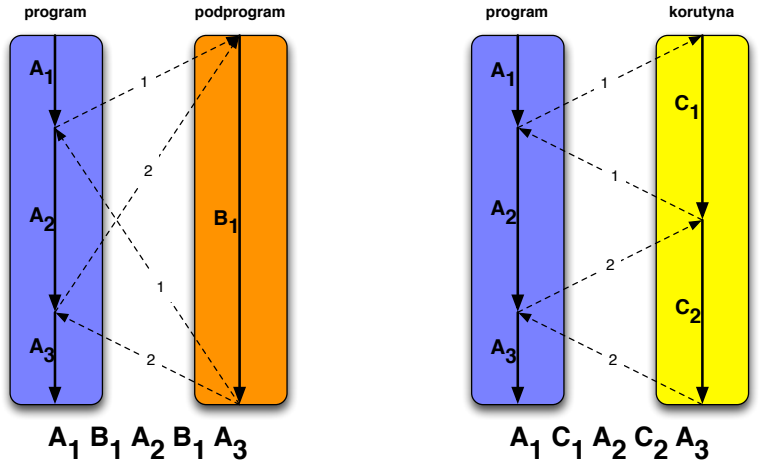
# Programowanie w Logice

## Korutyny i wątki

Przemysław Kobylański

## Korutyny i wątki

### Korutyna (współprogram)




## Korutyny i wątki

### Odraczanie celu

- Cel `X = 2, X > 1` kończy się powodzeniem.
- Cel `X > 1, X = 2` kończy się błędem.
- Sprawdzenie warunku `X > 1` należy odroczyć do czasu gdy zmienna `X` przyjmie wartość.
- Dzięki odraczaniu Prolog może być znowu bardziej deklaratywny (koniunkcja jest przemienna).

## Korutyny i wątki

### Predykat `freeze/2`

- Meta-predykat `freeze(Var, Goal)` odracza sprawdzenie celu `Goal` do chwili gdy zmienna `Var` przyjmie wartość.
- Cel sprawdzany jest natychmiast po tym jak zmienna przyjmie wartość.
- Niepowodzenie celu powoduje natychmiastowe wycofanie się.

## Korutyny i wątki

### Predykat freeze/2

```
?- freeze(X, X > 1), X = 2.
X = 2.

?- freeze(X, writeln(x=X)), freeze(Y, writeln(y=Y)),
   f(X, Y) = f(a, b).
x=a
y=b
X = a,
Y = b.

?- freeze(X, writeln(x=X)), freeze(Y, writeln(y=Y)),
   f(Y, X) = f(a, b).
y=a
x=b
X = b,
Y = a.
```

## Korutyny i wątki

### Predykat freeze/2

```
?- freeze(X, (writeln(x=X), Z = c)),
   freeze(Y, writeln(y=Y)),
   freeze(Z, writeln(z=Z)),
   f(X, Y) = f(a, b).
x=a
z=c
y=b
X = a,
Z = c,
Y = b.
```

## Korutyny i wątki

### Predykat freeze/2

```
?- X \= Y, X = a, Y = b.
false.

?- freeze(X, freeze(Y, X \= Y)), X = a, Y = b.
X = a,
Y = b.

?- freeze(X, freeze(Y, X \= Y)), X = a.
X = a,
freeze(Y, a\=Y).

?- freeze(X, freeze(Y, X \= Y)), Y = b.
Y = b,
freeze(X, freeze(b, X\=b)).
```

## Korutyny i wątki

### Predykat when/2

- Meta-predykat `when(Condition, Goal)` odracza sprawdzenie celu `Goal` do momentu gdy warunek `Condition` będzie spełniony.
- Warunek `Condition` może mieć następującą postać: (X ?= Y), nonvar(X), ground(X), (Cond1, Cond2) lub (Cond1; Cond2).
- `freeze(X, G)` jest równoważne `when(nonvar(X), G)` ale nie jest równoważne `when(ground(X), G)`.

## Korutyny i wątki

### Predykat when/2

```
?- freeze(X, writeln(x=X)), X = f(Y), Y = a.
x=f(_G1261)
X = f(a),
Y = a.

?- when(ground(X), writeln(x=X)), X = f(Y), Y = a.
x=f(a)
X = f(a),
Y = a.
```

## Korutyny i wątki

### Predykat when/2

```
?- when((nonvar(X), nonvar(Y)), writeln(f(X, Y))),
   X = a, Y = b.
f(a,b)
X = a,
Y = b.

?- when((nonvar(X); nonvar(Y)), writeln(f(X, Y))),
   X = a, Y = b.
f(a,_G1523)
X = a,
Y = b.
```

## Korutyny i wątki

### Predykat dif/2

- Predykat `dif(Term1, Term2)` narzuca ograniczenie, że termy `Term1` i `Term2` są różnymi termami.
- Jeśli `Term1` i `Term2` nie są unifikowalne, to `dif(Term1, Term2)` jest natychmiast spełniony.
- Jeśli `Term1` i `Term2` są identyczne, to `dif(Term1, Term2)` natychmiast zawodzi.
- Jeśli `Term1` i `Term2` mogą zunifikować się, wówczas `dif(Term1, Term2)` odracza warunki zapewniające różność `Term1` i `Term2`.

## Korutyny i wątki

### Predykat dif/2

```
?- dif(f(X, Y), f(X, Y)).
false.

?- dif(f(X, a), f(b, b)).
true.

?- dif(f(X, Y), f(Y, X)).
dif(Y, X).

?- dif(f(X, a), f(b, Y)).
dif(f(X, Y), f(b, a)).
```

## Korutyny i wątki
### Predykat dif/2

```
?- dif(f(X, a), f(b, Y)), X = a.
X = a.

?- dif(f(X, a), f(b, Y)), X = b.
X = b,
dif(f(b, Y), f(b, a)).

?- dif(f(X, a), f(b, Y)), X = b, Y = a.
false.
```

## Korutyny i wątki
### Listy otwarte

- Lista otwarta reprezentuje strumień termów, przy czym w strumieniu tym mogą pojawiać się kolejne termy.
- Pusty strumień reprezentowany jest nieukonkretnioną zmienną `_`.
- Kolejne termy wpisywane są do nieukonkretnionego ogona otwartej listy.
- Pusty strumień termów `S` spełnia warunek warunek `var(S)`.
- Aby zakończyć strumień termów `S` należy zunifikować go z listą pustą `[ ]`.
- W wyniku unifikacji niepustego strumienia `S` ze wzorcem `[H | T]` zostaje pod `H` podstawiony pierwszy element strumienia a `T` unifikuje się ze strumieniem kolejnych termów.

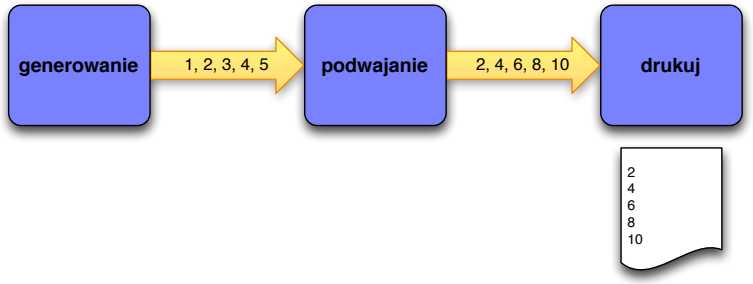
## Korutyny i wątki
### Listy otwarte

```
?- L1 = [a | L2], L2 = [b | L3], L3 = [c | L0].
L1 = [a, b, c|L0],
L2 = [b, c|L0],
L3 = [c|L0].

?- L1 = [a | L2], L2 = [b | L3], L3 = [c | L0], L0 = [].
L1 = [a, b, c],
L2 = [b, c],
L3 = [c],
L0 = [].
```

## Korutyny i wątki
### Korutyny przekazujące sobie dane w strumieniach

**Example (Podwajanie)**



## Example (cd.)

```
generowanie(I, J, S) :-
        (   I =< J
        ->  S = [I | T],
            I1 is I+1,
            generowanie(I1, J, T)
        ;   S = []).
```



## Example (cd.)

```
podwajanie(S1, S2) :-
        freeze(S1,
               (   S1 = [H1 | T1]
               ->  H2 is 2*H1,
                   S2 = [H2 | T2],
                   podwajanie(T1, T2)
               ;   S2 = [])).
```



## Example (cd.)

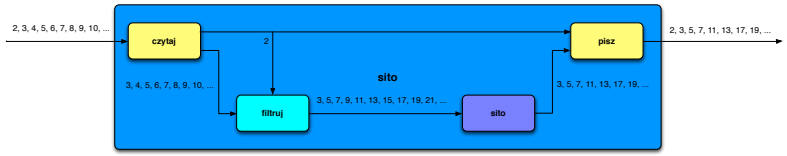
```
drukuj(S) :-
        freeze(S,
               (   S = [H | T]
               ->  writeln(H),
                   drukuj(T)
               ;   true)).
```



## Example (cd.)

```
?- drukuj(S2), podwajanie(S1, S2), generowanie(1, 5, S1).
2
4
6
8
10
S2 = [2, 4, 6, 8, 10],
S1 = [1, 2, 3, 4, 5].
```

## Example (Sito Eratostenesa)






## Example (cd.)

```
czytaj([H | T], H, T).

pisz(H, [H | T], T).

zamknij([]).
```



## Example (cd.)

```
sito(In, Out) :-
        freeze(In,
               (   czytaj(In, N, In_)
               ->  pisz(N, Out, Out_),
                   filtruj(N, In_, Out1),
                   sito(Out1, Out_)
               ;   zamknij(Out))).
```



## Example (cd.)

```
filtruj(N, In, Out) :-
        freeze(In,
               (   czytaj(In, I, In_)
               ->  (   I mod N =:= 0
                   ->  filtruj(N, In_, Out)
                   ;   pisz(I, Out, Out_),
                       filtruj(N, In_, Out_))
               ;   zamknij(Out))).
```

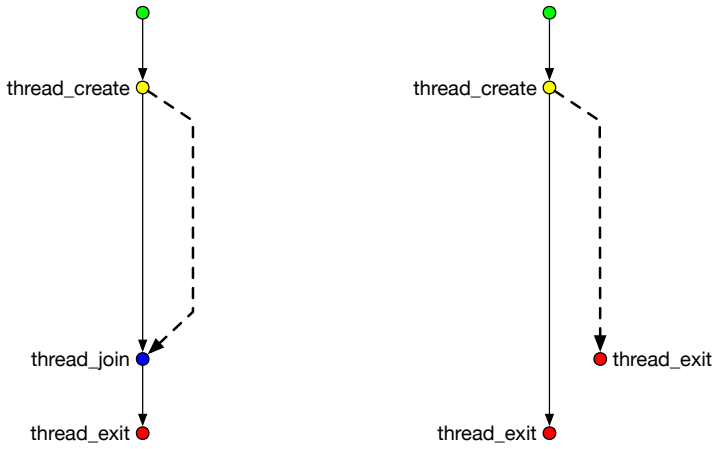
### Example (cd.)

```
?- sito(S1, S2), generowanie(2, 20, S1).
S2 = [2, 3, 5, 7, 11, 13, 17, 19],
S1 = [2, 3, 4, 5, 6, 7, 8, 9, 10|...].
```



- Do tworzenia wątków służy predykat `thread_create(Goal, ID, Options)` .
- Nowoutworzony wątek wykonuje zadany cel `Goal`.
- Identyfikator utworzonego wątku zwracany jest drugim parametrem `ID`.
- Tworzeniem wątku można sterować podając na liście `Options` termy ustalające parametry.






**detached(false)**
domyślne

**detached(true)**



### Example

Utworzony wątek nie współdzieli zmiennych z wątkiem, który go utworzył.

```
ex(Status) :-
    thread_create(X = b, ID, []),
    X = a,
    thread_join(ID, Status).
```

Jak widać w obu wątkach pod zmienną `X` podstawiane są dwie różne (nieunifikowalne) wartości:

```
?- ex(S).
S = true.
```

## Korutyny i wątki
### Tworzenie wątku

Wybrane opcje predykatu `thread_create/3`:

detached(Bool) Jeśli `false` (domyślnie), to trzeba połączyć się z tym wątkiem po jego zakończeniu wywołując predykat `thread_join/2` (zostaną swolnione wszystkie jego zasoby). Gdy `true`, to automatycznie po jego zakończeniu zostaną zwolnione wszystkie jego zasoby.

queue_max_size(Size) Ustalenie maksymalnego rozmiaru kolejki komunikatów.

## Korutyny i wątki
### Połączenie wątku

- Wywołanie predykatu `thread_join(+Id, -Status)` czeka na zakończenie wątku `Id` i oddaje `Status` jego zakończenia (następuję *połączenie*). Po zakończeniu wątku `Id` wszystkie jego zasoby zostają zwolnione. Wątek z atrybutem `detached(true)` nie może być połączony,

## Korutyny i wątki
### Komunikacja między wątkami

- Predykat `thread_send_message(+TID, +Term)` wysyła do wątku `TID` komunikat w postaci termu `Term`. Po wysłaniu wątek wysyłający kontynuuje pracę a komunikat zostaje dopisany na końcu kolejki komunikatów wątku `TID`.
- Predykat `thread_get_message(?Pattern)` pobiera z kolejki komunikatów ten, który unifikuje się z zadanym wzorcem `Pattern` i usuwa go z kolejki (komunikaty nieunifikujące się pozostają w kolejce). Jeśli kolejka jest pusta, to działanie wątku jest wstrzymane aż do chwili pojawienia się w kolejce nowego komunikatu.
- Predykat `thread_peek_message(?Pattern)` działa podobnie do `thread_get_message/1` ale z tą różnicą, że w przypadku pustej kolejki komunikatów zawodzi (nie wstrzymuje wątku).

## Korutyny i wątki
### Komunikacja między wątkami

Example (Ping-pong)

```
main :-
    thread_create(gracz, Id1, [detached(true)]),
    thread_create(gracz, Id2, [detached(true)]),
    thread_send_message(Id1, przeciwnik(Id2)),
    thread_send_message(Id2, przeciwnik(Id1)),
    thread_send_message(Id1, ping).
```

### Example (Ping-pong cd.)

```
gracz :-
    thread_get_message(przeciwnik(Id)),
    gracz(Id).

gracz(Id) :-
    thread_get_message(M1),
    writeln(M1),
    odbicie(M1, M2),
    thread_send_message(Id, M2),
    gracz(Id).

odbicie(ping, pong).
odbicie(pong, ping).
```



### Example (Ping-pong cd.)

Efekt działania:

```
?- main.
ping
pong
ping
pong
...
```



- Do synchronizacji wątków służą MUTEXy (ang. *MUTual EXclusion*).
- Jeśli jakiś zasób jest współdzielony przez dwa lub więcej wątków, to fragment kodu wymagający posiadania tego zasobu na wyłączność nazywa się **sekcją krytyczną**.
- Przed wejściem do sekcji krytycznej należy **zablokować** (ang. *lock*) mutex związany z zasobem a po wyjściu z sekcji krytycznej należy mutex **odblokować** (ang. *unlock*).
- Jeśli mutex jest zablokowany, to kolejny wątek, który chce go zablokować przed wejściem do sekcji krytycznej, zostanie **wstrzymany** do chwili gdy mutex zostanie odblokowany przez wątek opuszczający sekcję krytyczną.
- Dzięki mutexom mamy zagwarantowane, że w sekcji krytycznej związanej z danym zasobem znajduje się co najwyżej jeden wątek.



- Predykat `mutex_create(?Id)` tworzy mutex `Id`.
- Predykat `mutex_lock(+Id)` albo zablokuje mutex `Id` i skończy się powodzeniem albo wstrzyma działanie wątku, jeśli mutex `Id` jest zablokowany.
- Predykat `mutex_trylock(+Id)` działa podobnie jak `mutex_lock(+Id)` ale gdy mutex jest zablokowany, to kończy się niepowodzeniem zamiast wstrzymywać wątek.
- Predykat `mutex_unlock(+Id)` odblokowuje mutex `Id`.
- Predykat `with_mutex(+Id, Goal)` blokuje mutex `Id` na czas wykonywania celu `Goal`. Bez względu na wynik wykonania celu, zostają usunięte możliwe punktu nawrotu oraz odblokowywany jest mutex.
- Predykat `mutex_destroy(+Id)` usuwa mutex `Id`.

### Example (Błędne odliczanie)

```
main1 :-
    thread_create(powtarzaj, _, [detached(true)]),
    thread_create(powtarzaj, _, [detached(true)]).

powtarzaj :-
    odliczaj(9), nl,
    powtarzaj.

odliczaj(0).
odliczaj(N) :-
    N > 0,
    write(N),
    N1 is N-1,
    odliczaj(N1).
```



### Example (Błędne odliczanie cd.)

Przykład uruchomienia:

```
?- main1.
987654321
987654321
9876543219876543
9876521
987654321
9874321
987654654321
987654321
9876321
9876543543221
...
```



### Example (Poprawne odliczanie)

```
main2 :-
    mutex_create(Mutex),
    thread_create(powtarzaj(Mutex), _, [detached(true)]),
    thread_create(powtarzaj(Mutex), _, [detached(true)]).

powtarzaj(Mutex) :-
    with_mutex(Mutex, (odliczaj(9), nl)),
    powtarzaj(Mutex).
```



### Example (Poprawne odliczanie cd.)

Przykład uruchomienia:

```
?- main2.
987654321
987654321
987654321
987654321
987654321
987654321
987654321
987654321
987654321
987654321
987654321
...
```