Poprawność algorytmów

Jeśli uważasz, że jakiś program komputerowy jest bezbłędny, to się mylisz
- po prostu nie zauważyłeś jeszcze skutków błędu, który jest w nim zawarty. ☹

Można popełnić:

- **Błędy językowe**

np. zamiast **for** $i := 1$ **to** N **do** $X[i] := i$;
napisano **for** $i := 1$ **do** N **do** $X[i] := i$;

Powstają w wyniku naruszenia składni języka (programowania), którego używamy do zapisania algorytmu.
Możliwe skutki i znaczenie:
◊ zatrzymanie kompilacji lub interpretacji z komunikatem lub bez,
◊ przerwanie realizacji programu nawet jeśli kompilator błędu nie wykrył,
◊ błędy nieprzyjemne, ale zwykle niezbyt poważne - są względnie łatwe do poprawienia.

- **Błędy semantyczne**

np. sądziliśmy, że po zakończeniu iteracji **for** $i := 1$ **to** N **do** $X[i] := i$ zmienna i ma wartość N, a nie $N + 1$
Wynikają z niezrozumienia semantyki używanego języka programowania.
Możliwe skutki i znaczenie:
◊ program nie realizuje poprawnie algorytmu,
◊ błędy trudne do przewidzenia i potencjalnie groźne, ale są do uniknięcia przy większej wiedzy i starannym sprawdzaniu znaczenia używanych instrukcji.

- **Błędy logiczne**

np. w algorytmie zliczania zdań, w których występuje słowo „algorytm" nie zauważyliśmy, że sekwencja znaków „.. " może występować także wewnątrz zdania („Na rys. 2 pokazano schemat..."), a używaliśmy jej do wyszukiwania jego końca.
Możliwe skutki i znaczenie:
◊ algorytm przestaje być poprawnym rozwiązaniem zadania algorytmicznego,
◊ dla pewnych zestawów danych wejściowych algorytm podaje wyniki niezgodne z oczekiwanymi,
◊ procesor może nie być w stanie wykonać pewnych instrukcji (np. żądamy dzielenia przez 0),
◊ błędy bardzo groźne - mogą być trudne do znalezienia i pozostawać długo w ukryciu nawet w trakcie używania programu w postaci kodu.

- **Błędy algorytmiczne**

wynikają z wadliwie skonstruowanych struktur sterujących np. niewłaściwych zakresów iteracji, niewłaściwych warunków użytych do zatrzymywania iteracji warunkowych lub przeniesienia sterowania w niewłaściwe miejsce procesu w wyniku zastosowania wyboru warunkowego (lub instrukcji skoku).
Możliwe skutki i znaczenie:
◊ algorytm dla pewnych dopuszczalnych danych wejściowych daje niepoprawny wynik,
◊ wykonanie programu realizującego algorytm jest przerywane w trybie awaryjnym,
◊ program realizujący algorytm nie kończy w normalnym trybie swego działania.

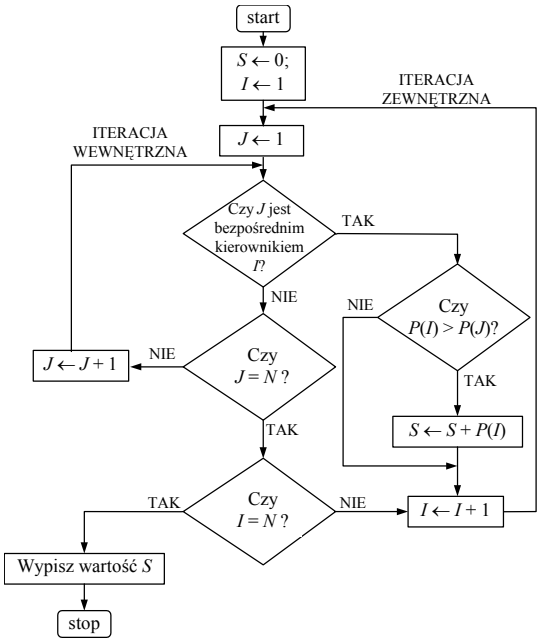
Rozmaitość źródeł błędów różnych typów

⇓

Złożone programy wymagają:
* testowania na licznych danych (zestawy testowe)
* uruchamiania (badanie wyników końcowych i pośrednich)

Przykład (błędu algorytmicznego)

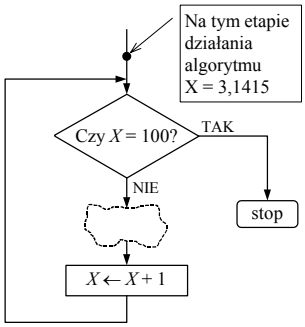
Algorytm sumowania zarobków pracowników, którzy zarabiają więcej niż ich bezpośredni przełożeni:
N jest zmienną o wartości równej liczbie pracowników, zmienne (indeksowe) I i J wskazują pracowników (kolejne elementy tablicy jednowymiarowej P, która zawiera płace pracowników), a zmienna S zawiera sumę zarobków;



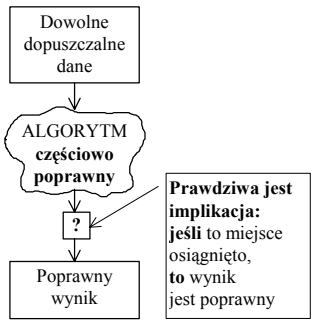
Czy ten schemat blokowy zawiera błąd czy nie?

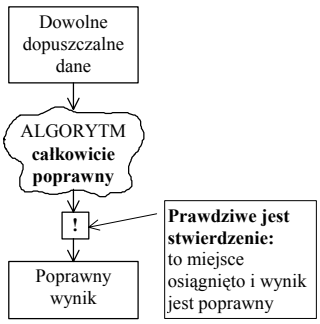
Przykład (błędu algorytmicznego)

Pętla nieskończona:



Algorytmy poprawne częściowo i całkowicie (ścisła definicja)





Metoda niezmienników i zbieżników

Częściowej poprawności algorytmu można dowodzić poprzez:

- wybranie **punktów kontrolnych**
- związanie z każdym punktem **asercji** (funkcji logicznej reprezentującej przypuszczenie)
- ustalenie **niezmienników** w obrębie iteracji
- dowiedzenie, że z prawdziwość jednej asercji wynika prawdziwość następnej, że niezmiennik pozostaje prawdziwy w kolejnych iteracjach i pociąga za sobą prawdziwość ostatniej asercji.

Całkowitej poprawności algorytmu można dowodzić poprzez dodatkowe

- ustalenie **zbieżnika** (wielkości zależnej od zmiennych i danych, która jest zbieżna)
- dowiedzenie, że po skończonej liczbie iteracji algorytm się zatrzyma w ostatnim punkcie kontrolnym.

Przykład zastosowania metody

Algorytm odwracający dowolny napis (procedura **odwrócone**):

odwrócone(„alama2koty”) = „ytok2amala”

- pomocnicze funkcje:

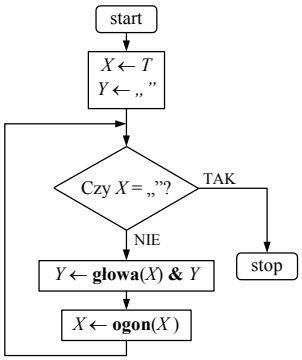
głowa(„alama2koty”) = „a” i **ogon**(„alama2koty”) = „lama2koty”

- operator konkatenacji (złożenia napisów):

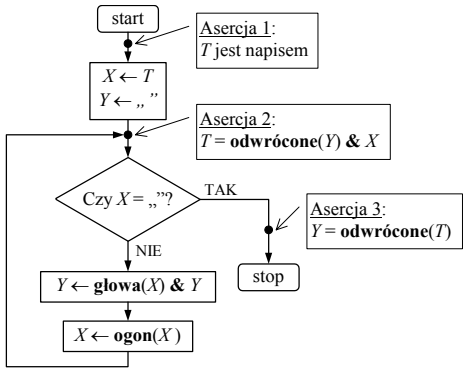
„alama” **&** „2koty” = „alama2koty”

czyli dla dowolnego napisu *T* zachodzi:

głowa(*T*) **& ogon**(*T*) = *T*



Przydział asercji:

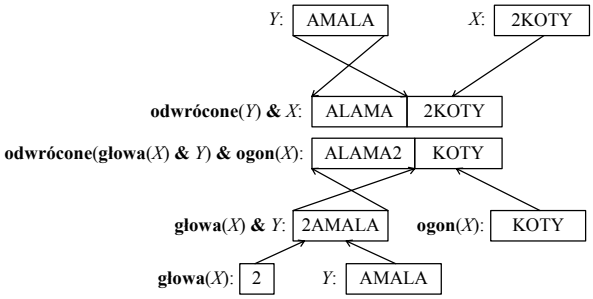


Aby wykazać częściową poprawność algorytmu należy udowodnić:

1. Jeśli asercja 1 jest prawdziwa, to 2 też jest prawdziwa (przed rozpoczęciem iteracji)
2. Jeśli w pewnym kroku iteracji asercja 2 jest prawdziwa, to w następnym kroku też jest ona prawdziwa (warunek z asercji 2 jest *niezmiennikiem* iteracji)
3. Jeśli w ostatnim kroku iteracji asercja 2 jest prawdziwa, to 3 jest też prawdziwa

Ad 1.: oczywiście zachodzi równość **odwrócone**(„ ”) **&** *T* = *T*

Ad 2.: trzeba sprawdzić czy
odwrócone(*Y*) **&** *X* = **odwrócone**(**głowa**(*X*) **&** *Y*) **& ogon**(*X*)
dla każdego *Y* i *X* ≠ „ ”



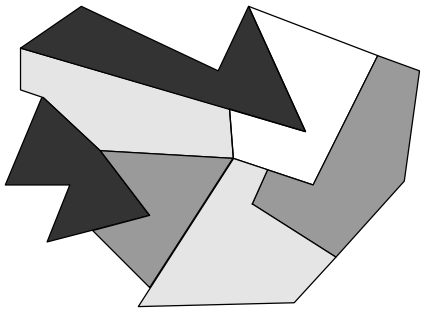
Ad 3.: oczywiście zachodzi równość
odwrócone(**odwrócone**(*Y*) **&** „ ”) = *Y*

Aby wykazać całkowitą poprawność algorytmu należy jeszcze dodatkowo udowodnić,

że dla każdego napisu *T* punkt kontrolny 2 jest przechodzony tylko skończoną liczbę razy tzn. 3 punkt kontrolny jest zawsze osiągany.

- długość napisu *X* jest *zbieżnikiem*, który może być w tym celu wykorzystany - w każdej iteracji długość *X* maleje o jeden znak i nie może stać się mniejsza od 0!

Problem z 1852 r.:



Rozwiązanie z 1976 r.:

Twierdzenie:

Cztery barwy wystarczą do pokolorowania dowolnej płaskiej mapy, tak aby każdy z dwóch sąsiadujących obszarów różnił się kolorem.

Dowód

Algorytmiczne rozwiązanie bardzo wielu podprzypadków szczególnych, które wyczerpują wszystkie możliwości – nikt formalnie nie dowodził poprawności tych algorytmów!

Metoda niezmienników i zbieżników może być zastosowana także dla dowodzenia poprawności **algorytmów rekurencyjnych**

Ale łatwiej jest skorzystać z tej metody po usunięciu rekurencji i zastąpieniu jej iteracją.

Wieże Hanoi (raz jeszcze)

Algorytm **iteracyjny** równoważny algorytmowi rekurencyjnemu!

Ustaw trzy kołki w kółko.

1. powtarzaj co następuje, aż do uzyskania po kroku 1.1 rozwiązania problemu:
   - 1.1. przenieś najmniejszy z dostępnych krążków z kołka, na którym się znajduje, na kołek następny w kierunku ruchu wskazówek zegara,
   - 1.2. wykonaj jedyne możliwe przeniesienie nie zmieniające położenia najmniejszego krążka, który został przeniesiony w kroku 1.1.