PRS

Process Rewrite Systems

czyli

jak uogólnić sieci Petriego

i automaty stosowe

Bisymulacja

- Jedna z definicji równoważności automatów/ gramatyk
- Silniejsza niż równość języków generowanych (może być równość języków przy braku równości bisymulacyjnej)
- Bierze się z semantyki, nie syntaktyki – sprawdza równość z punktu widzenia obserwatora

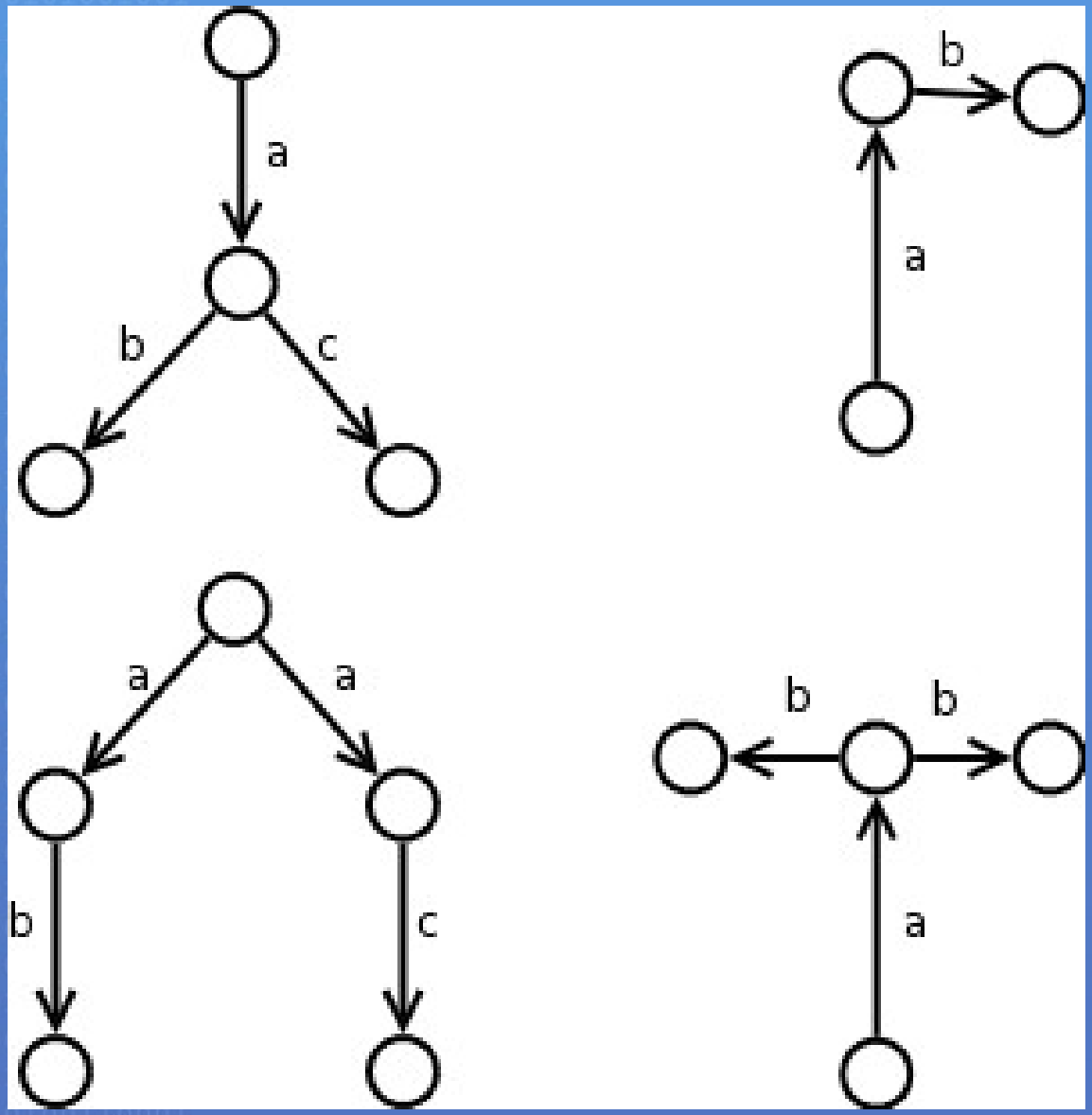
Bisymulacja

- Bisymulacja jest relacją między stanami jednego i drugiego automatu
- Taka relacja jest bisymulacją, jeśli istnieje strategia wygrywająca dla drugiego gracza w następującej grze:
  - Mamy dwa automaty z etykietowanymi przejściami, ustawionych na stany będące w relacji
  - Pierwszy gracz wybiera jeden z automatów i ruch

Bisymulacja c.d.

- Drugi gracz musi w drugim automacie wykonać ruch o takiej samej etykiecie, który kończy się w stanie będącym w relacji z tym na którym kończył się ruch pierwszego gracza
- Pierwszy gracz wygrywa jeśli może doprowadzić do stanu w którym drugi gracz nie może wykonać ruchu
- Drugi gracz wygrywa jeśli pierwszy gracz nie może wykonać ruchu do stanu, którego jeszcze nie odwiedził

Przykłady



Automat ze stosem (Pushdown Automaton - PDA)

- Skończony zbiór stanów P
- Skończony zbiór symboli stosowych S
- Skończony alfabet A
- Skończony zbiór przejść
- Stan początkowy a_0
- Akceptujemy pustym stosem
- Może być deterministyczny (DPDA), lub nie

Co można

- Rozpoznawać języki bezkontekstowe (PDA) lub deterministyczne języki bezkontekstowe (DPDA)
- Sprawdzać warunek stopu (czyli istnienie ścieżki w odpowiednim grafie przejść)
- Sprawdzać równość języków generowanych przez DPDA (wynik z 1997)
- Sprawdzać równoważność bisymulacyjną

Czego nie można

- Sprawdzać równości języków dla ogólnych PDA
- Sprawdzać zawierania języków
- Sprawdzić pełności języka
- Znaleźć przecięcia języków

Bisymulacja a równość języków

- W przypadku automatów (gramatyk) deterministycznych równoważność bisymulacyjna jest tym samym co równość języków generowanych przez automat
- Bisymulacja jest właściwą definicją równości automatów dla przypadków niedeterministycznych

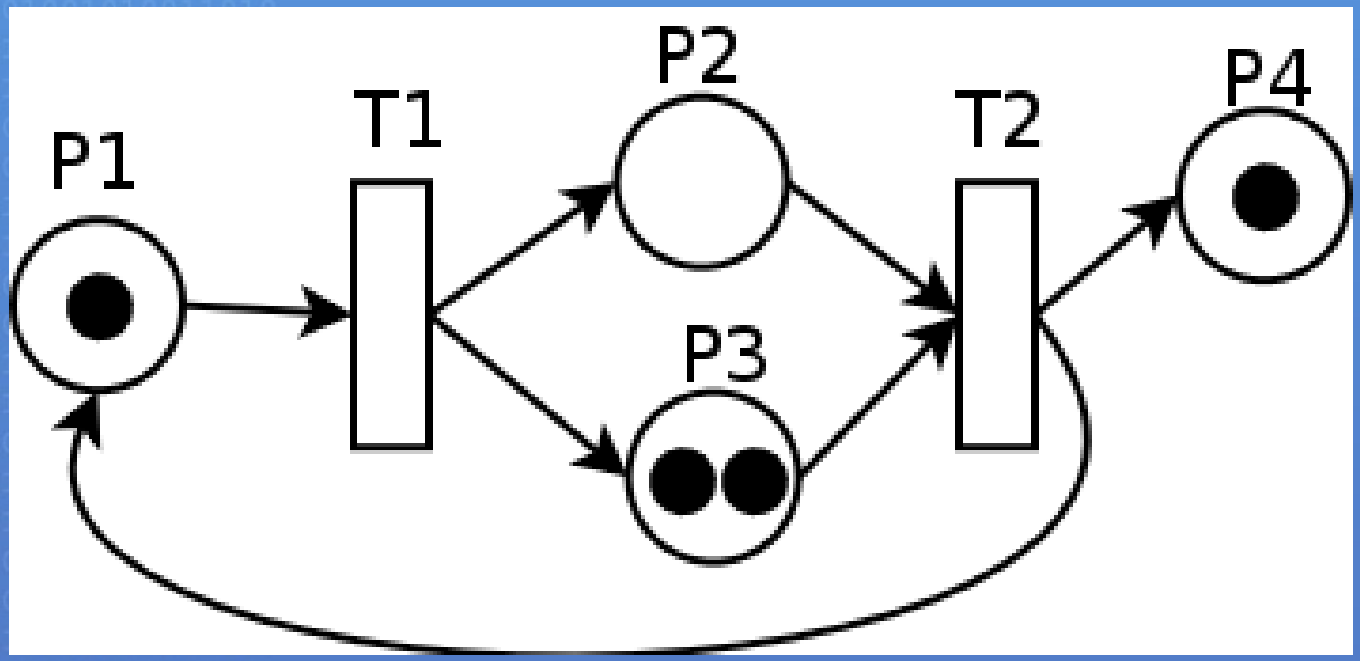
Notacja

- Stan automatu ze stosem możemy przedstawiać jako:
  - aX, gdzie a to jest stan w którym się znajdujemy, a X jest aktualnym stosem
- Przejścia możemy przedstawiać jako:
  - a.x -b-> c.Y, gdzie a j.w., x jest aktualnym wierzchołkiem stosu, b jest wczytanym symbolem (być może ε), c jest nowym stanem, a Y jest nowym wierzchołkiem stosu (nie koniecznie pojedynczym znakiem)
    - „.” jest łączna

Sieci Petriego

- Skończony zbiór stanów P
- Skończony zbiór zdarzeń S
- Skończony zbiór przejść między stanami, a tranzycjami lub vice versa A

Przykład



Co można

- Sprawdzić warunek stopu (osiągalności)
- Sprawdzić czy każde obliczenie sieci jest skończone

Czego nie można

- Sprawdzić równoważności w sensie bisymulacji
- Sprawdzić równości języków
- Sprawdzić zawierania języków

Notacja

- Przejścia możemy zapisywać jako:
  - X|X|Y -a-> A|B|A – gdzie każda stała odpowiada znacznikowi w odpowiadającym jej stanie
    - „|” jest łączne i przemienne
    - „a” jest oznaczeniem przejścia

Ciekawostka

- Jeśli w PDA z lewej strony przejścia pozwolimy na występowanie więcej niż dwóch symboli, to nie zwiększy się moc automatu.
  - Możemy kodować ciągi symboli nowymi znakami dorzucanymi do alfabetu stosowego
- Jeśli coś analogicznego zrobimy dla prostych sieci Petriego, to dostaniemy coś istotnie innego – MSA (multiset automaton)

Uogólnienie

- Będziemy używać zarówno „.” jak i „|”.
- Wprowadzimy klasy wyrażeń:
  - 1 – wyłącznie stałe (X, Y,...)
  - P (parallel) – termy z „|” (np. X|(Y|Z))
  - S (sequential) – termy z „.” (np. X.(Y.Z))
  - G (general) – termy z „|” lub „.” (np. X.(Y|Z))
- X,Y,... będą oznaczać stałe, a t_0,t_1,... termy nad tymi stałymi

Przejścia

- Nasz uogólniony system ma skończenie wiele przejść postaci t_1 -a-> t_2
  - rodzinę przejść oznaczamy Δ
- Mamy własności:
  - jeśli (t_1 -a-> t_1'), to t_1|t_2 -a-> t_1'|t_2
  - jeśli (t_1 -a-> t_1'), to t_1.t_2 -a-> t_1'.t_2

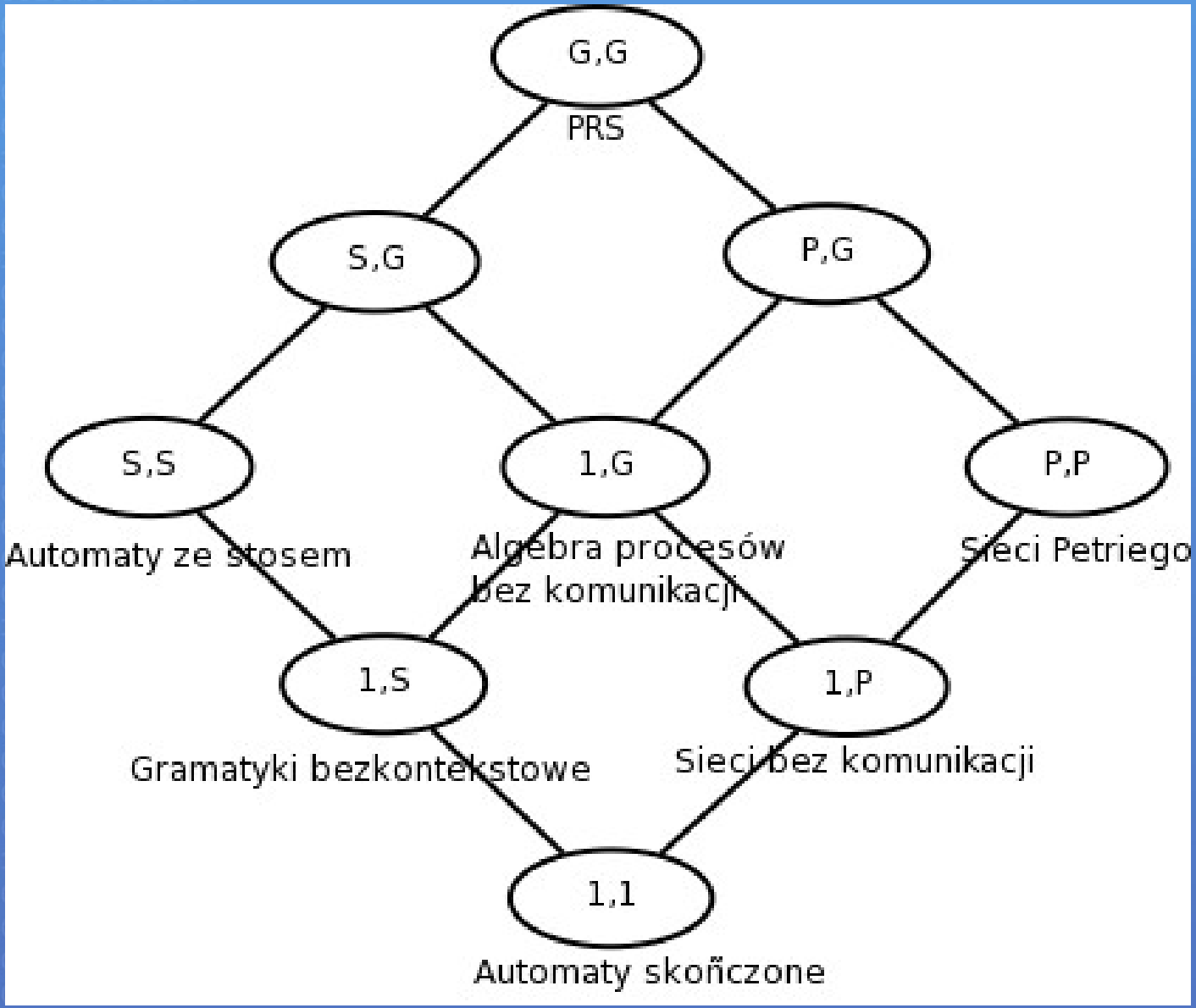
Klasy

- Możemy rozróżniać klasy po lewych i prawych stronach relacji $\Delta = \{(\alpha,\beta)\}$
- Jeśli α jest bardziej ogólne, lub nieporównywalne z β, to nasza relacja nie ma sensu
- W związku z tym możemy wyróżnić istotne klasy: (1,1), (1,S), (1,P), (1,G), (S,S), (P,P), (S,G), (P,G), (G,G)
- (G,G) nazywamy ogólnym PRS

Hierarchia

- Łatwo widać, że (1,1), to nic innego jak automaty skończone
- (1,S) odpowiada gramatykom bezkontekstowym w postaci normalnej Greibacha
- (S,S) zawiera automaty ze stosem, jeśli zastosujemy wprowadzoną notację
- (P,P) przy wprowadzonej notacji, to sieci Petriego

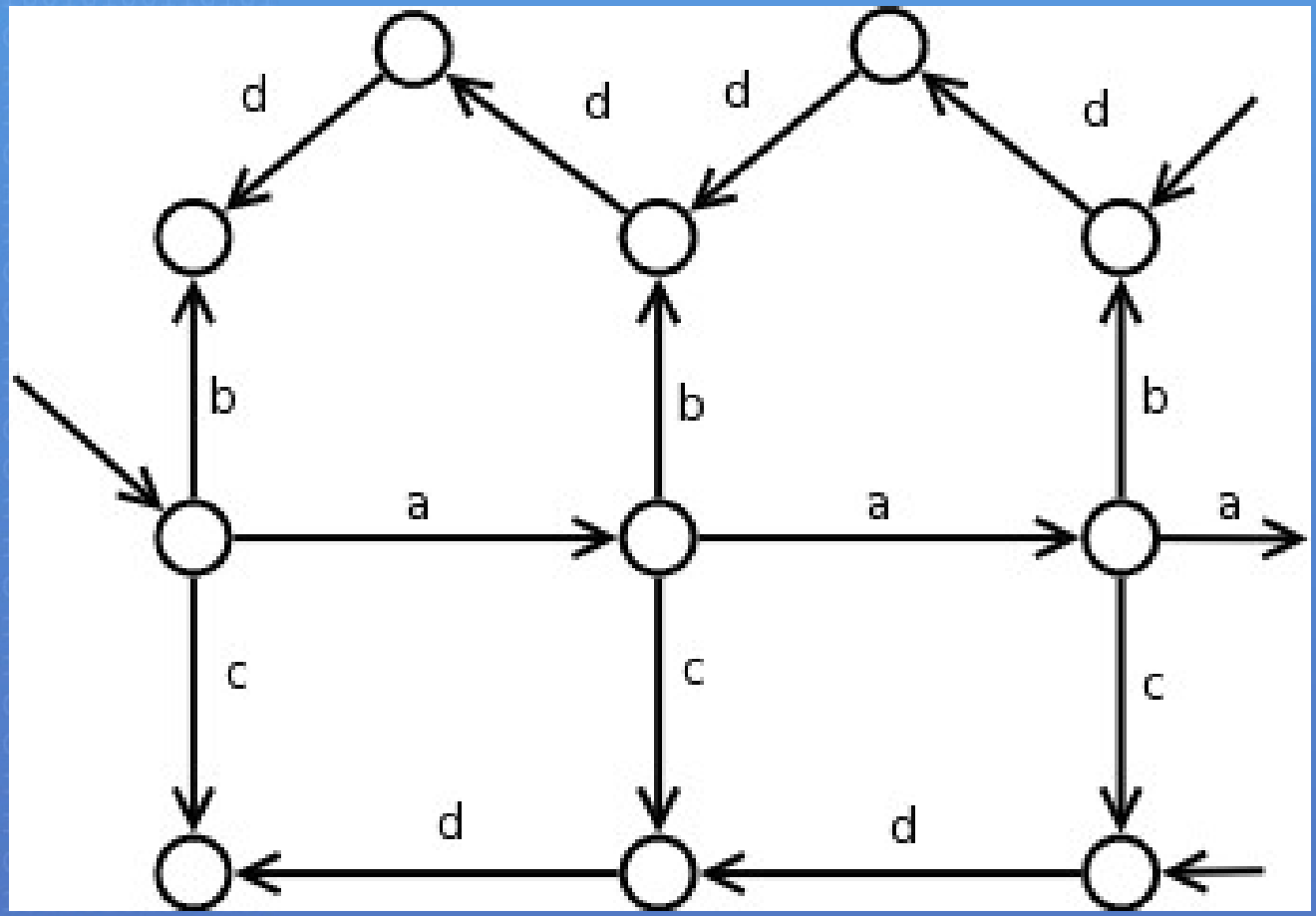
Hierarchia na rysunku



Ścisłość heirarchii

- Wiadomo, że z punktu widzenia równości języków ścisłości nie ma, bo (S,S) [PDA] i (1,S) [gramatyki bezkontekstowe] są tym samym
- Jeśli bierzemy pod uwagę równoważność bisymulacyjną ścisłość hierarchii jest i potrafimy odróżnić np. (S,S) od (1,S)

BPA != PDA



Osiągalność dla PRS

- Dla automatów ze stosem jak i sieci Petriego można sprawdzić czy ze stanu A da się dojść do stanu B
- Okazuje się, że w przypadku ogólnego PRS, ten problem także jest rozstrzygalny, a dowodzi się to redukując problem w przypadku ogólnym do przypadków szczególnych

Szkic dowodu

- Definiujemy postać normalną PRS jako taką, gdzie w przejściach występuje złożenie co najwyżej dwóch stałych i to co najwyżej po jednej stronie
  - X|Y -> Z, X -> Y|Z, X -> Y
  - X.Y -> Z, X -> Y.Z, X -> Y

Norma

- wielkością termu nazywamy:
  - s(X) = 1
  - s(t_1 | t_2) = s(t_1) + s(t_2) + 1
  - s(t_1.t_2) = s(t_1) + s(t_2) + 1
- normę przejścia definiujemy:
  - d(t_1 -> t_2) = size(t_1) + size(t_2)
- przez k_i oznaczamy liczbę przejść wielkości i nie będących w postaci normalnej
- normą Δ nazywamy (k_1,k_2,...,k_n)

Każdy PRS można sprowadzić do postaci normalnej

- Jeśli norma $\Delta = 0$, to Δ jest w postaci normalnej
- Jeśli norma Δ != 0, to weźmy największe niezerowe k_i i przejście $\delta : n(\delta) = i$.
- Poprawmy δ, tak żeby miało mniejszą normę
- Jeśli zawsze potrafimy to zrobić, to w końcu wyzerujemy normę Δ, czyli sprowadzimy PRS do postaci normalnej

Normalizacja PRS

- Dla δ weźmy term t nie będący stałą
- Zamieńmy wszystkie wystąpienia t w Δ przez nową zmienną X
- Dodajmy dwa nowe przejścia: t -> X i X -> t
- Tym samym zmniejszyliśmy normę Δ, zwiększając liczbę przejść w Δ o dwa
- Nowe reguły trzeba stosować w dobrej kolejności, ale to nie przeszkadza w badaniu osiągalności stanu

Kluczowy lemat

- Jeśli Δ jest PRS w postaci normalnej i istnieją stałe X i Y, takie że Y jest osiągalny z X, ale w Δ nie ma przejścia X -> Y, to Δ zawiera stałe X' i Y', takie że Δ nie zawiera przejścia X' -> Y', ale Y' jest osiągalny z X' po ścieżce składającej się wyłącznie z przejść jednego rodzaju (równoległych lub sekwencyjnych)

Dowód lematu

- Wybieramy X' i Y', takie żeby przejście z X' do Y' było najkrótsze możliwe w Δ
- Pokażemy, że jest to przejście jednego typu
  - Załóżmy, że zawiera dwa różne rodzaje przejść i pokażmy, że potrafimy wskazać krótszą ścieżkę między jakimiś stałymi X'' i Y''

Jeden z przypadków

- Załóżmy, że ostatnim nietrywialnym przejściem (innej postaci niż A -> B) jest przejście równoległe.
- Wcześniej występuje jakieś przejście sekwencyjne X1 -> X2.X3
- X2 musi się kiedyś zakończyć
- Musi istnieć X4 takie, że X2.X3 -...-> X4
- Czyli mamy przejście X1 -...-> X4 krótsze niż nasze badane, co przeczy założeniu

Ostateczny algorytm

- Zrób Δ w postaci normalnej
- Dla każdej pary stałych X i Y, takich że Y jest osiągalne z X wyłącznie po przejściach sekwencyjnych (czyli tak jak w automacie stosowym), albo po przejściach równoległych (jak w sieciach Petriego) dodaj przejście z X do Y do Δ
- Powtarzaj aż nie będzie co poprawiać

Działanie

- W tak skonstruowanym Δ, żeby sprawdzić czy ze stanu A da się dojść do stanu B wystarczy zobaczyć, czy istnieje krawędź z A do B.
- Złożoność algorytmu jest rzędu n^4, gdzie n jest złożonością algorytmu sprawdzającego osiągalność dla sieci Petriego lub automatu stosowego