorytmów iteracyjnych, nie zaprzestano prac nad ich rozwojem. Na uniwersytecie Maryland, zespół Pugh`a opracował bibliotekę do przeprowadzania obliczeń symbolicznych na zbiorach i relacjach z ograniczeniami afinicznymi [71]. Wyposażono ją w mechanizmy umożliwiające obliczanie tranzytywnego domknięcia relacji, zgodnie z przedstawionym poniżej algorytmem 5.2, który kończy działanie gdy obliczono dokładne tranzytywne domknięcie relacji, lub jego dolne przybliżenie.

Idea algorytmu 5.2, zaproponowana w pracy [72], sprowadza się do takiego wyboru prostej relacji zależności *A* (linia nr. 3), będącej częścią złożonej relacji *W*, aby jej tranzytywne domknięcie (linia nr. 4) obliczone zgodnie z podejściem przedstawionym w podrozdziałach 4.1.2 i 4.1.3, składało się tylko i wyłącznie z pojedynczej koniunkcji. Ponieważ, złożeniem dwóch relacji prostych jest nadal relacja prosta, pozwala to na wyznaczenie tranzytywnego domknięcia z zastosowaniem znanych algorytmów przedstawionych w podrozdziałach 4.1.2 i 4.1.3 dla relacji, powstałych w wyniku złożenia A^+, z pozostałymi relacjami $C_i \in W$. Kolejność w jakiej wykonywane są złożenia nie jest przypadkowa i zależy od ułożenia relacji zależności względem siebie w przestrzeni iteracji.

Algorytm 5.2 Obliczanie tranzytywnego domknięcia zaproponowane w pracy [72].

Wejście : $R = \bigcup_{i=1}^{m} C_i$, gdzie m to liczba koniunkcji z których składa się relacja R

Wyjście: R^+ lub R^+_{LB}

Niezmiennik pętli : $\left(R^+ \supseteq T \cup W^+\right) \wedge \left(\text{exact} \Rightarrow R^+ = T \cup W^+\right)$

(1) $T = \varnothing$; $W = R$; *exact = true*

(2) **while** not $\left(W = \varnothing \text{ or "accept } W \text{ as } W^+_{LB}\right)$ **do**

(3) choose a conjunct $A \in W$; remove A from W

(4) **if** A^+ is known **then**

(5) $T = T \cup A^+$

(6) $W_{new} = \varnothing$

(7) **for** all conjuncts $C_i \in W$ **do**

(8) **if** $\left(A^? - A^+\right) \circ C_i \circ \left(A^? - A^+\right) \equiv C_i$ **then** $W_{new} = W_{new} \cup \left(A^? \circ C_i \circ A^?\right)$

(9) **else if** $C_i \circ \left(A^? - A^+\right) \equiv C_i$ **then** $W_{new} = W_{new} \cup \left(C_i \circ A^?\right) \cup \left(A^+ \circ C_i \circ A^?\right)$

(10) **else if** $\left(A^? - A^+\right) \circ C_i \equiv C_i$ **then** $W_{new} = W_{new} \cup \left(A^? \circ C_i\right) \cup \left(A^? \circ C_i \circ A^+\right)$

(11) **else** $W_{new} = W_{new} \cup \left(C_i \circ A^+\right) \cup \left(A^+ \circ C_i\right) \cup \left(A^+ \circ C_i \circ A^+\right) \cup C_i$

(12) **endfor**

(13) $W = W_{new}$

(14) **else**

(15) $T = T \cup A^+_{LB}$

(16) $W = \left(W \circ A^+_{LB}\right) \cup \left(A^+_{LB} \circ W \cup A^+_{LB}\right) \circ \left(W \circ A^+_{LB}\right) \cup W$

(17) *exact = false*

(18) **endwhile**

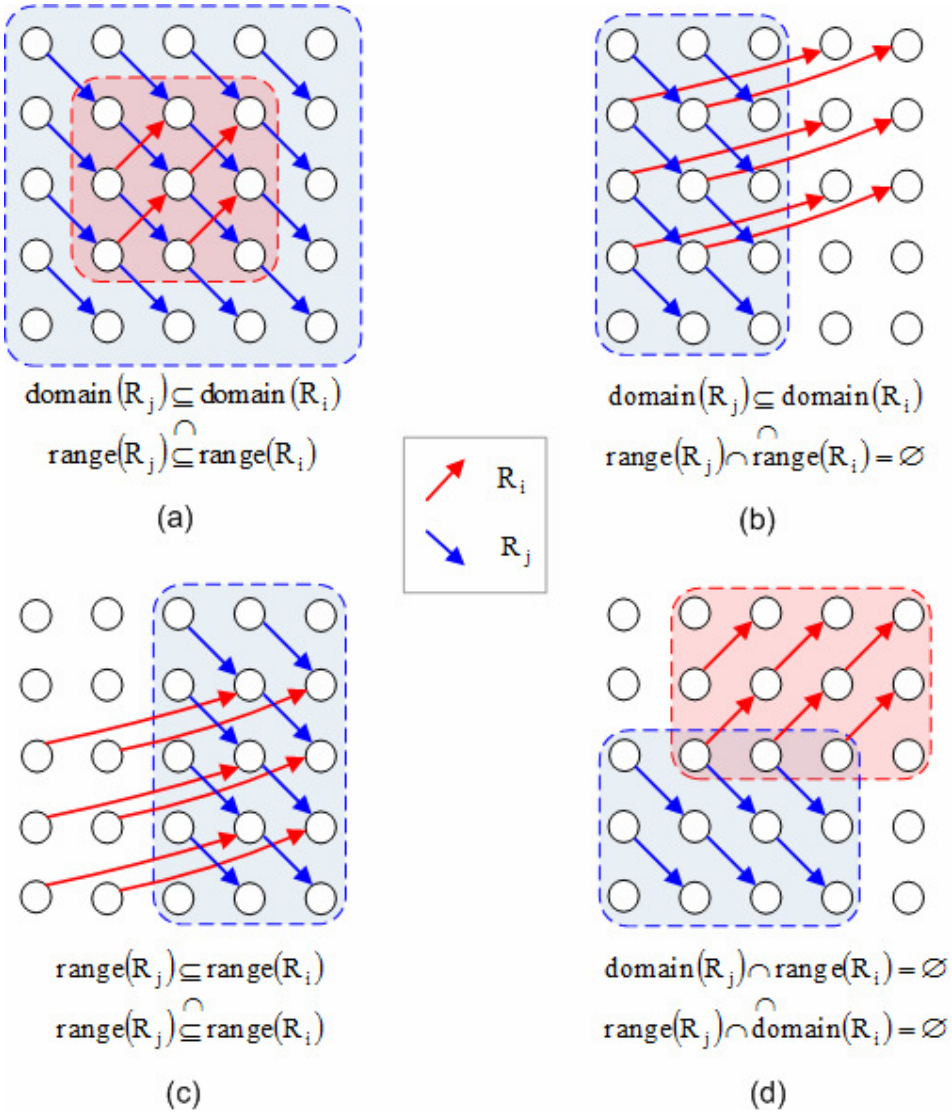
(19) **if** $\left(W = \varnothing \text{ and } exact = true\right)$ or $\left(T \cup W\right) \circ \left(T \cup W\right) \subseteq \left(T \cup W\right)$ **then**

(20) $R^+ = T \cup W$

(21) **else**

(22) $R^+_{LB} = T \cup W$

Koniec



Rys. 5.1 Możliwe warianty lokalizacji relacji w przestrzeni rozpatrywane przez algorytm 5.2 [op. własne]

Autorzy algorytmu 5.2, rozważyli trzy podstawowe warianty połączenia dwóch relacji zależności. Ich przykłady przedstawia rysunek 5.1 (a) dla linii nr 8 , (b) dla linii nr 9 i (c) dla linii nr 10. Ostatni wariant (linia nr. 11) zarezerwowano dla pozostałych możliwych połączeń pomiędzy parą relacji zależności. Rozpatruje on, również przypadek, w którym relacje nie są połączone (rysunek 5.1d).

Dokonywanie złożeń relacji A^+, które bardzo często przyjmują postać relacji zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej z pozostałymi relacjami $C_i \in W$ (linie 8-11) utrudnia późniejsze obliczenie tranzytywnego domknięcia nowo powstałych relacji. Dla przykładowej relacji zależności A :

$$A = \{[i, j] \rightarrow [i+1, j+1] \mid 1 \leq i < n \wedge 1 \leq j < n\}, \tag{5.2}$$

jej tranzytywne domknięcie A^+, obliczone zgodnie z algorytmem przedstawionym w podrozdziale 4.1.3 przyjmuje następującą postać :

$$A^+ = \{[i, j] \rightarrow [i', j-i+i'] \mid 1 \leq i < i' \leq n \wedge j+i' \leq m+i \wedge 1 \leq j\}. \tag{5.3}$$

Jakiekolwiek kolejne złożenie $A^+ \circ C_2 \circ A^+$ relacji A^+, z dowolną relacją zależności reprezentującą graf o topologii łańcucha, powiedzmy że będzie to relacja C_2 postaci :

$$C_2 = \{[i, j] \rightarrow [i+1, j-1] \mid 1 \leq i < n \wedge 2 \leq j \leq m\}, \tag{5.4}$$

skutkuje otrzymaniem relacji, (5.5) dla której obliczenie jej tranzytywnego domknięcia z zastosowaniem algorytmów przedstawionych w podrozdziałach 4.1.2 i 4.1.3, nie będzie możliwe, z uwagi na konieczność wprowadzenia nieafinicznych ograniczeń, w celu uzyskania dokładnego rozwiązania.

$$A^+ \circ C_2 \circ A^+ = \{[i, j] \rightarrow [i', j-i+i'-2] \mid 1 \leq i \leq i'-3 \wedge i' \leq n \wedge 1 \leq j \wedge j+i' \leq 2+m+i\}. \tag{5.5}$$

Na podstawie analizy algorytmu 5.2, można wysunąć następujący wniosek, iż wygenerowanie relacji postaci (5.5) w trakcie dokonywanych obliczeń, stwarza niebezpieczeństwo zakończenia jego działania z wynikiem w postaci aproksymacji tranzytywnego domknięcia dla rozpatrywanej złożonej relacji zależności.

Kolejnym, znanym autorowi sposobem umożliwiającym obliczenie tranzytywnego domknięcia złożonej sparametryzowanej relacji zależności jest podejście nieiteracyjne, przedstawione w pierwotnej formie w pracy [9] i jego udoskonalona wersja, zaprezentowana w podrozdziale 4.2.2. Zakres stosowalności tego rozwiązania, sprowadza się do sytuacji, w której złożona relacja zależności opisuje zależności na całej przestrzeni iteracji pętli. Jeśli jednak relacje zależności dokonują podziału przestrzeni iteracji na podprzestrzenie, co przedstawiono na rysunku 4.7b w podrozdziale 4.2.3, wówczas w celu obliczenia tranzytywnego domknięcia złożonej, sparametryzowanej relacji zależności, sprawdzonym rozwiązaniem, stosowanym przez wielu autorów okazało się zaadoptowanie algorytmów rozwiązujących ten problem, dla grafów o znanej liczbie wierzchołków.

Każda z wymienionych powyżej technik, realizowana jest w ramach biblioteki ISL (ang. *Integer Set Library*) [107], rozwijanej od początku 2006 roku. Jej autorem

jest Sven Verdoolaege, pracujący w instytucie badawczym INRIA Saclay (Francja). Zdecydował się on, na implementację wielu znanych sposobów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności, począwszy od podejść nieiteracyjnych po adaptację algorytmów charakterystycznych dla grafów ogólnych, takich jak algorytmy: Tarjan [102, 103] i Floyd-Warshall [109] (przedstawiony również w podrozdziale 2.1.6). Algorytmy te wywoływane są kolejno w przypadku porażki pozostałych podejść. W sytuacji, gdy żadne z nich nie pozwala na uzyskanie dokładnego rozwiązania, proponowane jest przybliżenie górne tranzytywnego domknięcia sparametryzowanej relacji zależności.

W przypadku algorytmu Tarjan [102, 103], jedna z modyfikacji polega na tym, że wierzchołki grafu reprezentowane są przez relacje zależności. Kolejna dotyczy krawędzi łączących te wierzchołki. Pomiędzy dowolną parą wierzchołków R_i i R_j, $1 \le i \ne j \le m$, gdzie m to liczba relacji, istnieje krawędź wtedy i tylko wtedy gdy

$$R_i \circ R_j \not\subset R_j \circ R_i, \tag{5.6}$$

Następnie, na podstawie istniejących połączeń między wierzchołkami, identyfikowane są silnie spójne składowe grafu (podrozdział 2.1.5), dla których tranzytywne domknięcie obliczane jest zgodnie z podejściem przedstawionym w dokumentacji biblioteki *ISL* [107] (podrozdział 2.4.2). Jeżeli dla dowolnej ze składowych obliczenie jej dokładnego tranzytywnego domknięcia nie będzie możliwe, wówczas algorytm kończy działanie z wynikiem w postaci przybliżenia górnego dla całego grafu. W przeciwnym wypadku, po wyznaczeniu tranzytywnego domknięcia dla każdej silnie spójnej składowej, należy je połączyć zgodnie z ich topologicznym porządkiem. Zaletą tego podejścia jest to, że silnie spójne składowe wykrywane są równolegle, wraz z procesem obliczania ich tranzytywnego domknięcia. Wadą natomiast jest konieczność łączenia składowych zgodnie z porządkiem topologicznym, co w przypadku grafów sparametryzowanych, nie zawsze jest jednoznaczne, ze względu na różne wartości zmiennych symbolicznych, determinujących położenie relacji względem siebie w przestrzeni.

Problem ten nie występuje, w przypadku zmodyfikowanego algorytmu Floyd-Warshall, zaproponowanego w pracy [72], gdzie tak jak w podejściu hybrydowym (podrozdział 4.2.3) uwzględniono wszystkie możliwe połączenia między relacjami o wspólnych elementach należących do zbioru dziedziny i zakresu relacji zależności. Verdoolaege w [107], dziedziny i zakresy połączonych ze sobą relacji nazwał „*partycjami*". Nie sprecyzował jednak metody ich wyszukiwania. Stwierdził tylko, że

dla złożonej relacji zależności $R = \bigcup_{i=1}^{m} R_i$, poprawne działanie algorytmu 5.3 wymaga wyznaczenia n^2 relacji postaci :

$$R_{pq} = \bigcup_{i \text{ s.t.} \operatorname{dom} R_i \subseteq P_p \wedge \operatorname{ran} R_i \subseteq P_q} R_i \,, \tag{5.7}$$

reprezentujących bezpośrednie połączenia pomiędzy daną parą partycji p i q, $1 \le p, q \le n$, gdzie n to liczba wszystkich rozpoznanych partycji.

Algorytm 5.3. Zmodyfikowany algorytm Floyd-Warshall [72, 107]

Wejście : R_{pq}, $0 \le p, q < n$, gdzie n to liczba partycji

Wyjście: $R^+ = \bigcup_{1 \le p,q \le n} R^+_{pq}$ - tranzytywne domknięcie składające się z połączeń bezpośrednich i pośrednich pomiędzy partycjami.

(1) **for** $r \in [0, n-1]$ **do**
(2) $R_{rr} = R^+_{rr}$
(3) **for** $p \in [0, n-1]$ **do**
(4) **for** $q \in [0, n-1]$ **do**
(5) **if** $p \neq r$ **OR** $q \neq r$ **then**
(6) $R_{pq} = R_{pq} \cup (R_{rq} \circ R_{pr}) \cup (R_{rq} \circ R_{rr} \circ R_{pr})$

Efektem działania algorytmu 5.3, jest relacja R^+ zawierająca wszystkie bezpośrednie i pośrednie połączenia pomiędzy partycjami. W linii numer 2 analizowanego algorytmu następuje jego rekurencyjne wywołanie, wraz z ponowieniem procesu identyfikacji partycji dla rozważanego zestawu relacji. Ma ono na celu dążenie do wyszukiwania przede wszystkim relacji prostych, (relacji składających się z pojedynczej koniunkcji) dla których możliwe staje się zastosowanie algorytmów przedstawionych w podrozdziale 4.1. Założenie to, staje się niewątpliwie słabą stroną tego podejścia, ponieważ sytuacja w której partycje łączą się za pośrednictwem tylko i wyłącznie pojedynczej relacji, nie często występuje i prowadzi zgodnie z przyjętą tendencją do zaproponowania rozwiązania w postaci przybliżenia górnego.

Wynikiem działania sekwencji omówionych algorytmów zaimplementowanych w ramach biblioteki *ISL* [107], dla przykładu 4.11 z podrozdziału 4.2.3 jest relacja R^+ postaci :

```
R+ = ({[i, j] -> [o0, o1] : (exists (e0 = [(-i - j - o0 + o1)/2]: 2e0
= -i - j - o0 + o1 and i >= 1 and i <= 5 and j >= 1 and j <= 4 and o0
>= 2 and o1 >= 2 and o1 <= 6 and o1 <= 3 + o0 and o0 <= 7 and o1 >= -4
+ o0 and o1 >= i + j - o0 and o1 <= -i + j + o0 and o0 >= 1 + i)) or
(exists (e0 = [(-i - j - o0 + o1)/2]: 2e0 = -i - j - o0 + o1 and i >=
2 and i <= 6 and j >= 4 and j <= 7 and o0 >= 2 and o1 >= 2 and o1 <= 6
and o1 <= 3 + o0 and o0 <= 7 and o1 >= -4 + o0 and o1 >= i + j - o0
and o1 <= -i + j + o0 and o0 >= 1 + i)) }, False),
```

gdzie wartość ostatniego parametru „**False**”, oznacza iż uzyskano wynik w postaci przybliżenia górnego, podczas gdy zastosowanie podejścia hybrydowego, pozwala uzyskać rozwiązanie dokładne (przykład 4.11, podrozdział 4.2.3).

Na koniec, warto też wspomnieć o algorytmie obliczania tranzytywnego domknięcia grupy relacji o ograniczeniach nałożonych na różnicę składowych krotek wejściowej i wyjściowej (ang. *difference bound constraint relations, octagonal constraint relations, relations with finite monoid affine transformations*), zaproponowanym w pracach [32, 33]. Relacja $R(x)$, gdzie $x = \{[x_1, x_2, ..., x_N]\}$ jest zbiorem składowych krotki wejściowej i wyjściowej, należy do grupy relacji o ograniczeniach nałożonych na różnicę składowych krotek wejściowej i wyjściowej [32], jeżeli zawiera skończoną liczbę nierówności postaci :

$$x_i - x_j \le a_{ij},\ 1 \le i, j \le N,\ i \ne j, \tag{5.8}$$

gdzie $a_{ij} \in Z$. Algorytm obliczania tranzytywnego domknięcia zaproponowany w pracach [32, 33], został opracowany z myślą o powyższych relacjach, w przypadku gdy składają się one z pojedynczej koniunkcji i zaimplementowany w ramach narzędzia o nazwie FLATA [122]. Dodatkowo, gwarantuje on uzyskanie dokładnego wyniku w przypadku wymienionych powyżej relacji, gdy nie zawierają one ograniczeń, nakładanych na zakres wartości, jakie mogą przyjmować składowe krotek wejściowej i wyjściowej. Własność ta, pomimo że ułatwia proces wyznaczania tranzytywnego domknięcia, tego typu relacji, na obecną chwilę, dyskwalifikuje je, jako formę reprezentacji, w sposób jak najbardziej dokładny, zależności występujących w pętlach programowych. Dlatego, też dokładna analiza rozwiązań, zawartych w pracach [32, 33], nie stała się przedmiotem zainteresowania autora tej pracy.

Wyniki przeprowadzonej analizy porównawczej w niniejszym rozdziale potwierdzają, iż zgodnie z bieżącym stanem wiedzy nie istnieje sformalizowane podejście umożliwiające obliczenie dokładnego tranzytywnego domknięcia sparametryzowanej relacji zależności w ogólnym przypadku. Wysiłki wielu badaczy zajmujących się tą tematyką, koncentrowały się wokół opracowania kolejnych, nowych rozwiązań, takich jak zaadoptowanie powszechnie znanych i sprawdzonych algorytmów stosowanych dla grafów o znanej liczbie wierzchołków, czy też stosowania technik iteracyjnych i nieiteracyjnych, w celu obliczenia dokładnego tranzytywnego domknięcia sparametryzowanych relacji zależności z ograniczeniami afinicznymi. Porównania oferowanych przez nich możliwości dokonano w kolejnym rozdziale, dotyczącym badań eksperymentalnych dla popularnych zestawów testowych NAS [115] i UTDSP [114].

6. Badania eksperymentalne

W celu przeprowadzenia badań eksperymentalnych, wybrane zostały zestawy testów NAS (ang. *NAS Parallel Benchmarks – NPB*) [7, 67, 108, 115] oraz UTDSP (ang. *University of Toronto Digital Signal Processing*) [86, 114].

Pierwszy z nich, to zbiór aplikacji opracowanych przez NASA do szacowania wydajności systemów równoległych. Zestaw NPB zawiera źródła pięciu jąder oraz trzech aplikacji CFD (ang. *Computional Fluid Dynamics* – symulacja dynamiki przepływu). Specyfikacja testów NAS udostępniona jest w postaci algorytmów przedstawiająca problem w taki sposób, aby istniało jedno unikatowe rozwiązanie i możliwe było zweryfikowanie poprawności wyników. Kwestia implementacji, wyboru struktur danych oraz sposobu użycia pamięci pozostaje otwarta dla programisty w celu uniezależnienia się od konkretnej architektury systemowej. Zbiór testów NAS jest często określany jako wyznacznik wydajności komputerów sekwencyjnych i równoległych. Do badań wykorzystano wersję 3.2 [115].

Drugim badanym zestawem testów jest UTDSP, opracowany na uniwersytecie w Toronto. UTDSP został stworzony do oszacowania jakości kodu wygenerowanego przez kompilator języka wysokiego poziomu (np. język C) stosowanego w programowalnych procesorach do cyfrowej obróbki sygnałów (DSP – ang. *Digital Signal Processing*). Oszacowanie te, zostało użyte w rozwijaniu specjalnych optymalizacji kompilatora w celu polepszenia jakości kodu i modyfikacji pod kątem architektury docelowego procesora dla uproszczenia zadania kompilatora. Aplikacje DSP napisane pierwotnie w języku niskiego poziomu (assembler) w celu stworzenia zestawu UTDSP zostały przepisane w języku C. Zestaw UTDSP, został podzielony na dwie klasy programów: 6 jąder (ang. *kernels*) oraz 10 aplikacji reprezentujących

popularne obliczenia z zakresu DSP. Do badań skorzystano z wersji 97.02.12. W poniższej tabeli 6.1 przedstawiono jądra i aplikacje zestawów testów NAS i UTDSP. W kolumnach zamieszczono ich nazwę oraz krótki opis problemu obliczeniowego.

Tabela 6.1. Jądra i aplikacje w badanych zestawach testów NAS i UTDSP [84]

NAS	**Jądra**	
	FT	Test pomiaru czasu obliczenia równań trójwymiarowej różniczki cząstkowej za pomocą prostej i odwrotnej Szybkiej Transformaty Fouriera
	MG	Test, w którym rozwiązywany jest układ równań Poisson'a w celu zbadania przemieszczenia danych na krótkie i długie dystansy
	CG	Test oparty o gradient sprzężenia (ang. *Conjugate Gradient*) oblicza przybliżoną wartość własną dużej, nieregularnej i rzadkiej macierzy. W testach zawarto nieustaloną strukturę komunikacji i obliczeń przy użyciu macierzy z losowymi lokalizacjami wspisów
	EP	Test (ang. *Embarrassingly Parallel benchmark*) zawiera losową parę odchyleń Gaussowskich zgodnie z zadanym schematem. Celem testu jest oszacowanie górnej granicy wydajności obliczeń zmiennoprzecinkowych.
	UA	Test (ang. *Unstructured Adaptive*) mierzy wpływ ciągłych i nieregularnych odwołań do pamięci.
	Aplikacje CFD	
	BT	Test do rozwiązania trójwymiarowego układu równań Navier-Stokes'a. Skończona liczba różnic rozwiązań problemu oparta jest o przybliżoną faktoryzację metody kierunków naprzemiennych (ang. *Alternating Direction Implicit - ADI*), rozdzielającej wymiary x,y,z.
	SP	Skończona liczba różnic rozwiązań problemu bazuje na przybliżonej faktoryzacji schematu Beam-Warming'a, która rozdziela wymiary x,y,z.
	LU	Symulacja aplikacji, w której zastosowano symetryczną metodę kolejnych nadrelaksacji (ang. *symmetric successive over-relaxation - SSOR*) do rozwiązania siedmio blokowego diagonalnego systemu poprzez podział na dolne i górne trójkątne systemy.
	LU-HP	hiperpłaszczyznowa wersja testów LU
UTDSP	**Jądra**	
	FFT	Szybka transformata Fouriera w wersji Radix-2 DIT (decymacja w czasie).
	FIR	Filtr o skończonej odpowiedzi impulsowej (ang. *Finite Impulse Response filter - FIR filter*)
	IIR	Filtr o nieskończonej odpowiedzi impulsowej (ang. *Infinite Impulse Response filter - FIR filter*)

	LATNRM	Znormalizowany filtr opóźniający (ang. *Normalized lattice filter*)
	LMSFIR	Adaptacyjny filtr FIR z zastosowanym algorytmem najmniejszych średnich kwadratów (ang. *Least mean squares -LMS*)
	MULT	Mnożenie macierzy
	Aplikacje DSP	
	G721_A, G721_B	Dwie implementacje kodera mowy ITU G.721 ADPCM
	V32.MODEM	Koder i dekoder modemu V.32
	ADPCM	Adaptacyjna różnicowa modulacja kodowo-impulsowa kodera mowy
	COMPRESS	Kompresja obrazu z wykorzystaniem dyskretnej transformacji cosinusowej (ang. *Discrete Cosine Transform - DCT*)
	EDGE_DETECT	Wykrywanie krawędzi 2D za pomocą operacji splotu i filtry Sobela
	HISTOGRAM	Rozszerzanie obrazu za pomocą wyrównania histogramu
	LPC	Predykcja liniowa LPC w kodowaniu mowy
	SPECTRAL	Analiza widmowa wykorzystująca uśrednianie periodogramu
	TRELLIS	Dekoder Trellis'a

Algorytmy przedstawione w rozdziałach 3 i 4, umożliwiające rozpoznanie klasy relacji zależności i obliczenie jej tranzytywnego domknięcia, wykorzystane w trakcie przeprowadzonych badań, zaimplementowano w ramach zmodyfikowanej wersji biblioteki *Omega v2.1.0* [121].

6.1 Kryteria wyboru pętli do badań eksperymentalnych

Wybór pętli z zestawów NAS i UTDSP dokonano na podstawie struktury pętli oraz relacji opisujących zależności. Pętle posiadają górną i dolną granicę w postaci stałej i sparametryzowanej oraz są dowolnie zagnieżdżone. W ciele pętli dopuszczalne są odwołania do zmiennych skalarnych oraz zmiennych tablicowych. Pętla nie może zawierać instrukcji skoku, instrukcji zmieniających krok pętli oraz odwołań do niezdefiniowanych zewnętrznych funkcji, które mogą generować dodatkowe nieznane zależności danych i tym samym uniemożliwić dokonanie dokładnej analizy zależności.

Zestaw pętli NAS zawiera 431 pętli. Uwzględniając powyższe kategorie zakwalifikowano 257 pętli, co stanowi około 60% pętli. Następnym kryterium jest obecność w pętli zależności danych. 133 pętli z 257 posiada zależności co stanowi 52% wybranych pętli.

Zestaw UTDSP zawiera 77 pętli. Odrzucając pętle zawierające instrukcje skoku (np. goto, continue, break), wywołania funkcji oraz pętle nie posiadające żadnych

zależności danych otrzymano 34 pętli (około 45%). Statystyki zawarto w tabeli 6.2.

Tabela 6.2. Statystyki liczbowe i procentowe dla testów NPB i UTDSP [84]

Nazwa testu	Wszystkich pętli	Pętli spełniających kryterium struktury		Pętli zakwalifikowany--ch do analizy		Pętli odrzuconych	
		Liczba	Procent (%)	Liczba	Procent (%)	Liczba	Procent (%)
NAS Parallel Benchmark							
BT	57	37	64,9	19	33,3	38	66,7
CG	24	9	37,5	5	20,8	19	79,2
EP	1	0	0	0	0,0	1	100
FT	13	3	23,1	2	15,4	11	84,6
LU-HP	46	32	69,6	17	37,0	29	63
LU	45	32	71,1	17	37,8	28	62,2
MG	31	16	51,6	12	38,7	19	61,3
SP	50	37	74	22	44,0	28	56,0
UA	164	91	55,5	39	23,8	125	76,2
Zbiorczo	**431**	**257**	**59,6**	**133**	**30,9**	**298**	**69,1**
UTDSP Benchmark							
fft	2	2	100	0	0	2	100
fir	2	2	100	2	100	0	0
iir	1	1	100	1	100	0	0
latnrm	3	3	100	3	100	0	0
lmsfir	3	3	100	2	66	1	33,3
mult	2	2	100	2	100	0	0
adpcm	1	0	0	0	0	1	100
compress	4	3	75	3	75	1	25
edge_det.	4	4	100	4	100	0	0
histogram	4	4	100	3	75	1	25
lpc	10	10	100	9	90	1	10
spectral	9	3	33,3	1	11,1	8	88,9
trellis	10	4	40	2	20	8	80
g721, g722	9	1	11,1	1	11,1	8	88,9
v32_modem	4	0	0	0	0	4	100
jpeg	9	1	11,1	1	11,1	8	88,9
Zbiorczo	**77**	**43**	**55,8**	**34**	**44,2**	**43**	**55,8**

6.2 Klasyfikacja relacji zależności dla badanych zestawów pętli programowych

Algorytmy umożliwiające rozpoznanie danej klasy relacji zależności, przedstawione w rozdziale 3, zostały zrealizowane w ramach biblioteki *Omega v2.1.0* [71]. Zastosowano również program o nazwie PETIT [70], służący do przeprowadzenia analizy zależności występujących w pętlach programowych. Po jej przeprowadzeniu otrzymano zbiór relacji zależności występujących w badanych pętlach, wchodzących w skład dwóch popularnych benchmarków NAS [115] i UTDSP [114]. Celem przeprowadzonych badań, stało się rozpoznanie jaki jest procent relacji, należących do klas zdefiniowanych w rozdziale 3 dla poszczególnych pętli, oraz zbadanie możliwości obliczenia ich tranzytywnego domknięcia. Tabele 6.3a, 6.3b i 6.3c przedstawiają wyniki przeprowadzonych badań.

Tabela 6.3a *Statystyki liczbowe i procentowe klas relacji zależności*

Benchmark	Liczba wszystkich relacji	Relacje zależności klasy delta (ang. *d-form relations*)	% Relacji zależności klasy delta
UTDSP	700	327	46,71 %
NAS	52098	31975	61,37 %

Tabela 6.3b *Statystyki liczbowe i procentowe klas relacji zależności*

Benchmark	Relacje zależności reprezentujące graf o topologii łańcucha o stałym wektorze dystansu	% Relacji zależności reprezentujących graf o topologii łańcucha o stałym wektorze dystansu	Relacje zależności o zmiennym wektorze dystansu reprezentujące graf o topologii łańcucha	% Relacji zależności o zmiennym wektorze dystansu reprezentujących graf o topologii łańcucha	Relacje zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej	% Relacji zależności o powiązanych elementach składowych krotek wejściowej lub wyjściowej
UTDSP	124	17,71 %	0	0	2	0,29 %
NAS	8359	16,04 %	10	0,02%	247	0,48 %

Tabela 6.3c *Statystyki liczbowe i procentowe klas relacji zależności*

Benchmark	Relacje zależności o niepowiązanych elementach składowych krotek wejściowej i wyjściowej	% Relacji zależności o niepowiązanych elementach składowych krotek wejściowej i wyjściowej	Relacje zależności o różnych wymiarach krotek wejściowej i wyjściowej	% Relacji zależności o różnych wymiarach krotek wejściowej i wyjściowej
UTDSP	3	0,43 %	244	34,86 %
NAS	48	0,09 %	11459	22 %

Tabela. 6.4 Liczba i procent relacji prostych dla których wyznaczono R^+

Benchmark	Relacje dla których obliczono R^+	% Relacji dla których obliczono R^+
UTDSP	700	100 %
NAS	52098	100 %

Z analizy wyników zawartych w tabelach 6.3a, 6.3b i 6.3c, nasuwają się następujące wnioski. Dla badanych benchmarków, najliczniejszą grupę stanowią trzy klasy relacji zależności : relacje zależności o zmiennym wektorze dystansu klasy delta (ang. *d-form relations*) – 61 % dla zestawu pętli NAS [115] i 47 % dla zestawu pętli UTDSP [114], relacje zależności o różnych wymiarach krotek wejściowej i wyjściowej – 22 % dla zestawu pętli NAS, 34,86 % dla zestawu pętli UTDSP i relacje zależności o stałym wektorze dystansu – 16,04 % dla zestawu pętli NAS i 17,71 % dla zestawu pętli UTDSP. Pozostałe klasy relacji zależności, szczególnie relacje zależności o powiązanych elementach składowych krotki wejściowej lub wyjściowej – 0,48 % dla zestawu pętli NAS i 0,29 % dla zestawu pętli UTDSP, charakteryzują się niewielkim odsetkiem relacji zależności, które do nich należą, co pozwala założyć iż pomimo trudności związanych z obliczeniem dokładnego tranzytywnego domknięcia tych

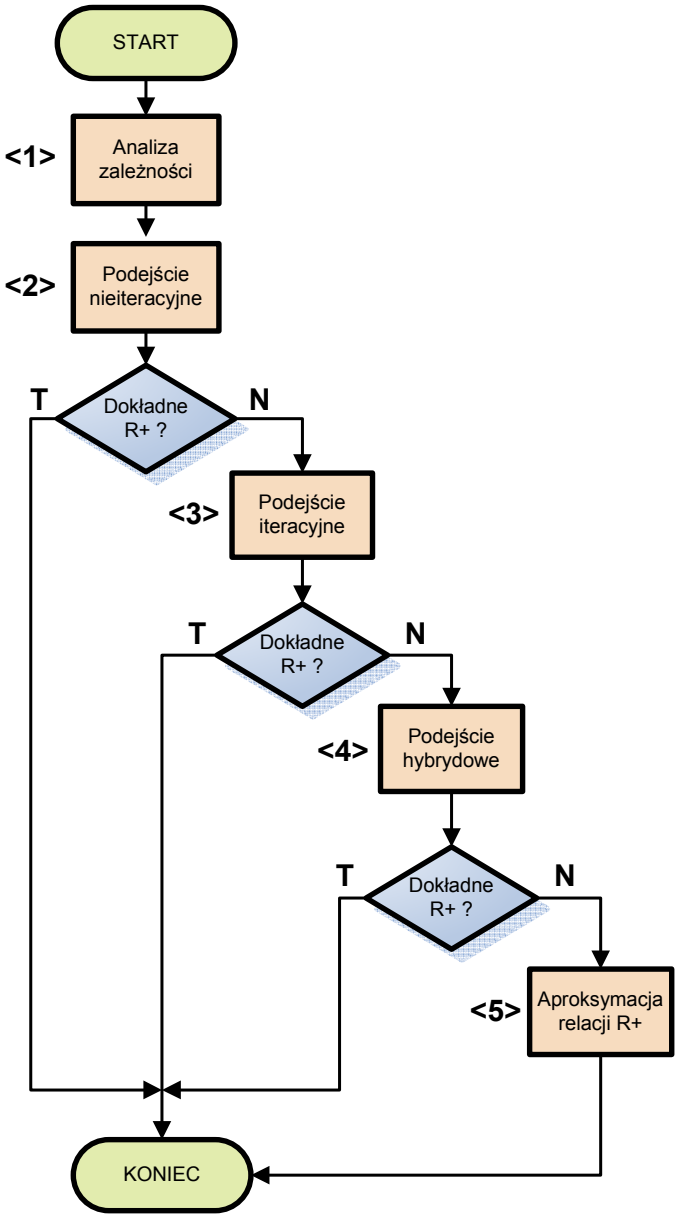
relacji (szczegóły przedstawiono w podrozdziale 4.1.6), operacja ta, zakończy się sukcesem w większości przypadków. Potwierdzenie tej prognozy przedstawiają wyniki zawarte w tabeli 6.4, przedstawiające liczbę i procent relacji prostych (składających się z pojedynczej koniunkcji), dla których wyznaczono ich dokładne tranzytywne domknicie.

6.3 Wybór algorytmu obliczania tranzytywnego domknięcia złożonych relacji zależności

Na rysunku 6.1 przedstawiono schemat, dotyczący wyboru algorytmu w zależności od wyniku otrzymanego w poprzednich próbach obliczenia dokładnego tranzytywnego domknięcia, złożonej relacji zależności. Wynika z niego, iż obliczanie tranzytywnego domknięcia złożonych relacji zależności zaczyna się analizą zależności pętli programowej <1>.W kolejnym kroku <2>, następuje próba obliczenia dokładnego tranzytywnego domknięcia relacji zależności w sposób nieiteracyjny z wykorzystaniem podejścia z podrozdziału 4.2.2. Okazuje się on być wystarczającym, gdy złożona, sparametryzowana relacja zależności opisuje zależności na całej przestrzeni iteracji pętli. W przypadku uzyskania niedokładnego wyniku, co można zweryfikować stosując twierdzenie 4.1 przedstawione w podrozdziale 4.3, następuje przejście do kroku <3> w którym tranzytywne domknięcie obliczane jest w sposób iteracyjny, zgodnie z algorytmem przedstawionym w podrozdziale 4.2.1. Jeśli i on zawiedzie, co może nastąpić w sytuacji gdy składowe wektora dystansu przyjmują zarówno wartości dodatnie jak i ujemne, wówczas wywoływany jest <4> algorytm hybrydowy, przedstawiony w podrozdziale 4.2.3, będący połączeniem technik iteracyjnego i nieiteracyjnego obliczania tranzytywnego domknięcia złożonych relacji zależności. Ostatecznie jeśli żadna z zaproponowanych technik, nie była w stanie uzyskać dokładnego wyniku, wtedy obliczane jest przybliżenie górne poszukiwanego rozwiązania <5>.

W tabeli 6.5 przedstawiono nazwę pętli, czas wykonywanych obliczeń (z dokładnością co do milisekundy, uzyskany na komputerze, którego parametry techniczne opisano w załączniku A) oraz rodzaj otrzymanego wyniku w zależności od zastosowanego podejścia. W przypadku podejść bazujących na iteracyjnym i hybrydowym (połączenie technik iteracyjnych i nieiteracyjnych) obliczaniu tranzytywnego domknięcia złożonej relacji zależności, w tabeli 6.5 zawarto kolumnę

dotyczącą liczby iteracji algorytmu, które zostały wykonane do momentu osiągnięcia bądź nie warunku zbieżności.



Rys 6.1 Schemat blokowy procesu obliczania tranzytywnego domknięcia złożonych relacji zależności i ścieżka doboru odpowiedniego algorytmu. [op. własne]

Poniżej przedstawiono legendę z opisem znaczenia poszczególnych symboli zawartych w tabeli 6.5. Uzyskane wyniki czasowe, zapisano w formacie (*m:s:ms*), gdzie

m to wartość zanotowanego pomiaru czasu w minutach, *s* określa sekundy a *ms* to milisekundy.

Legenda:

● – obliczono dokładne tranzytywne domknięcie

○ – nie uzyskano żadnego wyniku z powodu błędów biblioteki (zbyt duża liczba relacji zależności, zakończenie działania aplikacji spowodowane wystąpieniem wyjątku)

■ – uzyskano wynik w postaci przybliżenia górnego poszukiwanego rozwiązania

□ – nie uzyskano żadnego wyniku z powodu braku możliwości określenia dokładnych ograniczeń w ramach relacji opisujących zależności w badanej pętli, przez analizator zależności

Tabela 6.5 Zakres stosowalności zaproponowanych podejść dotyczących obliczania tranzytywnego domknięcia złożonych relacji zależności

Pętla	Podejście nieiteracyjne		Podejście iteracyjne			Podejście hybrydowe		
	Rodzaj wyniku	Czas obliczeń (m:s:ms)	Rodzaj wyniku	Liczba iteracji	Czas obliczeń (m:s:ms)	Rodzaj wyniku	Liczba iteracji	Czas obliczeń (m:s:ms)
NAS Benchmark								
BT_error.f2p_2	■	2:53:297	●	7	6:48:537	●	9	8:05:239
BT_error.f2p_3	■	0:00:013	●	1	0:00:013	●	2	0:00:452
BT_error.f2p_5	●	0:00:012	●	1	0:00:039	●	2	0:00:303
BT_error.f2p_6	■	0:00:020	●	1	0:00:067	●	2	0:00:772
BT_exact_rhs.f2p_2	○	-	○	-	-	○	-	-
BT_exact_rhs.f2p_3	○	-	○	-	-	○	-	-
BT_exact_rhs.f2p_4	○	-	○	-	-	○	-	-
BT_initialize.f2p_2	■	0:19:514	●	4	0:33:616	●	5	0:51:723
BT_initialize.f2p_3	■	0:18:233	●	4	0:31:827	●	5	0:52:403
BT_initialize.f2p_4	■	0:20:029	●	4	0:33:409	●	5	0:49:757
BT_initialize.f2p_5	■	0:19:917	●	4	0:34:319	●	5	0:50:638
BT_initialize.f2p_6	■	0:19:831	●	4	0:31:722	●	5	0:51:329
BT_initialize.f2p_7	■	0:18:946	●	4	0:32:245	●	5	0:52:643
BT_initialize.f2p_8	●	0:00:025	●	1	0:00:021	●	2	0:00:302
BT_initialize.f2p_9	●	0:00:011	●	1	0:00:009	●	1	0:00:013
BT_rhs.f2p_1	●	0:00:012	●	1	0:00:028	●	1	0:00:012
BT_rhs.f2p_3	○	-	○	-	-	○	-	-
BT_rhs.f2p_4	○	-	○	-	-	○	-	-
BT_rhs.f2p_5	■	3:31:805	●	6	5:23:144	●	6	9:01:438
CG_cg.f2p_3	●	0:00:010	●	1	0:00:009	●	1	0:00:010
CG_cg.f2p_4	●	0:00:014	●	1	0:00:010	●	2	0:00:046
CG_cg.f2p_6	●	0:00:010	●	1	0:00:008	●	1	0:00:010
CG_cg.f2p_7	●	0:00:013	●	1	0:00:011	●	1	0:00:074
CG_cg.f2p_8	●	0:00:010	●	1	0:00:009	●	1	0:00:023

FT_auxfnct.f2p_1	●	0:00:010	●	1	0:00:008	●	1	0:00:017
FT_auxfnct.f2p_2	●	0:00:010	●	1	0:00:008	●	1	0:00:011
LU_blts.f2p_1	○	-	○	-	-	○	-	-
LU_buts.f2p_1	○	-	○	-	-	○	-	-
LU_erhs.f2p_2	■	1:02:519	●	4	3:46:072	●	5	5:24:533
LU_erhs.f2p_3	○	-	○	-	-	■	4	4:37:782
LU_erhs.f2p_4	○	-	○	-	-	■	4	3:58:941
LU_erhs.f2p_5	○	-	○	-	-	■	4	4:11:548
LU_HP_blts.f2p_1	○	-	○	-	-	○	-	-
LU_HP_buts.f2p_1	○	-	○	-	-	○	-	-
LU_HP_erhs.f2p_2	■	1:14:823	●	4	3:49:857	●	5	6:41:836
LU_HP_erhs.f2p_3	○	-	○	-	-	■	4	6:12:952
LU_HP_erhs.f2p_4	○	-	○	-	-	■	4	5:54:129
LU_HP_erhs.f2p_5	○	-	○	-	-	■	4	6:02:974
LU_HP_jacld.f2p_1	○	-	○	-	-	○	-	-
LU_HP_jacu.f2p_1	○	-	○	-	-	○	-	-
LU_HP_lnorm.f2p.2	●	0:00:706	●	1	0:00:136	●	2	0:00:725
LU_HP_pintgr.f2p_2	●	0:00:911	●	1	0:00:279	●	2	0:01:557
LU_HP_pintgr.f2p_3	●	0:00:502	●	1	0:00:024	●	2	0:00:810
LU_HP_pintgr.f2p_7	●	0:00:487	●	1	0:00:026	●	2	0:00:909
LU_HP_pintgr.f2p_11	●	0:00:399	●	1	0:00:009	●	1	0:00:537
LU_HP_rhs.f2p_1	●	0:00:493	●	1	0:00:010	●	2	0:00:102
LU_HP_rhs.f2p_2	○	-	○	-	-	■	4	5:37:947
LU_HP_rhs.f2p_3	○	-	○	-	-	■	4	5:49:541
LU_HP_rhs.f2p_4	○	-	○	-	-	■	4	6:03:934
LU_jacld.f2p_1	○	-	○	-	-	○	-	-
LU_jacu.f2p_1	○	-	○	-	-	○	-	-
LU_l2norm.f2p_2	●	0:00:801	●	1	0:00:903	●	1	0:00:507
LU_pintgr.f2p_2	●	0:00:909	●	1	0:00:919	●	2	0:01:604
LU_pintgr.f2p_3	●	0:00:013	●	1	0:00:010	●	1	0:00:301
LU_pintgr.f2p_7	●	0:00:014	●	1	0:00:011	●	1	0:00:297
LU_pintgr.f2p_11	●	0:00:013	●	1	0:00:013	●	1	0:00:312
LU_rhs.f2p_1	●	0:00:016	●	1	0:00:018	●	1	0:00:215
LU_rhs.f2p_2	○	-	○	-	-	■	4	5:53:854
LU_rhs.f2p_3	○	-	○	-	-	■	4	6:07:462
LU_rhs.f2p_4	○	-	○	-	-	○	-	-
MG_mg.f2p_1	●	0:00:010	●	1	0:00:009	●	1	0:00:009
MG_mg.f2p_3	■	0:00:013	●	1	0:00:048	●	1	0:00:501
MG_mg.f2p_4	●	0:00:009	●	1	0:00:011	●	1	0:00:010
MG_mg.f2p_5	■	0:21:649	■	11	0:53:699	●	5	2:24:499
MG_mg.f2p_6	■	1:22:917	■	11	1:07:131	●	6	2:53:647
MG_mg.f2p_7	○	-	○	-	-	■	4	6:11:185
MG_mg.f2p_8	○	-	○	-	-	■	1	0:07:973
MG_mg.f2p_9	■	2:57:732	●	5	5:11:388	●	5	5:36:258
MG_mg.f2p_10	■	0:01:654	●	2	0:05:437	●	2	1:45:831
MG_mg.f2p_11	●	0:00:030	●	1	0:00:012	●	1	0:00:012
MG_mg.f2p_12	●	0:00:028	●	1	0:00:013	●	1	0:00:010
MG_mg.f2p_13	●	0:00:033	●	1	0:00:009	●	1	0:00:011
SP_error.f2p_2	■	4:11:136	■	11	7:25:741	●	6	9:06:919
SP_error.f2p_3	■	0:00:301	●	1	0:00:203	●	2	0:00:610
SP_error.f2p_5	■	0:00:105	●	1	0:00:101	●	1	0:00:102
SP_error.f2p_6	■	0:00:103	●	1	0:00:109	●	1	0:00:109
SP_exact_rhs.f2p_2	○	-	○	-	-	○	-	-

SP_exact_rhs.f2p_3	○	-	○	-	-	○	-	-
SP_exact_rhs.f2p_4	○	-	○	-	-	○	-	-
SP_initialize.f2p_2	■	0:12:347	●	4	0:34:575	●	6	1:39:414
SP_initialize.f2p_3	■	0:11:843	●	4	0:35:162	●	5	0:58:836
SP_initialize.f2p_4	■	0:12:104	●	4	0:32:499	●	5	1:01:743
SP_initialize.f2p_5	■	0:12:231	●	4	0:34:318	●	5	0:59:951
SP_initialize.f2p_6	■	0:11:859	●	4	0:35:104	●	6	1:03:126
SP_initialize.f2p_7	■	0:11:934	●	4	0:35:109	●	5	0:35:309
SP_initialize.f2p_8	●	0:00:020	●	1	0:00:010	●	1	0:00:401
SP_ninvr.f2p_1	●	0:00:017	●	1	0:00:009	●	1	0:00:119
SP_pinvr.f2p_1	●	0:00:014	●	1	0:00:010	●	1	0:00:157
SP_rhs.f2p_1	●	0:00:105	●	1	0:00:511	●	3	0:01:347
SP_rhs.f2p_3	○	-	○	-	-	○	-	-
SP_rhs.f2p_4	○	-	○	-	-	○	-	-
SP_rhs.f2p_5	■	0:23:345	●	5	0:47:126	●	7	1:15:421
SP_txinvr.f2p_1	○	-	○	-	-	■	2	1:49:725
SP_tzetar.f2p_1	○	-	○	-	-	■	2	2:03:481
UA_adapt.f2p_1	■	0:10:738	●	3	0:27:318	●	3	0:47:611
UA_adapt.f2p_2	●	0:00:099	●	1	0:00:103	●	2	0:01:821
UA_adapt.f2p_9	■	0:00:107	●	2	0:00:317	●	4	0:51:927
UA_adapt.f2p_10	■	0:00:105	●	2	0:00:719	●	4	0:51:911
UA_adapt.f2p_11	■	0:00:106	●	2	0:00:515	●	3	0:47:258
UA_diffuse.f2p_1	●	0:00:024	●	1	0:00:010	●	1	0:00:605
UA_diffuse.f2p_2	●	0:00:011	●	1	0:00:007	●	1	0:00:302
UA_diffuse.f2p_3	●	0:00:091	●	1	0:00:901	●	1	0:00:403
UA_diffuse.f2p_4	●	0:00:095	●	1	0:00:856	●	1	0:00:414
UA_diffuse.f2p_5	●	0:00:101	●	1	0:00:903	●	1	0:00:427
UA_mason.f2p_18	●	0:00:010	●	1	0:00:012	●	1	0:00:012
UA_precond.f2p_3	●	0:00:102	●	1	0:00:013	●	1	0:00:804
UA_precond.f2p_4	●	0:00:099	●	1	0:00:014	●	1	0:00:807
UA_precond.f2p_5	●	0:17:831	■	11	0:29:737	●	5	0:47:415
UA_setup.f2p_1	●	0:00:010	●	1	0:00:010	●	1	0:00:010
UA_setup.f2p_6	■	0:00:023	●	1	0:00:022	●	1	0:00:032
UA_setup.f2p_14	■	0:08:531	●	4	0:17:374	●	5	1:38:744
UA_setup.f2p_15	■	0:10:702	●	4	0:21:406	●	4	1:49:308
UA_setup.f2p_16	●	0:00:061	●	2	0:00:028	●	2	0:00:503
UA_transfer.f2p_1	●	0:00:010	●	1	0:00:011	●	1	0:00:011
UA_transfer.f2p_2	●	0:00:011	●	1	0:00:012	●	1	0:00:010
UA_transfer.f2p_3	●	0:00:013	●	1	0:00:010	●	1	0:00:011
UA_transfer.f2p_4	●	0:06:146	●	3	0:19:113	●	3	0:19:139
UA_transfer.f2p_5	●	0:00:010	●	1	0:00:010	●	1	0:00:010
UA_transfer.f2p_6	●	0:00:026	●	1	0:00:015	●	1	0:00:013
UA_transfer.f2p_7	●	0:02:338	●	3	0:05:473	●	3	0:05:749
UA_transfer.f2p_8	●	0:03:011	●	3	0:04:237	●	3	0:04:951
UA_transfer.f2p_9	●	0:02:414	●	3	0:08:281	●	3	0:08:129
UA_transfer.f2p_10	●	0:00:033	●	1	0:00:010	●	1	0:00:010
UA_transfer.f2p_11	■	0:05:574	●	5	0:07:367	●	5	0:27:942
UA_transfer.f2p_12	■	0:02:286	●	3	0:04:445	●	5	0:19:823
UA_transfer.f2p_13	●	0:00:091	●	2	0:00:056	●	2	0:00:523
UA_transfer.f2p_14	■	0:09:152	●	4	0:16:292	●	5	0:45:117
UA_transfer.f2p_15	●	0:00:009	●	1	0:00:009	●	1	0:00:010
UA_transfer.f2p_16	■	0:01:063	●	2	0:03:317	●	2	0:03:626
UA_transfer.f2p_17	■	0:03:591	●	3	0:06:024	●	4	1:18:331

UA_transfer.f2p_18	●	0:00:010	●	1	0:00:007	●	1	0:00:014
UA_transfer.f2p_19	■	0:00:579	●	2	0:02:834	●	2	0:02:427
UA_utils.f2p_12	■	0:05:344	●	2	0:08:344	●	3	0:15:641
UTDSP Benchmark								
compress_1	■	0:00:069	●	1	0:00:012	●	1	0:00:301
compress_2	■	0:00:103	●	2	0:00:109	●	3	0:11:548
compress_3	■	0:00:104	●	2	0:00:112	●	3	0:13:146
edge_detect1	●	0:00:071	●	2	0:00:057	●	2	0:01:603
edge_detect2	●	0:00:059	●	1	0:00:033	●	1	0:00:011
edge_detect3	■	0:00:109	●	2	0:00:319	●	2	0:47:532
fir_256_64	■	0:00:902	●	2	0:00:504	●	3	1:04:617
fir_32_1	■	0:00:998	●	2	0:00:709	●	3	1:36:958
g721_w_1	●	0:00:014	●	1	0:00:010	●	1	0:00:009
histogram_1	□	-	□	-	-	□	-	-
histogram_2	●	0:00:073	●	1	0:00:493	●	1	0:00:011
histogram_3	■	0:00:197	●	2	0:00:529	●	3	1:24:648
iir_4_64	■	0:09:521	■	11	0:16:423	●	4	1:01:332
jpeg_1	●	0:00:101	●	2	0:00:127	●	2	0:00:504
latnrm_32_64	■	0:04:516	■	11	0:11:745	●	3	0:52:749
latnrm_8_1_a	●	0:10:263	●	4	0:18:905	●	6	2:14:953
latnrm_8_1_b	●	0:00:038	●	1	0:00:017	●	1	0:00:074
lmsfir_32_64	■	0:05:197	■	11	0:13:391	●	4	0:59:645
lmsfir_8_1_a	●	0:00:090	●	1	0:00:010	●	1	0:00:011
lmsfir_8_1_b	●	0:00:075	●	1	0:00:010	●	1	0:00:013
lpc_1	■	0:00:102	●	2	0:00:794	●	3	0:32:548
lpc_2	■	0:00:103	●	3	0:00:803	●	3	0:31:813
lpc_3	●	0:00:008	●	1	0:00:011	●	1	0:00:012
lpc_4	●	0:02:116	●	2	0:04:395	●	2	0:14:049
lpc_5	■	0:02:447	●	2	0:02:147	●	3	0:25:342
lpc_6	■	0:00:086	●	1	0:00:020	●	1	0:00:302
lpc_7	□	-	□	-	-	□	-	-
lpc_8	●	0:00:015	●	1	0:00:012	●	1	0:00:012
lpc_9	■	0:00:104	●	2	0:00:107	●	2	0:27:613
lpc_10	●	0:00:009	●	1	0:00:010	●	1	0:00:012
mult_10_10	■	0:00:884	●	2	0:00:381	●	4	1:01:624
spectral_1	■	0:00:952	●	3	0:01:603	●	4	2:32:533
trellis_1	□	-	□	-	-	□	-	-
trellis2	■	0:00:883	●	2	0:00:992	●	2	1:02:207

W tabeli 6.6, w kolumnie drugiej umieszczono liczbę relacji zależności pętli wejściowej. Poniżej przedstawiono również legendę z opisem znaczenia poszczególnych symboli zawartych w tabeli 6.6.

Legenda:

ł, d, g – w kolumnie trzeciej topologia oznaczają odpowiednio łańcuch, drzewo, graf,

Tabela 6.6 Topologia grafu i liczba relacji zależności dla badanych zestawów pętli

Pętla	Liczba relacji zależności	Topo-logia
NAS Benchmark		
BT_error.f2p_2	110	g
BT_error.f2p_3	8	g
BT_error.f2p_5	34	g
BT_error.f2p_6	8	g
BT_exact_rhs.f2p_2	**1553**	**g**
BT_exact_rhs.f2p_3	**1553**	**g**
BT_exact_rhs.f2p_4	**1553**	**g**
BT_initialize.f2p_2	42	g
BT_initialize.f2p_3	42	g
BT_initialize.f2p_4	42	g
BT_initialize.f2p_5	42	g
BT_initialize.f2p_6	42	g
BT_initialize.f2p_7	42	g
BT_initialize.f2p_8	3	ł
BT_initialize.f2p_9	2	ł
BT_rhs.f2p_1	46	ł
BT_rhs.f2p_3	**702**	**g**
BT_rhs.f2p_4	**510**	**g**
BT_rhs.f2p_5	128	g
CG_cg.f2p_3	1	ł
CG_cg.f2p_4	11	g
CG_cg.f2p_6	1	g
CG_cg.f2p_7	2	ł
CG_cg.f2p_8	1	ł
FT_auxfnct.f2p_1	1	g
FT_auxfnct.f2p_2	1	ł
LU_blts.f2p_1	**4885**	**g**
LU_buts.f2p_1	**5640**	**ł**
LU_erhs.f2p_2	66	ł
LU_erhs.f2p_3	**683**	**g**
LU_erhs.f2p_4	**683**	**g**
LU_erhs.f2p_5	**683**	**g**
LU_HP_blts.f2p_1	**3232**	**g**
LU_HP_buts.f2p_1	**3593**	**g**
LU_HP_erhs.f2p_2	66	g
LU_HP_erhs.f2p_3	**683**	**g**
LU_HP_erhs.f2p_4	**683**	**g**
LU_HP_erhs.f2p_5	**683**	**g**
LU_HP_jacld.f2p_1	**2634**	**ł**
LU_HP_jacu.f2p_1	**2634**	**g**
LU_HP_lnorm.f2p.2	12	ł
LU_HP_pintgr.f2p_2	109	g
LU_HP_pintgr.f2p_3	6	g
LU_HP_pintgr.f2p_7	6	g
LU_HP_pintgr.f2p_11	4	g
LU_HP_rhs.f2p_1	17	d
LU_HP_rhs.f2p_2	**683**	**g**
LU_HP_rhs.f2p_3	**683**	**g**

LU_HP_rhs.f2p_4	**683**	**g**
LU_jacld.f2p_1	**2594**	**ł**
LU_jacu.f2p_1	**2594**	**ł**
LU_l2norm.f2p_2	9	g
LU_pintgr.f2p_2	109	g
LU_pintgr.f2p_3	6	ł
LU_pintgr.f2p_7	6	ł
LU_pintgr.f2p_11	6	g
LU_rhs.f2p_1	17	d
LU_rhs.f2p_2	**683**	**g**
LU_rhs.f2p_3	**683**	**g**
LU_rhs.f2p_4	**1412**	**g**
MG_mg.f2p_1	1	ł
MG_mg.f2p_3	3	ł
MG_mg.f2p_4	1	ł
MG_mg.f2p_5	24	ł
MG_mg.f2p_6	29	ł
MG_mg.f2p_7	**510**	**ł**
MG_mg.f2p_8	**55**	**ł**
MG_mg.f2p_9	18	ł
MG_mg.f2p_10	21	g
MG_mg.f2p_11	1	ł
MG_mg.f2p_12	1	ł
MG_mg.f2p_13	1	ł
SP_error.f2p_2	107	ł
SP_error.f2p_3	6	g
SP_error.f2p_5	34	g
SP_error.f2p_6	8	g
SP_exact_rhs.f2p_2	**1553**	**ł**
SP_exact_rhs.f2p_3	**1553**	**ł**
SP_exact_rhs.f2p_4	**1553**	**ł**
SP_initialize.f2p_2	42	g
SP_initialize.f2p_3	42	g
SP_initialize.f2p_4	42	g
SP_initialize.f2p_5	42	g
SP_initialize.f2p_6	42	g
SP_initialize.f2p_7	42	g
SP_initialize.f2p_8	3	ł
SP_ninvr.f2p_1	103	ł
SP_pinvr.f2p_1	103	ł
SP_rhs.f2p_1	64	g
SP_rhs.f2p_3	**699**	**ł**
SP_rhs.f2p_4	**699**	**ł**
SP_rhs.f2p_5	127	ł
SP_txinvr.f2p_1	**271**	**ł**
SP_tzetar.f2p_1	**288**	**ł**
UA_adapt.f2p_1	32	g
UA_adapt.f2p_2	8	ł
UA_adapt.f2p_9	32	g
UA_adapt.f2p_10	22	g
UA_adapt.f2p_11	22	g
UA_diffuse.f2p_1	5	ł
UA_diffuse.f2p_2	5	g
UA_diffuse.f2p_3	4	ł
UA_diffuse.f2p_4	4	ł
UA_diffuse.f2p_5	4	ł
UA_mason.f2p_18	1	ł
UA_precond.f2p_3	4	ł

UA_precond.f2p_4	4	ł
UA_precond.f2p_5	29	g
UA_setup.f2p_1	1	ł
UA_setup.f2p_6	5	ł
UA_setup.f2p_14	31	d
UA_setup.f2p_15	19	d
UA_setup.f2p_16	4	ł
UA_transfer.f2p_1	1	ł
UA_transfer.f2p_2	1	ł
UA_transfer.f2p_3	1	ł
UA_transfer.f2p_4	7	g
UA_transfer.f2p_5	1	g
UA_transfer.f2p_6	2	g
UA_transfer.f2p_7	4	ł
UA_transfer.f2p_8	4	ł
UA_transfer.f2p_9	4	ł
UA_transfer.f2p_10	1	g
UA_transfer.f2p_11	11	g
UA_transfer.f2p_12	11	g
UA_transfer.f2p_13	4	ł
UA_transfer.f2p_14	15	g
UA_transfer.f2p_15	1	ł
UA_transfer.f2p_16	8	g
UA_transfer.f2p_17	15	g
UA_transfer.f2p_18	1	ł
UA_transfer.f2p_19	4	g
UA_utils.f2p_12	39	g
UTDSP Benchmark		
compress_1	20	d
compress_2	29	g
compress_3	29	g
edge_detect1	21	g
edge_detect2	2	g
edge_detect3	7	g
fir_256_64	51	g
fir_32_1	44	g
g721_w_1	1	ł
histogram_1	**-**	**-**
histogram_2	4	ł
histogram_3	12	g
iir_4_64	10	ł
jpeg_1	3	ł
latnrm_32_64	5	ł
latnrm_8_1_a	37	g
latnrm_8_1_b	3	ł
lmsfir_32_64	9	g
lmsfir_8_1_a	1	ł
lmsfir_8_1_b	1	ł
lpc_1	19	g
lpc_2	19	ł
lpc_3	1	ł
lpc_4	4	g
lpc_5	5	g
lpc_6	3	g
lpc_7	**-**	**-**
lpc_8	1	ł
lpc_9	5	g
lpc_10	1	ł

mult_10_10	24	g
spectral_1	23	g
trellis_1	**-**	**-**
trellis2	11	g

W tabelach 6.7 i 6.8 podano statystykę topologii pętli, dla których obliczono i nie obliczono dokładnego tranzytywnego domknięcia, złożonej relacji zależności. Najczęściej występującą topologią jest graf ogólny, następnie łańcuch. W obydwu testach najżadziej występującą topologią okazało się drzewo. W przypadku trzech pętli z zestawu testowego UTDSP : **histogram_1**, **lpc_7** i **trellis_1**, zastosowany analizator zależności – PETIT [70] nie uzyskał żadnego wyniku, a jego działanie zakończyło się z powodu zaistniałego wyjątku, co uniemożliwiło dokonanie rozpoznania topologii tychże pętli. Zatem w ich przypadku topologia pozostaje nieokreślona.

Tabela 6.7 Topologie badanych pętli dla których obliczono tranzytywne domknięcie złożonych relacji zależności

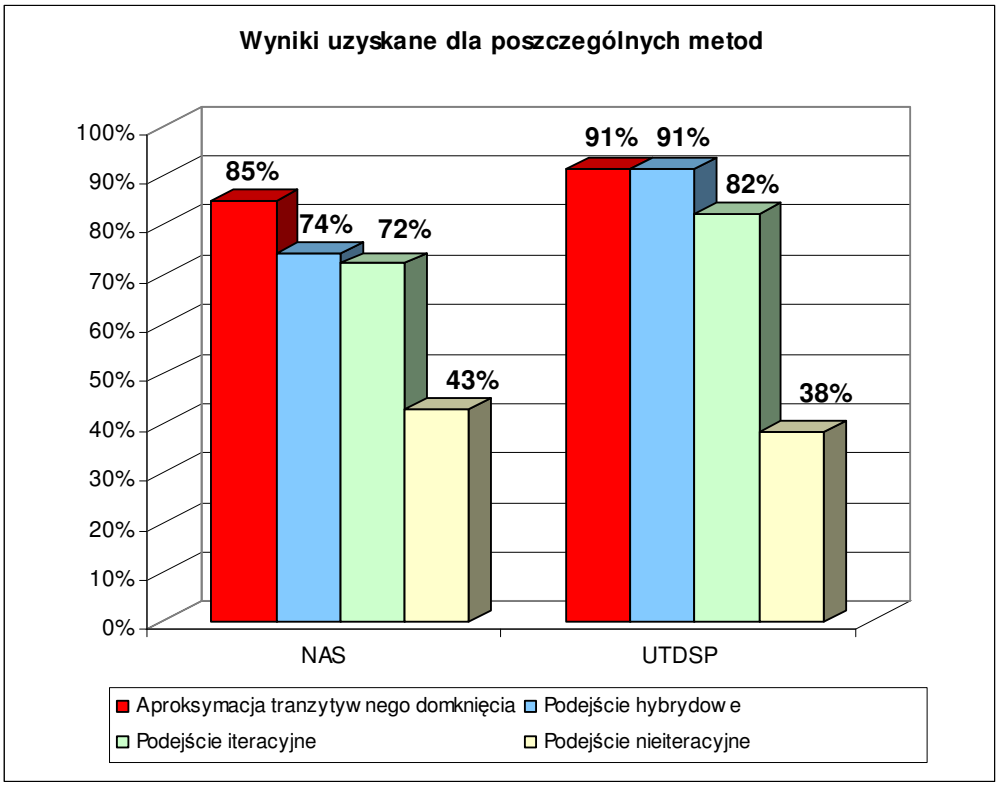
Benchmark	**Liczba bad. pętli**	**Topologia**							
		Łańcuch		**Drzewo**		**Graf ogólny**		**Nieokreślona**	
NAS	133	45	34 %	4	3 %	50	38 %	0	0 %
UTDSP	34	12	35 %	1	3 %	18	53 %	0	0 %

Tabela 6.8 Topologie badanych pętli dla których nie obliczono tranzytywnego domknięcia złożonych relacji zależności

Benchmark	**Liczba bad. pętli**	**Topologia**							
		Łańcuch		**Drzewo**		**Graf ogólny**		**Nieokreślona**	
NAS	133	13	10 %	0	0 %	21	15 %	0	0 %
UTDSP	34	0	0 %	0	0 %	0	0 %	3	9 %

Z analizy wyników przedstawionych na rysunku 6.2 wynika, iż zaproponowany w podrozdziale 4.2.3, algorytm 4.6, łączący techniki iteracyjnego i nieiteracyjnego obliczania tranzytywnego domknięcia złożonych relacji zależności (podejście hybrydowe), umożliwił obliczenie dokładnego tranzytywnego domknięcia sparametryzowanych relacji zależności dla 99 pętli z benchmarka NAS [115], co stanowi 74 % ich całkowitej liczby poddanej badaniu. Niewiele gorsze wyniki uzyskano w przypadku obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności w sposób iteracyjny (przedstawiony w podrozdziale 4.2.1). W przypadku

benchmarka NAS [115], wynik dokładny zanotowano dla 72 % pętli i 91 % w przypadku benchmarka UTDSP [114].



Rys.6.2 Zakres stosowalności technik obliczania tranzytywnego domknięcia złożonych relacji zależności [op. własne]

Najmniejszy odsetek pętli, dla których obliczono dokładne tranzytywne domknięcie ich relacji zależności, zaobserwowano w przypadku zastosowania podejścia nieiteracyjnego (opisanego w podrozdziale 4.2.2). Jest to kolejno 43 % pętli dla benchmarka NAS [115] i 38 % dla benchmarka UTDSP [114]. Wynik ten sygnalizuje, że dla analizowanego zestawu benchmarków, relacje zależności w znacznej cześci pętli (31 % - NAS i 53 % - UTDSP) dokonują podziału przestrzeni iteracji na podprzestrzenie, co stanowi ograniczenie tego podejścia (szczegóły przedstawiono w podrozdziale 4.2.3). Aproksymację tranzytywnego domknięcia relacji zależności, w postaci przybliżenia górnego obliczono dla 85 % pętli w przypadku zestawu NAS i dla 91 % pętli z zestawu UTDSP.

Pozostałe 26 % pętli z zestawu NAS [115] i 9 % pętli z zestawu UTDSP [114] to przypadki, dla których każda z analizowanych metod zawiodła z powodu ograniczeń

biblioteki, z których najbardziej znaczące to zbyt duża liczby relacji zależności tj. głównie od 200 i więcej, co utrudnia proces zarządzania pamiecią przewidzianą na przechowywanie ograniczeń w postaci równań i nierówności dla danych relacji zależności, przerwanie obliczeń z powodu wystąpienia wyjątku i brak możliwości identyfikacji istniejących zależności przez zastosowany analizator zależności PETIT [70]. Ostatnie z ograniczeń dotyczą bezpośrednio możliwości prezentowanych algorytmów.

Tabela 6.9 Ograniczenia uniemożliwiające obliczenie dokładnego tranzytywnego domknięcia sparametryzowanej relacji zależności.

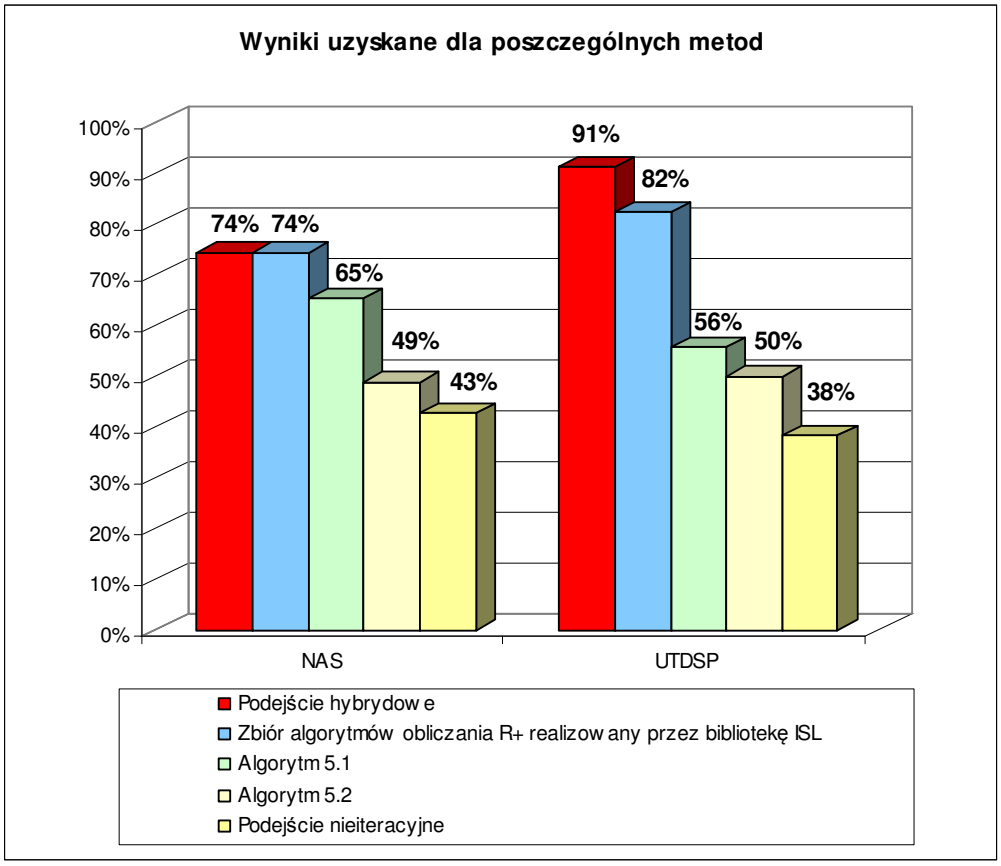
Benchmark	**Liczba badanych pętli**	**Techniki obliczania tranzytywnego domknięcia relacji zależności**	**Przyczyny braku mozliwości obliczenia relacji R+ w sposób dokładny**			
			ograniczenia biblioteki		**ograniczenia algorytmu**	
			Liczba pętli	**Procent pętli**	**Liczba pętli**	**Procent pętli**
NAS	133	Podejście nieiteracyjne	34	26 %	42	31 %
		Podejście iteracyjne			4	3 %
		Podejście hybrydowe			0	0 %
UTDSP	34	Podejście nieiteracyjne	3	9 %	18	53 %
		Podejście iteracyjne			4	12 %
		Podejście hybrydowe			0	0 %

W przypadku podejścia nieiteracyjnego ograniczeniem jest dokonywanie podziału przestrzeni iteracji przez relacje zależności na podprzestrzenie, co prowadzi do wprowadzenia nieistniejących (fałszywych) zależności przechodnich i wyniku w postaci przybliżenia górnego tranzytywnego domknięcia relacji zależności (szczegóły przedstawiono w podrozdziale 4.2.3).

W przypadku iteracyjnego obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności istotne jest, czy składowe wektora dystansu dla poszczególnych relacji przyjmują tylko wartości dodatnie, bądź tylko wartości ujemne (szczegóły przedstawiono w podrozdziale 4.2.2). Udział każdego z tych ograniczeń dla każdej z metod przedstawiono w tabeli 6.9.

W ramach przeprowadzonych badań, dokonano również porównania wyników uzyskanych dla algorytmów przedstawionych w rozdziale 5, dotyczącym prac pokrewnych. Rysunek 6.3 przedstawia zestawienie dotyczące liczby pętli z benchmarka NAS[115] i UTDSP [114], dla których obliczono dokładne tranzytywne domknięcie

złożonej relacji zależności, stosując algorytmy omówione w rodziale 5. Ponieważ w przypadku biblioteki *ISL* (ang. *Integer Set Library*) [107], obliczanie tranzytywnego domknięcia relacji, realizowane jest przy pomocy wielu algorytmów przedstawionych w rozdziale 5, wywoływanych kolejno w przypadku porażki każdego z poprzedników i niemożliwe staje się zidentyfikowanie, które z podejść doprowadziło do uzyskania dokładnego wyniku, dlatego też na potrzeby poniższego zestawienia posłużono się terminem *„zbioru algorytmów obliczania R^+, realizowanego przez bibliotekę ISL”*.



Rys 6.3 Procent pętli dla których obliczono tranzytywne domknięcie relacji zależności z zastosowaniem przedstawionych algorytmów [op. własne]

Wyniki uzyskane przez zbiór algorytmów obliczania R^+, realizowanego przez bibliotekę *ISL* (rysunek 6.3) i podejście hybrydowe przedstawione w pracy [25] są niemal identyczne. Dla benchmarka NAS [115], domknięcie przechodnie złożonej, sparametryzowanej relacji zależności obliczono odpowiednio dla 99 pętli za pomocą zbioru algorytmów obliczania R^+, realizowanego przez bibliotekę *ISL* oraz dla 99 pętli za pomocą podejścia hybrydowego. Stanowi to 74 % wszystkich pętli poddanych

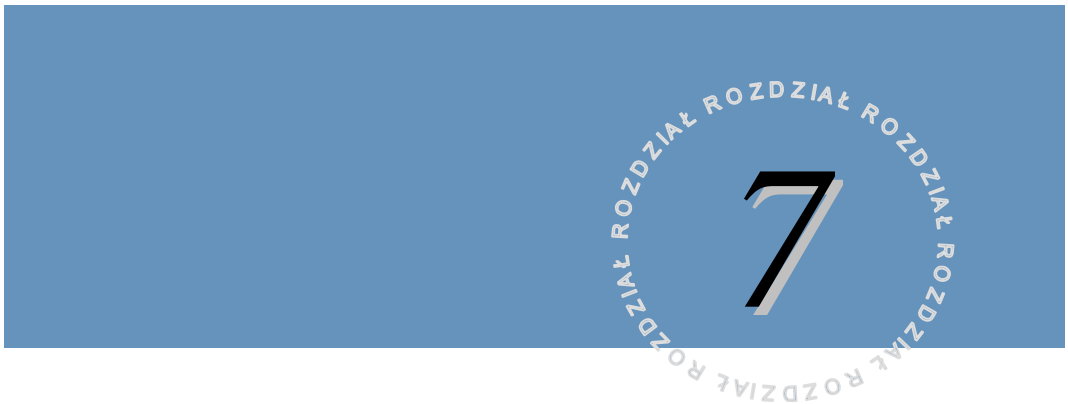
badaniu. Niewiele gorszy okazał się algorytm 5.1 (naiwny algorytm iteracyjny), z wynikiem w postaci 65 % pętli dla zestawu NAS [115] i 56 % pętli w przypadku zestawu UTDSP [114]. Zestawienie zamykają algorytm 5.2 realizowany w ramach biblioteki *Omega* [71] z wynikiem w postaci 49 % pętli dla zestawu NAS i 50 % pętli dla zestawu UTDSP, wraz z podejściem nieiteracyjnym.

Dla pozostałych pętli nie udało się wyznaczyć dokładnego tranzytywnego domknięcia złożonej, sparametryzowanej relacji zależności z powodów ograniczeń bibliotek, omówionych w bieżącym rozdziale, jak i również z powodu ograniczeń teoretycznych zastosowanych algorytmów omówionych w rozdziale 5, dotyczącym prac pokrewnych.

6.4 Podsumowanie

Wyniki przeprowadzonych badań eksperymentalnych w niniejszym rozdziale, pozwalają wysunąć następujacy wniosek, iż została potwierdzona teza pracy – zaproponowane w niniejszej pracy algorytmy, umożliwiają obliczanie dokładnego tranzytywnego domknięcia sparametryzowanych relacji zależności z ograniczeniami afinicznymi dla większego spektrum pętli programowych w porównaniu z istniejącymi metodami [9, 48, 72, 106, 107].

Pełne zestawienie wyników czasowych i statystyk z przeprowadzonych badań umieszczono w załączniku B na płycie CD. Zestawienie statystyk badanych zestawów testów NAS [115] i UTDSP [114] wraz z dodatkowymi relacjami zależności poddanym testom zawarto także w załączniku B.



7. Zastosowanie opracowanych algorytmów

Obliczenie relacji R^+ dla danej relacji *R*, symbolizującej zależności zawarte w pętli programowej stwarza możliwości jej wielorakiego zastosowania. Począwszy od testów legalności wykonywanych transformacji pętli programowych [72], poprzez usuwanie nadmiarowych operacji synchronizacji [72], po minimalizację liczby zmiennych indeksujących pętli [72], poszukiwanie niezależnych fragmentów kodu poddanych zrównolegleniu [10, 15, 84, 99], na generowaniu kodu [72] skończywszy.

Szczególnie interesujące okazały się jednak te, które dotyczą bezpośrednio transformacji pętli programowych. Aktualnie najmocniejsze z nich to przekształcenia afiniczne, polegające na wyznaczeniu harmonogramu (ang. *schedule*) dla każdej instancji instrukcji zawartej wewnątrz pętli. Harmonogram jest to odwzorowanie, które określa czas wykonania każdej iteracji poprzez wyznaczenie funkcji $\theta(S,I):\Omega\times N^{n_D}\rightarrow N$ (n_D - rozmiar przestrzeni iteracji , Ω - zbiór instrukcji $\{S_1,...,S_k\}$ w pętli), w taki sposób, aby uwzględniać występujące pomiędzy nimi zależności. Spośród wszystkich możliwych harmonogramów instancji instrukcji wyróżniony jest tak zwany harmonogram swobodny (ang. *free schedule* [50]). Polega on na tym, że iteracje wykonywane są natychmiast, kiedy tylko są dostępne (już obliczone) wszystkie ich operandy. $\theta_{free}(S,I)$ można przedstawić w formie :

$$\theta_{free}(S,I)=\begin{Bmatrix}0 \quad if \ \neg(\exists_{S'(I')}\ S'(I')\rightarrow S(I))\\ 1+\max_{\theta_{free}(S',I')}(S'(I')\rightarrow S(I))\end{Bmatrix}. \tag{7.1}$$

Taki harmonogram jest optymalny i znajduje maksymalną drobnoziarnistą równoległość zawartą w pętli. W przypadku gruboziarnistej równoległości Pugh i

Rosser [87], Pałkowski [84] i Siedlecki [99] w swoich pracach przedstawili sposób zastosowania podziału przestrzeni iteracji, poprzez wyszukanie jak największej liczby niezależnych fragmentów kodu, w celu optymalizacji komunikacji między procesorowej. Wyznaczenie fragmentu kodu (ang. *slice*), wymaga zastosowania operacji tranzytywnego domknięcia, do obliczenia tranzytywnych zależności dla wszystkich iteracji. Dzięki temu, możliwe staje się wyznaczenie zbioru osiągalnych iteracji w kierunku zgodnym z zależnością jak i odwrotnym. Następnie zbiór zależnych iteracji przeszukiwany jest w celu uzyskania niezależnych fragmentów kodu, które mogą być wykonywane bez konieczności synchronizacji pomiędzy nimi. Szczegóły dotyczące realizacji każdej z wyżej wymienionych technik przedstawiono w pracach [84, 87, 99]. Poniżej zamieszczono przykład zastosowania relacji R^k, wyznaczonej zgodnie z podejściem przedstawionym w rozdziale 4.1.4. Jest to przykład ideowy, dlatego mogą wystąpić różnice w sposobie generowania zbiorów dokonujących ekstrakcji równoległości w porównaniu z algorytmami przedstawionymi w pracach [84, 99]. Dla poniższej pętli (7.2)

```
for i = 1 to n do
  for j = 1 to m do
    a(i, j) = a(i,2*j)
  endfor
endfor
```
(7.2)

Petit [70], odszukał następującą relację zależności :

$$R = \{[i, j] \rightarrow [i, 2j] \mid 1 \le i \le n \wedge 1 \le 2j \le m\}. \tag{7.3}$$

Relacja R^k obliczona zgodnie z podejściem przedstawionym w podrozdziale 4.1.4

$$R^k = \left\{ [t_1, t_2] \rightarrow [x_1^{k,t_1}, x_2^{k,t_2}] \mid k \ge 1 \wedge 1 \le t_1 \le n \wedge 1 \le 2t_2 \le m \wedge x_1^{k,t_1} = t_1 \wedge x_2^{k,t_2} = t_2 \cdot 2^k \wedge \right. \\ \left. \exists\, \alpha : 2 \cdot \alpha = x_2^{k,t_2} \wedge 1 \le x_1^{k,t_1} \le n \wedge 2 \le x_2^{k,t_2} \le m \right\}. \tag{7.4}$$

Na początek rozważmy przykład zastosowania relacji R^k, w transformacji metodą wyznaczenia harmonogramów swobodnych. Jako pierwsze, czyli w czasie $\theta_{free}(S, I) = 0$ wykonywane są iteracje niezależne - *IND* i te należące do zbioru USV (definicja zawarta w podrozdziale 4.1.7), otrzymane w następujący sposób:

$$LD = \{[i, j] : 1 \le i \le n \ \wedge \ 1 \le j \le m\}, \tag{7.5}$$

$$IND = LD - (\text{domain}(R) \cup \text{range}(R)), \tag{7.6}$$

$$USV = (\text{domain}(R) - \text{range}(R)) = \{[i, j] : \exists\, \alpha : 2 \cdot \alpha = 1 + j \wedge 1 \le i \le n \wedge 1 \le 2j \le m - 2\}, \tag{7.7}$$

gdzie *LD* (7.5) to przestrzeń iteracji pętli (7.2), a *domain*(*R*) i *range*(*R*) to zbiory kolejno zawierające początki i końce zależnych iteracji. Kolejne iteracje wykonywane są zgodnie z porządkiem narzuconym przez relację R^k. Wyznaczając wartość $k_{\max}$:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \cdot 2^k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \le \begin{bmatrix} n \\ m \end{bmatrix} \text{ otrzymujemy } \quad t_1 \le n,$$

$$t_2 \cdot 2^k \le m \iff 2^k \le \frac{m}{t_2} \iff k \le \log_2 \frac{m}{t_2} \text{ (podstawiając minimalną}$$

wartość dla składowej t_2 ze zbioru *USV* (7.7)),

$$\min(t2) = 1, \text{ więc } k \le \log_2 \frac{m}{1} \iff k \le \log_2 m$$

$$k_{\max} = \lfloor \log_2 m \rfloor \tag{7.8}$$

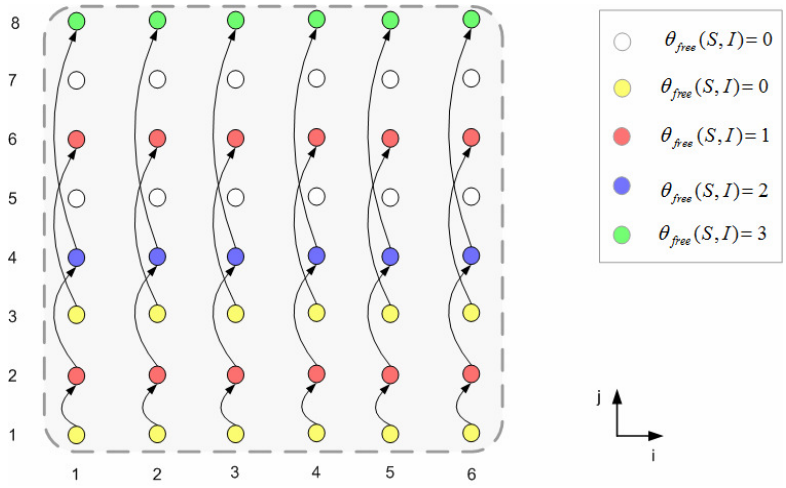
i ograniczając dziedzinę relacji R^k do zbioru wierzchołków *USV*: $R^k \setminus USV$, możemy przystąpić do wygenerowania zbioru określającego kolejne harmonogramy swobodne w zależności od wartości parametru *k*:

$$S(k) = \text{range}(R^k) \tag{7.9}$$

$$S(k) = \left\{ [t_1, t_2] : \exists\, \alpha :\ 2 \cdot 2^k \cdot \alpha = 2^k + t_2 \ \wedge 1 \le t_1 \le n \ \wedge \right. \\ \left. 2,\ 1 \cdot 2^k \le 2t_2 \le m, m \cdot 2^k - 2^{k+1} \right\}. \tag{7.10}$$

Jest to etap końcowy, umożliwiający wygenerowanie pętli skanujących poszczególne iteracje należące do zbiorów : *IND* (7.6), *USV* (7.7) i *S*(*k*) (7.9). Rysunek 7.1 przedstawia nowy porządek wykonywania iteracji pętli (7.2) dla wartości *n*=6 i *m*=8, zgodny z techniką transformacji pętli opartą na wyszukiwaniu harmonogramów swobodnych.

Zgodnie z rysunkiem 7.1 w czasie $\theta_{free}(S, I) = 0$ wykonywane są iteracje niezależne - *IND* wraz z iteracjami należącymi do zbioru *USV*. Następnie w czasie $\theta_{free}(S, I) = 1$ wykonywane są iteracje zbioru *S*(*k*=1), by w czasie $\theta_{free}(S, I) = 2$ wykonać iteracje *S*(*k*=2). Ostatecznie w czasie $\theta_{free}(S, I) = 3$ wykonywane są iteracje $S(k_{\max} = 3)$.



Rys 7.1 Ekstrakcja drobnoziarnistej równoległości metodą harmonogramów swobodnych [op. własne]

Poniższy listing 7.1 i 7.2, przedstawia kod równoległy, wygenerowany ręcznie ze względu na nieafiniczne ogranicznia relacji (7.10)

Listing 7.1

```
# kod skanujący wierzchołki niezależne codegen IND

 parfor (t1 = 1; t1 <= 6; t1++) {
   parfor (t2 = 5; t2 <= 7; t2 += 2) {
     a(t1,t2) = a(t1, 2*t2);
   }
 }

# kod skanujący wierzchołki UDS codegen UDS

 parfor (t1 = 1; t1 <= 6; t1++) {
   parfor (t2 = 1; t2 <= 3; t2 += 2) {
     a(t1,t2) = a(t1, 2*t2);
   }
 }
```

Listing 7.2

```
# kod skanujący wierzchołki zbioru S(k) wygenerowany ręcznie

 for (k=1; k<=3; k++) {
    parfor (t1 = 1; t1 <= 6; t1++) {
      parfor (t2 =max( 2, 2^k ); t2 <= min( 8, 3*2^k );  t2 += 2*2^k ) {
        a(t1,t2) = a(t1, 2*t2);
      }
    }
 }
```

Relację R^k można wykorzystać również do poszukiwaniu iteracji nie wymagających synchronizacji.

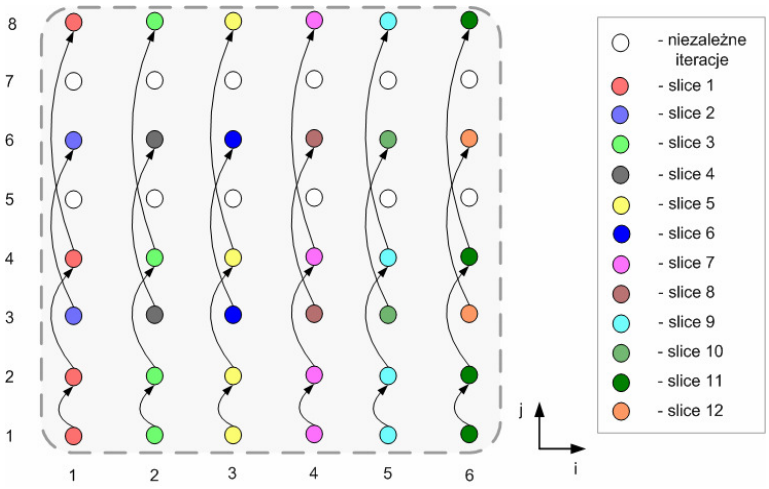
Listing 7.3

```
# kod skanujący wierzchołki niezależne codegen IND

  parfor (t1 = 1; t1 <= 6; t1++) {
    parfor (t2 = 5; t2 <= 7; t2 += 2) {
      a(t1,t2) = a(t1,2*t2);
    }
  }
```

Listing 7.4

```
# kod skanujący wierzchołki zbioru S(k) wygenerowany ręcznie

  parfor (t1 = 1; t1 <= 6; t1++) {
     parfor (t2 = 1; t2 <= 3; t2 += 2) {
       for ( k=1; k<= floor( log(8)/log(t2)); k++ ){
           a(t1, t2*2^k) = a(t1,t2*2^(k+1));
        }
  }
```

Podejście to dla rozważanego przykładu, nie różni się znacząco od metody transformacji pętli polegającej na wyznaczeniu harmonogramów swobodnych i dotyczy głownie sposobu generowania pętli przebierającej iteracje należące do zbioru $S(k)$. Różnice te przedstawia listing 7.4, który dokonuje ekstrakcji równoległości polegającej na wyszukiwaniu niezależnych fragmentów kodu [84, 99].



Rys 7.2 Ekstrakcja gruboziarnistej równoległości metodą łańcuchów pozbawionych synchronizacji [op. własne]

Liczba takich fragmentów (ang. *slice*) dla analizowanej pętli (7.2) o wartościach n=6 i m=8 zgodnie z rysunkiem 7.2 wynosi 13. Każdy z nich może być wykonywany niezależnie, bez konieczności wzajemnej synchronizacji.

Rozważana pętla (7.2), charakteryzuje się relacją zależności, składającą się z pojedynczej koniunkcji. W przypadku takich relacji obliczenie i zastosowanie relacji R^k do ekstrakcji równoległości, nie jest zagadnieniem trudnym. Inaczej, wygląda to dla relacji złożonych.

Obecnie, techniki umożliwiające obliczenie relacji R^k dla relacji składającej się z wielu koniunkcji, sprowadzają się do zaledwie kilku przypadków. Interesujące rozwiązanie, szczególnie z punktu widzenia algorytmów zaprezentowanych w tej pracy, przedstawił Sven Verdoolaege [107]. Zaproponował on obliczenie relacji R^k na podstawie obliczonej wcześniej relacji R^+. Zatem relacja R^+, może być wyznaczona za pomocą dowolnych technik: iteracyjnych, nieiteracyjnych bądź hybrydowych. Ostatecznie i tak dokonywana jest jej konwersja do relacji R^k, która moża być wykorzystywana do rozwiązania wielu problemów praktycznych. Szczegóły dotyczące tego podejścia, przedstawiono w pracy [107]. Możliwości jej zastosowania, dla ekstrakcji drobno- (***FS*** – ang. *free schedules*) [50] i gruboziarnistej (***SFS*** – ang. *synchronization free slices*) [84, 87, 99] równoległości w przypadku popularnego benchmarka NAS [115] przedstawiono w tabelach 7.1 i 7.2.

Tabela 7.1 Wyniki ekstrakcji równoległości z wykorzystaniem relacji R^k i R^+ obliczonych zgodnie z algorytmami zaproponowanymi w tej pracy.

Liczba pętli	**Algorytm transformacji pętli**	**Rodzaj uzyskanej równoległości**	**Liczba pętli**	**Procent pętli**
133	Harmonogramy swobodne (ang. *free schedules*)	pełna	**45**	**34%**
		częściowa	**27**	**20%**
		nie zrównoleglono	**61**	**46%**
	Niezależne fragmenty kodu (ang. synchronization free slices)	pełna	**42**	**32%**
		pojedynczy fragment kodu	**6**	**4%**
		nie zrównoleglono	**85**	**64%**

Z analizy wyników przedstawionych w tabeli 7.1 wynika, iż w przypadku ekstrakcji drobnoziarnistej równoległości metodą wyszukiwania harmonogramów swobodnych [40, 49, 50] pełną równoległość uzyskano dla 45 pętli, co stanowi 34% pętli poddanych

badaniu. Równoległość *„częściowa”* rozumiana jest przez autora jako niepełna, tzn. nie uzyskano pełnej równoległości zawartej w pętli programowej z powodu braku możliwości obliczenia relacji R^k w sposób dokładny (posłużono się przybliżeniem górnym, zawierającym nieistniejące (fałszywe) zależności przechodnie (nie wynikające z przebiegu zależności bezpośrednich), które zmniejszają procent dostępnej równoległości). Dla 61 pętli, co stanowi 46 % poddanych badaniu, nie udało się dokonać zrównoleglenia z uwagi na zbyt dużą liczbę zależności, która nie umożliwiła obliczenie relacji R^+, bądź sekwencyjny charakter przetwarzania instrukcji należących do utworzonego harmonogramu.

W przypadku ekstrakcji gruboziarnistej równoległości metodą wyszukiwania niezależnych fragmentów kodu [84, 99], równoległość uzyskano dla 42 pętli, co stanowi 32 % pętli poddanych badaniu. Dla 4 % pętli zanotowano pojedynczy fragment kodu, który wraz z 85 pętlami (64 % pętli poddanej badaniu), dla których działanie biblioteki z powodów wymienionych wcześniej, zakończyło się błędem, uniemożliwiło ich zrównoleglenie tą metodą.

Dla porównania, w tabeli 7.2 przedstawiono również wyniki, które uzyskano w przypadku ekstrakcji gruboziarnistej równoległości, metodą wyszukiwania niezależnych fragmentów kodu, przy zastosowaniu relacji tranzytywnego domknięcia, obliczonej zgodnie z algorytmem zaproponowanym przez Kelly i Pugh[72].

Tabela 7.2 Wyniki ekstrakcji gruboziarnistej równoległości z wykorzystaniem relacji R^+ obliczonej zgodnie z algorytmem 5.2.

Liczba pętli	**Algorytm transformacji pętli**	**Rodzaj uzyskanej równoległości**	**Liczba pętli**	**Procent pętli**
133	Niezależne fragmenty kodu (ang. synchronization free slices)	pełna	**42**	**32%**
		pojedynczy fragment kodu	**0**	**0%**
		nie zrównoleglono	**91**	**68%**

Z analizy wyników przedstawionych w tabeli 7.2 wynika, iż liczba pętli, dla których dokonano identyfikacji i zrównoleglenia niezależnych fragmentów kodu, przy zastosowaniu relacji tranzytywnego domknięcia, obliczonej zgodnie z algorytmem 5.2, zaproponowanym przez Kelly i Pugh [72] jest taka sama jak w przypadku zastosowania algorytmów obliczania tranzytywnego domknięcia relacji, zaproponowanych w niniejszej dysertacji i wynosi 42, co stanowi 32% pętli poddanych badaniu. Jednak dla

91 pętli (68 %), nie dokonano zrównoleglenia, z powodu błędów biblioteki, omówionych wcześniej i faktu istnienia pojedynczego fragmentu kodu, w przypadku którego zastosowanie algorytmów obliczania tranzytywnego domknięcia, przedstawionych w niniejszej pracy w odróżnieniu od algorytmu zaproponowanego przez Kelly i Pugh [72] pozwoliło na ich identyfikację w liczbie 6 pętli, co stanowi 4 % pętli poddanych badaniu.

Należy jednak dodać, iż wyniki zawarte w tabelach 7.1 i 7.2, dotyczą zastosowania przedstawionych w niej transformacji pętli programowych, bezpośrednio dla analizowanych pętli, tzn. pętle te ze zostały poddane żadnym dodatkowym modyfikacjom takim jak : rozszerzenie skalaru (ang. *scalar expansion*), czy też propagacja stałych (ang. *constant propagation*), które pozwoliłyby na znaczną redukcję liczby zależności, co w konsekwencji przełożyłoby się na uproszczenie grafu zależności oraz zmniejszyłoby liczbę relacji o wspólnych końcach. Zagadnienia dotyczące redukcji relacji zależności stały się przedmiotem obecnie prowadzonych badań w Katedrze Inżynierii Oprogramowania, Wydziału Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego w Szczecinie.

Zaproponowane w niniejszej pracy algorytmy obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności, zostały zaimplementowane w ramach narzędzia autorstwa dr inż. Marka Pałkowskiego, dokonującego ekstrakcji drobno- i gruboziarnistej równoległości, zawartej w pętlach programowych, dostępnego na uczelnianym serwerze Zakładu Inżynierii Oprogramowania Wydziału Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego w Szczecinie, pod adresem : http://www.sfs.zut.edu.pl i jako rozszerzenie biblioteki Omega [71] o nowej nazwie Omega+, w wersji 2.1.6, rozwijanej przez doktora Chun Chen, pracownika Wydziału Informatyki Uniwersytetu Utah, dostępnej pod adresem http://www.cs.utah.edu/~chunchen/omega.

8. Podsumowanie

W niniejszej pracy, zaprezentowano algorytmy umożliwiające obliczenie tranzytywnego domknięcia sparametryzowanych relacji zależności (rozdział 4), zarówno w sposób dokładny jak i przybliżony. Ekspozycję algorytmów poprzedzono wprowadzeniem teoretycznym, niezbędnym do zrozumienia prezentowanych rozwiązań, z naciskiem na wyjaśnienie podstaw arytmetyki Presburgera (rozdziały 1 i 2). W rozdziale 3, przedstawiono aktualnie obowiązującą klasyfikację relacji zależności występujących w pętlach programowych, wraz z algorytmami dokonującymi ich rozpoznania w sposób formalny. Prawidłowe rozpoznanie klasy, do której należy dana relacja zależności, umożliwia późniejsze zastosowanie odpowiedniego algorytmu obliczania jej tranzytywnego domknięcia.

Zaproponowano autorski algorytm iteracyjnego obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności, którego innowacyjność polega na rozpoznaniu klasy, każdej z relacji zależności wchodzącej w skład unii relacji wejściowych [19], zgodnie z algorytmami przedstawionymi w rozdziale 3, na początku i w każdym kolejnym jego przebiegu. Po rozpoznaniu klas relacji zależności, następuje obliczenie ich tranzytywnego domknięcia, zgodnie z algorytmami przedstawionymi w podrozdziale 4.1.

Udoskonalono również rozwiązania, bazujące na nieiteracyjnym obliczaniu tranzytywnego domknięcia sparametryzowanych relacji zależności (podrozdział 4.2.2), zachowując ograniczenia, które zgodnie z podejściem przedstawionym w [9] były z nich usuwane. Pozwoliło to, na uzyskanie dokładnego rozwiązania dla większej grupy relacji zależności.

Ostatecznie, w przypadku braku możliwości osiągnięcia warunku zbieżności rozwiązań iteracyjnych, zaproponowano podejście hybrydowe (podrozdział 4.2.3),

będące połączeniem technik iteracyjnego (podrozdział 4.2.1) i nieiteracyjnego (podrozdział 4.2.2) obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności, poprzez dokonanie podziału przestrzeni iteracji pętli na podprzestrzenie. Tranzytywne domknięcie relacji zależności obliczane jest wówczas w sposób nieiteracyjny, indywidualnie w ramach danej podprzestrzeni i iteracyjnie na całej przestrzeni iteracji pętli, w celu połączenia otrzymanych wcześniej wyników, uzyskanych na poszczególnych podprzestrzeniach.

W rozdziale piątym, zawarto opis prac pokrewnych do zagadnienia poruszanego w niniejszej dysertacji. Poruszono kwestię ograniczeń istniejących rozwiązań, oraz pokazano zalety zaproponowanego podejścia. Dokonano również porównania obecnie znanych algorytmów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności [9, 48, 63, 72, 106], z podejściem iteracyjnym i hybrydowym.

W rozdziale 6, przedstawiono metodykę badań eksperymentalnych, charakterystykę systemów badawczych oraz wyniki przeprowadzonych eksperymentów dla zestawów testowych NAS [115] i UTDSP [114].

Główną zaletą opracowanych algorytmów jest możliwość obliczenia tranzytywnego domknięcia sparametryzowanych relacji zależności dla szerszego spektrum pętli programowych, w porównaniu ze znanymi autorowi metodami. Implementacja proponowanych algorytmów, oraz badania eksperymentalne pozwalają stwierdzić, że możliwe jest użycie opracowanych rozwiązań jako kluczowy fragment transformacji pętli programowych, dokonujących ekstrakcji zarówno drobno- jak i gruboziarnistej równoległości.

Opracowane algorytmy w ramach zmodyfikowanej wersji biblioteki *Omega v2.1.0* [121], dostępne są pod adresem : http://www.sfs.zut.edu.pl/files/omega3.tar.gz i http://www.cs.utah.edu/~chunchen/omega w wersji 2.1.6.

8.1 Wnioski końcowe

W oparciu o zaproponowane algorytmy, oraz przeprowadzone badania eksperymentalne sformułowano następujące wnioski końcowe:

- Zastosowanie zbioru algorytmów 3.1-4, pozwala na rozpoznanie klas relacji zależności występujących w pętlach programowych w sposób formalny.
- W podrozdziałach 4.1.1-3, przedstawiono znane techniki obliczania tranzytywnego domknięcia sparametryzowanej relacji zależności o stałym (ang.

uniform relation) i zmiennym (ang. *d-form relation*) wektorze dystansu, składającej się z pojedynczej koniunkcji.

- W podrozdziale 4.1.4, sformalizowano sposób obliczania tranzytywnego domknięcia sparametryzowanej relacji zależności reprezentującej graf o topologii łańcucha o zmiennym wektorze dystansu, polegający na utworzeniu i rozwiązaniu układu równań rekurencyjnych.
- Podrozdziały 4.1.5 i 4.1.6, zawierają propozycję autora, dotyczącą uzupełnienia technik obliczania tranzytywnego domknięcia relacji zależności składających się z pojedynczej koniunkcji dla klas relacji o niepowiązanych (ang. *non-coupled index variables*) i powiązanych (ang. *coupled index variables*) elementach składowych krotek wejściowej i / lub wyjściowej.
- Zastosowanie algorytmu 4.2, umożliwia rozpoznanie nieciągłości występujących w łańcuchach zależności dla każdej z klas relacji zależności.
- Dla relacji składającej się z pojedynczej koniunkcji o przerwanym łańcuchu zależności, zastosowanie algorytmu 4.1 pozwala na usunięcie nieprawidłowych zależności przechodnich powstałych w wyniku obliczenia jej tranzytywnego domknięcia, zgodnie z technikami przedstawionymi w podrozdziałach 4.1.1-6.
- W przypadku złożonych relacji zależności (składających się z unii skończonej liczby relacji prostych), algorytm 4.3, polegający na iteracyjnym obliczaniu tranzytywnego domknięcia sparametryzowanych relacji zależności, gwarantuje uzyskanie dokładnego wyniku dla relacji, których składowe wektora dystansu przyjmują tylko wartości dodatnie bądź tylko wartości ujemne.
- Obliczenie dokładnego tranzytywnego domknięcia złożonej, sparametryzowanej relacji zależności w sposób nieiteracyjny, nie wymaga aby składowe wektora dystansu tej relacji przyjmowały tylko wartości dodatnie, bądź ujemne. Jednak zasadniczym ograniczeniem tego podejścia jest to, że złożona, sparametryzowana relacji zależności musi opisywać zależności na całej przestrzeni iteracji pętli. Podział przestrzeni iteracji na podprzestrzenie, nie pozwala na uzyskanie dokładnego wyniku.
- Połączenie technik iteracyjnego i nieiteracyjnego obliczania tranzytywnego domkniecia złożonych, sparametryzowanych relacji zależności pozwala wyodrębnić pozytywne aspekty każdego z rozwiązań i tym samym umożliwia

obliczenie tranzytywnego domknięcia dla szerszego spektrum relacji zależności, w porównaniu z zastosowaniem każdego z owych podejść z osobna.

- W przypadku, gdy niemożliwe jest obliczenie dokładnego tranzytywnego domknięcia sparametryzowanej relacji zależności, algorytm 4.3 dokonuje również aproksymacji poszukiwanego rozwiązania w postaci przybliżenia górnego bądź dolnego.
- Zaproponowane w niniejszej pracy algorytmy umożliwiają obliczenie dokładnego tranzytywnego domknięcia sparametryzowanych relacji zależności z ograniczeniami afinicznymi, dla większego spektrum pętli programowych w porównaniu z aktualnie opracowanymi algorytmami.
- W oparciu o zaproponowane algorytmy możliwe jest opracowanie kompilatorów automatycznie generujących kod równoległy, którego czas wykonania za pomocą wielu procesorów jest krótszy niż jego sekwencyjny odpowiednik.
- Zaproponowane algorytmy znajdują zastosowanie w transformacjach pętli programowych, dokonujących ekstrakcji drobno- i gruboziarnistej równoległości i tym samym pozwalają po części na udzielenie odpowiedzi na szereg następujących pytań:
  (i) Czy możliwe jest zwiększenie ekstrakcji równoległości w porównaniu z aktualnie znanymi metodami i jakie koszty się z tym wiążą ?
  (ii) W jakim stopniu przybliżenie górne tranzytywnego domknięcia sparametryzowanej relacji zależności, obliczone z powodu braku możliwości uzyskania wyniku dokładnego, umożliwia osiągnięcie zadowalającego przyspieszenia obliczeń równoległych ?
  (iii) Jaki odsetek równoległości został pominięty w wyniku zastosowania obecnie znanych transformacji ?

8.2 Dalsze badania

W trakcie badań napotkano na wiele ograniczeń i problemów, których rozwiązanie umożliwiłoby zwiększenie i tak już szerokiego zakresu stosowalności proponowanych rozwiązań:

- Niezbędnym jest rozszerzenie możliwości istniejących bibliotek o mechanizmy pozwalające na implementację algorytmów obliczania tranzytywnego domknięcia sparametryzowanych relacji zależności w przypadku, gdy niemożliwe jest opisanie go przy pomocy afinicznych ograniczeń.
- Rozszerzenie możliwości istniejących narzędzi o nowe algorytmy w zakresie upraszczania ograniczeń zawartych w opisie relacji i zbiorów.
- Opracowanie algorytmów umożliwiających obliczenie dokładnego tranzytywnego domknięcia złożonych sparametryzowanych relacji o przerwanym łańcuchu zależności.

Powyższymi zagadnieniami autor pragnie zająć się w swoich przyszłych badaniach.

Załącznik A

SPECYFIKACJA WYKORZYSTANEGO SPRZĘTU DO BADAŃ EKSPERYMENTALNYCH

Badania eksperymentalne przeprowadzono z wykorzystaniem komputera **HP Compaq 6710b, Intel (R) Core (TM) 2 Duo**. Charakterystyka maszyny badawczej została zamieszczona poniżej.

HP Compaq 6710b

Procesor	Intel (R) Core (TM) 2 Duo, T7300, 2. 00 Ghz
Liczba procesorów	1 dwurdzeniowy
Pamięć operacyjna	2 GB
Pojemność dyskowa	80 GB
System operacyjny	Fedora Linux
Kompilator	gcc 4.3.0 20080428

Załącznik B

PŁYTA CD

Na dołączonej płycie CD w zależności od katalogu, umieszczono następujące dodatki:

- *Omega Library v. 2.1.0* – implementacja algorytmów, w ramach zmodyfikowanej wersji biblioteki - źródła i dokumentacja.
- Experiments – wyniki badań:
  - zestawienie wszystkich wyników ilościowych i czasowych dla badanych pętli z rozdziału 6,
  - zródła pętli zestawów NAS i UTDSP oraz ich format w języku wejściowym narzędzia Petit,
- Tools – narzędzia pomocnicze: oryginalne źródła biblioteka Omega Calculator i Petit w wersji 2.1.

Publikacje własne

[1] Bielecki W., Klimek T., Trifunovič K., Obliczenie potęgi k znormalizowanej afinicznej relacji. Metody Informatyki Stosowanej Nr 2/2008 (Tom 15)

[2] Bielecki W., Klimek T., Trifunovič K., Calculating Exact Transitive Closure for a Normalized Affine Integer Tuple Relation. Electronic Notes in Discrete Mathematics 33 (2009), pages 7–14.

[3] Bielecki W., Klimek T., Pietrasik M., Obliczenie tranzytywnego domknięcia sparametryzowanych relacji zależności nie należących do klasy d-form. Metody Informatyki Stosowanej Nr 2/2009 (19)

[4] Bielecki W., Klimek T., Pietrasik M., An experimental study on recognizing classes of dependence relations. Wydawnictwo Pomiary Automatyka i Kontrola Nr 10/2009 vol 55.

[5] Bielecki W., Klimek T. Rola operacji tranzytywnego domknięcia w przetwarzaniu równoległym i rozproszonym. Stargardzkie Zeszyty Naukowe

[6] Bielecki W., Pałkowski M., Klimek T., Wyznaczenie punktów reprezentatywnych niezależnych fragmentów kodu w grafie zależności pętli programowych. Metody Informatyki Stosowanej, Nr 1/2010 (22)

[7] Bielecki W., Klimek T., Pałkowski M. Beletska A. An iterative algorithm of computing the transitive closure of a union of parameterized affine integer tuple relations. In COCOA 2010, volume 6508 of Lecture Notes in Computer Science, pages 104-113. Springer, 2010.

[8] Bielecki W., Klimek T., Zwiększenie zbieżności iteracyjnego algorytmu obliczania tranzytywnego domknięcia unii sparametryzowanych relacji zależności. Metody Informatyki Stosowanej Nr 3/2010

Bibliografia

[1] Abdali S. K. and Saunders B. D. Transitive closure and related semiring properties via eliminants. Theoretical Computer Science, 40(2,3), pages 257-274, 1985.

[2] Agrawal R. Alpha : An extension of relational algebra to express a class of recursive queries. In Proceedings of the IEEE 3rd International Conference on Data Engineering, pages 879-885, Los Angeles, CA, February 1987.

[3] Agrawal R. and Jagadish H. V. Direct algorithms for computing the transitive closure of database relations. In Proceedings of the 13th International VLDB Conference, 1987.

[4] Agrawal R. and Jagadish H. V. Hybrid transitive closure algorithms. In Proceedings of the 16h International VLDB Conference, pages 326-334, Brisbane, Australia, 1990.

[5] Agrawal R., Dar S. and Jagadish H. V. Direct transitive closure algorithms: Design and performance evaluation. ACM Transactions on Database Systems, 15(3), pages 427-458, September 1990.

[6] Allen R., Kennedy K. Optimizing compilers for modern architectures: A Dependence based Approach. Morgan Kaufmann Publishers, Inc., 2001.

[7] Bailey D., Barszcz E., Barton J., Browning R., Carter R., Dagum L., Fatoohi R., Fineberg S., Frederickson P., Lasinski T., Schreiber R., Simon H., Venkatakrishnan V., Weeratunga S. The NAS

[8] Barthou D., Feautrier P., Redon X. On the equivalence of two systems of affine recurrence equations. In: Monien, Feldmann B. R. L. (eds.) Euro-Par 2002. LNCS, vol 2400, pages 309-313. Springer, Heidelberg (2002)

[9] Beletska A., Barthou D., Bielecki W., Cohen A. Computing the Transitive Closure of a Union of Affine Integer Tuple Relations. In COCOA 2009, volume 5573 of Lecture Notes in Computer Science, pages 98-109. Springer, 2009.

[10] Beletska A., Bielecki W., Siedlecki K., San Pietro P. Finding Synchronization-Free Slices of Operations in Arbitrarily Nested Loops. In Proceedings of ICCSA'2008, LNCS.

[11] Beletska A., San Pietro P. Extracting Coarse-Grained Parallelism with the Affine Transformation Framework and its Limitations, Electronic Modelling, no. 5, 2006.

[12] Bielecki W., Beletska A., Pałkowski M., San Pietro P. Extracting synchronization-free chains of dependent iterations in non-uniform loops, Advanced Computer Systems, 14th International Multi-Conference, Polish Journal of Environmental Studies, Vol. 16 No. 5B, Międzyzdroje 2007, pages 165-171.

[13] Bielecki W., Beletska A., Pałkowski M., San Pietro P. Extracting synchronization-free trees composed of non-uniform loop operations. Algorithms and Architectures for Parallel Processing, Lecture Notes in Computer Science Volume 5022/2008, Springer Berlin / Heidelberg, 2008, s. 185-195.

[14] Bielecki W., Beletska A., San Pietro P. Extracting Coarse Grained Parallelism in Program Loops with the Slicing Framework, In Proceedings of ISPDC, IEEE, 2007.

[15] Bielecki W., Drążkowski R. Approach to building free schedules for loops with affine dependences represented with a single dependence relation. WSEAS Transactions on Computers, Issue 11, Volume 4, 2005

[16] Bielecki W., Klimek T. Rola operacji tranzytywnego domknięcia w przetwarzaniu równoległym i rozproszonym. Stargardzkie Zeszyty Naukowe

[17] Bielecki W., Klimek T. Zwiększenie zbieżności iteracyjnego algorytmu obliczania tranzytywnego domknięcia unii sparametryzowanych relacji zależności. Metody Informatyki Stosowanej

[18] Bielecki W., Klimek T., Pałkowski M., Beletska A. An iterative algorithm of computing the transitive closure of a union of parameterized affine integer tuple relations. In COCOA 2010, volume 6508 of Lecture Notes in Computer Science, pages 104-113. Springer, 2010.

[19] Bielecki W., Klimek T., Pietrasik M. An experimental study on recognizing classes of dependence relations. Wydawnictwo Pomiary Automatyka i Kontrola Nr 10/2009 vol 55

[20] Bielecki W., Klimek T., Pietrasik M. Obliczenie tranzytywnego domknięcia sparametryzowanych relacji zależności nie należących do klasy d-form. Metody Informatyki Stosowanej Nr 2/2009 (19)

[21] Bielecki W., Klimek T., Trifunovič K. Calculating Exact Transitive Closure for a Normalized Affine Integer Tuple Relation. Electronic Notes in Discrete Mathematics 33 (2009), pages 7–14.

[22] Bielecki W., Klimek T., Trifunovič K. Obliczenie potęgi k znormalizowanej afinicznej relacji. Metody Informatyki Stosowanej Nr 2/2008 (Tom 15)

[23] Bielecki W., Pałkowski M. Extracting coarse-grained parallelism in computer simulation applications, ACS 2006, 13th International Multi-Conference, Vol. II, Szczecin 2006, s. 237-246.

[24] Bielecki W., Pałkowski M., Klimek T. Wyznaczenie punktów reprezentatywnych niezależnych fragmentów kodu w grafie zależności pętli programowych. Metody Informatyki Stosowanej, Nr 1/2010 (22)

[25] Bielecki W., Pałkowski M., Sierbin M. Zastosowanie bibliotek Omega i ISL do ekstrakcji niezależnych fragmentów kodu w pętlach programowych. Metody Informatyki Stosowanej

[26] Bielecki W., Siedlecki K. Extracting synchronization-free threads in perfectly nested loops using the Omega project software. WSEAS SEPADS, SALZBURG 2005.

[27] Bielecki W., Siedlecki K. Finding Free Schedules for Non-Uniform Loops. Electronic Modeling 2004.

[28] Bielecki W., Siedlecki K. Wyszukiwanie początków niezależnych wątków w dowolnie zagnieżdżonych pętlach programowych. PS, M IX SNI, 2004.

[29] Bielecki W., Siedlecki K. Wyznaczanie niezależnych wątków w pętlach idealnie zagnieżdżonych. PS, M VII SNI, 2002

[30] Boigelot B. Symbolic Methods for Exploring Infinite State Spaces. PhD thesis, Université de Liège (1998)

[31] Boigelot B., Wolper P. Symbolic verification with periodic sets. In: Dill, D.L. (ed,) CAV 1994. LNCS vol. 818, pages 55-67. Springer, Heidelberg (1994)

[32] Bozga M., Girlea C., Iosif R. Iterating octagons. In TACAS' 09, pages 337-351. Springer, 2009. 1,1.0.1,4,4.2,4.2.1, 5

[33] Bozga M., Iosif R., and and Yassine Lakhnech. Flat parametric counter automata. In ICALP, LNCS 4052, pages 577-588. Springer-Verlag, 2006

[34] Chartrand G. Introductory Graph Theory, Dover 1985

[35] Comon H., Jurski Y. Multiple counters automata, safety analysis and presburger arithmetic. In: Vardi, M. Y. (ed.) CAV 1998. LNCS, vol 1427, pages 268-279. Springer, Heidelberg (1998)

[36] Cormen T., Leiserson C. E., Rivest R. Wprowadzenie do algorytmów. Wydawnictwo Naukowo-Techniczne, Warszawa 1997

[37] Cruz I. F. and T. S. Norvell. Aggregative closure: An extension of transitive closure. In Proceedings of the IEEE 5th International Conference on Data Engineering, pages 84-391, Los Angeles, California, February 1989.

[38] Dar S. and Jagadish H. V. A spanning tree transitive closure algorithm. In Proceedings of the IEEE 8th International Conference on Data Engineering, pages 2-11, Tempe, Arizona, February 1992.

[39] Dar S. and Ramakrishnan R. A performance study of transitive closure algorithms. In Proceedings of the ACM-SIGMOD 1994 Conference on Management of Data, 1994.

[40] Darte A., Robert Y., Vivien F. Scheduling And Automatic Parallelization. Birkhaüser (2000)

[41] Deo N. Teoria grafów i jej zastosowania w technice i informatyce. Wydawnictwo PWN, Warszawa 1980

[42] Diestel R. Graph Theory. Electronic Edition. Springer-Verlag Heidelberg, New York 2005

[43] Ding-Kai Chen. Compiler optimizations for parallel loops with fine-grained synchronization. PhD thesis University of Illinois at Urbana-Champaign, 1994

[44] Dolan A. K., Aldous J. Networks and Algorithms: An Introductory Approach, Wiley-Interscience, 1993

[45] Drozdek A., Simon Donald L. Struktury danych w języku C. Wydawnictwo Naukowo Techniczne, Warszawa 1996

[46] Ebert J. A sensitive transitive closure algorithm. Information Processing Letters, 12, pages 255-258, 1981.

[47] El-Rewini H., Abd-El-Barr M. Advanced computer architecture and parallel processing. A John Wiley & Sons, Inc publication, 2005.

[48] Eve J. and Kurki-Suonio R. On computing the transitive closure of a relation. Acta Informatica, 8, pages 303-314, 1977.

[49] Feautrier P. Some efficient solutions to the affine scheduling problem, part I, One Dimensional Time. International Journal of Parallel Programming 21. 1992, pages 313-348.

[50] Feautrier P. Some efficient solutions to the affine scheduling problem, part II, Multidimensional Time. International Journal of Parallel Programming 21. 1992, pages 389- 420.

[51] Fischer M. J. and Meyer A. R. Boolean matrix multiplication and transitive closure. In Conference Record 1971 12th Annual Symposium on Switching and Automata Theory, pages 129-131, East Lansing, Michigan, October 1971. IEEE Computer Society.

[52] Ganguly S., Krishnamurthy R. and Silberschatz A. An analysis technique for transitive closure algorithms: A statistical approach. In Proceedings of the IEEE

7th International Conference on Data Engineering, pages 728-735, Kobe, Japan, April 1991.

[53] Goralcikova A., and Koubek V. A reduct and closure algorithm for graphs. In Mathematical Foundations of Computer Science, volume 74 of Lecture Notes in Computer Science, pages 301-307. Springer-Verlag, 1979.

[54] Griebl M. Automatic Parallelization of Loop Programs for Distributed Memory Architectures. Habilitation thesis, Universität Passau, Fakultät für Mathematik und Informatik, 2004

[55] Guh K. C. and Yu C. T. Evaluation of transitive closure in distributed database systems. IEEE Journal on Selected Areas in Communications, 7(3), pages 399-407, April 1989.

[56] Hua K. A., Su J. X. W. and Hua C. M. Efficient evaluation of traversal recursive queries. In Proceedings of the IEEE 9th International Conference on Data Engineering, pages 549-558, Vienna, Austria, April 1993

[57] Hua K. and Hannenhalli G. Parallel transitive closure computations using topological sort. In Proceedings of the 1st International Conference on Parallel and Distributed Systems, pages 122-129, Miami Beach, Florida, December 1991.

[58] Ioannidis Y. E. and Ramakrishnan R. Efficient transitive closure algorithms. In Proceedings of the 14th VLDB Conference, pages 335-346, Los Angeles, California, 1988.

[59] Ioannidis Y. E., Ramakrishnan R. and Winger L. Transitive closure algorithms based on graph traversal. ACM Transactions on Database Systems, 18(3), pages 512-576, September 1993

[60] Ioannidis Y. E., Ramakrishnan R. and Winger L. Transitive closure algorithms based on graph traversal. Technical report, Computer Science Department, University of Wisconsin, Madison, WI, 1991.

[61] Italiano G. Finding paths and deleting edges in directed acyclic graphs. Information Processing Letters, 28(1), pages 5-11, 1988.

[62] Jagadish H. V., Agrawal R. and Ness L. A study of transitive closure as a recursion mechanism. In Proceedings of the ACM-SIGMOD 1987 Conference on Management of Data, pages 331-344, San Francisco, California, 1987.

[63] Jakobson H. Mixed approach algorithms for transitive closure. In Proceedings of the 10th ACM Symposium on Principles of Database Systems, pages 199-205, 1991.

[64] Jakobson H. On tree based techniques for query evaluation. In Proceedings of the 11th ACM Symposium on Principles of Database Systems, pages 380-392, San Diego, California, 1992.

[65] Jaumard B., and Minoux M. An efficient algorithm for the transitive closure and a linear worst-case complexity result for a class of sparse graphs. Information Processing Letters, 22, pages 163-169, 1986

[66] Jiang B. Traversing graphs in paging environment, BFS or DFS? Information Processing Letters, 37(3), pages 143-147, 1991.

[67] Jin H., Frumkin M., Yan J. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. October 1999. NAS Technical Report NAS-99-011.

[68] Jun M. and Takaoka T. An O(n2) parallel algorithm to compute the all pairs shortest paths and transitive closure. Journal of Information Processing, 12(2), pages 119-124, 1989.

[69] Kabler R, Ioannidis Y. E. and Carey M. Performance evaluation of algorithms for transitive closure. Information Systems, 17(5), pages 415-441, September 1992.

[70] Kelly W., Maslov V., Pugh W., Rosser E., Shpeisman T., Wonnacott D. New User Interface for Petit and other Extensions. Technical Report, Dept. of Computer Science, University of Maryland, College Park, December 1996

[71] Kelly W., Maslov V., Pugh W., Rosser E., Shpeisman T., Wonnacott D. The Omega Library, Version 1.1.0, Interface Guide. November 18, 1996.

[72] Kelly W., Pugh W., Rosser E., Shpeisman T. Transitive Closure of Infinite Graphs and its Applications. International Journal of Parallel Programming. 1996, Volume 24, 6, pages 579-598

[73] La Poutre J. A. and van Leeuwen J. Maintenance of transitive closures and transitive reductions of graphs. In H. Gottler and H. J. Schneider, editors, Graph-Theoretic Concepts in Computer Science, volume 314 of Lecture Notes in Computer Science, pages 106-120, Berlin, 1988, Springer-Verlag

[74] Lehmann D. J. Algebraic structures for transitive closure. Theoretical Computer Science, 4(1), pages 59-76, 1977.

[75] Lilja D. J. A Multiprocessor Architecture Combining Fine-Grained and Coarse-Grained Parallelism Strategies, Parallel Computing, Vol. 20, No.5, 1994, pages 729-751.

[76] Lim A. W., Cheong G. I., Lam M. S. An affine partitioning algorithm to maximize parallelism and minimize communication. In ICS'99, pages 228-237. ACM Press, 1999.

[77] Lim A. W., Lam M. S. Communication-free parallelization via affine transformations. Proceedings of the Seventh Workshop on Languages and Compilers for Parallel Computing. 1994, pages 92-106.

[78] Lim A. W., Lam M. S. Maximizing parallelism and minimizing synchronization with affine transforms. Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1997.

[79] Lu H. New strategies for computing the transitive closure of a database relation. In Proceedings of the 13th International VLDB Conference, pages 267-274, 1987.

[80] Lu H., Mikkilineni K. and Richardson J. Design and evaluation of algorithms to compute the transitive closure of a database relation. In Proceedings of the 3rd International Conference on Data Engineering, pages 112-119, Los Angeles, California, February 1987.

[81] Maslov V., Pugh W. Simplifying Polynomial Constraints Over Integers to Make Dependence Analysis More Precise, Lecture Notes in Computer Science, 1994, Volume 854/1994, pages 737-748

[82] McHugh J. A. Algorithmic Graph Theory. Prentice-Hall, Englewood Cli s, N.J., 1990.

[83] Nuutila E. Efficient Transitive Closure Computation in Large Digraphs. Mathematics and Computing in Engineering Series No. 74 PhD thesis Helsinki University of Technology. Helsinki 1995.

[84] Pałkowski M. Algorytmy zwiększające ekstrakcję równoległości w pętlach programowych - praca doktorska. Szczecin, Wydział Informatyki, Politechnika Szczecińska, 2008

[85] Pałkowski M. Upraszczanie relacji zależności i zbiorów iteracji w celu podniesienia wydajności generowanego kodu równoległego. Publikacja planowana do Kwartalnika Politechniki Szczecińskiej, 2008.

[86] Peng, Sean Hsienen. UTDSP: A VLIW Programmable DSP Processor. Department of Electrical and Computer Engineering, University of Toronto, 1999.

[87] Pugh W., Rosser E. Iteration Space Slicing and Its Application to Communication Optimization. Proceedings of the International Conference on Supercomputing. 1997, 221-228.

[88] Pugh W. and Wonnacott D. An exact method for analysis of value-based array data dependences. In In Sixth Annual Workshop on Programming Languages and Compilers for Parallel Computing. Springer-Verlag, 1993.

[89] Pugh W. Counting solutions to Presburger formulas: how and why. ACM SIGPLAN Notices Volume 29, Issue 6, June 1994

[90] Pugh W. Definitions of dependence distance. ACM Letters on Programming Languages and Systems (LOPLAS), September 1992, Volume 1 , Issue 3, pages 261-265

[91] Pugh W. The Omega test: a fast and practical integer programming algorithm for dependence analysis. Proceedings of the 1991 ACM/IEEE conference on Supercomputing, Albuquerque, New Mexico, United States, pages 4-13

[92] Purdom P. A transitive closure algorithm. BIT, 10, pages 76-94, 1970

[93] Reddy C. R., Loveland D. W. Presburger Arithmetic with Bounded Quantifier Alternation. Proceedings of the tenth annual ACM symposium on Theory of computing, San Diego, 1978

[94] Rosenthal A., Heiler S., Dayal U., and Manola F. Traversal recursion: a practical approach to supporting recursive applications. In Proceedings of the ACM-SIGMOD 1986 International Conference on Management of Data, pages 166-176, Washington D.C, May 1986.

[95] Ross K. A., Wright. Ch. R. B. Matematyka Dyskretna. Wydawnictwo Naukowe PWN, Warszawa 1996

[96] Schmitz L. An improved transitive closure algorithm. Computing, 30, pages 359-371, 1983.

[97] Schrijver A. Theory of Linear and Integer Programming. John Wiley & Sons. 1986

[98] Shashidhar K. C., Bruynooghe K. M, Catthoor F., Janssens G. An automatic verification technique for loop and data reuse transformations based on geometric modeling of programs. Journal of Universal Computer Science 9(3), pages 248-269 (2003)

[99] Siedlecki K. Algorytmy wyszukiwania drobno- i gruboziarnistej równoległości w pętlach programowych z zależnościami afinicznymi - praca doktorska. Szczecin, Wydział Informatyki, Politechnika Szczecińska, 2008.

[100] Simon K. An improved algorithm for transitive closure on acyclic digraphs. Theoretical Computer Science, 58(1-3), pages 325-346, 1988

[101] Sippu S. and Soisalon-Soininen S. A generalized transitive closure for relational queries. In Proceedings of the 7th ACM Symposium on Principles of Database Systems, pages 325-332, 1988.

[102] Tarjan R. E. A unified approach to path problems. Journal of the ACM, 28(3), pages 577-593, 1981

[103] Tarjan R. E. Depth first search and linear graph algorithms. SIAM Journal of Computing, 1(2), pages 146-160, June 1972.

[104] Tucker A. Applied Combinatorics, wyd. 2, Wiley, 1984

[105] Van Leeuwen J. Graph Algorithms. Transitive reduction and transitive closure. In Jan Van Leeuwen, editor, Handbook of Theoretical Computer Science, volume A: Algorithms and Complexity, chapter 10, section 1.4, pages 539-544. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.

[106] Van Leeuwen J. On efficiently computing the product of two binary relations. International Journal of Computer Mathematics, 5, pages 193-201, 1976

[107] Verdoolaege S. Integer Set Library: Manual, www.kotnet.org/~skimo/isl/manual.pdf, 2010

[108] Waheed A., Yan J. Parallelization of NAS Benchmarks for Shared Memory Multiprocessors. March 1998. NAS Technical Report NAS-98-010.

[109] Warren H. S. A modifcation of Warshall's algorithm for the transitive closure of binary relations. Communications of the ACM, 18(4), pages 218-220, 1975.

[110] Wilson R. J. Wprowadzenie do teorii grafów. Wydawnictwo Naukowe PWN, Warszawa 2008

[111] Wilson R. J., Beineke L. W. Applications of Graph Theory. Academic Press, 1979

[112] http://www.wolfram.com/

[113] http://mathworld.wolfram.com/RecurrenceEquation.html

[114] http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html

[115] http://www.nas.nasa.gov/Software/NPB

[116] http://www.netlib.org/benchmark/livermore

[117] http://www.maplesoft.com

[118] http://maxima.sourceforge.net

[119] http://www.mupad.com

[120] http://www.sfs.zut.edu.pl

[121] http://www.sfs.zut.edu.pl/files/omega3.tar.gz

[122] http://www-verimag.imag.fr/~async/flata/flata.html