Brulion – this version for your eyes only
Dowód poprawności pewnej klasy

Grażyna Mirkowska
G. Mirkowska@uksw.edu.pl
Dąbrowa Leśna

Andrzej Salwicki
salwicki@mimuw.edu.pl
Dąbrowa Leśna

Streszczenie. Zamierzamy udowodnić, poprawność klasy implementującej kolejki priorytetowe w drzewach binarnych poszukiwań. Po przypomnieniu algorytmicznej specyfikacji struktury kolejek priorytetowych ATPQ , zbadamy klasę PQ rozszerzającą klasę BST drzew binarnych poszukiwań i realizującą metody *insert, delete* i *member*. Następnie udowodnimy własności tych metod, np.

$$PQ \models \forall_{n\in BST}\forall_{e\in E}\,[\textbf{call}\ insert(n,e)]member(n,e)$$

Po wykazaniu, że każda własność specyfikacji ATPQ jest prawdziwa w modelu realizowanym przez klasę PQ jesteśmy uprawnieni do stwierdzenia, że klasa PQ jest modelem specyfikacji ATPQ, inaczej mówiąc klasa ta poprawnie implementuje specyfikację ATPQ.
Dwa są powody dla których podejmujemy ten temat:

1° Jesteśmy przekonani o słuszności tezy, że praktyka programowania powinna być powiązana z dowodzeniem poprawności programów. Ta praca jest próbą dowodu poprawności klasy.

2° W wielu podręcznikach można znaleźć dowody poprawności algorytmów insert, delete i member w strukturze BST. Dowody te opowiadają jak przebiega wykonanie algorytmu – są jego opowiedzeniem, jednak nie są to dowody wyprowadzające tezę z aksjomatów (w tym przypadku z aksjomatów struktury BST).

1. Wprowadzenie

Dzisiaj coraz więcej instytucji i firm podziela pogląd wyrazany przez nas [**?**] i innych[**?**, **?**, **?**], że wytwarzane oprogramowanie powinno być dostarczane zamawiającemu wraz z dowodem poprawności.
Trochę wolniej, ale niepowstrzymanie dokonuje się delegacja uprawnień programisty na rzecz kompilatora. Na początku programowano w kodzie maszyny. Programista musiał pamiętać, że w komórce pamięci o numerze 1013 znajduje się wartośc całkowitoliczbowa, a w sąsiedniej zapisywane są wyniki obliczeń zmienno-przecinkowych. Wynikały stąd liczne błędy. Wprowadzono język Fortran i programowanie obliczania wartości wyrażeń stało się znacznie prostsze. Program nie zawierał deklaracji

zmiennych. Trudno więc było wykryć błędy w rodzaju: w programie pojawiają się zmienne var1, var, a miała być tylko jedna zmienna o nazwie var1.

Nikt dzisiaj nie neguje potrzeby sprawdzania przez kompilator takich cech programu jak:

A) poprawność składniowa programu, tj. czy dany tekst można wyprowadzić w bezkontekstowej gramatyce $\mathcal{G}$ języka programowania $\mathcal{L}$.

B) well-formedness programu tj. własność polegająca na tym, że w programie używamy nazw uprzednio zadeklarowanych, że zgadzają sie typy argumentów z typem funkcji, że ...

C) dana klasa C implementuje określony interfejs I (w Javie: `class C implements interface I`).

Uważamy, że w przyszłości narzędzia mocniejsze od kompilatorów powinny sprawdzać jeszcze mocniejsze własności np. czy dana klasa C jest modelem specyfikacji S:

`class C` **models** `specification S.`

W dalszym ciągu określimy oba pojęcia: co to jest *specyfikacja* oraz co to znaczy *być modelem*.

W tytule książki N. Wirtha znajduje się takie równanie:

$$Program = Algorytm + Struktura\ danych$$

Podobną myśl można znaleźć w pracy C.A.R. Hoare'a z r. 1974. Strukturę danych $\mathcal{T}$ można zaimplementować tworząc odpowiednią klasę $\mathcal{K}_\mathcal{T}$. Algorytm $\mathcal{A}$ wykorzystujący strukturę $\mathcal{T}$ do przeprowadzenia obliczeń wygodnie jest zaimplementować jako klasę $\mathcal{K}_\mathcal{A}$ anonimową (w terminologii Javy) lub jako blok prefiksowany nazwa klasy (w terminologii Simuli67 i Loglanu'82). Potrzebna jest jeszcze specyfikacja $\mathcal{S}$. Jej rola jest trojaka:

- specyfikacja $\mathcal{S}$ ma w sposób możliwie pełny opisywać własności struktury danych $\mathcal{T}$,
- klasa $\mathcal{K}_\mathcal{T}$ ma spełniać wymagania specyfikacji $\mathcal{S}$,
- dowód pożądanych własności algorytmu $\mathcal{A}$ powinien wykorzystywać wymagania (własności) wyliczone w specyfikacji $\mathcal{S}$.

Poniższy diagram ilustruje nasz tok myśli i pracy

$$\begin{array}{ccccc} \mathcal{T} & = & \mathcal{S} & = & \mathcal{K}_\mathcal{T} \\ & & & & \downarrow \\ \mathcal{A} & \longrightarrow & \beta & \longrightarrow & \mathcal{K}_\mathcal{A} \end{array}$$

Struktura $\mathcal{T}$ jest realizowana jako klasa $\mathcal{K}_\mathcal{T}$. Z drugiej strony struktura $\mathcal{T}$ jest opisana przez specyfikację $\mathcal{S}$. Pomiędzy klasą $\mathcal{K}_\mathcal{T}$ a specyfikacją $\mathcal{S}$ ma zachodzić relacja: klasa $\mathcal{K}_\mathcal{T}$ jest modelem specyfikacji $\mathcal{S}$. Algorytm $\mathcal{A}$ może być zaprogramowany jako rozszerzenie klasy $\mathcal{K}_\mathcal{T}$ w postaci klasy anonimowej lub bloku prefiksowanego nazwą klasy. Oczekiwane własności algorytmu zostaną wyrażone przez formułę β.
Dowód formuły β ze specyfikacji $\mathcal{S}$ jest decydującym argumentem na rzecz tezy: wykonanie algorytmu $\mathcal{A}$ w strukturze $\mathcal{T}$ ma pożądane własności ...

W dalszym ciągu tej pracy postępujemy zgodnie z przedstawionym powyżej schematem:

a) Pierwotnie sformułowane zadanie to wykazać, że algorytm $insert(e, n)$ wykonywany w strukturze BST drzew binarnych poszukiwań ma następującą własność

$$\beta : \qquad \forall_{n \in BST} \forall_{e \in E} mb(e, insert(e, n)).$$

b) W rozdziale 2 przypominamy zwyczajowo używany opis struktury BST.

c) nieformalny opis algorytmu insert: ...

d) Specyfikacja struktury drzew binarnych poszukiwań to algorytmiczna teoria drzew BST, zobacz rozdział 3. W rozdziale tym badamy jakość specyfikacji i wykazujemy, ...

e) Klasa BST ...

f) algorytm insert najlepiej jest zapisać jako blok prefiksowany nazwą klasy BST
pref BST block

g) przeprowadzimy dowód naszej formuły z aksjomatów teorii ATBST.

W podobny sposób można dodać algorytm $delete$ i przeprowadzić dowód jego własności.
W końcu będziemy mogli udowodnić, że klasa PQ rozszerzająca klasę BST
unit PQ: **class extends** BST : ...
jest poprawną implementacją struktury kolejek priorytetowych.

========== Ale nic nie jest za darmo. Na to by kompilator mógł zbadać czy program jest poprawnie zbudowany musimy zadeklarować typy zmienych, funkcji, etc. Jeżeli chcemy by powstały nowe narzędzia mocniejsze od kompilatora i sprawdzające np. poprawność programu względem warunków początkowego i końcowego to musimy te warunki gdzieś zapisać. W językach programowania Eiffel i Euclid znajdujemy próby stworzenia fraz specyfikujących algorytmy. Logiki programów [**?**], które powstały ponad czterdzieści lat temu, starały się dostarczyć narzędzi do badania własności algorytmów: *poprawność, częściowa poprawność, własność stopu, etc.*. Na razie nie widać zadowalających narzędzi do badania programów obiektowych. W niniejszym opracowaniu analizujemy przypadek pewnej klasy PQ i wykazujemy, że jest ona poprawną implementacją struktury kolejek priorytetowych. W programowaniu obiektowym pojawiają się nowe własności semantyczne, **nowe zjawiska** nieznane programom strukturalnym tj deterministycznym programom iteracyjnym.
Przykłady.
Obiekty niepotrzebne i obiekty niedostępne
Wyciek pamięci.
Wiszące referencje.
Śmieci.

Plan

1. Drzewa binarnych poszukiwań BST
2. Algorytmiczna teoria BST
   - sygnatura
     dziedziny operacji i sygnały błędu
   - aksjomaty
   - tw. o reprezentacji
3. Interpretacja kolejek priorytetowych
   - lemat o operacji insert
   - lemat o operacji delete
   - fakt: każda skończona liczba operacji insert i delete wykonana na drzewie BST zostawia drzewo BST
   - **lemat** Po wykonaiu insert(e, s), operacja member znajduje ten element w drzewie s.
     W dowodzie wykorzystamy tw. o reprezentacji i regułę wnioskowania

$$\frac{\{\Gamma(e,s) \Rightarrow \beta(e,s)\}_{e\in E, s\in T}}{\forall_{e\in E, s\in T}\beta(e,s)}$$

4. drzewa AVL

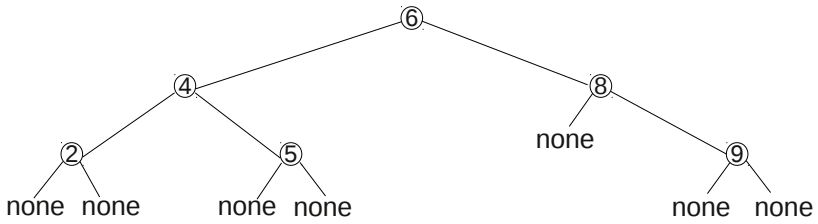
2. Drzewa BST

Co to są drzewa BST?
Niech E będzie zbiorem elementów, uporządkowanym liniowo przez relację $<$ (w programach będziemy używać nazwy *less*). Ponadto rozważać będziemy relację równość $=$ w zbiorze E.

Definicja 1. *Drzewem poszukiwań binarnych* (*ang*. BST-tree) jest każde skończone drzewo binarne d o tej własności, że każdemu wierzchołkowi n przyporządkowana jest pewna wartość $n.v$ ze zbioru E i ponadto dla każdego wierzchołka n drzewa d zachodzą dwie własności:

- największa wartość zapisana w lewym poddrzewie $n.l$ wierzchołka n jest mniejsza od wartości $n.v$,
- najmniejsza wartość zapisana w prawym poddrzewie $n.r$ wierzchołka n jest większa od wartości $n.v$.

Przykład 1. Niech d będzie jakimś drzewem BST, np.



□

Każde drzewo binarne można zapisać w postaci tzw. *S*-wyrażenia (tj. w notacji z nawiasami).

Definicja 2. *Zapis* jest funkcją określoną na zbiorze drzew binarnych, o wartościach w zbiorze *S*-wyrażeń:

- Zapisem drzewa pustego jest S-wyrażenie $()$,
- Zapisem drzewa zawierającego dokładnie jeden element e jest *S*-wyrażenie $(()e())$,
- Niech d będzie drzewem binarnym, niech w korzeniu tego drzewa będzie zapisany element e i niech drzewa binarne $d.l$ i $d.r$ będą, odpowiednio, lewym i prawym poddrzewem drzewa d. Zapisem drzewa $d.l$ jest T_1, a T_2 jest zapisem drzewa $d.r$. Zapisem drzewa d jest *S*-wyrażene $(T_1 e T_2)$.

□

Zapisy drzew binarnych poszukiwań spełniają dodatkowy **warunek**: dla każdego wierzchołka wewnętrznego n drzewa d, elementy zbioru E występujące w zapisie lewego poddrzewa $n.l$ są mniejsze od elementu zapisanego w wierzchołku n i odpowiednio, elementy występujące w zapisie prawego poddrzewa $n.r$ są większe od elementu zapisanego w wierzchołku n.

Łatwo zaobserwować nastepujący

Fakt 1. Odwzorowanie *zapis* jest funkcją różnowartościową i **na** ze zbioru BST drzew binarnych poszukiwań na zbiór S_{BST}, S-wyrażeń spełniających ten warunek.

$$zapis : BST \xrightarrow[\text{na}]{1-1} S_{BST}$$

3. Algorytmiczna teoria drzew BST - specyfikacja

W tym rozdziale przypominamy aksjomaty struktury danych - drzewa BST. Tj. przypominamy definicję algorytmicznej teorii drzew ABST i podstawowe twierdzenie o reprezentacji.

Algorytmiczna teoria drzew binarnych poszukiwań

SYGNATURA TEORII, programiści mówią o interfejsie

Na uniwersum składają się dwa typy: zbiór E elementów oraz zbiór T drzew binarnych.

Działania

$new\_T : E \to T$

$v : T \to E \qquad l : T \to T \qquad r : T \to T$

$ul : T \times T \to T \qquad ur : T \times T \to T$

Relacje

$isnone : T \to B_0 \qquad <: E \times E \to B_0$

$empty : T \to B_0 \qquad =: E \times E \to B_0$

AKSJOMATY

$$v(new\_T(e)) = e \wedge isnone(l(new\_T(e))) \wedge isnone(r(new\_T(e))) \tag{BST1}$$

$$(isnone(n) \vee \mathbf{S}\, true) \tag{BST2}$$

$$mb(e, n.l) \Rightarrow e < v(n) \wedge mb(e, n.r) \Rightarrow v(n) < e \tag{BST3}$$

$$(n = n' \equiv (isnone(n) \wedge isnone(n')) \vee (n.v = n'.v \wedge n.l = n'.l \wedge n.r = n'.r)) \tag{BST4}$$

$$(n.r = n'' \wedge n.v = e \wedge (\mathbf{K}_1 bool \vee isnone(n'))) \Rightarrow$$
$$[n3 := ul(n', n)](n3.r = n'' \wedge n3.v = e \wedge n3.l = n') \tag{BST5}$$

$$(n.l = n'' \wedge n.v = e \wedge (\mathbf{K}_2 bool \vee isnone(n'))) \Rightarrow$$
$$[n3 := ur(n', n)](n3.r = n' \wedge n3.v = e \wedge n3.r = n'') \tag{BST6}$$

aksjomaty liniowego porządku $\leq$ i równości $=$ (BST7)

aksjomaty o Exception - do wymyslenia (BST8)

Programy występujące w powyższych formułach: *mb*, **S**, $\mathbf{K}_1$, $\mathbf{K}_2$ są zamieszczone poniżej:

S: **begin**
n1:=n; continue := true;
while (n1 $\neq$ none) $\wedge$ continue
do
if e =n1.val **then** continue := false
else
if e<n1.val **then** n1 := n1.l **else** n1 := n1.r **fi**
fi;
done
end

$\mathbf{K}_1$: **begin**
n2 := n';
while $\neg$ isnone(n2.r) **do** n2 := n2.r **done**;
bool := n2.v < n.v
end

```
K2: begin
      n2 := n';
      while ¬ isnone(n2.l) do n2 := n2.l done;
      bool := n.v < n2.v
    end
```

```
mb(e, n) ≡ begin n1 := n; result := false;
             while ¬ result ∧ n1≠none
             do
                 if e =n1.v then result := true
                 else
                    if e <n1.v then n1:=n1.l else n1 :=n1.r fi
                 fi
             done
           end result
```

Twierdzenie 1. (o niesprzeczności)
Wymagania podane powyżej są niesprzeczne.

Dowód:
Skonstruujemy model, dalej będziemy go nazywać *modelem standardowym* teorii ABST. Elementami tego modelu są S-wyrażenia nad zbiorem E i sam zbiór E.
Jedynym S-wyrażeniem n dla którego zachodzi $isnone(n)$ jest S-wyrażenie $()$.
Wynikiem operacji $new\_T(e)$ jest S-wyrażenie $(()e())$.
Niech $x = (T_1 e T_2)$ wtedy $x.v = e$ oraz $x.l = T_1$ i $x.r = T_2$.
Jeśli $x = (T_1 e T_2)$ i S-wyrażenie T ma tę własność, że największy element e' w nim zapisany jest mniejszy od elementu e, to określony jest wynik operacji $ul(T, x) = (T e T_2)$ lub czytając to inaczej operacji $x.l := T$. Jeśli warunek ten jest niespełniony to wynikiem jest wyjątek $ExceptionLL$.
Podobnie, jeśli $x = (T_1 e T_2)$ i S-wyrażenie T ma tę własność, że najmniejszy element e' w nim zapisany jest większy od elementu e, to określony jest wynik operacji $ur(T, x) = (T_1 e T)$ lub czytając to inaczej operacji $x.r := T$. Jeśli warunek ten jest niespełniony to wynikiem jest wyjątek $ExceptionRR$.
Przy tych definicjach dość łatwo jest sprawdzić, że wszystkie własności (BST1 – BST6) wymienione w sekcji AKSJOMATY są prawdziwe w tej interpretacji. □

Twierdzenie 2. (o reprezentacji)
Każdy model $\mathcal{M}$ teorii ABST jest izomorficzny z modelem standardowym nad zbiorem E elementów modelu $\mathcal{M}$.

Wnioskiem z twierdzenia o reprezentacji jest następujące

Twierdzenie 3. (o opisowej zupełności)
Teoria $ATBST$ jest *opisowo zupełna* ze względu na zbiór S-wyrażeń.

Tzn. dla dowolnej formuły $\alpha(x)$ z jedną zmienną wolną x typu T, jeśli dla każdego S-wyrażenia należącego do zbioru T, formuła $\alpha(x/s)$ jest twierdzeniem teorii $ABST$, to formuła $\forall x\, \alpha(x)$ jest twierdzeniem $ABST$.

□

Niech Δ_s będzie formułą opisującą drzewo s tj. formułą postaci ... Powyższe twierdzenie jest uzasadnieniem poprawności następujacej reguły wnioskowania

$$\frac{\{\Delta_s \Rightarrow \alpha(x/s)\}_{s\in S}}{\forall x\, \alpha(x)}$$

Co się stanie jeśli ze zbioru aksjomatów usuniemy formuły algorytmiczne?
Taki zbór formuł pierwszego rzędu dopuszcza modele niestandardowe.

Zamknijmy ten rozdział modułem specyfikacji drzew BST

```
unit ABST: specification (type E; function less(e,e1: E):Boolean,
            function eq(e,e1: E):Boolean);
    unit T: class(v:E);
uzupełnij
end ABST
```

Moduły typu specification różnią się od interfejsów (interface) tym, że występuje w nich część o nazwie Aksjomaty. W tej części wyliczamy postulaty (czasami mówi sie niezmienniki) jakie maja byc prawdziwe w środowisku realizowanym przez odpowiednią klasę. Relacja klasa C implementuje interfejs I jest znacznie słabsza od relacji klasa C modeluje specyfikację S. Kompilator bez trudu sprawdzi czy wszystkie funkcje wyliczone w interfejsie I zostały zadeklarowane w klasie C. Do sprawdzenia relacji `class C models S` potrzebne są inwencja i wiedza jaką dysponuje człowiek.

4. Implementacja drzew BST

A oto implementacja

```
unit BST: models ABST class(type E; function less(e,e1: E):Boolean,
            function eq(e,e1: E):Boolean);
    unit T: class(v:E);
        private var l,r: T;
        unit cl: function: T; begin result := l end cl;
        unit cr: function: T; begin result := r end cr;
        unit ul: procedure(n: T);
        begin
            if less(max(n), e) then l := n else throw ExceptionLL endif
        end ul;
        unit ur: procedure(n: T);
        begin
            if less(e, min(n)) then r := n else throw ExceptionRR endif
```

```
        end ur;
        unit min: function(n: T): E;
        begin
            if n=none then raise ExceptionEmpty else
            while n.l =/= none do n := n.l done; result := n.e endif
        end min;
        unit max: function(n: T): E;
        begin
            if n=none then raise ExceptionEmpty else
            while n.r =/= none do n := n.r done; result := n.e endif
        end max;
    end T;
end BST
```

Można sprawdzić, że wszystkie własności wymienione w części AKSJOMATY specyfikacji ABST są formułami prawdziwymi w środowisku realizowanym przez klase BST.

A więc mamy następujący

Fakt 2. Klasa BST rzeczywiście modeluje specyfikację ABST.

5. Deklaracje procedur insert i delete

5.1. insert

Zapiszmy procedurę insert

```
unit insert: procedure(e:element, n: node);
   var dalej: Boolean, n1: node;
begin
   if n=/= none then
      dalej := true;
      n1:= n;
      while dalej
      do
        if e= n1.v
        then
           dalej := false
        else
           if e < n1.v
           then
              if n1.l =none
              then
                 ul(n1, new node(e)); (* n1.l := new node(e) *)
                 n1 := n1.l
              else
                 n1 :=n1.l
              fi;
           else (* e > n1.v *)
```

```
               if n1.r =none
               then
                  ur(n1, new node(e)); (* n1.r := new node(e) *)
                  n1 := n1.r
               else
                  n1 := n1.r
               fi;
            fi
         fi
      od
   else
      n:= new node(e)
   fi
end insert;
```

5.2. member

funkcję member

```
unit member: function(e:element, n: node): Boolean;
   var dalej: Boolean, n1: node;
begin
   if n=/= none then
      dalej := true;
      n1:= n;
      while dalej
      do
        if e= n1.v
        then
           dalej := false; result := true
        else
           if e < n1.v
           then
              if n1.l =none
              then
                 dalej := false; result := false
              else
                 n1 :=n1.l
              fi
           else (* e > n1.v *)
              if n1.r =none
              then
                 dalej := false; result := false
              else
                 n1 := n1.r
              fi
           fi
        fi
      od
```

```
   else
      result := false
   fi
end member;
```

5.3. aksjomat skończoności drzew BST

oraz napiszmy na nowo program S występujący w aksjomacie (BST2) procedurę search – przyjęliśmy jako aksjomat drzew BST, że algorytm search zawsze kończy swe obliczenia, tj. każde drzewo BST jest skończone.

Uwaga. Nie mozna pominąć tego aksjomatu bo wtedy nie mamy jak odrzucić modeli niestandardowych, z gałęziami nieskończonymi. **Koniec uwagi**

```
unit search: procedure(e:element, n: node);
   var dalej: Boolean, n1: node;
begin
   if n=/= none
   then
      dalej := true;
      n1:= n;
      while dalej
      do
        if e= n1.v
        then
           dalej := false;
        else
           if e < n1.v
           then
              if n1.l =none
              then
                 dalej := false;
              else
                 n1 :=n1.l
              fi
           else (* e > n1.v *)
              if n1.r =none
              then
                 dalej := false;
              else
                 n1 := n1.r
              fi
           fi
        fi
      od
   else (* n=/= none *)
      dalej := false
   fi
end search;
```

6. Dowód

6.1. przypadek - n = none

Ten przypadek trzeba rozpatrzeć osobno. Najpierw zauważmy, że

$$n = none \Rightarrow [insert(e, n)(n = new\ node(e))$$

Z kolei łatwo widać, że $member(e, new\ node(e))$.

6.2. przypadek - n $\neq$ none

Zacznijmy od tego, że jest tautologią każda formuła o nastepujacym schemacie

$$S\alpha \Rightarrow S\alpha$$

a więc także formuła postaci

$$S : \left\{ \begin{array}{l} P; \\ M \end{array} \right\} \alpha \Rightarrow S : \left\{ \begin{array}{l} P; \\ M \end{array} \right\} \alpha$$

Niech program P składa się z dwu instukcji $P : \delta := true; n1 := n$, a o programie M wiadomo, że nie zmienia wartości zmiennej n, a jego obliczenie zależy tylko wartości zmiennych δ oraz $n1$. Przy tych założeniach prawdziwa jest następująca równoważność

$$\{P; M\}\, \alpha \Rightarrow \{P; M; P; M\}\, \alpha$$

Jeśli założenia o programie M odziedziczy program K, to mamy wtedy taką tautologię

$$S : \left\{ \begin{array}{l} P : \delta := true; n1 := n; \\ M : \left(\begin{array}{l} \textbf{if } \delta \textbf{ then } K \textbf{ fi} \end{array} \right)^i \end{array} \right\} \alpha \Rightarrow S^2 : \left\{ \begin{array}{l} P : \delta := true; n1 := n; \\ M : \left(\begin{array}{l} \textbf{if } \delta \textbf{ then } K \textbf{ fi} \end{array} \right)^i; \\ P : \delta := true; n1 := n; \\ M : \left(\begin{array}{l} \textbf{if } \delta \textbf{ then } K \textbf{ fi} \end{array} \right)^i \end{array} \right\} \alpha$$

ustawmy to inaczej

$$S : \left\{ \begin{array}{l} P : \delta := true; n1 := n; \\ M : \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad K \\ \textbf{fi} \end{array} \right)^i \end{array} \right\} \alpha \Rightarrow S^2 : \left\{ \begin{array}{l} P : \delta := true; n1 := n; \\ M : \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad K \\ \textbf{fi} \end{array} \right)^i; \\ P : \delta := true; n1 := n; \\ M : \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad K \\ \textbf{fi} \end{array} \right)^i \end{array} \right\} \alpha$$

zamiast K wstawiam $\{\textbf{if } \gamma \textbf{ then } M_1 \textbf{ else } M_2 \textbf{ fi}\}$ nadal zastrzegając, że mają być spełnione zastrzeżenia
...
otrzymuję

$$S: \left\{ \begin{array}{l} P: \delta := true; n1 := n; \\ \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad M: \left(\begin{array}{l} \textbf{if } \gamma \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} \textbf{if } \gamma \\ \textbf{then} \\ \quad M_1 \\ \textbf{else} \\ \quad M_2 \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right) \\ \textbf{fi} \end{array} \right)^i \end{array} \right\} \alpha \Rightarrow S^2: \left\{ \begin{array}{l} P: \delta := true; n1 := n; \\ \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad M: \left(\begin{array}{l} \textbf{if } \gamma \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} \textbf{if } \gamma \\ \textbf{then} \\ \quad M_1 \\ \textbf{else} \\ \quad M_2 \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right) \\ \textbf{fi} \end{array} \right)^i ; \\ P: \delta := true; n1 := n; \\ \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad M: \left(\begin{array}{l} \textbf{if } \gamma \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} \textbf{if } \gamma \\ \textbf{then} \\ \quad M_1 \\ \textbf{else} \\ \quad M_2 \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right) \\ \textbf{fi} \end{array} \right)^i \end{array} \right\} \alpha$$

Kładę $e = n1.v$ jako γ i zamiast M_1 kładę $\delta := false$.

$$S: \left\{ \begin{array}{l} P: \delta := true; n1 := n; \\ \quad \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad M: \left(\begin{array}{l} \textbf{if } \gamma : e = n1.v \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} M_1 : \delta := false \\ \textbf{else} \\ \quad M_2 \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right) \end{array} \right)^i \end{array} \right\} \alpha \Rightarrow S^2: \left\{ \begin{array}{l} P: \delta := true; n1 := n; \\ \quad \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad M: \left(\begin{array}{l} \textbf{if } \gamma : e = n1.v \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} M_1 : \delta := false \\ \textbf{else} \\ \quad M_2 \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right) \end{array} \right)^i ; \\ P: \delta := true; n1 := n; \\ \quad \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad M: \left(\begin{array}{l} \textbf{if } \gamma : e = n1.v \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} M_1 : \delta := false \\ \textbf{else} \\ \quad M_2 \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right) \end{array} \right)^i \end{array} \right\} \alpha$$

Program M_2 ma nastepująca strukturę $if\ \kappa\ then\ M_3\ else\ M_4\ fi$

$$
S: \left\{ \begin{array}{l} P: \delta := true; n1 := n; \\ M: \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} \textbf{if } \gamma: e = n1.v \\ \textbf{then} \\ M_1: \delta := false \\ \textbf{else} \\ \quad M_2: \left(\begin{array}{l} \textbf{if } \kappa \\ \textbf{then} \\ \quad M_3 \\ \textbf{else} \\ \quad M_4 \\ \textbf{fi} \end{array} \right) \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right)^i \end{array} \right\} \alpha \Rightarrow S^2: \left\{ \begin{array}{l} P: \delta := true; n1 := n; \\ M: \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} \textbf{if } \gamma: e = n1.v \\ \textbf{then} \\ M_1: \delta := false \\ \textbf{else} \\ \quad M_2: \left(\begin{array}{l} \textbf{if } \kappa \\ \textbf{then} \\ \quad M_3 \\ \textbf{else} \\ \quad M_4 \\ \textbf{fi} \end{array} \right) \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right)^i ; \\ P: \delta := true; n1 := n; \\ M: \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \quad K: \left\{ \begin{array}{l} \textbf{if } \gamma: e = n1.v \\ \textbf{then} \\ M_1: \delta := false \\ \textbf{else} \\ \quad M_2: \left(\begin{array}{l} \textbf{if } \kappa \\ \textbf{then} \\ \quad M_3 \\ \textbf{else} \\ \quad M_4 \\ \textbf{fi} \end{array} \right) \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right)^i \end{array} \right\} \alpha
$$

Pozostaje wstawić programy oznaczone jako M_3 i M_4.

Uporządkujmy to nieco

$$
S:\left\{\begin{array}{l}
P:\ \delta := true;\ n1 := n;\\
M:\left(\begin{array}{l}
\textbf{if } \delta\\
\textbf{then}\\
\quad K:\left\{\begin{array}{l}
\textbf{if } \gamma: e = n1.v\\
\textbf{then}\\
M_1:\ \delta := false\\
\textbf{else}\\
\quad M_2:\left\{\begin{array}{l}
\textbf{if } \kappa: e < n1.v\\
\textbf{then}\\
M_3:\ \textbf{if } n1.l = none \textbf{ then } \delta := false \textbf{ else } n1 := n1.l \textbf{ fi}\\
\textbf{else}\\
M_4:\ \textbf{if } n1.r = none \textbf{ then } \delta := false \textbf{ else } n1 := n1.r \textbf{ fi}\\
\textbf{fi}
\end{array}\right.\\
\textbf{fi}
\end{array}\right.\\
\textbf{fi}
\end{array}\right)^{i}
\end{array}\right\}\alpha
$$

$\Rightarrow$

$$
S^2:\left\{\begin{array}{l}
P:\delta := true; n1 := n;\\
M:\left(\begin{array}{l}
\textbf{if } \delta\\
\textbf{then}\\
\quad K:\left(\begin{array}{l}
\textbf{if } \gamma: e = n1.v\\
\textbf{then}\\
M_1:\ \delta := false\\
\textbf{else}\\
\quad M_2:\left(\ \textbf{if } \kappa \textbf{ then } \quad M_3 \textbf{ else } \quad M_4 \textbf{ fi}\right.\\
\textbf{fi}
\end{array}\right.\\
\textbf{fi}
\end{array}\right)^{i}\\
P:\delta := true; n1 := n;\\
M:\left(\begin{array}{l}
\textbf{if } \delta\\
\textbf{then}\\
\quad K:\left(\begin{array}{l}
\textbf{if } \gamma: e = n1.v\\
\textbf{then}\\
M_1:\ \delta := false\\
\textbf{else}\\
\quad M_2:\left(\ \textbf{if } \kappa \textbf{ then } \quad M_3 \textbf{ else } \quad M_4 \textbf{ fi}\right.\\
\textbf{fi}
\end{array}\right.\\
\textbf{fi}
\end{array}\right)^{i}
\end{array}\right\}\alpha
$$

a może tak

$$
S\alpha \Rightarrow S^2 : \left\{
\begin{array}{l}
P : \delta := true; n1 := n; \\
M : \left(
\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
K : \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
M_2 : \left(\textbf{if } \kappa \textbf{ then } M_3 \textbf{ else } M_4 \textbf{ fi} \right) \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right)^i ; \\
P : \delta := true; n1 := n; \\
M : \left(
\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
K : \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
M_2 : \left(\textbf{if } \kappa \textbf{ then } M_3 \textbf{ else } M_4 \textbf{ fi} \right) \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right)^i
\end{array}
\right\} \alpha
$$

rozwińmy oznaczenia κ, M_3, M_4

$$
S\alpha \Rightarrow S^2 : \left\{
\begin{array}{l}
P : \delta := true; n1 := n; \\
\left(
\begin{array}{l}
M : \\
\left(
\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
\quad K : \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
\quad M_2 : \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3 : \textbf{if } n1.l = none \textbf{ then } \delta := false \textbf{ else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4 : \textbf{if } n1.r = none \textbf{ then } \delta := false \textbf{ else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right)^i \\
; \\
P : \delta := true; n1 := n; \\
\left(
\begin{array}{l}
M : \\
\left(
\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
\quad K : \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
\quad M_2 : \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3 : \textbf{if } n1.l = none \textbf{ then } \delta := false \textbf{ else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4 : \textbf{if } n1.r = none \textbf{ then } \delta := false \textbf{ else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right)^i
\end{array}
\right\} \alpha
$$

A czy to jeszcze jest tautologia? Chciałbym.

$$
S\alpha \Rightarrow \left\{
\begin{array}{l}
P : \delta := true; n1 := n; \\
M : \left(\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
K : \left(\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
M_2 : \left(\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3 : \textbf{if } n1.l = none \textbf{ then } \left| \begin{array}{l} n1.l := \textbf{new } node(e); \\ n1 := n1.l \end{array} \right. \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4 : \textbf{if } n1.r = none \textbf{ then } \left| \begin{array}{l} n1.r := \textbf{new } node(e); \\ n1 := n1.r \end{array} \right. \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array} \right) \\
\textbf{fi}
\end{array} \right) \\
\textbf{fi}
\end{array} \right)^i ; \\
P : \delta := true; n1 := n; \\
M : \left(\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
K : \left(\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false; result := true \\
\textbf{else} \\
M_2 : \left(\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3 : \textbf{if } n1.l = none \textbf{ then } \left| \begin{array}{l} \delta := false; \\ result := false \end{array} \right. \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4 : \textbf{if } n1.r = none \textbf{ then } \left| \begin{array}{l} \delta := false; \\ result := false \end{array} \right. \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array} \right) \\
\textbf{fi}
\end{array} \right) \\
\textbf{fi}
\end{array} \right)^i
\end{array}
\right\} \alpha
$$

ale tak nie jest ponieważ instrukcję $\delta := false$ zamieniłem na $n1.l := \textbf{new } node(e); n1 := n1.l$

Nie wszystko jest stracone. Zauważmy, że po tej zamianie zachodzi trochę bardziej skomplikowana formuła w której $\alpha : \neg\delta$, w programie M_3 instrukcję $\delta := false$ zastąpiono przez $n1.l :=$ **new** $node(e); n1 := n1.l$ lub ... Wtedy jest tautologią formuła

$$\vdash (S\neg\delta \Rightarrow \left\{ \begin{array}{l} P : \delta := true; n1 := n; \\ M' : \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ K' : \\ \textbf{fi} \end{array} \right)^{i+1} ; \\ P : \delta := true; n1 := n; \\ M'' : \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ K'' : . \\ \textbf{fi} \end{array} \right)^{i+1} \end{array} \right\} \neg\delta)$$

w której program K zastąpiono odpowiednio przez programy K' oraz K''.

$$
S\neg\delta \Rightarrow \left\{
\begin{array}{l}
P : \delta := true; n1 := n; \\
\left(
\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
\quad \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
\quad \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3' : \textbf{if } n1.l = none \textbf{ then } \left[\begin{array}{l} n1.l := \textbf{new } node(e); \\ n1 := n1.l \end{array} \right] \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4' : \textbf{if } n1.r = none \textbf{ then } \left[\begin{array}{l} n1.r := \textbf{new } node(e); \\ n1 := n1.r \end{array} \right] \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right) \quad M_2' : \\
\textbf{fi}
\end{array}
\right) \quad K' : \\
\textbf{fi}
\end{array}
\right)^{i+1} \quad M' : \\
;\\
P : \delta := true; n1 := n; \\
\left(
\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
\quad \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false; result := true \\
\textbf{else} \\
\quad \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3'' : \textbf{if } n1.l = none \textbf{ then } \left[\begin{array}{l} \delta := false; \\ result := false \end{array} \right] \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4'' : \textbf{if } n1.r = none \textbf{ then } \left[\begin{array}{l} \delta := false; \\ result := false \end{array} \right] \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right) \quad M_2'' : \\
\textbf{fi}
\end{array}
\right) \quad K'' : \\
\textbf{fi}
\end{array}
\right)^{i+1} \quad M'' : \\
result
\end{array}
\right\}
$$

Z górnej połówki wyprowadzę insert (tj. pętlę w insert)

posługuję się następującym schematem tautologii: dla każdej liczby naturalnej j i dla dowolnych formuł γ i α oraz dla dowolnych programów N i K jest tautologią poniższa implikacja:

$$\boxed{\vdash N; (\mathbf{if}\ \gamma\ \mathbf{then}\ K\ \mathbf{fi})^j\ (\neg\gamma \wedge \alpha) \Rightarrow N; \mathbf{while}\ \gamma\ \mathbf{do}\ K\ \mathbf{od}\ (\neg\gamma \wedge \alpha)}$$

stosując ten schemat przyjęliśmy: zamiast N, program P,
zamiast γ, formułę δ,
zamiast K, program K' napisany powyżej,
zamiast α, formułę $P; (M'')^{i+1} result$.

W ten sposób udowodniliśmy tautologię

$$
S\neg\delta \Rightarrow \left\{
\begin{array}{l}
P : \delta := true; n1 := n; \\
\left(
\begin{array}{l}
\textbf{while } \delta \\
\textbf{do} \\
\quad \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
\quad \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3' : \textbf{if } n1.l = none \textbf{ then } \left[\begin{array}{l} n1.l := \textbf{new } node(e); \\ n1 := n1.l \end{array} \right] \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4' : \textbf{if } n1.r = none \textbf{ then } \left[\begin{array}{l} n1.r := \textbf{new } node(e); \\ n1 := n1.r \end{array} \right] \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right)_{M_2'} \\
\textbf{fi}
\end{array}
\right)_{K'} \\
\textbf{od}
\end{array}
\right)_{M'} ; \\
P : \delta := true; n1 := n; \\
\left(
\begin{array}{l}
\textbf{if } \delta \\
\textbf{then} \\
\quad \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false; result := true \\
\textbf{else} \\
\quad \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3'' : \textbf{if } n1.l = none \textbf{ then } \left[\begin{array}{l} \delta := false; \\ result := false \end{array} \right] \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4'' : \textbf{if } n1.r = none \textbf{ then } \left[\begin{array}{l} \delta := false; \\ result := false \end{array} \right] \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right)_{M_2''} \\
\textbf{fi}
\end{array}
\right)_{K''} \\
\textbf{fi}
\end{array}
\right)_{M''}^{i+1}
\end{array}
\right\}_{result}
$$

Potem ponownie korzystamy z tego samego schematu by otrzymać

$$
S\neg\delta \Rightarrow \left\{
\begin{array}{l}
P : \delta := true; n1 := n; \\
M : \left(
\begin{array}{l}
\textbf{while } \delta \\
\textbf{do} \\
\quad K : \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false \\
\textbf{else} \\
\quad M_2 : \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3 : \textbf{if } n1.l = none \textbf{ then } \left|
\begin{array}{l}
n1.l := \textbf{new } node(e); \\
n1 := n1.l
\end{array}
\right. \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4 : \textbf{if } n1.r = none \textbf{ then } \left|
\begin{array}{l}
n1.r := \textbf{new } node(e); \\
n1 := n1.r
\end{array}
\right. \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right) \\
\textbf{od}
\end{array}
\right) ;
\end{array}
\right\} ;
$$

$$
\left\{
\begin{array}{l}
P : \delta := true; n1 := n; \\
M : \left(
\begin{array}{l}
\textbf{while } \delta \\
\textbf{do} \\
\quad K : \left(
\begin{array}{l}
\textbf{if } \gamma : e = n1.v \\
\textbf{then} \\
M_1 : \delta := false; result := true \\
\textbf{else} \\
\quad M_2 : \left(
\begin{array}{l}
\textbf{if } \kappa : e < n1.v \\
\textbf{then} \\
M_3 : \textbf{if } n1.l = none \textbf{ then } \left|
\begin{array}{l}
\delta := false; \\
result := false
\end{array}
\right. \\
\qquad \textbf{else } n1 := n1.l \textbf{ fi} \\
\textbf{else} \\
M_4 : \textbf{if } n1.r = none \textbf{ then } \left|
\begin{array}{l}
\delta := false; \\
result := false
\end{array}
\right. \\
\qquad \textbf{else } n1 := n1.r \textbf{ fi} \\
\textbf{fi}
\end{array}
\right) \\
\textbf{fi}
\end{array}
\right) \\
\textbf{od}
\end{array}
\right) \\
\end{array}
\right\} result
$$

Zauważmy, że programy występujące w następniku implikacji to są treści procedury insert oraz

funkcji boolowskiej member. Mogę więc zastosować aksjomat regułę kopii
procedure $m(f1, f2)Body\_m$**end**m

$$\boxed{\text{[call m(x,y)]}\alpha \Leftrightarrow [f1 := x; f2 := y; Body\_m]\alpha.}$$

A więc dla każdego $i \in N$ jest tautologią

$$\left\{ \begin{array}{l} P: \delta := true;\ n1 := n; \\ \qquad \left(\begin{array}{l} \textbf{if } \delta \\ \textbf{then} \\ \qquad \left\{ \begin{array}{l} \textbf{if } \gamma : e = n1.v \\ \textbf{then} \\ M_1 : \delta := false \\ \textbf{else} \\ \qquad \left\{ \begin{array}{l} \textbf{if } \kappa : e < n1.v \\ \textbf{then} \\ M_3 : \textbf{if } n1.l = none \textbf{ then } \delta := false \\ \qquad \textbf{else } n1 := n1.l \textbf{ fi} \\ \textbf{else} \\ M_4 : \textbf{if } n1.r = none \textbf{ then } \delta := false \\ \qquad \textbf{else } n1 := n1.r \textbf{ fi} \\ \textbf{fi} \end{array} \right. \\ M_2 : \\ \textbf{fi} \end{array} \right. \\ \textbf{fi} \end{array} \right)^i \\ M: \end{array} \right\} \neg\delta$$

$$\Rightarrow [insert(e,n)]member(e,n)$$

Teraz posłużę się regułą R3, regułą ω o przeliczalnej liczbie przesłanek

$$\boxed{\frac{\{P; (\textbf{if } \delta \textbf{ then } K \textbf{ fi})^i \neg\delta \Rightarrow insert(e,n); member(e,n)result\}_{i\in N}}{P; \textbf{while } \delta \textbf{ do } K \textbf{ od } \neg\delta \Rightarrow insert(e,n); member(e,n)result}}$$

i otrzymam

$$\vdash P; \textbf{while } \delta \textbf{ do } K \textbf{ od } \neg\delta \Rightarrow [insert(e,n)]member(e,n)$$

Formuła $P; \textbf{while } \delta \textbf{ do } K \textbf{ od } \neg\delta$ jest aksjomatem drzew BST, to oznacza, że

$$BST \vdash [insert(e,n)mamber(e,n)$$

co czytamy: w każdym modelu struktury drzew BST wykonanie operacji insert(e,n) jest udane (tj. bez zapętlenia i bez podniesienia wyjątku) i potem zachodzi member(e,n).

7. Zakończenie

Tu napisać ....