Współprogramy III

Ten wykład ma na celu pokazanie kolejnej ciekawej możliwości, którą oferują współprogramy.

Wspólprogramy reprezentujące wyrażenia regularne kooperują w celu wypisania języka regularnego.

Drugim naszym celem jest opanowanie techniki dowodzenia przez indukcję ze względu na hierarchię klas (w tym przypadku coroutin).

Wyrażenia regularne

Niech A będzie ustalonym zbiorem. Jego elementy nazywać będziemy znakami alfabetu, a on sam nazywany będzie *alfabetem*. Niech A^* oznacza zbiór wszystkich skończonych ciągów znaków alfabetu A. Zbiór A^* zawiera ciąg pusty, oznaczany $\varnothing$.

Zakładamy, że znaki: $\cup$, $\bullet$, $*$, $($, $)$ nie należą do alfabetu A.

Definicja. Zbiorem *wyrażeń regularnych* WR nad alfabetem A jest najmniejszy zbiór wyrażeń taki, że:
wr1) każdy element zbioru A jest elementem zbioru WR,
wr2) jeśli dwa wyrażenia α i β należą do zbioru WR, to do zbioru WR należą też wyrażenia

$$(\alpha \cup \beta), \quad (\alpha \bullet \beta), \quad \alpha^*$$

Przykłady.
Niech $A = \{x, y, z\}$. Napisy x, z, $((x)\bullet(y \cup z))$, $(x \cup y)^*$ są wyrażeniami regularnymi, napis $(x \cup t)^*$ nie jest wyrażeniem regularnym nad alfabetem A.

Języki regularne

Z każdym wyrażeniem regularnym α można związać zbiór $|\alpha|$ ciągów skończonych znaków z alfabetu A w następujący sposób.

Definicja.

Jeżeli $\alpha = a$ jest atomowym wyrażeniem regularnym $\alpha \in A$ to $|\alpha| = \{a\}$,

Jeżeli α jest wyrażeniem regularnym postaci $(\beta \cup \gamma)$ to $|\alpha| = |\beta| \cup |\gamma|$,

jeżeli α jest wyrażeniem regularnym postaci $(\beta \bullet \gamma)$ to

$|\alpha| = \{uw: u \in |\beta| \text{ i } w \in |\gamma|\}$,

jeżeli α jest wyrażeniem regularnym postaci β^*, to

$|\alpha| = |\beta^*| = \{\varnothing \cup |\beta| \cup |\beta \bullet \beta| \cup |(\beta \bullet \beta) \bullet \beta| \cup \ldots \cup |\beta \bullet \beta \bullet \ldots \bullet \beta| \cup \ldots\}$

Elementy zbioru $|\alpha|$ nazywamy słowami języka regularnego $|a|$.

Przykład.

$|(x \bullet (y \cup z)| = \{xy, xz\}$

$|(x \cup y)^*| = \{\varnothing, x, y, xx, xy, yx, yy, xxy, xyx, \ldots\}$ ten język regularny zawiera wszystkie słowa skończone zawierające znaki x i y.

Zastosowania wyrażeń regularnych

Wyrażenia i języki regularne znalazły wiele zastosowań:

- w kompilatorach,
- w opisie procesów,
- w lingwistyce formalnej,
- w teorii złożoności,
- w aplikacjach dotyczących sterowania.

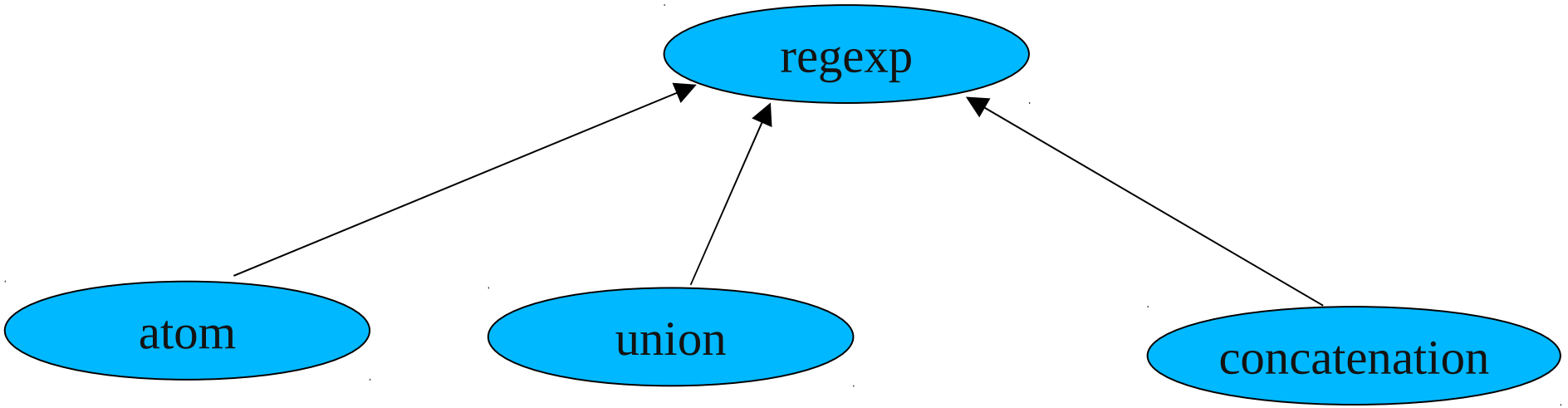
Obliczanie wartości wyrażeń regularnych

Na ogól chcemy znaleźć odpowiedź na pytanie czy dane słowo należy do określonego języka regularnego.
Czasami jednak chcemy wypisać wszystkie słowa należące do języka regularnego opisanego wyrażeniem regularnym α.

Jest oczywiste, że nie da się tego zrobić dla języków opisanych wyrażeniami zawierającymi znak *.

Wobec tego w dalszym ciągu tego wykładu *ograniczamy* się do wyrażeń regularnych, w których nie występuje operator *.

Hierarchia coroutin: regexp



Klasa bazowa hierarchii: regexp

```
unit REGEXP:coroutine;
    var B:BOOL; (* B  all the words of the language were shown *)
begin
   return
   inner;
   B := true
end REGEXP;
```

klasa Atom

```
unit ATOM: REGEXP class(C:CHAR);
begin
   do
        I:=I+1;    (* update the position *)
        Word(I):=C;
        B:=TRUE;
        detach
   od
end ATOM;
```

klasa Union

```
unit UNION: REGEXP
                class(L,R:REGEXP);
  var M: INTEGER;
begin
   do
     M:=I;
     do
        attach(L);
        if L.B then exit fi;
        detach;
        I:=M
     od;
     L.B:=FALSE;
     do
        detach;
        I:=M;
        attach(R);
        if R.B then exit fi;
     od;
     R.B:=FALSE;
     B:=TRUE;
     detach;
   od;
end UNION;
```

klasa Concatenation

unit CONCATENATION: regexp class(L, R: regexp);

```
    var N,M:INTEGER;
begin
  do
     M:=I;
     do
        attach(L);
        N:=I;
        do
          attach(R);
          if R.B then if L.B
          then exit exit
          else exit fi fi;
         detach;
         I:=N
     od;
        R.B:=FALSE;
        detach;
        I:=M
   od;
   R.B,L.B:=FALSE;
   B:=TRUE;
   detach
od;
end CONCATENATION;
```

Twierdzenie

Dla każdego obiektu o spełniającego relację o in regexp, następujący program Pr wydrukuje wszystkie słowa języka regularnego reprezentowanego przez ten obiekt o i zatrzyma się.

```
Pr:  I:=0;
     do
        attach(o);
        drukuj zawartość tablicy Word
                        for J:=1 to I
                        do
                            write(Word(J))
                        od;       writeln;
        if W.B then exit fi
     od
```

Załóżmy, że miejsca tablicy Word są wypełnione aż do pozycji I. Załóżmy ponadto, że pewne słowa z języka L(o) zostały już wcześniej wytworzone i wydrukowane przez wcześniejsze aktywowanie współprogramu o. (W tablicy Word nie ma więc już po nich śladu.)

Lemat

Wykonanie instrukcji **attach(o)** ma następujący efekt: kolejne słowo języka L(o) zostanie dopisane do tablicy Word poczynając od pozycji Word(I+1). Atrybut B przyjmie wartość true wtedy i tylko wtedy gdy wszystkie słowa języka L(o) zostaną wydrukowane.

dowód lematu

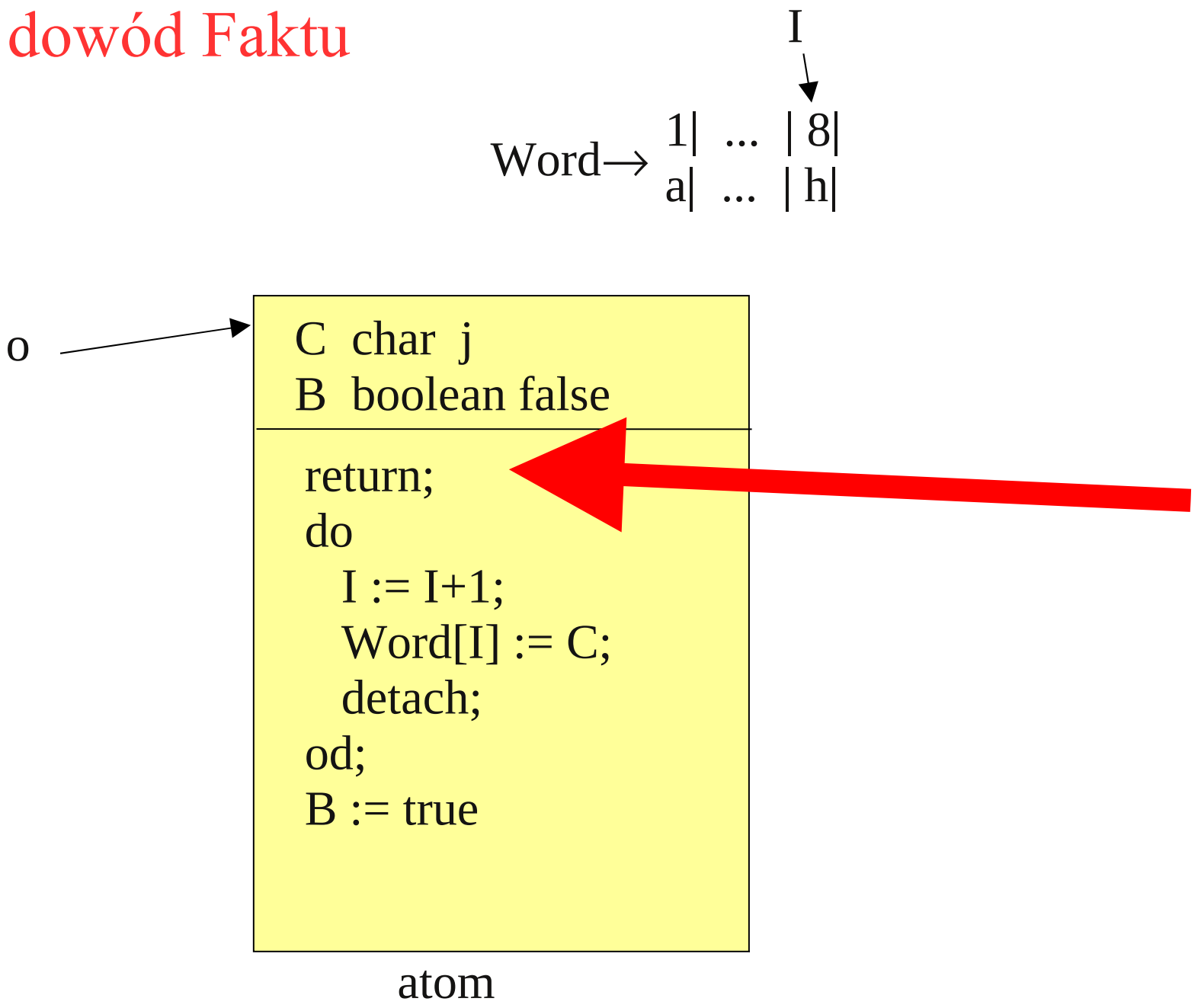
Obiekt o należy do klasy Atom, lub do klasy Union lub do klasy Concatenation.
W każdym z tych przypadków jest on korzeniem pewnego drzewa obiektów.

Dowód będzie przebiegać ze względu na wysokość tego drzewa.

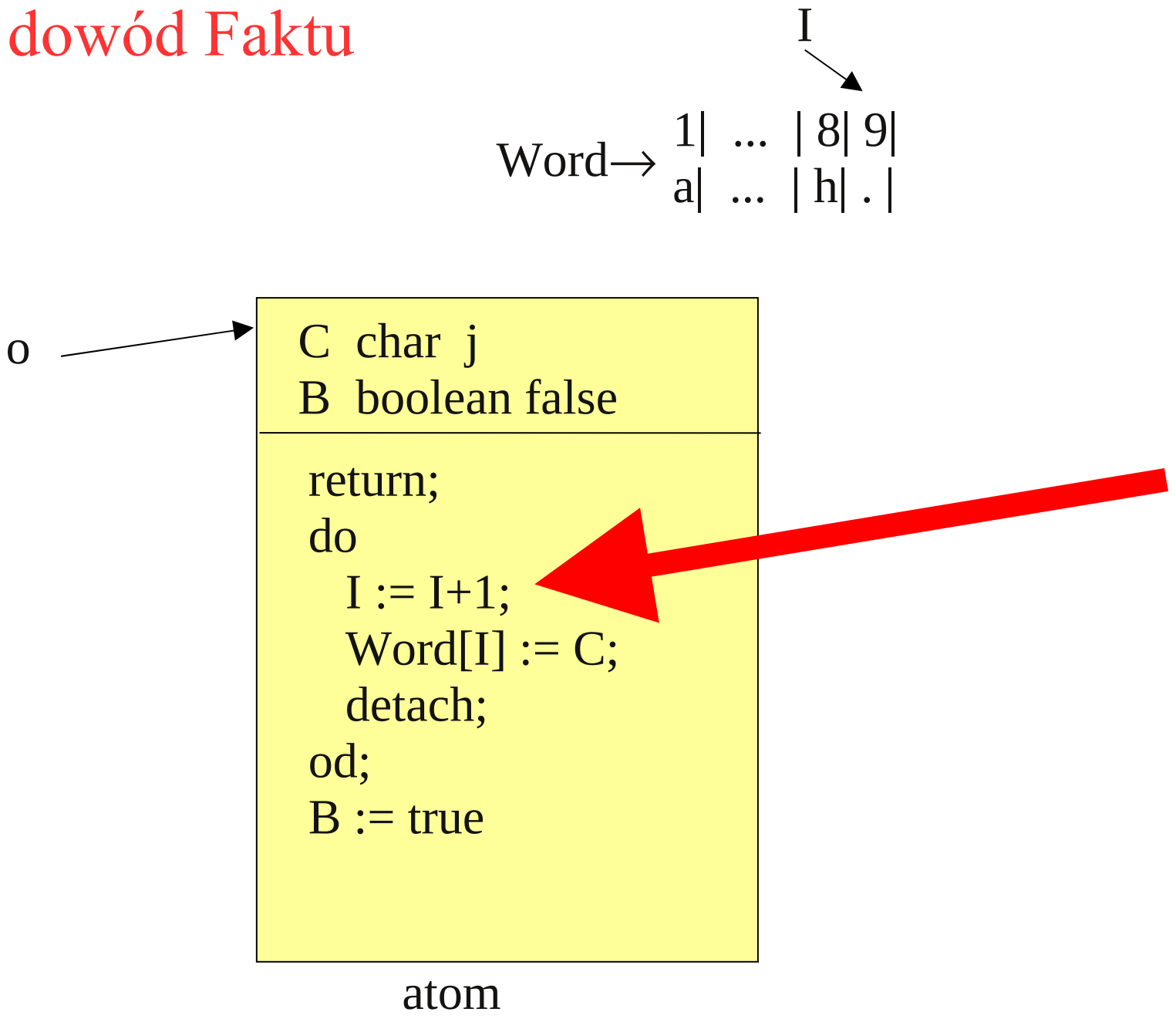
W przypadku gdy obiekt o jest atomem, drzewo ma wysokość zero, i zachodzi następujący

Fakt.
Wykonanie instrukcji **attach(o)** zastosowanej do obiektu o takiego, że o **in** Atom spowoduje umieszczenie litery C na I+1 miejscu tablicy Word i atrybut B w tym obiekcie przyjmie wartość true.

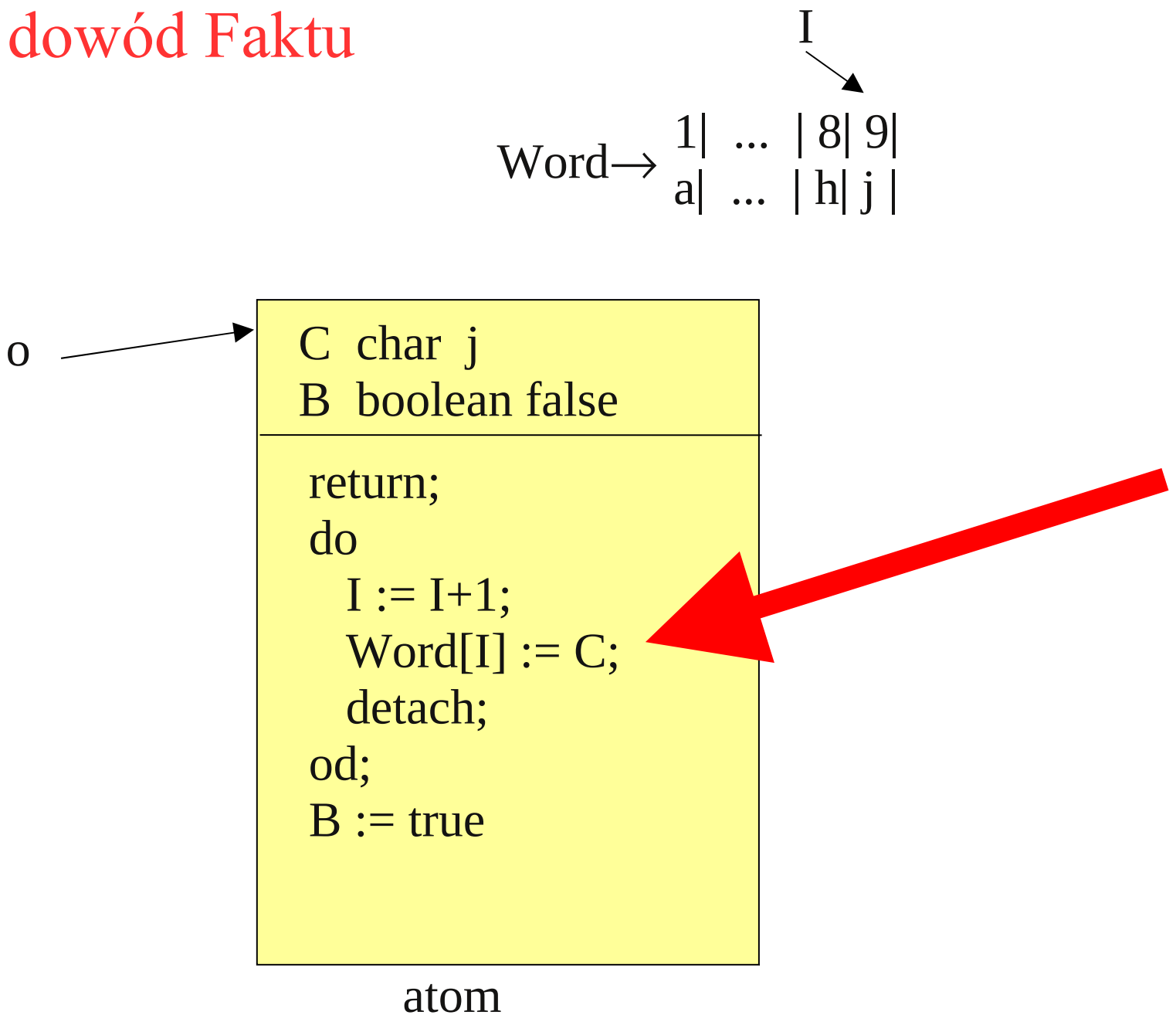
dowód Faktu


I
Word→
1| ... | 8|
a| ... | h|
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

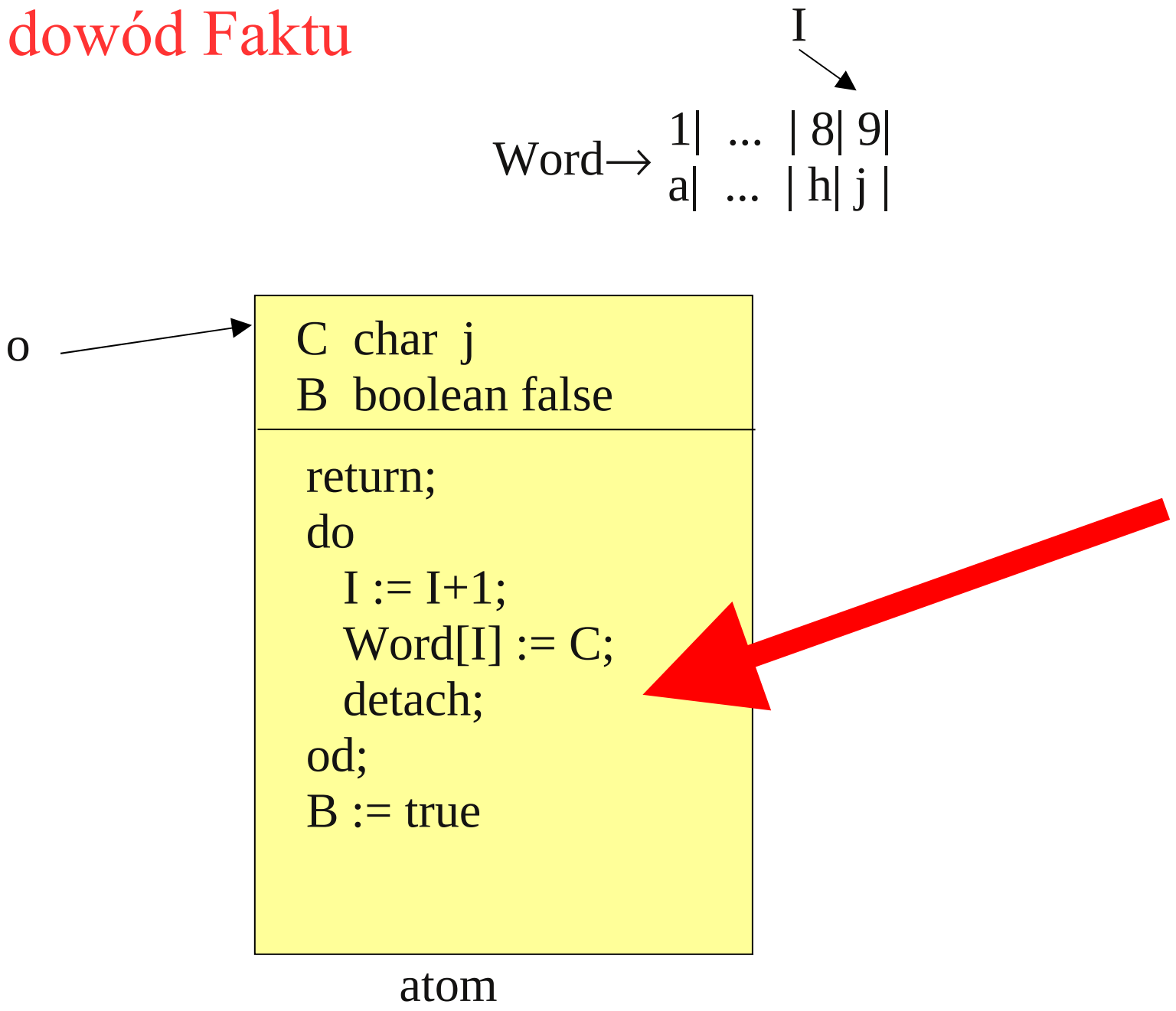
dowód Faktu


I
Word→
1| ... | 8| 9|
a| ... | h| . |
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

dowód Faktu


I
Word→
1| ... | 8| 9|
a| ... | h| j |
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

dowód Faktu


I
Word→
1| ... | 8| 9|
a| ... | h| j |
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

dowód lematu c.d.

Pozostaje nam do wykazania, że jeśli lemat jest spełniony dla obiektów L i R będących korzeniami lewego i prawego poddrzewa drzewa o korzeniu o, to lemat jest spełniony dla samego obiektu o.

Przypadek A) o in Union

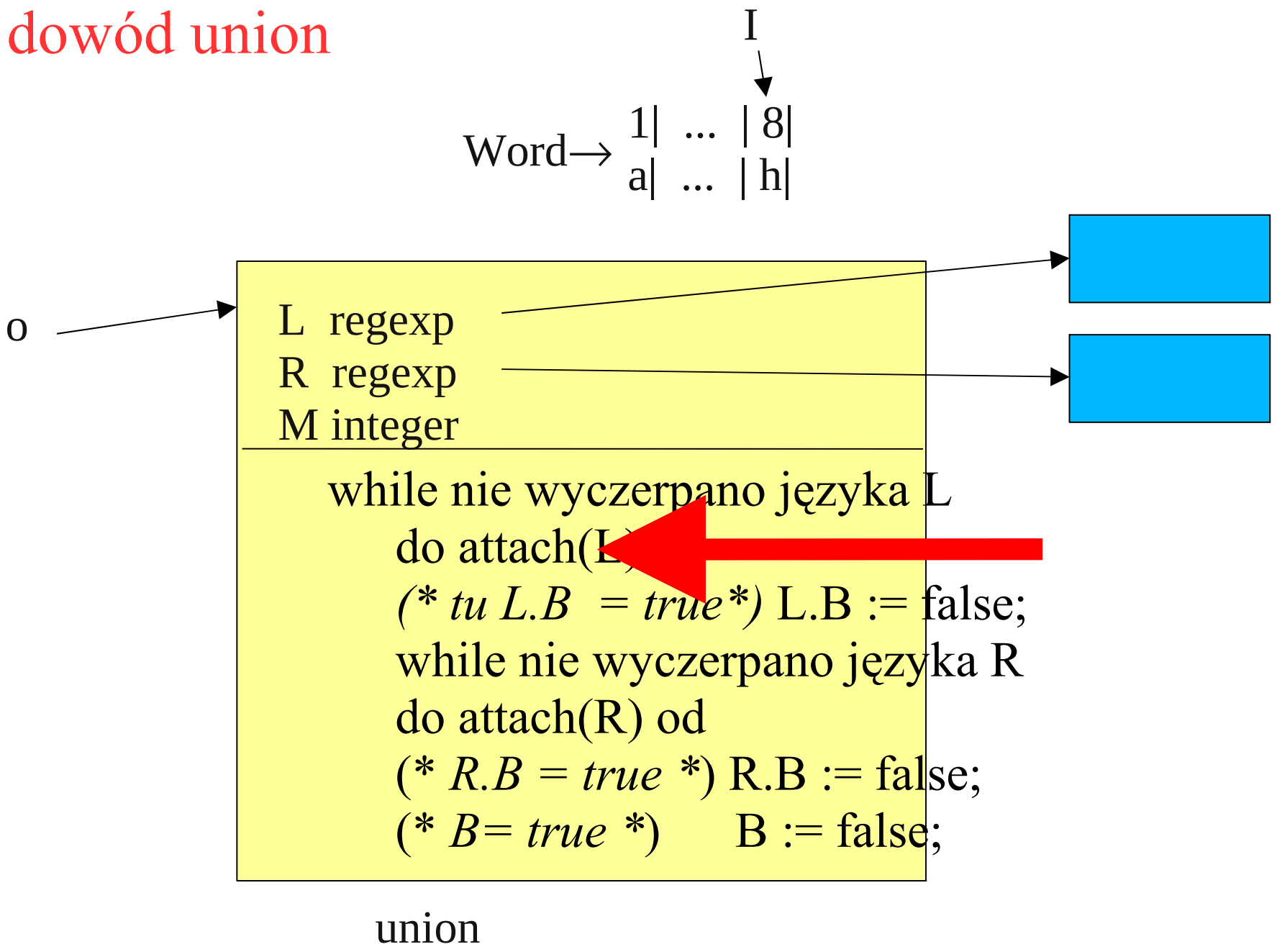
Struktura instrukcji we współprogramie o jest następująca

```
while nie wyczerpano języka L
do attach(L) od        -- z założenia indukcyjnego
(* tu L.B  = true*) L.B := false;
while nie wyczerpano języka R
do attach(R) od
(* R.B = true *) R.B := false;
(* B= true *)      B := false;
```

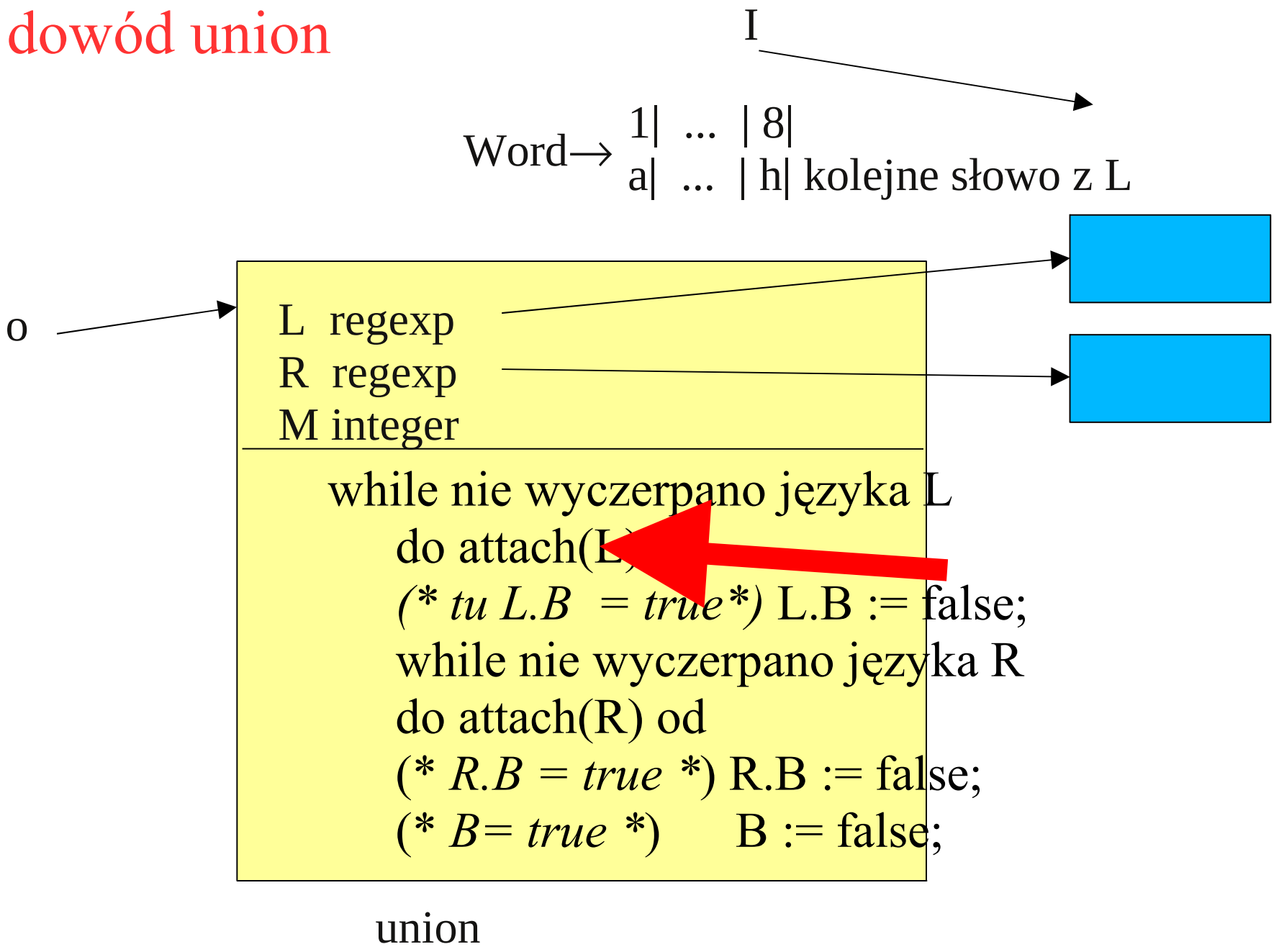
dowód lematu c.d.

```
unit Union: regexp  class(L, R: regexp);
     var M: integer;
begin
   do
     M:=I; (* I is the position of the lastly generated letter.  *)
     do
       attach(L)   (* by the inductive assumption this statement causes that one word will be generated of
                      the language L and it will be concatenated to the content of WORD(1) , ... , WORD(I)   *)
       if L.B then exit fi;
       detach;
       I:=M          (* reestablish the position in the table WORD for the next word  *)
     od;
     L.B:=FALSE;  (* restart language L *)
     do
       detach;
       I:=M;          (* reestablish the position in the table WORD for the next word  *)
       attach(R);   (* by  the inductive assumption this statement causes that one word will be generated of
                      the language R and it will be concatenated to the content of WORD(1) , ... , WORD(I)    *)
       if R.B then exit fi;
     od;
     R.B:=FALSE;  (* restart language R *)
     B:=TRUE;
     detach;
   od;
 end Union;
```

dowód union


I
Word→
1| ... | 8|
a| ... | h|
o
L regexp
R regexp
M integer
while nie wyczerpano języka L
do attach(L)
(* tu L.B = true*) L.B := false;
while nie wyczerpano języka R
do attach(R) od
(* R.B = true *) R.B := false;
(* B= true *) B := false;
union

dowód union


I
Word→
1| ... | 8|
a| ... | h| kolejne słowo z L
o
L regexp
R regexp
M integer
while nie wyczerpano języka L
do attach(L)
(* tu L.B = true*) L.B := false;
while nie wyczerpano języka R
do attach(R) od
(* R.B = true *) R.B := false;
(* B= true *) B := false;
union

dowód lematu c.d.

Przypadek B) o in Concatenation

Struktura instrukcji we współprogramie o jest następująca

```
while nie wyczerpano języka L
do
    zapamietaj I;
    attach(L);        -- słowo z języka L
    attach(R);        --  a po nim słowo z języka R
    detach;           --  a więc słowo z języka (L • R)
    odtwórz I;
 od;
```

dowód lematu c.d.

```
unit CONCATENATION: REGEXP class(L,R:REGEXP);
      var N,M:INTEGER;
 begin
   do
     M:=I; (*begin of first language word position *)
     do
       attach(L);
       N:=I; (* begin of the second language word position *)
       do
         attach(R);
         if R.B then if L.B then exit exit else exit fi fi;
         detach; I:=N (* restart language R word generation position *)
       od;
       R.B:=FALSE; (* restart language R *)
       detach; I:=M (* restart language L word generation position *)
     od;
     R.B,L.B:=FALSE; B:=TRUE; detach
   od;
 end CONCATENATION;
```

Program główny

```
const N=50;   (* DIMENSION FOR ARRAY WORD *)
     var          A,B,C,D,E,W,V,L,O,G,II,NN:REGEXP, I,J,N,M:INTEGER;
                    (* I = GLOBAL POSITION POINTER FOR ARRAY WORD *)
     var WORD: arrayof CHAR; (* BUFFER FOR WORDS GENERATION *)
 begin
    A:=new ATOM('A'); B:=new ATOM('B'); C:=new ATOM('C');    D:=new ATOM('D');
     E:=new ATOM('E');  L:=new ATOM('L'); G:=new ATOM('G');    II:=new ATOM('I');
     NN:=new ATOM('N');  O:=new ATOM('O');  W:=new UNION(A,L);
     W:=new CONCATENATION(W,new UNION(D,O));
     V:=new CONCATENATION(II,C);
     V:=new UNION(V,new CONCATENATION(L,new CONCATENATION(A,NN)));
     V:=new CONCATENATION(G,V);  V:=new UNION(A,V);
     W:=new CONCATENATION(W,V);
     writeln(" WE HAVE LANGUAGE DEFINED BY THE FOLLOWING
EXPRESSION");
     writeln(" (A∪L)•(D∪O)•(A∪G•(I•C∪L•A•N))");
     array WORD dim(1:N);
    do
      attach(W);
      write(" ");      for J:=1 to I  do   write(WORD(J))   od;
      if W.B then exit fi
   od
 end
```

Czy wiemy co wydrukuje ten program?

Czy potrzebne jest nam do tego wykonanie programu?

==========

dodaj klasę iteration pochodną klasy regexp

Współprogramy IV

Współprogram może pozwolić na szybsze wykonanie

- Przygotujmy obiekty coroutin odpowiadające rekordom aktywać procedur
- Przykład – wieże Hanoi

Procedura

```
unit Hanoi: procedure(ile, z , na: integer);
   var k: integer;
begin
   k:=6 -(z+na) ;    (* z + na +k = 6 *)
   if ile>1 then call Hanoi(ile-1, z, k) fi;
   przenieś krążek z wieży z na wieżę na;
   if ile>1 then call Hanoi(ile-1, k, na) fi
end Hanoi;
```

Tablica współprogramów I

Utwórzmy tablicę współprogramów – każdemu możliwemu rekordowi aktywacji procedury będzie odpowiadać jeden współprogram

var obiekty: arrayof arrayof arrayof CHanoi;

Wartość ile zmienia się od 1 do n= liczba krążków
wartości z oraz na zmieniają się od 1 do 3

Tablica współprogramów II

Wypełniamy tablicę obiekty:

```
for ile := 1 to n do
   newarray obiekty(ile) dim (1:3);
   for z := 1 to 3 do
      newarray obiekty(ile, z) dim (1:3)
      for na := 1 to 3 do
         if z =/= na then obiekty(ile,z,na):=new CHanoi(ile,z,na)
      od
   od
od
```

Współprogram CHanoi

```
unit CHanoi: coroutine(ile, z, na : integer);
    var k: integer;
begin
    k:= 6-(z + na);
    return;
    do
       if ile>1 then attach(obiekty[ile-1, z, k]) fi;
       przenieś krążek z wieży z na wieżę na;
       if ile>1 then attach(obiekty[ile-1, k, na]) fi;
    od
end CHanoi;
```

Zmontuj to razem i … wykonaj

```
program WieżeHanoi;
   unit Chanoi: coroutine…

   var obiekty: arrayof arrayof arrayof Chanoi,
        i,j,m: integer;
   const n=64;
begin
   (* utwórz tablicę obiekty *)
    ...
   attach(obiekty[n, 1, 3])
end WieżeHanoi;
```

Spostrzeżenia

- Obie wersje programu WieżeHanoi: *rekurencyjna* i *ze współprogramami* są równoważne!
- Druga wersja będzie działać znacznie szybciej – ponieważ zaoszczędzimy na tworzeniu rekordów aktywacji.
- Oczywiście sam problem ma złożoność wykładniczą i nic na to nie poradzimy.
  Ale w wielu sytuacjach można sporo zyskać, np. zastępując procedury modyfikujące strukturę danych np. DRZEWO przez odpowiednie współprogramy.