Programowanie

Egzamin — rozwiązania

25 czerwca 2001

Fragmenty napisane kursywą i oznaczone „Komentarz do zadania” nie są częścią rozwiązania zadań, lecz stanowią dodatkowe objaśnienia. Przedstawione niżej rozwiązania wykładowca uważa za wzorowe. Nie oznacza to jednak, że wykładowca uważa je za jedynie słuszne.

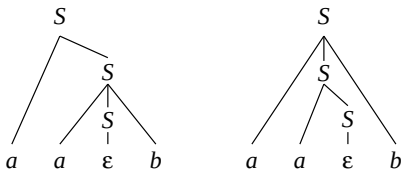
Zadanie 1. Niech $L(G)$ oznacza język generowany przez gramatykę G. Pokażemy wpierw, że $L(G) \subseteq L$. Ponieważ dane słowo należy do języka generowanego przez gramatykę wtedy i tylko wtedy, gdy istnieje drzewo wyprowadzenia tego słowa, to wystarczy pokazać, że dla każdego drzewa wyprowadzenia t, jeśli drzewo to jest drzewem wyprowadzenia słowa u, to $u \in L$, tj. $u = a^n b^m$ dla pewnych $n, m \in \mathbb{N}$, takich, że $n \geq m$. Dowód tego faktu przeprowadzimy przez indukcję względem wysokości drzewa t. Podstawa indukcji: jedynym drzewem wysokości 1 jest

Ponieważ $\epsilon = a^0 b^0$, więc $u \in L$. Krok indukcyjny: przypuśćmy, że dla każdego drzewa wyprowadzenia t wysokości k, jeśli t jest drzewem wyprowadzenia słowa u, to $u \in L$. Niech t' będzie drzewem wyprowadzenia wysokości $k+1$ i niech będzie ono drzewem wyprowadzenia pewnego słowa v. Pokażemy, że $v \in L$. Skoro wysokość drzewa t' wynosi $k+1 > 1$, więc w korzeniu tego drzewa może wystąpić jedynie produkcja $S \to aSb$ lub $S \to aS$. Oba przypadki są przedstawione na poniższym rysunku:

gdzie t jest drzewem wyprowadzenia wysokości k. Na mocy założenia indukcyjnego słowo, którego drzewem wyprowadzenia jest t jest postaci $a^n b^m$ dla pewnych $n \geq m \geq 0$. Zatem w pierwszym przypadku $v = a(a^n b^m) = a^{n+1} b^m$, w drugim zaś $v = a(a^n b^m)b = a^{n+1} b^{m+1}$. W obu przypadkach słowo v jest żądanej postaci, należy więc do języka L.

Pokażemy teraz, że $L \subseteq L(G)$. Rozważmy słowo $u = a^n b^m \in L$ dla pewnych $n \geq m \geq 0$. Pokażemy, że $u \in L$, tj. że $S \stackrel{*}{\Rightarrow}_G u$. Dowód przeprowadzimy przez indukcję względem n. Podstawa indukcji: dla $n = 0$ słowo $a^0 b^0 = \epsilon$ można wyprowadzić z gramatyki G, zawiera ona bowiem produkcję $S \to \epsilon$. Krok indukcyjny: przypuśćmy, że dla ustalonego $n \geq 0$ wszystkie słowa postaci $a^n b^m$, dla $m \leq n$ można wyprowadzić z gramatyki G. Rozważmy słowo $u = a^{n+1} b^m$. Jeśli $m = n+1$, to $u = a(a^n b^n)b$. Z założenia indukcyjnego $S \stackrel{*}{\Rightarrow}_G a^n b^n$. Zatem $S \Rightarrow_G aSb \stackrel{*}{\Rightarrow}_G a(a^n b^n)b = u$. Jeśli natomiast $m < n+1$, to $u = a(a^n b^m)$, przy czym $m \leq n$, więc na mocy założenia indukcyjnego $S \stackrel{*}{\Rightarrow}_G a^n b^m$. Zatem $S \Rightarrow_G aS \stackrel{*}{\Rightarrow}_G a(a^n b^m) = u$.

Powyższa gramatyka jest niejednoznaczna, gdyż np. słowo aab ma dwa różne drzewa wyprowadzenia:



Jednoznaczną gramatyką generującą ten język jest gramatyka $G' = \langle \Sigma, V', S, P' \rangle$, gdzie $V' = \{S, T\}$ i

$$P' = \{S \to aS, S \to T, T \to aTb, T \to \epsilon\}$$

Zadanie 2.

$$
\begin{aligned}
F^{\mathfrak{M}} &= 0 \\
T^{\mathfrak{M}} &= 1 \\
not^{\mathfrak{M}}(0) &= 1 \\
not^{\mathfrak{M}}(1) &= 0 \\
nil^{\mathfrak{M}} &= \epsilon \\
cons^{\mathfrak{M}}(b,c) &= \langle b,c\rangle \\
hd^{\mathfrak{M}}(\langle b,c\rangle) &= b \\
hd^{\mathfrak{M}}(\epsilon) &= 0 \\
tl^{\mathfrak{M}}(\langle b,c\rangle) &= c \\
tl^{\mathfrak{M}}(\epsilon) &= \epsilon \\
appn^{\mathfrak{M}}(\epsilon,c) &= c \\
appn^{\mathfrak{M}}(\langle b,c\rangle,c') &= \langle b, appn^{\mathfrak{M}}(c,c')\rangle \\
rev^{\mathfrak{M}}(\epsilon) &= \epsilon \\
rev^{\mathfrak{M}}(\langle b,c\rangle) &= appn^{\mathfrak{M}}(rev^{\mathfrak{M}}(c), \langle b,\epsilon\rangle) \\
inv^{\mathfrak{M}}(\epsilon) &= \epsilon \\
inv^{\mathfrak{M}}(\langle 0,c\rangle) &= \langle 1, inv^{\mathfrak{M}}(c)\rangle \\
inv^{\mathfrak{M}}(\langle 1,c\rangle) &= \langle 0, inv^{\mathfrak{M}}(c)\rangle \\
null^{\mathfrak{M}}(\epsilon) &= 1 \\
null^{\mathfrak{M}}(\langle b,c\rangle) &= 0
\end{aligned}
$$

Zadanie 3.

$$\frac{}{F \to F} \qquad \frac{}{T \to T} \qquad \frac{}{nil \to nil} \qquad \frac{e_1 \to e_1' \quad e_2 \to e_2'}{cons(e_1,e_2) \to cons(e_1',e_2')}$$

$$\frac{e \to T}{not(e) \to F} \qquad \frac{e \to F}{not(e) \to T} \qquad \frac{e \to cons(e_1,e_2)}{hd(e) \to e_1} \qquad \frac{e \to nil}{hd(e) \to F} \qquad \frac{e \to cons(e_1,e_2)}{tl(e) \to e_2} \qquad \frac{e \to nil}{tl(e) \to nil}$$

$$\frac{e_1 \to nil \quad e_2 \to e_2'}{appn(e_1,e_2) \to e_2'} \qquad \frac{e_1 \to cons(e_1',e_1'') \quad appn(e_1'',e_2) \to e_2'}{appn(e_1,e_2) \to cons(e_1',e_2')}$$

$$\frac{e \to nil}{rev(e) \to nil} \qquad \frac{e \to cons(e_1,e_2) \quad appn(rev(e_2), cons(e_1,nil)) \to e'}{rev(e) \to e'}$$

$$\frac{e \to nil}{inv(e) \to nil} \qquad \frac{e \to cons(T,e') \quad inv(e') \to e''}{inv(e) \to cons(F,e'')} \qquad \frac{e \to cons(F,e') \quad inv(e') \to e''}{inv(e) \to cons(T,e'')}$$

$$\frac{e \to nil}{null(e) \to T} \qquad \frac{e \to cons(e_1,e_2)}{null(e) \to F}$$

Zadanie 4. Semantyka algebraiczna:

$$
\begin{aligned}
not(T) &= F && (1)\\
not(F) &= T && (2)\\
hd(cons(x,l)) &= x && (3)\\
hd(nil) &= F && (4)\\
hd(cons(x,l)) &= l && (5)\\
hd(nil) &= nil && (6)\\
appn(nil,l) &= l && (7)\\
appn(cons(x,l_1),l_2) &= cons(x, appn(l_1,l_2)) && (8)
\end{aligned}
$$

$$
\begin{aligned}
rev(nil) &= nil && (9)\\
rev(cons(x,l)) &= appn(rev(l), cons(x, nil)) && (10)\\
inv(nil) &= nil && (11)\\
inv(cons(x,l)) &= cons(not(x), inv(l)) && (12)\\
null(nil) &= T && (13)\\
null(cons(x,l)) &= F && (14)
\end{aligned}
$$

gdzie $x : obj$ i $l, l_1, l_2 : list$. Konstruktorami są stałe F, T i nil oraz symbol $cons$. Symbole hd i tl są destruktorami, not, $appn$, rev i inv operatorami, $null$ zaś obserwatorem. Zasada indukcji dla gatunku $list$:

$$\frac{\Phi(nil) \quad \Phi(l) \Rightarrow \Phi(cons(x,l))}{\Phi(l')}$$

gdzie $l, l' : list$ i $x : obj$. Korzystając z powyższej zasady indukcji udowodnimy lemat:

$$inv(appn(l_1, l_2)) = appn(inv(l_1), inv(l_2)) \quad (15)$$

Dowód przez indukcję względem l_1. Podstawa indukcji: $l_1 = nil$. Wówczas

$$L = inv(appn(nil, l_2)) \overset{(7)}{=} inv(l_2) \overset{(7)}{=} appn(nil, inv(l_2)) \overset{(11)}{=} appn(inv(nil), inv(l_2)) = P$$

Krok indukcyjny: załóżmy, że

$$inv(appn(l_1, l_2)) = appn(inv(l_1), inv(l_2)) \quad (16)$$

Pokażemy, że

$$inv(appn(cons(x, l_1), l_2)) = appn(inv(cons(x, l_1)), inv(l_2))$$

Istotnie,

$$L = inv(appn(cons(x, l_1), l_2)) \overset{(8)}{=} inv(cons(x, appn(l_1, l_2))) \overset{(12)}{=}$$
$$cons(not(x), inv(appn(l_1, l_2))) \overset{(16)}{=} cons(not(x), appn(inv(l_1), inv(l_2))) \overset{(8)}{=}$$
$$appn(cons(not(x), inv(l_1)), inv(l_2))) \overset{(12)}{=} appn(inv(cons(x, l_1)), l_2) = P$$

Z pomocą powyższego lematu i zasady indukcji udowodnimy twierdzenie:

$$inv(rev(l)) = rev(inv(l))$$

Podstawa indukcji: $l = nil$. Wówczas

$$L = inv(rev(nil)) \overset{(9)}{=} inv(nil) \overset{(11)}{=} nil \overset{(9)}{=} rev(nil) \overset{(11)}{=} rev(inv(nil)) = P$$

Krok indukcyjny: załóżmy, że

$$inv(rev(l)) = rev(inv(l)) \quad (17)$$

Pokażemy, że

$$inv(rev(cons(x,l))) = rev(inv(cons(x,l)))$$

Faktycznie, mamy

$$L = inv(rev(cons(x,l))) \overset{(10)}{=} inv(appn(rev(l), cons(x, nil))) \overset{(15)}{=}$$
$$appn(inv(rev(l)), inv(cons(x, nil))) \overset{(17)}{=} appn(rev(inv(l)), inv(cons(x, nil))) \overset{(12)}{=}$$
$$appn(rev(inv(l)), cons(not(x), inv(nil))) \overset{(11)}{=} appn(rev(inv(l)), cons(not(x), nil)) \overset{(10)}{=}$$
$$rev(cons(not(x), inv(l))) \overset{(11)}{=} rev(inv(cons(x,l))) = P$$

Zadanie 5. 1. [*Komentarz do zadania: Rozważmy język programowania, w którym wszystkie procedury są globalne (np. język C). Wówczas zmienne występujące w treści procedury są dwojakiego rodzaju: mogą to być albo zmienne lokalne, zadeklarowane w danej procedurze, albo zmienne globalne (zadeklarowane poza treścią procedur). Kompilator projektuje strukturę rekordu aktywacji każdej procedury i statycznie ustala położenie zmiennych lokalnych względem początku tego rekordu. W czasie biegu programu bezwzględny adres zmiennej lokalnej oblicza się na postawie adresu względnego i adresu początku rekordu aktywacji zwanego linkiem sterowania. Zmienne globalne są przechowywane w segmencie danych statycznych. Struktura tego segmentu jest ustalana w czasie kompilacji i nie ulega zmianie w trakcie biegu programu (zmienne globalne istnieją przez cały czas pracy programu). W czasie biegu programu bezwzględny adres zmiennej globalnej oblicza się na postawie jej położenia w segmencie danych statycznych i adresu początku tego segmentu. Jeżeli wolno deklarować procedury lokalne, sytuacja znacznie się komplikuje.*]

W treści procedur lokalnych mogą pojawiać się zmienne zadeklarowane wewnątrz procedur otoczających te procedury swym zasięgiem. Zmienne te nie są ani lokalne w treści danej procedury (zmienne lokalne łatwo zlokalizować w rekordzie aktywacji znajdującym się na szczycie stosu), nie są też globalne (zmienne globalne również łatwo zlokalizować, są one bowiem przechowywane w segmencie danych statycznych, którego struktura nie zmienia się podczas wykonania programu). W trakcie biegu programu znajdują się one w rekordach aktywacji procedur wywołanych przed wywołaniem danej procedury. Aby w trakcie działania programu można było łatwo wyznaczyć adresy rekordów aktywacji zawierających takie zmienne, każdy rokord aktywacji posiada dodatkowe pole, zwane *linkiem dostępu*. Jest to adres rekordu aktywacji procedury, która w treści programu bezpośrednio otacza daną procedurę, tj. procedury, w której dana procedura jest lokalna. Algorytm wyznaczania i przetwarzania linków dostępu jest następujący. W trakcie kompilowania procedury z każdą zmienną występującą w jej treści i z każdą procedurą wywoływaną w jej treści związuje się tzw. *indeks odniesienia*, tj. liczbę całkowitą, określającą liczbę poziomów zagnieżdżenia, które dzielą miejsce deklaracji tej zmiennej (procedury) i miejsce jej użycia (wywołania). Indeks odniesienia zmiennych lokalnych w danej procedurze wynosi zero, indeks zmiennych zadeklarowanych w procedurze bezpośrednio otaczającej daną procedurę wynosi jeden itd. Indeks odniesienia procedury, która jest zadeklarowana jako lokalna w danej procedurze wynosi 0, indeks odniesienia procedury zadeklarowanej na tym samym poziomie zagnieżdżenia (w szczególności indeks odniesienia procedury w niej samej) wynosi 1, indeks odniesienia procedury, dla której dana procedura jest lokalna wynosi 2 itd.

Niech indeksem odniesienia zmiennej x w procedurze P będzie $k \geq 0$. Aby zlokalizować rekord aktywacji zawierający tę zmienną, należy w czasie biegu procedury P wykonać następujące czynności:

$a :=$ adres rekordu aktywacji procedury P;
wykonaj k razy:
    $a := a\uparrow.$*link_dostępu*;
a jest adresem rekordu zawierającego zmienną x;

gdzie $a\uparrow.$*link_dostępu* oznacza zawartość pola *link_dostępu* rekordu aktywacji a.

Podczas wywołania procedury strona wywołująca musi zainicjować link dostępu w nowotworzonym rekordzie aktywacji. Niech indeks odniesienia procedury Q wywoływanej w procedurze P wynosi k. Przed przekazaniem sterowania do procedury Q procedura P wykonuje następujące czynności:

$a :=$ adres rekordu aktywacji procedury P;
wykonaj k razy:
    $a := a\uparrow.$*link_dostępu*;
wstaw a jako link dostępu do rekordu aktywacji procedury P;

Jako link dostępu procedur globalnych przyjmuje się adres segmentu danych statycznych (zadeklarowanych na poziomie zagnieżdżenia równym zero).

2. Rozważmy program przedstawiony w tablicy 3 w języku Pascal rozszerzonym o możliwość przekazywania procedur lokalnych jako parametrów. Ponieważ procedura Q jest parametrem formalnym procedury P, jej indeks odniesienia nie jest stały i zależy od tego, z jakim parametrem faktycznym jest związana w danym wywołaniu nazwa Q. Podczas wywołania $P(Q1)$ wynosi on 2, podczas wywołania $P(Q2)$ zaś 1. Algorytm statycznego wyznaczania indeksów odniesienia jest więc w tym przypadku niewystarczający.

```
var x;

procedure Q1;
begin
    x := 1;
end;

procedure R;
    var y;
    procedure Q2;
    begin
        x := y
    end;
    procedure P (procedure Q);
    begin
        Q
    end;
begin
    y := 2;
    P(Q1);
    P(Q2)
end
```

Tablica 3: Fragment programu do zadania 5.2

[*Komentarz do zadania: Jeżeli ograniczymy, tak jak np. w Pascalu, przekazywanie procedur jako parametrów jedynie do procedur globalnych, to wówczas algorytmu ustalania linków dostępu nie trzeba modyfikować. Tworząc rekordy aktywacji procedur przekazanych jako parametr, jako link dostępu należy zawsze przyjmować adres segmentu danych statycznych.*]

3. Istotnie, problem przedstawiony w poprzednim punkcie polega na tym, że nie można poprawnie wyznaczyć linku dostępu procedury Q przekazanej jako parametr do procedury P. Właściwym otoczeniem procedury Q jest bowiem nie miejsce jej wywołania w treści procedury P, lecz miejsce, w którym nazwa Q została związana z parametrem faktycznym (procedurą $Q1$ bądź $Q2$) w chwili wywołania procedury P. Dlatego link dostępu należy wyznaczyć nie w chwili wywołania procedury Q (tworzenia jej własnego rekordu aktywacji), lecz wcześniej, w chwili wiązania parametrów procedury P (w chwili tworzenia rekordu aktywacji procedury P). Link ten należy wyznaczyć tak, jakby odpowiedni parametr faktyczny (procedura $Q1$ bądź $Q2$) była w tym miejscu wywołana. Do procedury P należy przekazać tzw. *domknięcie funkcji*, tj. ten link dostępu wraz z adresem kodu procedury Q. Podczas wywołania procedury Q wewnątrz procedury P do nowotworzonego rekordu aktywacji procedury Q należy skopiować link dostępu z jej domknięcia.

4. Rozważmy następujący program w SML-u, języku, w którym można przekazywać procedury jako wyniki innych procedur:

```
fun f () =
   let
      val x = ref 0;
      fun g () = x := !x + 1
   in
      g
   end
val h = f ()
```

Na skutek wykonania tego ciągu deklaracji z nazwą `h` zostanie związana funkcja `g` zadeklarowana wewnątrz

```
program Zadanie_6_2 (Output);
    function P (var x : integer;  function e : integer) : integer;
    begin
        x := x + 1;
        P := e
    end; (* P *)

    var z : integer;

    function e : integer;
        begin
        e := z * z + 1
        end;
begin
    write(P(z, e))
end.
```

Tablica 4: Tłumaczenie wywołania przez nazwę na kombinację wywołania przez zmienną i procedurę

funkcji `f`. Funkcja `g` odwołuje się do zmiennej `x` umieszczonej w rekordzie aktywacji funkcji `f`. Jeżeli rekord aktywacji funkcji `f` zostałby odłożony na stos i usunięty z chwilą zakończenia pracy funkcji `f`, to funkcja `g` odwoływałaby się do obszaru pamięci, który został już usunięty. Statycznie nie sposób ustalić, jak długo rekord aktywacji funkcji `f` będzie jeszcze „potrzebny". Dlatego rekordów aktywacji nie można umieszczać na stosie.

Zadanie 6. 1. Niech parametr formalny x procedury P będzie przekazywany przez nazwę. Wywołanie $P(e)$ procedury P z parametrem faktycznym e, który może być wyrażeniem, ma dokładnie takie znaczenie, jak gdyby:

1. dokonano zmiany nazw wszystkich zmiennych lokalnych zadeklarowanych w procedurze P na takie, które nie występują nigdzie indziej w programie (aby nie dochodziło do kolizji nazw zmiennych lokalnych i zmiennych występujących w parametrze faktycznym e),
2. wstawiono tekst wyrażenia e (ujmując je tam, gdzie to składniowo możliwe, w nawiasy) w każde miejsce wystąpienia parametru formalnego x treści procedury P,
3. wykonano tak zmodyfikowaną treść procedury P.

Program przedstawiony w tablicy 1 wypisze liczbę 10. Jeżeli przynajmniej jeden z parametrów x i e jest przekazywany przez wartość, wówczas zostanie wypisana liczba 5.

2. Istotnie, podczas przekazywania parametru x przez nazwę należy sprawdzić, czy w treści procedury wykorzystuje się tylko jego R-wartość, czy także L-wartość. W drugim przypadku odpowiadający mu parametr faktyczny musi posiadać L-wartość. Jeśli jest to zmienna prosta, wówczas znaczenie programu jest dokładnie takie, jakby zmienna ta była przekazana przez zmienną (referencję). Jeśli zaś parametr faktyczny e nie posiada L-wartości, np. jest wyrażeniem, to w treści procedury mogą występować jedynie odwołania do jego R-wartości. Wówczas znaczenie programu jest takie samo, jak gdyby przed wywołaniem danej procedury utworzono dodatkową, bezparametrową procedurę funkcyjną obliczającą wartość wyrażenia e i przekazaną ją jako parametr do treści funkcji. Tłumaczenie programu z tablicy 1 jest przedstawione w tablicy 4.

[*Komentarz do zadania: Opisane wyżej tłumaczenie nie jest poprawne w przypadku przekazywania przez nazwę zmiennych indeksowanych. Zmienne indeksowane posiadają L-wartości, zgodnie z powyższym tłumaczeniem powinny być więc przekazywane przez zmienną. Jednak np. przy przekazywaniu zmiennej $A[i]$ przez nazwę, wartość indeksu i jest wyznaczana w chwili odwołania do zmiennej, podczas gdy przy przekazywaniu przez zmienną — podczas wiązania parametru faktycznego z formalnym.*]

Zadanie 7. 1. Program w SML-u:

```
fun mrgsort _ [] = []
  | mrgsort _ [x] = [x]
  | mrgsort op< ss =
      let
         fun revapp (x::xs,ys) = revapp(xs,x::ys)
           | revapp (nil,ys) = ys
         fun merge (xs as x::xs', ys as y::ys', zs) =
               if x<y
                  then merge (xs',ys,x::zs)
                  else merge (xs,ys',y::zs)
           | merge (xs,[],zs) = revapp(zs,xs)
           | merge ([],ys,zs) = revapp(zs,ys)
         fun split (xs,ys,z1::z2::zs) = split (z1::xs,z2::ys,zs)
           | split (xs,ys,zs) =
                merge (mrgsort op< (zs@xs), mrgsort op< ys, nil)
      in
         split ([],[],ss)
      end
```

2. Program w Prologu:

```
split([H1,H2|T],[H1|X],[H2|Y]) :-
   split(T,X,Y).
split([H],[H],[]).
split([],[],[]).

mrg([H1|T1],[H2|T2],[H1|X]) :-
   H1<H2,
   mrg(T1,[H2|T2],X).
mrg([H1|T1],[H2|T2],[H2|X]) :-
   H1>=H2,
   mrg([H1|T1],T2,X).
mrg(X,[],X).
mrg([],X,X).

mrgsort([],[]) :- !.
mrgsort([X],[X]) :- !.
mrgsort(X,Y) :-
   split(X,X1,X2),
   mrgsort(X1,Y1),
   mrgsort(X2,Y2),
   mrg(Y1,Y2,Y).
```

Zadanie 8. Rozważmy zbiór równości (1), (2), (3). Na mocy aksjomatu (3) wyrażenia różniące się jedynie rozstawieniem nawiasów są równoważne w sensie relacji $\sim$. Aksjomaty (1) i (2) mówią natomiast, że dodanie zera w dowolnym miejscu wyrażenia nie zmienia jego wartości (nie wyprowadza poza klasę abstrakcji relacji $\sim$). Zatem relacja $\sim$ utożsamia wszystkie wyrażenia, które zawierają tyle samo jedynek. Ponieważ $[0]_\sim$ jest zbiorem wszystkich wyrażeń nie zawierających stałej 1, $[1]_\sim$ — zbiorem wszystkich wyrażeń stałych zawierających dokładnie jedno wystąpienie stałej 1, zaś liczba wystąpień stałej 1 w wyrażeniu e_1 + e_2 jest sumą liczby wystąpień stałej 1 w wyrażeniach e_1 i e_2, więc algebra $\mathfrak{P} = \langle \mathcal{T}(\Sigma, \mathcal{X})/_\sim, \cdot^{\mathfrak{P}} \rangle$ jest izomorficzna z algebrą liczb naturalnych z zerem, jedynką i zwykłym dodawaniem. Formalny dowód tego faktu przedstawimy w dalszej części rozwiązania.

Rozważmy teraz zbiór równości (2), (3). Ponieważ brakuje w tym przypadku aksjomatu (1), który pozwala dopisać/usunąć zero z lewej strony wyrażenia, to np. termy $0 + 1$ i 1 nie są $\sim$-równoważne. Wszystkie klasy abstrakcji z poprzedniego przypadku, z wyjątkiem $[0]_\sim$, rozpadają się na dwie części — na zbiory termów zawierające zero z lewej strony i nie zawierające zera. Zbiór równości (2), (3) definiuje więc algebrę $\mathfrak{A}$ o nośniku $\{0\} \cup \{0,1\} \times (\mathbb{N} \setminus \{0\})$, w której

$$\begin{aligned}
0^{\mathfrak{A}} &= 0 \\
1^{\mathfrak{A}} &= \langle 1, 1\rangle \\
+^{\mathfrak{A}}(b, 0) &= b \\
+^{\mathfrak{A}}(0, \langle c, a\rangle) &= \langle 0, a\rangle, \quad c \in \{0,1\}, a \in \mathbb{N} \setminus \{0\} \\
+^{\mathfrak{A}}(\langle 0, a_1\rangle, \langle c, a\rangle) &= \langle 0, a_1 + a_2\rangle, \quad c \in \{0,1\}, a_1, a_2 \in \mathbb{N} \setminus \{0\} \\
+^{\mathfrak{A}}(\langle 1, a_1\rangle, \langle c, a\rangle) &= \langle 1, a_1 + a_2\rangle, \quad c \in \{0,1\}, a_1, a_2 \in \mathbb{N} \setminus \{0\}
\end{aligned}$$

Zbadajmy teraz zbiór złożony z pojedynczej równości (3). W tym przypadku relacja $\sim$ utożsamia termy różniące się ustawieniem nawiasów, jednak liczba i kolejność zarówno zer, jak i jedynek we wszystkich równoważnych termach musi być taka sama. Aksjomat łączności (3) definiuje więc algebrę niepustych, skończonych słów nad alfabetem $\{0,1\}$, w której interpretacją 0 jest słowo 0, interpretacją 1 słowo 1, interpretacją $+$ zaś operacja konkatenacji słów.

Jeśli oprócz równości (3) skorzystamy także z aksjomatu przemienności (4), wówczas nie tylko rozstawienie nawiasów, lecz także kolejność zer i jedynek w $\sim$-równoważnych wyrażeniach może być dowolna. Natomiast liczba zarówno zer, jak i jedynek we wszystkich wyrażeniach należących do jednej klasy abstrakcji jest stała. Zbiór równości (3), (4) definiuje więc algebrę niezerowych dwuwymiarowych wektorów $\mathbb{N} \times \mathbb{N} \setminus \{\langle 0,0\rangle\}$, w której 0 i 1 są wektorami jednostkowymi $\langle 1,0\rangle$ i $\langle 0,1\rangle$, zaś operacja $+$ jest dodawaniem wektorów (po współrzędnych).

Zauważmy, że jeśli do zbioru (3), (4) dołączymy przynajmniej jeden z aksjomatów (1) lub (2), to na powrót otrzymamy definicję zbioru liczb naturalnych.

Zdefiniujemy teraz izomorfizm $\phi : \mathfrak{P} \to \mathfrak{N}$.

Niech napis $n \cdot 1$ oznacza term

$$0 + \underbrace{1 + \ldots + 1}_{n \text{ razy}}$$

zaś $|e|_s$ — liczbę symboli $s \in \mathcal{X} \cup \{0,1\}$ w termie e, tj.

$$\begin{aligned}
|x|_s &= \begin{cases} 1, & \text{gdy } s = x \\ 0, & \text{w p.p} \end{cases} \\
|0|_s &= \begin{cases} 1, & \text{gdy } s = 0 \\ 0, & \text{w p.p} \end{cases} \\
|1|_s &= \begin{cases} 1, & \text{gdy } s = 1 \\ 0, & \text{w p.p} \end{cases} \\
|e_1 + e_2|_s &= |e_1|_s + |e_2|_s
\end{aligned}$$

Lemat 1. $n_1 \cdot 1 + n_2 \cdot 1 \sim (n_1 + n_2) \cdot 1$, dla dowolnych liczb $n_1, n_2 \geq 0$.

Dowód. Dowód przeprowadzimy przez indukcję względem n_2. Podstawa indukcji: $n_2 = 0$. Ponieważ

$$\dfrac{\dfrac{}{x + 0 = x}\,(2) \quad \dfrac{}{n_1 \cdot 1 = n_1 \cdot 1}\,(\text{Refl})}{n_1 \cdot 1 + 0 = n_1 \cdot 1}\,(\text{Mon})$$

czyli $n_1 \cdot 1 + 0 \sim n_1 \cdot 1$, więc $n_1 \cdot 1 + n_2 \cdot 1 \equiv n_1 \cdot 1 + 0 \sim n_1 \cdot 1 \equiv (n_1 + n_2) \cdot 1$. Krok indukcyjny: przypuśćmy, że $n_1 \cdot 1 + n_2 \cdot 1 \sim (n_1 + n_2) \cdot 1$. Pokażemy, że $n_1 \cdot 1 + (n_2 + 1) \cdot 1 \sim (n_1 + n_2 + 1) \cdot 1$. Istotnie, $(n_2 + 1) \cdot 1 \equiv n_2 \cdot 1 + 1$ i $(n_1 + n_2 + 1) \cdot 1 \equiv (n_1 + n_2) \cdot 1 + 1$, oraz

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(x + y) + z = x + (y + z)}\,(3) \quad \dfrac{}{n_1 \cdot 1 = n_1 \cdot 1}\,(\text{Refl})}{(n_1 \cdot 1 + y) + z = n_1 \cdot 1 + (y + z)}\,(\text{Mon}) \quad \dfrac{}{n_2 \cdot 1 = n_2 \cdot 1}\,(\text{Refl})}{(n_1 \cdot 1 + n_2 \cdot 1) + z = n_1 \cdot 1 + (n_2 \cdot 1 + z)}\,(\text{Mon}) \quad \dfrac{}{1 = 1}\,(\text{Refl})}{(n_1 \cdot 1 + n_2 \cdot 1) + 1 = n_1 \cdot 1 + (n_2 \cdot 1 + 1)}\,(\text{Mon})$$

więc $n_1 \cdot 1 + (n_2 + 1) \cdot 1 \equiv n_1 \cdot 1 + (n_2 \cdot 1 + 1) \sim (n_1 \cdot 1 + n_2 \cdot 1) + 1 \sim (n_1 + n_2) \cdot 1 + 1 \equiv (n_1 + n_2 + 1) \cdot 1$.

Twierdzenie 2. Dla każdego termu stałego e istnieje $n \geq 0$, takie, że $e \sim n \cdot 1$.

Dowód. Przez indukcję względem struktury termu e. Jeżeli $e \equiv 0$, to $n = 0$, bowiem $0 \cdot 1 \equiv 0$. Jeśli $e \equiv 1$, to ponieważ

$$\dfrac{\dfrac{}{0 + x = x}\ (1) \quad \dfrac{}{1 = 1}\ (\text{Refl})}{0 + 1 = 1}\ (\text{Mon})$$

więc $1 \sim 0 + 1 \equiv 1 \cdot 1$ i $n = 1$. Przypuśćmy teraz, że $e \equiv e_1 + e_2$ i załóżmy, że istnieją liczby $n_1, n_2 \geq 0$, takie, że $e_1 \sim n_1 \cdot 1$ i $e_2 \sim n_2 \cdot 1$. Na mocy lematu 1 mamy $e_1 + e_2 \sim n_1 \cdot 1 + n_2 \cdot 1 \sim (n_1 + n_2) \cdot 1$.

Lemat 3. Jeżeli $x \neq s$, przy czym $x \in \mathcal{X}$ i $s \in \mathcal{X} \cup \{0, 1\}$, to $|e_1[x/e_2]|_s = |e_1|_s + |e_1|_x \times |e_2|_s$ dla dowolnych termów e_1 i e_2. Ponadto $|e_1[x/e_2]|_x = |e_1|_x \times |e_2|_x$ dla dowolnych termów e_1 i e_2 i dowolnej zmiennej $x \in \mathcal{X}$.

Dowód. Rozważmy przypadek $x \neq s$. Dowód przeprowadzimy przez indukcję względem struktury termu e_1. Podstawa indukcji: jeśli $e_1 \equiv x$, wówczas $|e_1[x/e_2]|_s = |e_2|_s = 0 + 1 \times |e_2|_s = |e_1|_s + |e_1|_x \times |e_2|_s$. Jeśli $e_1 \in \mathcal{X} \cup \{0, 1\}$ i $e_1 \neq x$, to $|e_1[x/e_2]|_s = |e_1|_s = |e_1|_s + 0 \times |e_2|_s = |e_1|_s + |e_1|_x \times |e_2|_s$. Krok indukcyjny: niech $e_1 = e_1' + e_1''$. Przypuśćmy, że $|e_1'[x/e_2]|_s = |e_1'|_s + |e_1'|_x \times |e_2'|_s$ i $|e_1''[x/e_2]|_s = |e_1''|_s + |e_1''|_x \times |e_2|_s$. Wówczas $|e_1[x/e_2]|_s = |(e_1' + e_1'')[x/e_2]|_s = |e_1'[x/e_2] + e_1''[x/e_2]|_s = |e_1'[x/e_2]|_s + |e_1''[x/e_2]|_s = |e_1'|_s + |e_1'|_x \times |e_2'|_s + |e_1''|_s + |e_1''|_x \times |e_2|_s = |e_1' + e_1''|_s + (|e_1' + e_1''|_x) \times |e_2|_s = |e_1|_s + |e_1|_x \times |e_2|_s$. Analogicznie dowodzi się drugiej części lematu.

Lemat 4. Jeśli $e_1 \sim e_2$, to $|e_1|_s = |e_2|_s$ dla $s \in \mathcal{X} \cup \{1\}$. Innymi słowy, jeśli równość $e_1 = e_2$ jest w naszym systemie dowodliwa, to liczba wystąpień dowolnej zmiennej oraz stałej 1 po obu stronach równości jest taka sama. Zauważmy, że liczba wystąpień stałej 0 po obu strona równości może być różna (tak jest np. w aksjomacie (1)).

Dowód. Przez indukcję względem struktury dowodu $\vdash e_1 = e_2$. Podstawa indukcji: dowód jest instancją aksjomatu (1), (2), (3) lub (Refl). Twierdzenie jest oczywiste. Krok indukcyjny: przypuśćmy, że ostatnim krokiem dowodu jest instancja reguły

$$\dfrac{e_1 = e_2}{e_2 = e_1}\ (\text{Sym})$$

Z założenia indukcyjnego $|e_2|_s = |e_1|_s$, zatem też $|e_1|_s = |e_2|_s$. Podobnie jeśli ostatnim krokiem dowodu jest instancja reguły

$$\dfrac{e_1 = e_2 \quad e_2 = e_3}{e_1 = e_3}\ (\text{Trans})$$

to z założeń indukcyjnych $|e_1|_s = |e_2|_s$ i $|e_2|_s = |e_3|_s$ wynika natychmiast, że $|e_1|_s = |e_3|_s$. Na koniec rozważmy przypadek, gdy ostatnim krokiem dowodu jest instancja reguły

$$\dfrac{e_1 = e_2 \quad e_3 = e_4}{e_1[x/e_3] = e_2[x/e_4]}\ (\text{Mon})$$

Z założeń indukcyjnych $|e_1|_s = |e_2|_s$ i $|e_3|_s = |e_4|_s$. Należy pokazać, że $|e_1[x/e_3]|_s = |e_2[x/e_4]|_s$. Niech $s \in \mathcal{X} \cup \{1\}$. Jeśli $x \neq s$, to $|e_1[x/e_3]|_s \overset{(*)}{=} |e_1|_s + |e_1|_x \times |e_3|_s = |e_2|_s + |e_2|_x \times |e_4|_s = |e_2[x/e_4]|_s$, gdzie równość $(*)$ wynika z lematu 3. Ponadto $|e_1[x/e_3]|_x \overset{(*)}{=} |e_1|_x \times |e_3|_x = |e_2|_x \times |e_4|_x = |e_2[x/e_4]|_x$, gdzie równość $(*)$ również wynika z lematu 3.

Twierdzenie 5. Jeśli $n_1 \neq n_2$, to $n_1 \cdot 1 \not\sim n_2 \cdot 1$.

Dowód. Twierdzenie wynika natychmiast z lematu 4, gdyż $|n \cdot 1|_1 = n$ dla dowolnego $n \geq 0$.

Wniosek 6. Dla dowolnego termu stałego e zachodzi $e \sim |e|_1 \cdot 1$.

Dowód. Istotnie, $|n \cdot 1|_1 = n$. Wniosek wynika natychmiast z twierdzeń 2 i 5.

Definiujemy odwzorowanie $\phi : \mathfrak{P} \to \mathfrak{N}$ wzorem

$$\phi([n \cdot 1]_\sim) \quad = \quad n$$

Na mocy twierdzeń 2 i 5 w każdej klasie abstrakcji relacji $\sim$ istnieje dokładnie jeden reprezentant postaci $n \cdot 1$. Odwzorowanie ϕ jest więc poprawnie określone i jest bijekcją. Ponadto na mocy wniosku 6 i lematu 1

$$\begin{aligned}
\phi([0]_\sim) &= \phi([0 \cdot 1]_\sim) = 0 \\
\phi([1]_\sim) &= \phi([1 \cdot 1]_\sim) = 1 \\
\phi([e_1 + e_2]_\sim) &= \phi([|e_1|_1 \cdot 1 + |e_2|_1 \cdot 1]_\sim) = \phi([(|e_1|_1 + |e_2|_1) \cdot 1]_\sim) = \\
&\quad (|e_1|_1 + |e_2|_1) = \phi([|e_1|_1 \cdot 1]_\sim) + \phi([|e_2|_1 \cdot 1]_\sim) = \phi([e_1]_\sim) + \phi([e_2]_\sim)
\end{aligned}$$

Odwzorowanie ϕ jest zatem izomorfizmem.