

Wstęp do teorii obliczeń

Paweł Urzyczyn
urzy@mimuw.edu.pl

6 kwietnia 2011, godzina 21:10

1 Funkcje częściowo rekurencyjne

Pojęcie funkcji obliczalnej i częściowo obliczalnej może zostać zdefiniowane na wiele różnych sposobów. Dziś najczęściej odwołujemy się w tym celu do maszyn Turinga, ale czasem wygodnie jest użyć pochodzącej od Kleene'go definicji funkcji (częściowo) rekurencyjnych. Rozważamy funkcje częściowe nad \mathbb{N} o dowolnej liczbie argumentów. Funkcje częściowo rekurencyjne to funkcje otrzymane z funkcji bazowych (zero, następnik, rzutowania) za pomocą trzech operacji: składania, rekursji prostej i minimum. Sformułujemy teraz ścisłą definicję tych pojęć. Napis $f : \mathbb{N}^k \dashrightarrow \mathbb{N}$ czytamy „ f jest k -argumentową funkcją częściową o argumentach i wartościach w \mathbb{N} ”.

Funkcje bazowe: Jako *funkcje bazowe* przyjmujemy:

- następnik, $\text{succ}(n) = n + 1$;
- rzuty, $\Pi_i^k(n_1, \dots, n_k) = n_i$, gdzie $k \geq 1$ oraz $i \leq k$;
- funkcje stale równe zero, $Z_k(n_1, \dots, n_k) = 0$, gdzie $k \geq 1$.

Składanie: Funkcja $f : \mathbb{N}^k \dashrightarrow \mathbb{N}$ powstaje przez *składanie* funkcji $h : \mathbb{N}^\ell \dashrightarrow \mathbb{N}$ z funkcjami $g_1, \dots, g_\ell : \mathbb{N}^k \dashrightarrow \mathbb{N}$, jeżeli

- $\text{Dom}(f) = \{\vec{n} \mid \vec{n} \in \text{Dom}(g_i) \text{ dla wszystkich } i, \text{ oraz } (g_1(\vec{n}), \dots, g_\ell(\vec{n})) \in \text{Dom}(h)\}$;
- $f(\vec{n}) = h(g_1(\vec{n}), \dots, g_\ell(\vec{n}))$, dla dowolnego $\vec{n} \in \text{Dom}(f)$.

Rekursja: Funkcja $f : \mathbb{N}^{k+1} \dashrightarrow \mathbb{N}$ powstaje przez *rekursję prostą* z funkcji $h : \mathbb{N}^{k+2} \dashrightarrow \mathbb{N}$ i funkcji $g : \mathbb{N}^k \dashrightarrow \mathbb{N}$, jeżeli dla dowolnego $n \in \mathbb{N}$ i dowolnego $\vec{m} \in \mathbb{N}^k$ spełnione są równania

- $f(0, \vec{m}) = g(\vec{m})$;
- $f(\text{succ}(n), \vec{m}) = h(f(n, \vec{m}), n, \vec{m})$,

przy czym równanie uważamy za spełnione, gdy obie strony są określone i równe, lub obie strony są nieokreślone. Wartość wyrażenia $h(f(n, \vec{m}), n, \vec{m})$ jest określona wtedy i tylko wtedy gdy $(n, \vec{m}) \in \text{Dom}(f)$ oraz $(f(n, \vec{m}), n, \vec{m}) \in \text{Dom}(h)$.

Minimum: Funkcja $f : \mathbb{N}^k \rightarrow \mathbb{N}$ powstaje z funkcji $h : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ przez zastosowanie operacji minimum, co zapisujemy $f(\vec{n}) = \mu y[h(\vec{n}, y) = 0]$, gdy dla dowolnego $\vec{n} \in \mathbb{N}^k$:

- Jeśli istnieje takie m , że $h(\vec{n}, m) = 0$ oraz wszystkie wartości $h(\vec{n}, i)$ dla $i < m$ są określone i różne od zera, to $f(\vec{n}) = m$.
- Jeśli takiego m nie ma, to $f(\vec{n})$ jest nieokreślone.

Definicja 1.1 Klasa funkcji *częściowo rekurencyjnych* to najmniejsza klasa funkcji częściowych nad \mathbb{N} zawierająca funkcje bazowe i zamknięta ze względu na składanie, rekursję prostą i operację minimum. Funkcje *rekurencyjne* to te funkcje częściowo rekurencyjne, które są całkowite (zawsze określone).

Klasa funkcji *pierwotnie rekurencyjnych* to najmniejsza klasa funkcji całkowitych nad \mathbb{N} zawierająca funkcje bazowe i zamknięta ze względu na składanie i rekursję prostą.

Funkcje charakterystyczne: Z każdą relacją k -argumentową $r \subseteq \mathbb{N}^k$ wiążemy dwie funkcje

$$c_r(\vec{n}) = \begin{cases} 0, & \text{jeśli } \vec{n} \in r; \\ 1, & \text{w przeciwnym przypadku,} \end{cases}$$

$$\chi_r(\vec{n}) = \begin{cases} 0, & \text{jeśli } \vec{n} \in r; \\ \text{nieokreślone,} & \text{w przeciwnym przypadku.} \end{cases}$$

Pierwsza z nich to zwykła funkcja charakterystyczna zbioru r , drugą nazywamy *częściową funkcją charakterystyczną* r . Relacje reprezentujemy za pomocą ich funkcji charakterystycznych. Piszemy np. w skrócie $\mu y[r(\vec{n}, y)]$ zamiast $\mu y[\chi_r(\vec{n}, y) = 0]$.

Definicja 1.2 Mówimy, że zbiór $A \subseteq \mathbb{N}^k$ jest *rekurencyjny*, gdy funkcja c_A jest rekurencyjna, a *rekurencyjnie przeliczalny*, gdy funkcja χ_A jest częściowo rekurencyjna. (W tym drugim przypadku mówi się też, że *relacja A jest częściowo rekurencyjna*.)

Fakt 1.3 *Każdy zbiór rekurencyjny jest rekurencyjnie przeliczalny.*

Dowód: Wynika to z równości $\chi_A(\vec{n}) = \mu y[c_A(\vec{n}) = 0]$, którą można ściśle wyrazić z pomocą rzutowań. Dla $A \subseteq \mathbb{N}$ mamy na przykład $\chi_A(n) = \mu y[c_A(\Pi_1^2(n, y)) = 0]$. \square

Funkcje pierwotnie rekurencyjne

Najczęściej mamy do czynienia z funkcjami i relacjami pierwotnie rekurencyjnymi. Na przykład dodawanie określone jest przez rekursję prostą z pomocą funkcji następnika:

$$0 + m = m;$$

$$\text{succ}(n) + m = \text{succ}(n + m).$$

Definicja ta jest zgodna ze schematem rekursji prostej dla $k = 1$ oraz $g(m) = \Pi_1^1(m)$ i $f(l, n, m) = \text{succ}(\Pi_1^3(l, n, m))$. Podobnie definiujemy mnożenie i potęgowanie. Przykłady te

pokazują jak można manipulować argumentami: jeśli k -argumentowa funkcja f jest pierwotnie rekurencyjna, to także l -argumentowa funkcja $g(n_1, \dots, n_l) = f(n_{i_1}, \dots, n_{i_k})$ jest pierwotnie rekurencyjna, bo jest złożeniem funkcji f z odpowiednimi rzutowaniami.

Każda funkcja stała $f(\vec{n}) = m$ jest pierwotnie rekurencyjna jako złożenie $f(\vec{n}) = succ^m(Z_k(\vec{n}))$. Aby określić funkcję poprzednika $pred$ napiszemy najpierw równania

$$\begin{aligned} p(0, m) &= Z_1(m); \\ p(n+1, m) &= \Pi_2^3(p(n), n, m), \end{aligned}$$

a następnie użyjemy złożenia:

$$pred(n) := p(n, 0) = p(n, Z_1(n)).$$

Teraz można już zdefiniować (nieujemne) odejmowanie:

$$\begin{aligned} m \dot{-} 0 &= m; \\ m \dot{-} (n+1) &= pred(m \dot{-} n). \end{aligned}$$

Następna prosta, ale pożyteczna funkcja to test na zero:

$$\begin{aligned} sg(0) &= 0; \\ sg(n+1) &= 1. \end{aligned}$$

Wśród najprostszych relacji pierwotnie rekurencyjnych mamy nierówność ostrą i nieostrą:

$$c_{\leq}(m, n) = sg(m \dot{-} n), \quad c_{<}(m, n) = sg((m+1) \dot{-} n).$$

Operacje logiczne nie wyprowadzają poza klasę relacji pierwotnie rekurencyjnych:

$$c_{r \vee p}(\vec{n}) = c_r(\vec{n}) \cdot c_p(\vec{n}), \quad c_{r \wedge p}(\vec{n}) = sg(c_r(\vec{n}) + c_p(\vec{n})), \quad c_{\neg r} = 1 \dot{-} c_r(\vec{n}),$$

a więc np. równość, jako koniunkcja dwóch nierówności, też jest pierwotnie rekurencyjna. Klasa relacji pierwotnie rekurencyjnych jest też zamknięta ze względu na *kwantyfikatory ograniczone*. Oznacza to, że jeśli zdefiniujemy relację r warunkiem

$$r(n, \vec{m}) \equiv \exists y \leq n. s(y, \vec{m}),$$

w którym s jest pierwotnie rekurencyjna, to także r jest pierwotnie rekurencyjna. Istotnie:

$$\begin{aligned} r(0, \vec{m}) &= s(0, \vec{m}); \\ r(n+1, \vec{m}) &= r(n, \vec{m}) \vee s(n+1, \vec{m}). \end{aligned}$$

Zajmiemy się teraz jeszcze kilkoma sposobami definiowania funkcji pierwotnie rekurencyjnych. Najpierw definicje warunkowe. Jeśli funkcje g i h oraz relacja r są pierwotnie rekurencyjne i

$$f(\vec{n}) = \begin{cases} g(\vec{n}), & \text{jeśli } r(\vec{n}); \\ h(\vec{n}), & \text{w przeciwnym przypadku,} \end{cases}$$

to wtedy $f(\vec{n}) = g(\vec{n}) \cdot (1 \dot{-} c_r(\vec{n})) + h(\vec{n}) \cdot c_r(\vec{n})$ jest pierwotnie rekurencyjna. Możemy też definiować funkcje za pomocą *minimum ograniczonego*, rozumianego tak:

$$\mu y \leq n [r(y, \vec{m})] = \begin{cases} \mu y [y \leq n \wedge r(y, \vec{m})], & \text{jeśli minimum jest określone;} \\ 0, & \text{w przeciwnym przypadku.} \end{cases}$$

Minimum ograniczone definiujemy przez rekursję prostą (por. kwantyfikator ograniczony), więc jest to funkcja pierwotnie rekurencyjna, o ile tylko taka jest relacja r . Podobnie jest z operacją *maksimum ograniczonego*

$$\nu y \leq n[r(y, \vec{m})] := \mu y \leq n[r(y, \vec{m}) \wedge \forall z \leq n. r(z, \vec{m}) \rightarrow z \leq y].$$

Minimum i maksimum ograniczone to bardzo przyteczne narzędzia, pozwalające łatwo definiować rozmaite funkcje, np. dzielenie całkowite to:

$$\left\lfloor \frac{m}{n} \right\rfloor = \nu y \leq m[y \cdot n \leq m].$$

Kodowanie słów

Dwie liczby naturalne można zakodować za pomocą jednej liczby, używając funkcji pary

$$\langle m, n \rangle = \left\lfloor \frac{(m+n)(m+n+1)}{2} \right\rfloor + m.$$

Funkcje odwrotne do funkcji pary to takie funkcje ℓ i r , że $\langle \ell(n), r(n) \rangle = n$ zachodzi dla wszystkich $n \in \mathbb{N}$. Zarówno $\langle \cdot, \cdot \rangle$ jak i jej funkcje odwrotne, to funkcje pierwotnie rekurencyjne, na przykład $\ell(n) = \mu y \leq n[\exists x \leq n. \langle y, x \rangle = n]$. Jeśli trzeba zakodować ciąg liczb a_0, \dots, a_k o dowolnej długości $k+1$, to można użyć jednoznaczności rozkładu na czynniki pierwsze:

$$\text{kod}(a_0 \dots a_k) = 2^{a_0} 3^{a_1} \dots p_k^{a_k},$$

gdzie p_i dla $i \in \mathbb{N}$ to ciąg rosnący wszystkich liczb pierwszych. Uwaga: to kodowanie „skleja” ze sobą wszystkie ciągi kończące się zerami i nigdy nie daje w wyniku zera. Ale nam to akurat nie przeszkodzi, a nawet zaraz się przyda.

Aby swobodnie posługiwać się powyższym kodowaniem, zauważmy, że następujące funkcje i relacje są pierwotnie rekurencyjne:

- relacja „ n jest liczbą pierwszą”: $\neg \exists y \leq n \exists z \leq n (y \cdot z = n \wedge y \neq 1 \wedge z \neq 1)$;
- funkcja p_i dana¹ równaniami $p_0 = 2$, $p_{n+1} = \mu y \leq 2p_n [y \text{ pierwsze i } y > p_n]$;
- funkcja $@(m, n) := \mu y \leq m [p_n^y \mid m \wedge \neg(p_n^{y+1} \mid m)]$ — „ n -ty wyraz ciągu o kodzie m ”;
- funkcja $m[i \leftarrow n] := \mu y \leq m \cdot p_i^n [\forall z \leq m (z \neq i \rightarrow @(y, z) = @(m, z)) \wedge @(y, i) = n]$ — „zmiana i -tego wyrazu w ciągu o kodzie m na liczbę n ”.

Maszyny Turinga

Przypomnijmy, że (deterministyczną, jednotaśmową) *maszynę Turinga nad alfabetem \mathcal{A}* można zdefiniować jako krotkę $\mathcal{M} = \langle \Delta, Q, \delta, q_0, q_a, q_r \rangle$, gdzie:

- Δ jest skończonym alfabetem, zawierającym \mathcal{A} oraz symbol $B \notin \mathcal{A}$ (blank);

¹Wiadomo, że pomiędzy liczbami m i $2m$ zawsze jest liczba pierwsza.

- Q jest skończonym zbiorem *stanów*;
- $q_0 \in Q$ jest stanem *początkowym*;
- $q_a \in Q$ jest stanem *akceptującym*;
- $q_r \in Q$ jest stanem *odrzucającym*;
- $\delta : (Q - \{q_a, q_r\}) \times \Delta \rightarrow \Delta \times Q \times \{-1, 0, +1\}$ jest *funkcją przejścia*.

Zakładając, że zbiory Δ i Q są rozłączne, można zdefiniować *konfigurację* maszyny jako trójkę postaci (q, i, w) , gdzie $q \in Q$, $i \in \mathbb{N}$ oraz $w \in \Delta^*$, przy czym utożsamiamy konfiguracje (q, i, w) oraz (q, i, wB) . Interpretacja tej definicji jest następująca. Taśma maszyny jest nieskończona w prawo. Na początku taśmy zapisane jest słowo w , dalej w prawo są same blanki, a głowica maszyny znajduje się w pozycji i . Konfigurację postaci $\mathcal{C}_w = (q_0, 0, w)$, gdzie $w \in \mathcal{A}^*$, nazywamy *początkową*. Konfiguracje *akceptujące* (odp. *odrzucające*) są zaś postaci (q_a, i, w) (odp. (q_r, i, w)).

Jeśli $\mathcal{C} = (q, i, w)$ oraz $w = a_0 \dots a_k$, to *następna konfiguracja* \mathcal{C}' jest określona tak:

- Jeśli $\delta(q, a) = (b, p, +1)$ to $\mathcal{C}' = (p, i + 1, w[i \leftarrow b])$;
- Jeśli $\delta(q, a) = (b, p, 0)$ to $\mathcal{C}' = (p, i, w[i \leftarrow b])$;
- Jeśli $\delta(q, a) = (b, p, -1)$ to $\mathcal{C}' = (p, i - 1, w[i \leftarrow b])$.

Piszemy $\mathcal{C} \rightarrow_{\mathcal{M}} \mathcal{C}'$, a symbolem $\rightarrow_{\mathcal{M}}$ oznaczamy przechodnio-zwrotne domknięcie relacji $\rightarrow_{\mathcal{M}}$. Jeżeli $\mathcal{C}_w \rightarrow_{\mathcal{M}} \mathcal{C}'$, gdzie \mathcal{C}' jest konfiguracją *akceptującą* (odp. *odrzucającą*) to mówimy, że maszyna *akceptuje* (odp. *odrzuca*) słowo w . Maszyna *zatrzymuje się* dla wejścia w , wtedy i tylko wtedy, gdy słowo w jest *akceptowane* lub *odrzucone*.

Obliczenie rozpoczynające się od konfiguracji \mathcal{C}_0 to skończony lub nieskończony ciąg konfiguracji $\mathcal{C}_0 \rightarrow_{\mathcal{M}} \mathcal{C}_1 \rightarrow_{\mathcal{M}} \mathcal{C}_2 \rightarrow_{\mathcal{M}} \dots$. Obliczenie skończone jest *akceptujące* lub *odrzucające*, w zależności od ostatniego stanu. Oczywiście maszyna deterministyczna ma obliczenie nieskończone rozpoczynające się od \mathcal{C}_w wtedy i tylko wtedy, gdy nie zatrzymuje się dla wejścia w .

Przez $L(\mathcal{M})$ oznaczamy język złożony ze wszystkich słów *akceptowanych* przez maszynę \mathcal{M} . Zbiór słów $L \subseteq \mathcal{A}^*$ jest *częściowo obliczalny* wtedy i tylko wtedy, gdy $L = L(\mathcal{M})$ dla pewnej maszyny \mathcal{M} . Jeśli dodatkowo maszyna \mathcal{M} odrzuca dokładnie te słowa, które nie należą do L , to mówimy, że L jest *obliczalny*.

Fakt 1.4 *Następujące warunki są równoważne:*

1. Z jest zbiorem *obliczalnym*;
2. $\neg Z$ jest zbiorem *obliczalnym*;
3. Z i $\neg Z$ są *częściowo obliczalne*;

Dowód: Jeśli dwie różne deterministyczne maszyny akceptują języki Z i $\neg Z$, to należy uruchomić je jednocześnie i zobaczyć, która zaakceptuje słowo wejściowe (jedna musi). Tę pracę może wykonać jedna maszyna deterministyczna. \square

Jeśli $\mathcal{A} = \{1\}$, a każdą liczbę naturalną n zinterpretujemy jako ciąg jedynek długości n , to język $L(\mathcal{A})$ można uważać za podzbiór \mathbb{N} . Możemy też użyć maszyny Turinga do definiowania funkcji $f_{\mathcal{M}} : \mathbb{N} \rightarrow \mathbb{N}$. Jeśli \mathcal{M} zatrzymuje się dla wejścia n (tj. słowa 1^n), to za wartość funkcji $f_{\mathcal{M}}(n)$ przyjmujemy pozycję głowicy w odp. konfiguracji końcowej. W przeciwnym razie wartość $f_{\mathcal{M}}(n)$ uważamy za nieokreśloną. Rozdzielając ciągi jedynek znakiem $\#$, możemy także definiować maszyny akceptujące relacje wieloargumentowe i obliczające wieloargumentowe funkcje.

Definicja 1.5 Funkcja częściowa $f : \mathbb{N}^k \rightarrow \mathbb{N}$ jest *częściowo obliczalna*, wtedy i tylko wtedy, gdy jest postaci $f_{\mathcal{M}}$ dla pewnej maszyny \mathcal{M} . Jeśli przy tym maszyna \mathcal{M} zatrzymuje się² dla dowolnego wejścia, to mówimy, że f jest *obliczalna*.

Fakt 1.6 Każda funkcja (częściowo) rekurencyjna jest też (częściowo) obliczalna.

Dowód: Indukcja ze względu na definicję funkcji rekurencyjnej (ćwiczenie). □

Wniosek 1.7 Każdy zbiór rekurencyjny (rekurencyjnie przeliczalny) jest obliczalny (częściowo obliczalny).

Kodowanie maszyn Turinga

Chcemy udowodnić twierdzenie odwrotne do Faktu 1.6 (dla uproszczenia na razie dla funkcji jednoargumentowych). W tym celu najpierw umówimy się, że stany maszyny i symbole alfabetu są ponumerowane, przy czym symbol B ma numer 0. Od tej pory stany i symbole utożsamiamy z ich numerami. Konfiguracje reprezentujemy jako trójki $\langle q, i, w \rangle = \langle q, \langle i, w \rangle \rangle$, używając dwukrotnie zwykłej funkcji pary. Przypomnijmy, że jeśli $c = \langle q, i, w \rangle$ to $q = \ell(c)$, $i = \ell(r(c))$ i $w = r(r(c))$. Równość $c = \langle q, i, w \rangle$ jest więc relacją pierwotnie rekurencyjną, podobnie jak funkcja, która argumentowi $\langle q, i, w \rangle$ przypisuje trójkę $\langle p, i + \varepsilon, w[i \leftarrow b] \rangle$, gdzie $\varepsilon \in \{-1, 0, 1\}$. Dla danej konfiguracji c , kod następnej konfiguracji $N(c)$ wyraża się więc definicją warunkową:

$$N(c) = \begin{cases} \dots & \dots \\ \dots & \dots \\ \langle p, i + \varepsilon, w[i \leftarrow b] \rangle, & \text{jeśli } c = \langle q, i, w \rangle \text{ oraz } \delta(q, @(\langle w, i \rangle)) = (p, b, \varepsilon), \\ \dots & \dots \\ \dots & \dots \end{cases}$$

o tylu wierszach ile jest możliwych „klauzul” w definicji funkcji przejścia. Operacja zmiany konfiguracji jest więc pierwotnie rekurencyjna. Możemy ją iterować, aby wyznaczyć kolejne konfiguracje obliczenia dla danego wejścia n :

$$\begin{aligned} C(0, n) &= \langle q_0, 0, \text{kod}(1^n) \rangle; \\ C(m+1, n) &= N(C(m, n)). \end{aligned}$$

Teraz zdefiniujemy relację: „ \mathcal{M} akceptuje wejście n po m krokach z głowicą w pozycji k ”:

²Można zakładać, że maszyna akceptuje każde słowo postaci $1^{m_1}\#1^{m_2}\#\dots\#1^{m_k}$ i odrzuca pozostałe.

$$t_{\mathcal{M}}(n, k, m) \equiv \exists y \leq C(m, n). C(m, n) = \langle q_a, k, y \rangle.$$

Możemy już napisać jaka funkcja jest obliczana przez maszynę \mathcal{M} .

$$f_{\mathcal{M}}(n) = \ell(\mu y[t_{\mathcal{M}}(n, \ell(y), r(y))]).$$

Powyższą konstrukcję łatwo jest uogólnić na funkcje wieloargumentowe (ćwiczenie). Uwzględniając Fakt 1.6, udowodniliśmy więc, że

Twierdzenie 1.8 *Funkcje (częściowo) obliczalne i (częściowo) rekurencyjne to to samo.*

Jeśli dokładniej przyjrzymy się naszemu dowodowi, to zauważymy też następujący ważny fakt.

Twierdzenie 1.9 (o postaci normalnej Kleene'go) *Każdą funkcję częściowo rekurencyjną φ można przedstawić w postaci*

$$\varphi(\vec{n}) = \ell(\mu y[g(\vec{n}, y) = 0]),$$

gdzie g jest funkcją pierwotnie rekurencyjną.

Fakt 1.10 *Następujące warunki są równoważne dla $A \subseteq \mathbb{N}$:*

1. A jest rekurencyjnie przeliczalny.
2. A jest dziedziną pewnej funkcji częściowo rekurencyjnej.
3. Istnieje taki rekurencyjny zbiór $B \subseteq \mathbb{N}^2$, że dla wszystkich $n \in \mathbb{N}$ zachodzi równoważność

$$n \in A \Leftrightarrow \exists z. (n, z) \in B.$$
4. A jest pusty lub jest zbiorem wartości pewnej funkcji rekurencyjnej.³
5. A jest zbiorem wartości pewnej funkcji częściowo rekurencyjnej.

Dowód: Implikacja (1) \Rightarrow (2) jest oczywista, a (2) \Rightarrow (5) wynika stąd, że $A = \text{Dom}(\psi)$ implikuje $A = \text{Rg}(\varphi)$, gdzie $\varphi(n) = n + Z_1(\psi(n))$. Dla dowodu implikacji (5) \Rightarrow (3) założymy, że $A = \text{Rg}(\varphi)$. Na mocy twierdzenia Kleene'go mamy $\varphi(x) = \ell(\mu y[g(x, y) = 0])$ dla pewnej rekurencyjnej funkcji g . Mamy teraz

$$n \in A \Leftrightarrow \exists x \exists y (g(x, y) = 0 \wedge \forall i < y (g(x, i) \neq 0) \wedge \ell(y) = n),$$

a więc można przyjąć

$$B = \{(n, z) \mid g(\ell(z), r(z)) = 0 \wedge \forall i < r(z) (g(\ell(z), i) \neq 0) \wedge \ell(r(z)) = n\}.$$

Aby pokazać (3) \Rightarrow (4), przypuścimy, że $A \neq \emptyset$ i ustalmy jakieś $a \in A$. Wtedy $A = \text{Rg}(f)$, dla

$$f(n) = \begin{cases} \ell(n), & \text{jeśli } (\ell(n), r(n)) \in B; \\ a, & \text{w przeciwnym przypadku.} \end{cases}$$

Na koniec, (4) \Rightarrow (1) wynika z równości $\chi_A(m) = Z_1(\mu y[f(y) = m])$ dla $A = \text{Rg}(f)$. □

³To uzasadnia nazwę „rekurencyjnie przeliczalny”.

Ćwiczenia

1. Udowodnić, że klasa funkcji pierwotnie rekurencyjnych jest zamknięta ze względu na minimum ograniczone i ograniczony kwantyfikatory ogólny.
2. Pokazać, że jeśli f jest k -argumentową funkcją rekurencyjną to jest też rekurencyjną relacją $k + 1$ -argumentową.
3. Pokazać, że jeśli funkcja częściowa $f : \mathbb{N}^k \dashrightarrow \mathbb{N}$ jest rekurencyjnie przeliczalna jako relacja $k + 1$ -argumentowa to jest k -argumentową funkcją częściowo rekurencyjną.
4. Zdefiniować funkcje:
 - $|m|$ — „długość ciągu o kodzie m (bez końcowych zer)”;
 - $m \bullet n$ — „dopisanie n na końcu ciągu o kodzie m ”.

2 Inne definicje obliczalności

While-programy

While-programy to bardzo uproszczony model języka programowania, w którym programy są konstruowane za pomocą prostych pętli i w którym występuje tylko jeden typ danych (np. liczby naturalne, ale niekoniecznie).

Definicja 2.1 Ustalmy sygnaturę Σ . *While-program* nad Σ to wyrażenie jednej z następujących postaci:

- *Instrukcja przypisania*: $x := t$, gdzie x jest zmienną a t jest termem sygnatury Σ ;
- *Złożenie*: $P_1; P_2$, gdzie P_1 i P_2 są while-programami;
- *Instrukcja warunkowa*: **if** α **then** P_1 **else** P_2 **fi**, gdzie α jest formułą otwartą a P_1 i P_2 są while-programami;
- *Pętla*: **while** α **do** P **od**, gdzie α jest formułą otwartą a P jest while-programem.

Intuicja związana z interpretacją while-programu w danej strukturze sygnatury Σ powinna być oczywista: wykonanie programu zmienia wartościowanie zmiennych w sposób odpowiedni do zawartych w nim instrukcji.

Definicja 2.2 Dla dowolnej struktury \mathfrak{A} sygnatury Σ i dowolnego while-programu P definiujemy *relację wejścia-wyjścia*, którą oznaczamy przez $P^{\mathfrak{A}}$. Relacja ta zachodzi pomiędzy wartościowaniami w \mathfrak{A} i jest oczywiście określona przez indukcję.⁴

- Relacja $(x := t)^{\mathfrak{A}}$ składa się z par postaci $(\varrho, \varrho[x \mapsto a])$, gdzie $a = \llbracket t \rrbracket_{\varrho}$.
- Relacja $(P_1; P_2)^{\mathfrak{A}}$ to złożenie relacji $P_2^{\mathfrak{A}} \bullet P_1^{\mathfrak{A}}$.

⁴Napis $\varrho[x \mapsto a]$ oznacza wartościowanie różniące się od ϱ tylko tym, że $\varrho[x \mapsto a](x) = a$.

- Jeśli $P = \mathbf{if} \alpha \mathbf{ then } P_1 \mathbf{ else } P_2 \mathbf{ fi}$, to $(\varrho, \varrho') \in P^{\mathfrak{A}}$ wtedy i tylko wtedy, gdy zachodzi jeden z przypadków
 - $(\varrho, \varrho') \in P_1^{\mathfrak{A}}$ oraz $\mathfrak{A}, \varrho \models \alpha$;
 - $(\varrho, \varrho') \in P_2^{\mathfrak{A}}$ oraz $\mathfrak{A}, \varrho \not\models \alpha$.
- Jeśli $P = \mathbf{while} \alpha \mathbf{ do } P_1 \mathbf{ od}$, to $(\varrho, \varrho') \in P^{\mathfrak{A}}$ wtedy i tylko wtedy, gdy istnieje $m \geq 0$ i taki ciąg wartościowań $\varrho = \varrho_0, \varrho_1, \dots, \varrho_m = \varrho'$, że
 - $(\varrho_i, \varrho_{i+1}) \in P_1^{\mathfrak{A}}$, dla $i = 0, \dots, m-1$;
 - $\mathfrak{A}, \varrho_i \models \alpha$, dla $i = 0, \dots, m-1$;
 - $\mathfrak{A}, \varrho_m \not\models \alpha$.

Oczywiście, dla danego ϱ istnieje co najwyżej jedno wartościowanie ϱ' o własności $(\varrho, \varrho') \in P^{\mathfrak{A}}$. Jeśli istnieje, to mówimy, że program P *zatrzymuje się* dla danych ϱ z wynikiem ϱ' . While-program, w którym wyróżnimy pewną liczbę zmiennych *wejściowych* oraz jedną zmienną *wyjściową* można więc uważać za definicję funkcji częściowej.

Definicja 2.3 Funkcja częściowa $f : \mathfrak{A}^n \dashrightarrow \mathfrak{A}$ jest *obliczalna w \mathfrak{A}* , wtedy i tylko wtedy, gdy istnieje while-program P i zmienne x_1, \dots, x_n, y o takich własnościach:

- Zmienne x_1, \dots, x_n są różne;
- P zatrzymuje się dla ϱ wtedy i tylko wtedy, gdy $f(\varrho(x_1), \dots, \varrho(x_n))$ jest określone;
- Jeśli $(\varrho, \varrho') \in P^{\mathfrak{A}}$, to $\varrho'(y) = f(\varrho(x_1), \dots, \varrho(x_n))$.

Fakt 2.4 Funkcje (częściowo) obliczalne w strukturze $\langle \mathbb{N}, 0, \text{succ} \rangle$, to dokładnie funkcje (częściowo) rekurencyjne.

Dowód: Ćwiczenie. □

Gramatyki typu zero

Przez *gramatykę typu zero* rozumiemy twór postaci $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$, w którym \mathcal{A} jest alfabetem terminalnym, \mathcal{N} jest alfabetem nieterminalnym, $\xi_0 \in \mathcal{N}$ to symbol początkowy, a produkcje (czyli reguły) ze zbioru P mają kształt $u \Rightarrow v$, gdzie $u, v \in (\mathcal{A} \cup \mathcal{N})^*$ są zupełnie dowolnymi słowami, byle tylko $u \neq \varepsilon$. Relacja redukcji \rightarrow_G jest zdefiniowana podobnie jak dla gramatyk bezkontekstowych, mianowicie $x \rightarrow_G y$ (zapisywane też tak: $G \vdash x \rightarrow y$) zachodzi wtedy i tylko wtedy, gdy $x = x_1 u x_2$, $y = x_1 v x_2$, oraz $u \Rightarrow v$ jest pewną produkcją. Oczywiście notacja \rightarrow_G oznacza istnienie (być może pustego) ciągu redukcji, a język generowany przez gramatykę G definiujemy tak:

$$L(G) = \{w \in \mathcal{A}^* \mid \xi_0 \rightarrow_G w\}.$$

Na przykład język $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ jest generowany przez taką gramatykę typu zero:

$$\begin{aligned}\xi_0 &\Rightarrow \varepsilon, & \xi_0 &\Rightarrow \eta; \\ \eta &\Rightarrow a\beta\eta c, & \eta &\Rightarrow abc; \\ \beta a &\Rightarrow a\beta, & \beta b &\Rightarrow bb.\end{aligned}$$

Twierdzenie 2.5 *Język L jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy $L = L(G)$ dla pewnej gramatyki G .*

Dowód: (\Leftarrow) Niech $L = L(G)$. Konstruujemy maszynę niedeterministyczną \mathcal{M} z jedną taśmą pomocniczą.⁵ Na początku umieszcza się na tej taśmie symbol początkowy gramatyki, a następnie niedeterministycznie wykonuje redukcje. Tj. w każdej fazie pracy, maszyna wybiera redukcję $x \Rightarrow y$ i zastępuje na taśmie pewne wystąpienie słowa x przez y . (To może wymagać przesunięcia pozostałej zawartości taśmy w lewo lub w prawo.) Potem następuje sprawdzenie, czy zawartość obu taśm, wejściowej i roboczej, jest taka sama. Jeśli tak, to maszyna akceptuje.

(\Rightarrow) Niech $L = L(\mathcal{M})$ i załóżmy, że \mathcal{M} jest deterministyczną maszyną jednotaśmową. (Taśma jest nieskończona w obie strony.) Definiujemy gramatykę G , której alfabet nieterminalny składa się ze stanów i symboli maszyny \mathcal{M} , oraz dodatkowo z nawiasów kwadratowych (na oznaczenie początku i końca konfiguracji).

Zbiór produkcji gramatyki G dzieli się na dwie części. Pierwszą grupę stanowią produkcje pozwalające wyprowadzić dowolne słowo postaci $w[q_0w]$ (ćwiczenie: jak to zrobić?). Druga grupa produkcji pozwala na symulację obliczenia maszyny w prawej części słowa. Są to takie produkcje:

- $qa \Rightarrow bp$, gdy $\delta(q, a) = (b, p, +1)$;
- $qa \Rightarrow pb$, gdy $\delta(q, a) = (b, p, 0)$;
- $q] \Rightarrow bp]$, gdy $\delta(q, B) = (b, p, +1)$;
- $q] \Rightarrow pb]$, gdy $\delta(q, B) = (b, p, 0)$;
- $cqa \Rightarrow pcb$ i $[qa \Rightarrow [pBb$, gdy $\delta(q, a) = (b, p, -1)$;
- $cq] \Rightarrow pcb]$, gdy $\delta(q, B) = (b, p, -1)$;
- $aq_a \Rightarrow q_a$ i $q_a a \Rightarrow q_a$, gdy $a \neq [,]$;
- $[q_a] \Rightarrow \varepsilon$.

Zauważmy, że wówczas zachodzi równoważność:

$$[q_0w] \rightarrow_G [q_a] \quad \text{wtedy i tylko wtedy gdy} \quad w \in L(\mathcal{M}),$$

bo każdy ciąg postaci $[q_0w] \rightarrow_G x_1 \rightarrow_G x_2 \rightarrow_G \dots \rightarrow_G [q_a]$ musi reprezentować obliczenie maszyny dla słowa w , zakończone „wycieraniem” wszystkich symboli oprócz q_a .

⁵Czytelnik wie skądinąd, że języki akceptowane przez takie maszyny są częściowo obliczalne.

Jeśli odpowiednio dobierzemy produkcje z pierwszej grupy, to każdy ciąg redukcji, w którym występują słowa zawierające stany maszyny \mathcal{M} , musi zaczynać się od $\xi_0 \rightarrow_G w[q_0w]$, dla pewnego w .

A zatem wyprowadzenie w gramatyce G , kończące się słowem terminalnym może wyglądać tylko tak:

$$\xi_0 \rightarrow w[q_0w] \rightarrow w[q_a] \rightarrow w,$$

co oznacza, że $L(G) = L(\mathcal{M})$. □

Rachunek lambda

Lambda-termny definiuje się zwykle tak (przyjmujemy pewien ustalony nieskończony zbiór *zmiennych przedmiotowych*):

- zmienne przedmiotowe są termami;
- jeśli M i N są termami, to (MN) też;
- jeśli M jest termem i x jest zmienną, to (λxM) jest termem.

Konwencje notacyjne:

- opuszczamy zewnętrzne nawiasy;
- aplikacja wiąże w lewo, tj. MNP oznacza $(MN)P$;
- piszemy $\lambda x_1 \dots x_n.M$ zamiast $\lambda x_1 \dots \lambda x_n.M$.

Operator lambda-abstrakcji, λ , wiąże zmienne, tj. wszystkie wystąpienia x w termie λxM uważa się za *związane*. Zmienne *wolne* definiuje się tak:

- $FV(x) = \{x\}$;
- $FV(MN) = FV(M) \cup FV(N)$;
- $FV(\lambda xM) = FV(M) - \{x\}$.

Dwa termny uważa się za identyczne, jeśli różnią się tylko nazwami zmiennych związanych. Podstawienie $M[N/x]$ definiuje się tak, że N jest wstawiane tylko na wolne wystąpienia x , a zmienne związane ulegają w razie potrzeby przemianowaniu, tak aby nie doprowadzić do konfuzji. Na przykład: $((\lambda y.xy)x)[y/x] = (\lambda z.yz)y$. Szczegółowe definicje pomijamy.

Dla lambda-termów określa się relację *beta-redukcji* jako najmniejszą relację \rightarrow_β , taką, że:

- $(\lambda xM)N \rightarrow_\beta M[N/x]$;
- jeśli $M \rightarrow_\beta M'$ to $MN \rightarrow_\beta M'N$, $NM \rightarrow_\beta NM'$ oraz $\lambda xM \rightarrow_\beta \lambda xM'$.

Teraz pokażemy jak w rachunku lambda można zinterpretować pewne proste konstrukcje. Zaczniemy od wartości logicznych

$$\mathbf{true} = \lambda xy.x \qquad \mathbf{false} = \lambda xy.y$$

i instrukcji warunkowej

$$\mathbf{if } P \mathbf{ then } Q \mathbf{ else } R = PQR.$$

Łatwo sprawdzić, że $\mathbf{if } \mathbf{true} \mathbf{ then } Q \mathbf{ else } R \rightarrow_{\beta} Q$ oraz $\mathbf{if } \mathbf{false} \mathbf{ then } Q \mathbf{ else } R \rightarrow_{\beta} R$.

Następna konstrukcja to para uporządkowana. Przyjmiemy

$$\begin{aligned} \langle M, N \rangle &= \lambda x.xMN; \\ \pi_i &= \lambda x_1x_2.x_i \text{ dla } i = 1, 2. \end{aligned}$$

Jak należy się spodziewać, mamy $\langle M_1, M_2 \rangle \pi_i \rightarrow_{\beta} M_i$. Zauważmy jednak, że nie zachodzi równość $\langle M\pi_1, M\pi_2 \rangle =_{\beta} M$, tj. nasza para uporządkowana nie jest *surjektywna*.

Liczby naturalne reprezentujemy w rachunku lambda jako tzw. *liczebniki Churcha*:

$$c_n = \lambda fx.f^n(x),$$

gdzie notacja $f^n(x)$ oznacza oczywiście term $f(f(\dots(x)\dots))$, w którym f występuje n razy. Zwykle zamiast c_n będziemy pisać \mathbf{n} , co jest mniej precyzyjne ale wygodne. Więc na przykład:

$$\begin{aligned} \mathbf{0} &= \lambda fx.x; \\ \mathbf{1} &= \lambda fx.fx; \\ \mathbf{2} &= \lambda fx.f(fx), \text{ i tak dalej.} \end{aligned}$$

Powiemy, że funkcja częściowa $f : \mathbb{N}^k \rightarrow \mathbb{N}$ jest *definiowalna* w beztypowym rachunku lambda (λ -*definiowalna*) jeżeli istnieje term zamknięty F , spełniający następujące warunki dla dowolnych $n_1, \dots, n_k \in \mathbb{N}$:

- Jeżeli $f(n_1, \dots, n_k) = m$, to $F\mathbf{n}_1 \dots \mathbf{n}_k =_{\beta} \mathbf{m}$;
- Jeżeli $f(n_1, \dots, n_k)$ jest nieokreślone, to $F\mathbf{n}_1 \dots \mathbf{n}_k$ nie ma postaci normalnej.

Mówimy oczywiście, że F *definiuje* lub *reprezentuje* funkcję f w rachunku lambda. Kilka przykładów termów definiujących pewne funkcje mamy poniżej.

Przykład 2.6

- Następnik: $\mathbf{succ} \equiv \lambda nfx.f(nfx)$;
- Dodawanie: $\mathbf{add} \equiv \lambda mnfx.mf(nfx)$;
- Mnożenie: $\mathbf{mult} \equiv \lambda mnfx.m(nf)x$;
- Potegowanie: $\mathbf{exp} \equiv \lambda mnfx.mnfx$;
- Test na zero: $\mathbf{zero} \equiv \lambda m.m(\lambda y.\mathbf{false})\mathbf{true}$;

- Funkcja k -argumentowa stale równa zeru: $\mathbf{Z}_k = \lambda m_1 \dots m_k. \mathbf{0}$;
- Rzut k -argumentowy na i -tą współrzędną: $\mathbf{\Pi}_k^i = \lambda m_1 \dots m_k. m_i$.

Udowodnimy teraz, że każda funkcja częściowo rekurencyjna jest definiowalna w rachunku lambda. Zaczynamy od najłatwiejszego.

Lemat 2.7 *Funkcje bazowe są lambda-definiowalne.*

Dowód: Przykład 2.6. □

Lemat 2.8 *Jeśli funkcja całkowita f powstaje przez składanie lambda-definiowalnych funkcji całkowitych, to też jest lambda-definiowalna.*

Dowód: Oczywiście. Ale tylko dlatego, że mowa o funkcjach całkowitych. □

Lemat 2.9 *Jeśli funkcja całkowita f powstaje przez rekursję prostą z lambda-definiowalnych funkcji całkowitych, to też jest lambda-definiowalna.*

Dowód: To już nie jest oczywiste. Załóżmy, że f jest zdefiniowana równaniami

$$\begin{aligned} f(0, \vec{n}) &= g(\vec{n}); \\ f(n+1, \vec{n}) &= h(f(n, \vec{n}), n, \vec{n}), \end{aligned}$$

i że funkcje g i h są definiowalne odpowiednio za pomocą termów G i H . Zdefiniujmy pomocnicze termy

$$\begin{aligned} \mathbf{Step} &\equiv \lambda p. \langle \mathbf{succ}(p\pi_1), H(p\pi_2)(p\pi_1)x_1 \dots x_m \rangle; \\ \mathbf{Init} &\equiv \langle \mathbf{0}, Gx_1 \dots x_m \rangle. \end{aligned}$$

Funkcja f jest wtedy definiowalna termem

$$F \equiv \lambda x x_1 \dots x_m. x \mathbf{Step} \mathbf{Init} \pi_2.$$

Ta definicja wyraża następujący algorytm obliczania wartości funkcji f : generujemy ciąg par

$$(0, a_0), (1, a_1), \dots, (n, a_n),$$

gdzie $a_0 = g(n_1, \dots, n_m)$, $a_{i+1} = h(a_i, i, n_1, \dots, n_m)$ i na koniec $a_n = f(n, n_1, \dots, n_m)$. □

Wniosek 2.10 *Funkcje pierwotnie rekurencyjne są lambda-definiowalne.*

Ponieważ mamy twierdzenie Kleene'go, więc pozostaje już tylko pokazać lambda-definiowalność funkcji określonych przez minimum.

Twierdzenie 2.11 *Funkcje lambda-definiowalne to dokładnie funkcje częściowo rekurencyjne.*

Dowód: Implikacja z lewej do prawej wynika stąd, że proces ewaluacji termu do postaci normalnej może być zrealizowany za pomocą maszyny Turinga. Dowodzimy implikacji odwrotnej. Niech $f(\vec{n}) = \ell(\mu y[g(\vec{n}, y) = 0])$, gdzie g jest funkcją pierwotnie rekurencyjną. Na mocy Wniosku 2.10, zarówno g jak ℓ są lambda-definiowalne pewnymi termami G i L . Określmy pomocniczy term

$$W \equiv \lambda y. \mathbf{if\ zero}(G\vec{x}y) \mathbf{then\ } \lambda w. Ly \mathbf{ else\ } \lambda w. w(\mathbf{succ\ } y)w.$$

Funkcja f jest definiowana termem

$$F \equiv \lambda \vec{x}. W\mathbf{0}W.$$

Rzeczywiście, przypuśćmy najpierw, że $\mu y[f(\vec{n}, y) = 0] = n$, oraz $\ell(n) = r$. Wtedy mamy taki ciąg redukcji:

$$F\vec{n} \rightarrow_{\beta} \overline{W\mathbf{0}W} \rightarrow_{\beta} \overline{W\mathbf{1}W} \rightarrow_{\beta} \dots \rightarrow_{\beta} \overline{W\mathbf{n}W} \rightarrow_{\beta} L\mathbf{n} \rightarrow_{\beta} \mathbf{r}, \quad (1)$$

gdzie \overline{W} oznacza wynik podstawienia \vec{n} na \vec{x} .

Jeśli minimum jest nieokreślone, to znaczy, że wszystkie wartości $g(\vec{n}, m)$ są określone i różne od zera, bo funkcja g jest całkowita. Wtedy mamy nieskończony ciąg redukcji

$$F\vec{n} \rightarrow_{\beta} \overline{W\mathbf{0}W} \rightarrow_{\beta} \overline{W\mathbf{1}W} \rightarrow_{\beta} \dots \rightarrow_{\beta} \overline{W\mathbf{n}W} \rightarrow_{\beta} \dots$$

Można pokazać, że żadna inna redukcja termu $F\vec{n}$ nie prowadzi do postaci normalnej. \square

Ćwiczenia

1. Napisać gramatykę typu zero generującą język złożony ze wszystkich słów $w\bar{w} \in \{0, 1, 2\}^*$, gdzie słowo \bar{w} powstaje z w przez zamianę każdej jedynek na zero i każdej dwójki na jedynekę.
2. Udowodnić, że dodanie instrukcji **go to** do języka **while**-programów nie rozszerza klasy funkcji obliczalnych.
3. Napisać lambda-term definiujący funkcję $\lfloor \frac{m}{n} \rfloor$ dwóch zmiennych m i n .
4. Niech $\mathbf{Y} = \lambda f(\lambda x f(xx))(\lambda x f(xx))$. Pokazać, że $\mathbf{Y}F =_{\beta} F(\mathbf{Y}F)$ dla dowolnego termu F .

3 Funkcje pierwotnie i elementarnie rekurencyjne

Dla wszystkich $n \in \mathbb{N}$ definiujemy funkcje $a_n : \mathbb{N} \rightarrow \mathbb{N}$:

$$\begin{aligned} a_0(x) &= x + 1; \\ a_{n+1}(x) &= a_n^{x+1}(1). \end{aligned}$$

A więc $a_1(x) = x + 2$, $a_2(x) = 2x + 3$, co jeszcze wygląda dość niegroźnie. Ale dalej $a_3(x) > 2^x$, a oszacowanie dla $a_4(x)$ wymaga x razy iterowanego potęgowania. Wszystkie funkcje a_n są jednak pierwotnie rekurencyjne, bo są określone przez rekursję prostą.

Lemat 3.1 *Funkcje a_n mają następujące własności (dla wszystkich $x \in \mathbb{N}$):*

1. $x < a_n(x)$;
2. $x < a_n^x(1)$;
3. $a_n(x) < a_n(x+1)$.

Dowód: Dowód jest przez jednoczesną indukcję ze względu na n . Oczywiście (1) i (3) zachodzi dla $n = 0$. Zauważmy też, że (1) implikuje (2). Istotnie, jeśli $x < a_n(x)$ dla dowolnego x , to $1 < a_n(1) < a_n(a_n(1)) < \dots$ więc $a_n^x(1)$ jest równe co najmniej $x+1$.

Kolej na krok indukcyjny. Zaczynamy od części (3):

$$a_{n+1}(x+1) = a_n^{x+2}(1) = a_n^{x+1}(a_n(1)) > a_n^{x+1}(1) = a_{n+1}(x).$$

Teraz część (1):

$$a_{n+1}(x) = a_n^{x+1}(1) = a_n(a_n^x(1)) > a_n(x). \quad \square$$

Wniosek 3.2 Funkcje a_n są rosnące, oraz $a_n(x) < a_m(x)$, gdy $n < m$.

Dowód: Zauważmy, że $a_{n+1}(x) = a_n(a_n^x(1)) > a_n(x)$. □

Lemat 3.3 Następujące nierówności zachodzą dla wszystkich $n, x \in \mathbb{N}$:

1. $a_n^2(x) < a_{n+2}(x)$;
2. $a_n^{x+1}(x) \leq a_{n+3}(x)$.

Dowód: Zaczynamy od nierówności $a_m(x+1) \leq a_m(a_m^x(1)) = a_{m+1}(x)$, która zachodzi dla wszystkich m . Stąd $a_n(a_n(x)) < a_n(a_{n+1}(x)) = a_n^{x+2}(1) = a_{n+1}(x+1) \leq a_{n+2}(x)$. Dalej mamy $a_n^{x+1}(x) < a_n^{x+1}(a_n^x(1)) = a_n^{2x+1}(1) < (a_n^2)^{x+1}(1) < a_{n+2}^{x+1}(1) = a_{n+3}(x)$. □

Twierdzenie 3.4 Dla dowolnej funkcji pierwotnie rekurencyjnej $f : \mathbb{N}^k \rightarrow \mathbb{N}$ istnieje taka liczba n , że $f(\vec{x}) \leq a_n(\max \vec{x})$ dla wszystkich $\vec{x} \in \mathbb{N}^k$.

Dowód: Dowód jest przez indukcję ze względu na definicję funkcji f . Oczywiście funkcje bazowe są ograniczone przez funkcję następnika a_0 . Jeśli funkcja f jest złożeniem postaci $f(\vec{x}) = h(g_1(\vec{x}), \dots, g_\ell(\vec{x}))$, to niech m będzie dostatecznie duże na to, aby $g_i(\vec{x}) \leq a_m(\max \vec{x})$ oraz $h(\vec{y}) \leq a_m(\max \vec{y})$ dla dowolnych \vec{x}, \vec{y} . Teraz $f(\vec{x}) \leq a_m(a_m(\max \vec{x})) < a_{m+2}(\max \vec{x})$.

Jeśli f jest określona równaniami rekursji prostej:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}); \\ f(x+1, \vec{y}) &= h(f(x, \vec{y}), x, \vec{y}), \end{aligned}$$

oraz m jest dostatecznie duże, to przez indukcję ze względu na x udowodnimy

$$f(x, \vec{y}) \leq a_{m+1}^{x+1}(\max \vec{y}) \leq a_{m+3}(\max\{x, \vec{y}\}). \quad \square$$

Ćwiczenia

1. Skąd wiadomo, że funkcja Ackermanna jest rekurencyjna?
2. Niech e_0 będzie funkcją następnika i niech $e_{n+1}(x) = e_n^x(x)$. Udowodnić, że dla każdej funkcji pierwotnie rekurencyjnej f istnieje takie n , że $f(\vec{x}) \leq e_n(\max \vec{x})$, gdy $\max \vec{x} \geq 2$. Wywnioskować, że funkcja $e_\omega(x) = e_x(x)$ nie jest pierwotnie rekurencyjna.
3. W Definicji 2.1 zamieniamy pętlę **while** na pętlę postaci **for** $i = 1$ **to** x **do** P **od**, przy czym zmienna sterująca i nie występuje w P . Zdefiniować semantykę takiego języka w strukturze liczb naturalnych i pokazać, że funkcje obliczalne w tym języku to dokładnie funkcje pierwotnie rekurencyjne.
4. Udowodnić, że funkcja jest elementarnie rekurencyjna wtedy i tylko wtedy, gdy jest obliczalna w czasie elementarnie rekurencyjnym.

4 Numeracja

Funkcje częściowo rekurencyjne i zbiory rekurencyjnie przeliczalne można numerować na różne sposoby. Można na przykład zdefiniować numerację maszyn Turinga, lub przypisywać numery funkcjom częściowo rekurencyjnym przez indukcję ze względu na ich definicje. Zaczynamy wtedy od przypisania numerów funkcjom bazowym:

- Z_k ma numer $\langle 0, 0 \rangle$;
- Π_i^k ma numer $\langle 0, i \rangle$;
- $succ$ ma numer $\langle 0, 2 \rangle$.

Funkcję k -argumentową o numerze n będziemy oznaczać przez $\varphi_n^{(k)}$, pomijając górny indeks jeśli jest to nieszkodliwe. Jeśli teraz funkcja k -argumentowa ψ jest złożeniem postaci

$$\psi(\vec{x}) = \varphi_n^{(m)}(\varphi_{\ell_1}^{(k)}(\vec{x}), \dots, \varphi_{\ell_m}^{(k)}(\vec{x})),$$

to za numer funkcji ψ możemy przyjąć liczbę $\langle 1, \langle m, \langle n, \langle \ell_1, \dots, \langle \ell_{m-1}, \ell_m \rangle \rangle \rangle \rangle \rangle$. Podobnie, funkcja o $k+1$ argumentach i numerze $\langle 2, \langle m, n \rangle \rangle$ to funkcja zdefiniowana przez rekursję prostą z funkcji $\varphi_m^{(k)}$ i $\varphi_n^{(k+2)}$. A funkcji $\mu y[\varphi_n^{(k+1)}(\vec{x}, y) = 0]$ można przypisać numer $\langle 3, n \rangle$. Aby nie zostały żadne wolne numery przyjmujemy, że wszystkie liczby innej postaci niż wymienione wyżej są numerami funkcji Z_k . Niewiele to zmienia, bo i tak

każda funkcja częściowo rekurencyjna ma nieskończenie wiele numerów,

numerujemy bowiem w istocie nie funkcje, ale *algorytmy*.

W istocie nie ma większego znaczenia jaki dokładnie przyjęliśmy sposób numeracji. Można np. zamiast numerów funkcji używać ich definicji *in extenso* i nie wpłynęłoby to na otrzymane rezultaty. Używanie liczb naturalnych często jednak upraszcza sformułowanie tych rezultatów.

Zbiory rekurencyjnie przeliczalne numerujemy tak jak ich częściowe funkcje charakterystyczne: zbiór W_n to dziedzina funkcji φ_n .

Funkcje uniwersalne

Funkcja dwuargumentowa $\Phi(m, n) = \varphi_m^{(1)}(n)$ jest częściowo rekurencyjna. Nazywamy ją *funkcją uniwersalną* dla klasy funkcji częściowo rekurencyjnych. Dla całkowitych funkcji obliczalnych takiej funkcji nie ma (nie da się ich sensownie ponumerować).

Fakt 4.1 *Nie istnieje funkcja uniwersalna dla klasy funkcji rekurencyjnych, tj. nie istnieje dwuargumentowa funkcja $\Psi : \mathbb{N}^2 \rightarrow \mathbb{N}$ o tej własności, że każda funkcja rekurencyjna $f : \mathbb{N} \rightarrow \mathbb{N}$ ma dla pewnego n postać*

$$f(x) = \Psi(n, x).$$

Dowód: W przeciwnym razie funkcja $f(x) = \Psi(x, x) + 1$ też miałaby numer, tj. mielibyśmy $f(x) = \Psi(n, x)$, skąd w szczególności $\Psi(n, n) + 1 = f(n) = \Psi(n, n)$. \square

s-m-n-twierdzenie

Twierdzenie o tej dziwacznej nazwie jest intuicyjnie tak naturalne, że zwykle posługujemy się nim nie zauważając tego. Rozpatrzmy na początek wersję dwuwymiarową.

Fakt 4.2 *Jeśli $\varphi : \mathbb{N}^2 \rightarrow \mathbb{N}$ jest częściowo rekurencyjna, to istnieje taka rekurencyjna (całkowita) funkcja s , że*

$$\varphi(x, y) = \varphi_{s(x)}(y)$$

zachodzi dla dowolnych x, y . (Inaczej możemy napisać że $\varphi_{s(x)} = \lambda y. \varphi(x, y)$.)

Dowód: Dla danego $x \in \mathbb{N}$, liczba $s(x)$ jest numerem definicji funkcji

$$\xi = \lambda y. \varphi(x, \Pi_1^1(y)).$$

Taki numer można efektywnie obliczyć z danego x . Przy naszej numeracji⁷ mamy

$$s(x) = \langle 1, \langle 2, \langle p, \langle l(x), \langle 0, 1 \rangle \rangle \rangle \rangle \rangle,$$

gdzie p jest (ustalonym) numerem funkcji φ , natomiast $l(x)$ jest numerem funkcji stale równej x , tj. funkcji $\eta = \lambda y. \text{succ}^x(Z_1(y))$. Aby obliczyć $l(x)$, zauważmy, że:

- $l(0)$ to numer funkcji Z_1 , czyli $l(0) = \langle 0, 0 \rangle$;
- $l(x + 1)$ to numer funkcji $\lambda y. \text{succ}(x)$, czyli liczba $\langle 1, \langle 1, \langle \langle 0, 2 \rangle, l(x) \rangle \rangle \rangle$.

Widzimy więc, że funkcja l jest definiowalna przez rekursję prostą (i składanie). Szczegóły pozostawiamy czytelnikowi. Uwaga: nasza definicja funkcji l nie jest w istocie zgodna ze schematem rekursji prostej (por. definicję poprzednika w rozdziale 1). \square

Podobnie udowodnimy też ogólniejszą wersję powyższego faktu.

⁷Jeśli utożsamiać definicje funkcji z maszynami Turinga, to $s(x)$ jest numerem maszyny, która dla danego wejścia y dopisuje z przodu x i uruchamia maszynę obliczającą φ .

Twierdzenie 4.3 (s-m-n-twierdzenie) Dla dowolnych $m, n \geq 1$ istnieje funkcja rekurencyjna $s_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$, spełniająca dla dowolnych i, \vec{x}, \vec{y} równość

$$\varphi_i^{(m+n)}(\vec{x}, \vec{y}) = \varphi_{s_n^m(i, \vec{x})}^{(n)}(\vec{y})$$

Jeśli więc mówimy, że numer funkcji $\lambda y.y^n$, albo numer zbioru $\{k \mid k \text{ podzielne przez } n\}$ jest efektywnie obliczalny z n , to w istocie korzystamy z s-m-n-twierdzenia. W pierwszym przypadku mamy bowiem funkcję rekurencyjną s spełniającą równość $\varphi_{s(n)}(y) = y^n$, a w drugim mamy funkcję s o własności $\varphi_{s(n)}(y) = \chi_W(n, y)$, gdzie $W = \{\langle n, k \rangle \mid k \text{ podzielne przez } n\}$.

Uwaga: Należy pamiętać, że każda funkcja częściowo rekurencyjna ma nieskończenie wiele numerów. Nasze s-m-n-twierdzenie mówi, że istnieje algorytm pozwalający zawsze znaleźć *jakiś* numer poszukiwanej funkcji.

Twierdzenie o rekursji

Jako rozgrzewkę proponujemy czytelnikowi rozwiązanie Ćwiczenia 3. W razie kłopotów należy przeczytać dowód poniższego twierdzenia. Bywa ono nazywane „twierdzeniem o punkcie stałym”, ale w istocie nie stwierdza istnienia punktu stałego, bo przypisanie $\varphi_n \mapsto \varphi_{f(n)}$ nie jest dobrze określonym przekształceniem na funkcjach. Z warunku $\varphi_n = \varphi_m$ nie musi bowiem wynikać $\varphi_{f(n)} = \varphi_{f(m)}$.

Twierdzenie 4.4 (Twierdzenie o rekursji) Dla każdej rekurencyjnej funkcji $f : \mathbb{N} \rightarrow \mathbb{N}$ istnieje taka liczba n , że $\varphi_n = \varphi_{f(n)}$.

Dowód: Rozpatrzmy funkcję $\psi(x, y) = \varphi_{f(\varphi_x(x))}(y)$. Na mocy s-m-n-twierdzenia mamy $\psi(x, y) = \varphi_{s(x)}(y)$, dla pewnej rekurencyjnej funkcji s . (Uwaga: $s(x) \neq f(\varphi_x(x))$!) Oczywiście funkcja s też ma numer, powiedzmy, że $s = \varphi_m$. Dla dowolnych x, y zachodzi więc równość $\varphi_{\varphi_m(x)}(y) = \varphi_{f(\varphi_x(x))}(y)$, w szczególności $\varphi_{\varphi_m(m)}(y) = \varphi_{f(\varphi_m(m))}(y)$. A więc można przyjąć $n = s(m) = \varphi_m(m)$ i mamy $\varphi_n = \varphi_{f(n)}$. \square

Powyższy dowód opiera się na takim samym pomysle jak definicja kombinatora punktu stałego $\mathbf{Y} = \lambda f(\lambda x.f(xx))(\lambda x.f(xx))$ w rachunku lambda (ćwiczenie 2.4). Podobieństwo będzie bardziej widać, jeśli zamiast $\varphi_n(m)$ napiszemy $\underline{n}(m)$ i użyjemy nieformalnej lambda-notacji.

Rozpatrzmy funkcję $\lambda x.f(\underline{x}(x))$ i przypuśćmy, że ma ona numer m , tj. $\underline{m} = \lambda x.f(\underline{x}(x))$. Podstawiając m w miejsce x dostajemy równość $\underline{m}(m) = f(\underline{m}(m))$. Stąd $n = f(n)$ dla $n = \underline{m}(m)$. Wyrażenie $\underline{m}(m)$ uderzająco przypomina lambda-term $\mathbf{Y}f =_{\beta} (\lambda x.f(xx))(\lambda x.f(xx))$.

Oczywiście nie każda funkcja ma punkt stały, a błąd polega na tym, że wartość $\underline{m}(m)$ może być nieokreślona. Zamiast równości $\underline{m}(x) = f(\underline{x}(x))$ możemy jednak wziąć słabszy warunek $\underline{m}(x) = \underline{f(\underline{x}(x))}$. Dzięki s-m-n-twierdzeniu mamy całkowitą funkcję \underline{m} , która spełnia ten warunek. Wtedy dla $n = \underline{m}(m)$ otrzymamy $\underline{n} = \underline{f(n)}$.

Wniosek 4.5 Jeśli $\varphi : \mathbb{N}^2 \rightarrow \mathbb{N}$ jest częściowo rekurencyjna, to $\varphi_n = \lambda x.\varphi(n, x)$, dla pewnego n . W szczególności:

- Istnieje takie n , że $\varphi_n(x) = x^n$ dla dowolnego x .

- Istnieje takie k , że $W_k = \{k\}$.
- Istnieje takie m , że $\varphi_m(x) = m$ dla dowolnego x .

Dowód: Niech s będzie taką funkcją obliczalną, że $\varphi_{s(n)} = \lambda x.\varphi(n, x)$. Należy zastosować Twierdzenie 4.4 do funkcji s . \square

Ostatnia część Wniosku 4.5 to rozwiązanie Ćwiczenia 3 w języku programowania, gdzie programami są numery algorytmów. Niezależnie od danych wejściowych, program φ_m generuje w wyniku własny numer. Rozwiązanie w Pascalu to już tylko sprawa implementacji.

Inne zastosowania twierdzenia o rekursji mogą polegać na dowodzeniu poprawności (całkowitości) niektórych definicji rekurencyjnych. Na przykład częściową obliczalność funkcji Ackermanna⁸ można wykazać stosując twierdzenie o rekursji w następujący sposób. Niech $\Phi(m, n, x)$ będzie trójargumentową funkcją uniwersalną (dla funkcji dwuargumentowych). Określmy trójargumentową operację $B(m, n, x)$ za pomocą zwykłej definicji warunkowej:

$$\begin{aligned} B(m, 0, x) &= x + 1; \\ B(m, n + 1, 0) &= \Phi(m, n, 1); \\ B(m, n + 1, x + 1) &= \Phi(m, n, \Phi(m, n + 1, x)). \end{aligned}$$

Niech f będzie taką (całkowitą) funkcją, że $\Phi(f(m), x, y) = B(m, x, y)$. Dwuargumentowa wersja twierdzenia o rekursji zastosowana do funkcji f mówi, że istnieje taka liczba n , że dla dowolnych n, x zachodzi $\Phi(m, n, x) = \Phi(f(m), n, x) = B(m, n, x)$. Oznacza to, że $A(n, x)$ można zdefiniować jako $B(m, n, x)$.

Ćwiczenia

1. Udowodnić, że funkcja s_n^m , o której mowa w s-m-n-twierdzeniu, jest elementarnie rekurencyjna.
2. Wywnioskować z s-m-n-twierdzenia, że
 - (a) numer zbioru $W_n \cup W_m$;
 - (b) numer funkcji $\lambda x(\varphi_n(x) \cdot \varphi_m(x))$,
 są efektywnie obliczalne z m i n .
3. Napisać program, który drukuje własny tekst. Nie wolno odwoływać się do szczegółów implementacji (nazwy pliku, miejsca w pamięci itp.) Zadanie łatwe w Lispie, trudne w Pascalu.
4. Dlaczego w dowodzie twierdzenia o rekursji funkcje s i $\varphi_x(x)$ są na pewno różne?
5. Niech $\psi_n(x) = \Psi(n, x)$, gdzie Ψ jest taką funkcją częściowo rekurencyjną, że
 - istnieje funkcja rekurencyjna f spełniająca warunek $\varphi_n = \psi_{f(n)}$ dla wszystkich n .

Pokazać, że każda funkcja częściowo rekurencyjna ψ ma nieskończenie wiele wystąpień w ciągu ψ_n .
Wskazówka: W przeciwnym razie zbiór $\{k \mid \varphi_k = \psi\}$ byłby rekurencyjny.

⁸To, że funkcja Ackermanna jest częściowo rekurencyjna jest dla nas oczywiste, ale tylko dlatego, że znamy maszyny Turinga.

5 Nierozstrzygalność

Zbiór $\mathbb{N}^{\mathbb{N}}$ jest mocy continuum, a zbiór wszystkich funkcji rekurencyjnych jest zaledwie przeliczalny. Z pewnością więc istnieją funkcje całkowite, które nie są rekurencyjne. Możemy jednak podać konkretny przykład:

$$f(x) = \begin{cases} 0, & \text{jeśli } \varphi_x(x) \text{ jest określone;} \\ 1, & \text{w przeciwnym przypadku.} \end{cases}$$

Gdyby f była rekurencyjna, to funkcja

$$\psi(x) = \begin{cases} 0, & \text{jeśli } \varphi_x(x) \text{ jest nieokreślone;} \\ \text{nieokreślone,} & \text{w przeciwnym przypadku.} \end{cases}$$

byłaby częściowo rekurencyjna, zatem $\psi = \varphi_k$ dla pewnego k . Podstawiając k w miejsce x otrzymujemy sprzeczność: $\varphi_k(k)$ jest określone wtedy i tylko wtedy, gdy jest nieokreślone.

Zastosowana powyżej technika przekątniowa pozwala także wywnioskować, że nie każdą funkcję częściowo rekurencyjną można rozszerzyć do funkcji rekurencyjnej. Inaczej mówiąc: to, że pewne algorytmy są częściowe, jest nieuniknione.

Twierdzenie 5.1 *Istnieje taka funkcja częściowo rekurencyjna $\varphi : \mathbb{N} \dashrightarrow \mathbb{N}$, że żadna funkcja całkowita $f : \mathbb{N} \rightarrow \mathbb{N}$ zawierająca φ nie jest rekurencyjna.*

Dowód: Wystarczy przyjąć $\varphi(x) = \varphi_x(x) + 1$. Jeśli $\varphi \subseteq f$, gdzie f jest rekurencyjna, to dla pewnego k mamy $f = \varphi_k$. Ponieważ f jest całkowita, więc $\varphi_k(k)$ jest określone. Stąd wynika, że $\varphi(k)$ jest też określone i mamy $\varphi_k(k) = f(k) = \varphi(k) = \varphi_k(k) + 1$. \square

Metodą przekątniową łatwo też udowodnimy istnienie zbiorów nierekurencyjnych.

Twierdzenie 5.2 *Zbiory $K = \{n \mid \varphi_n(n) \text{ określone}\}$ i $S = \{\langle m, n \rangle \mid \varphi_m(n) \text{ określone}\}$ nie są rekurencyjne.⁹*

Twierdzenie 5.2 zwykle formułujemy tak: Pytanie czy dana funkcja częściowo rekurencyjna jest określona dla danego argumentu stanowi *problem nierozstrzygalny*. Jest to problem stopu dla maszyn Turinga wyrażony w języku funkcji częściowo rekurencyjnych. Ponieważ zbiory K i S są oczywiście rekurencyjnie przeliczalne, więc ich dopełnienia nie mogą być rekurencyjnie przeliczalne. A więc problem nieokreśloności funkcji nie jest nawet częściowo rozstrzygalny.

Dowody nierozstrzygalności konkretnych problemów zwykle polegają na *redukcji* jednego problemu do innego. Mówimy, że zbiór A *redukuje się* (lub jest *sprowadzalny*) do zbioru B , i piszemy $A \leq B$, gdy istnieje funkcja obliczalna f o takiej własności:¹⁰

$$x \in A \quad \text{wtedy i tylko wtedy, gdy} \quad f(x) \in B.$$

W praktyce oznacza to tyle, że istnieje algorytm przekształcający każdą możliwą instancję x problemu A w instancję $f(x)$ problemu B , w ten sposób, że pytanie „Czy $x \in A$?” można sprowadzić do pytania „Czy $f(x) \in B$?”.

⁹Możemy także napisać $K = \{n \mid n \in W_n\}$ i $S = \{\langle m, n \rangle \mid m \in W_n\}$.

¹⁰Relacje \leq oznacza się też przez \leq_m (od *many-one reducibility*). Jeśli zaś funkcja f jest różnowartościowa, to można napisać $A \leq_1 B$.

Fakt 5.3 Niech $A \leq B$. Jeśli B jest rozstrzygalny (rekurencyjnie przeliczalny) to A też jest rozstrzygalny (rekurencyjnie przeliczalny). Ponadto, jeśli $\neg B$ jest rekurencyjnie przeliczalny, to $\neg A$ też jest rekurencyjnie przeliczalny.

Dowód: Łatwy. □

Wniosek 5.4 Zbiór $A = \{n \mid \varphi_n \text{ jest funkcją całkowitą}\}$ nie jest rekurencyjny. Dokładniej, zbiór $\neg A$ nie jest rekurencyjnie przeliczalny.

Dowód: Pokażemy, że $K \leq A$, skąd na mocy Faktu 5.3 otrzymamy tezę. Dla $x \in \mathbb{N}$ rozpatrzmy funkcję $\vartheta_x(y) = Z_1(\varphi_x(x))$. W zależności od tego, czy $x \in K$ jest to funkcja stała lub nigdzie nie określona. Funkcja ϑ_x jest częściowo rekurencyjna, a jej numer jest obliczalny z x , tj. $\vartheta_x = \varphi_{s(x)}$ dla pewnej rekurencyjnej funkcji s . Ponadto, funkcja ϑ_x jest całkowita wtedy i tylko wtedy, gdy określone jest $\varphi_x(x)$. A więc $K \leq A$, bo

$$x \in K \quad \text{wtedy i tylko wtedy, gdy} \quad s(x) \in A. \quad \square$$

Twierdzenie Rice'a

Następne twierdzenie dostarcza prostego warunku wystarczającego na nierekurencyjność. Powiemy, że zbiór $A \subseteq \mathbb{N}$ jest *nietrywialny*, gdy $A \neq \emptyset$ i $A \neq \mathbb{N}$. Zbiór A jest zaś *adekwatny*, gdy z warunków $n \in A$ i $W_n = W_m$ wynika $m \in A$. Inaczej mówiąc, zbiór adekwatny określa pewną własność zbiorów rekurencyjnie przeliczalnych niezależną od wyboru indeksu. Poniższe twierdzenie możemy więc odczytać tak: Każda nietrywialna własność zbiorów rekurencyjnie przeliczalnych jest nierozstrzygalna.

Twierdzenie 5.5 (Rice'a) Jeśli zbiór rekurencyjnie przeliczalny jest nietrywialny i adekwatny, to nie jest rekurencyjny.

Dowód: Niech $X \subseteq \mathbb{N}$ będzie nietrywialny i adekwatny. Bez straty ogólności możemy przyjąć, że zbiór pusty nie ma własności X , tj. że $m \notin X$, gdy $W_m = \emptyset$ (w przeciwnym razie zamiast X rozważamy zbiór $\neg X$). Załóżmy jeszcze, że $k \in X$.

Pokażemy, że $K \leq X$. W tym celu, dla dowolnego n rozpatrzmy maszynę Turinga M^n , która dla dowolnego wejścia x :

- najpierw oblicza $\varphi_n(n)$;
- a potem (jeśli $\varphi_n(n)$ było określone) sprawdza, czy $x \in W_k$.

Maszyna M^n akceptuje pewien zbiór $W_{f(n)}$ i właśnie stwierdziliśmy, że

$$W_{f(n)} = \begin{cases} W_k, & \text{jeśli } n \in K; \\ \emptyset, & \text{w przeciwnym przypadku.} \end{cases}$$

Inaczej mówiąc $f(n) \in X$ wtedy i tylko wtedy, gdy $n \in K$, co dowodzi, że $K \leq X$. □

Wniosek 5.6 *Następujące zbiory nie są rekurencyjne:*

1. $\{n \mid W_n \text{ jest skończony}\};$
2. $\{n \mid W_n \text{ jest ko-skończony}\};$
3. $\{n \mid W_n = \emptyset\};$
4. $\{n \mid W_n \text{ jest rekurencyjny}\};$
5. $\{n \mid 0 \in W_n\}.$

Przykłady takie, jak we Wniosku 5.6 można łatwo mnożyć. Twierdzenie Rice'a nie jest jednak aż tak negatywne, jak może się wydawać. Własności algorytmów (maszyn Turinga) nie zawsze są adekwatne, tj. nie są własnościami rozpoznawanych przez nie zbiorów. Twierdzenie Rice'a nie stosuje się więc np. do problemu: *Czy dana niedeterministyczna maszyna Turinga ma dla każdego wejścia chociaż jedno obliczenie nieskończone?*

Ćwiczenia

1. Skąd wiadomo, że funkcja f w dowodzie twierdzenia Rice'a jest rekurencyjna?
Wskazówka: Z s-m-n-twierdzenia.
2. Sformułować i udowodnić twierdzenie Rice'a dla relacji wieloargumentowych i dla funkcji.
3. Udowodnić, że następujące zbiory są nierekurencyjne:
 - $\{n \mid \varphi_n(0) \text{ jest określone}\};$
 - $\{n \mid \varphi_n \text{ jest monotoniczna}\};$
 - $\{n \mid \text{Dom}(\varphi_n) = \emptyset\};$
 - $\{n \mid \varphi_n \text{ jest różnowartościowa}\}.$
4. Udowodnić, że $S \leq_m K$. *Wskazówka:* Dla danych n, k niech $A_{n,k} = \mathbb{N}$, gdy $n \in W_k$ oraz niech $A_{n,k} = \emptyset$, w przeciwnym przypadku. Jeśli teraz $A_{n,k} = W_{s(n,k)}$ to $\langle n, k \rangle \in S$ zachodzi wtedy i tylko wtedy, gdy $s(n, k) \in K$.

6 Stopnie nierozstrzygalności

Typową metodą dowodu nierozstrzygalności danego zbioru A jest redukcja postaci $S \leq_m A$ lub $K \leq_m A$. Zatem znane problemy nierozstrzygalne są w istocie co najmniej tak trudne jak problem stopu. Pokażemy teraz, że istnieją jednak zbiory nierekurencyjne, które nie mają tej własności, tj. nie są *zupelne* w klasie zbiorów rekurencyjnie przeliczalnych ze względu na m-sprowadzalność.

Mówimy, że zbiór $A \subseteq \mathbb{N}$ jest *produktywny*, gdy istnieje taka częściowo rekurencyjna funkcja ψ , że $\psi(i) \in A - W_i$ (w szczególności $\psi(i)$ jest określone) dla dowolnego $W_i \subseteq A$.

Przykładem zbioru produktywnego jest $\neg K = \{n \mid n \notin W_n\}$ (przy funkcji identycznościowej).

Lemat 6.1

1. Każdy zbiór produktywny zawiera nieskończony podzbiór rekurencyjnie przeliczalny.
2. Jeśli $-K \leq_m A$ to A jest produktywny.

Dowód: (1) Niech A będzie produktywny i niech ψ będzie odpowiednią funkcją. Ponieważ zbiór pusty jest oczywiście zawarty w A , więc $\psi(i) \in A$, gdy i jest numerem zbioru pustego. Zbiór $\{\psi(i)\}$ też ma jakiś numer j , więc $\psi(j) \in A$, przy czym $\psi(i) \neq \psi(j)$. I tak dalej.¹¹

(2) Niech f będzie taką funkcją, że $n \in -K$ wtedy i tylko wtedy, gdy $f(n) \in A$, i niech $W_i \subseteq A$. Zbiór $f^{-1}(W_i) \subseteq -K$ jest rekurencyjnie przeliczalny, co więcej (na mocy s-m-n-twierdzenia) jego numer $g(i)$ jest obliczalny i mamy $g(i) \in -K - W_{g(i)}$. Zatem $f(g(i)) \in A - W_i$. \square

Twierdzenie 6.2 *Istnieje rekurencyjnie przeliczalny ale nie rekurencyjny zbiór, do którego problem stopu nie jest m-sprowadzalny.*

Dowód: Problem stopu $S = \{\langle m, n \rangle \mid m \in W_n\}$, jako zbiór rekurencyjnie przeliczalny, jest obrazem pewnej funkcji rekurencyjnej f . Rozpatrzmy funkcję częściową

$$j(k) = \mu x [r(f(x)) = k \wedge \ell(f(x)) > 2k]$$

i rekurencyjnie przeliczalny zbiór $X = \text{Rg}(\ell \circ f \circ j)$. Zauważmy, że $n \in X$ zachodzi wtedy i tylko wtedy, gdy dla pewnego k para $\langle n, k \rangle$ jest, w kolejności wyznaczonej przez f , pierwszym elementem zbioru S spełniającym warunek $n > 2k$. Ponieważ wśród liczb od 0 do $2k$ występuje co najwyżej k elementów zbioru X , więc $-X$ jest zbiorem nieskończonym.

Zauważmy dalej, że w każdym zbiorze nieskończonym W_k są liczby większe od $2k$, zatem $j(k)$ jest określone i mamy $X \cap W_k \neq \emptyset$. A więc $-X$ nie ma nieskończonego podzbioru rekurencyjnie przeliczalnego. Wyciągamy stąd dwa wnioski. Po pierwsze, sam zbiór $-X$, jako nieskończony, nie jest rekurencyjnie przeliczalny, a więc X nie jest rekurencyjny. Po drugie, z Lematu 6.1 wynika, że $-K \not\leq_m -X$, czyli $K \not\leq_m X$. \square

Jeśli $A \leq_m B$ oraz $B \leq_m A$, to mówimy, że zbiory A i B mają ten sam *m-stopień* i piszemy $A \equiv_m B$. Z twierdzenia 6.2 wynika, że struktura m-stopni jest nietrywialna: istnieją co najmniej trzy różne m-stopnie rekurencyjnie przeliczalne. Istotnie, każdy zbiór rekurencyjnie przeliczalny jest m-sprowadzalny do problemu stopu,¹² mamy bowiem

$$x \in W_n \quad \text{wtedy i tylko wtedy, gdy} \quad \varphi_n(x) \text{ jest określone.}$$

Z drugiej strony, każdy zbiór rekurencyjny jest m-sprowadzalny do dowolnego zbioru. Mamy więc najmniejszy m-stopień zbiorów rekurencyjnych, największy m-stopień problemu stopu, wiemy też, że istnieją stopnie pośrednie.

Oprócz m-sprowadzalności rozważa się też inne pojęcia redukcji, w szczególności redukcje *w sensie Turinga*: piszemy $A \leq_T B$ gdy istnieje maszyna Turinga rozpoznająca zbiór A

¹¹Ścisłe rzecz biorąc, należy zdefiniować przez indukcję dwie funkcje rekurencyjne f i g , w ten sposób, że $W_{g(i)} = \{f(0), \dots, f(i-1)\}$ dla dowolnego i , oraz $\psi(g(i)) = f(i)$.

¹²A więc także do K , patrz ćwiczenie 4 do rozdziału 5.

z pomocą „wycroczni” odpowiadającej na pytania postaci „czy dane słowo należy do B ?”. Mówimy wtedy też, że A jest T -srowadzalny do B . Różnica między \leq_T i \leq_m polega m.in. na tym, że liczba możliwych „zapytań o B ” jest dowolna i nie jest a priori ograniczona. Zatem $A \leq_m B$ implikuje $A \leq_T B$, ale niekoniecznie na odwrót. Zauważmy, np. że $-K \leq_T K$.

Analogicznie do m -stopni definiujemy T -stopnie, zwane czasem po prostu *stopniami nierozstrzygalności*, jako klasy abstrakcji relacji

$$A \equiv_T B \quad \text{wtedy i tylko wtedy, gdy} \quad A \leq_T B \text{ oraz } B \leq_T A.$$

Każdy T -stopień jest sumą pewnych m -stopni, a więc problemy, które nie są m -równoważne mogą jednak być T -równoważne. Z Twierdzenia 6.2 nie wynika więc, że istnieją nietrywialne rekurencyjnie przeliczalne T -stopnie. To pytanie nazywano kiedyś *problemem Posta*.

6.1 Rozwiązanie problemu Posta

Skonstruujemy teraz takie dwa zbiory rekurencyjnie przeliczalne A i B , że $A \not\leq_T B$ oraz $B \not\leq_T A$. Aby tak było, musimy pokazać, że żadna maszyna z wycrocznią A nie rozstrzyga zbioru B i na odwrót.

Oznaczmy przez M_e^X deterministyczną maszynę Turinga o numerze e , która używa wycroczni X . Naszym celem jest konstrukcja zbiorów A i B jako sum ciągów przybliżeń A_n i B_n . Jednocześnie definiujemy ciąg „świadków” $F(j)$, spełniających następujące równoważności:¹³

$$\begin{aligned} F(2e) \in B & \quad \text{wtedy i tylko wtedy, gdy} \quad M_e^A \text{ odrzuca } F(2e); \\ F(2e+1) \in A & \quad \text{wtedy i tylko wtedy, gdy} \quad M_e^B \text{ odrzuca } F(2e+1). \end{aligned}$$

Oczywistym problemem jest to, że dodanie nowego elementu do zbioru A lub B może popsuć dotychczasowe własności „świadków”. Zatem ciąg F będzie „granica” ciągów F_n uzyskiwanych w poszczególnych fazach konstrukcji (w razie potrzeby znajdujemy nowych świadków).

Konstrukcja A_n , B_n i F_n przebiega przez indukcję ze względu na n i zaczyna się od

$$A_0 = \emptyset = B_0, \quad F_0(x) = 2^x.$$

Krok indukcyjny dla $n > 0$ zależy od pierwszej współrzędnej liczby n interpretowanej jako para $n = \langle l, r \rangle$.

Przypadek parzysty. Załóżmy, że $n = \langle 2e, \text{coś} \rangle$ oraz

1. $F_n(2e) \notin B_n$;
2. maszyna $M_e^{A_n}$ odrzuca wejście $F_n(2e)$.

Obliczenie maszyny $M_e^{A_n}$ odrzucającej $F_n(2e)$ jest skończone, zatem zadaje wycroczni A_n tylko skończenie wiele pytań. Niech k będzie takie, że wszystkie odwołania do A_n dotyczą liczb mniejszych od k . Definiujemy teraz $A_{n+1} = A_n$, $B_{n+1} = B_n \cup \{F_n(2e)\}$, oraz

$$F_{n+1}(x) = \begin{cases} 3^k \cdot F_n(x), & \text{jeśli } x > 2e \text{ i } x \text{ jest nieparzyste;} \\ F_n(x), & \text{w przeciwnym przypadku.} \end{cases}$$

Zauważmy, że liczba $3^k \cdot F_n(x)$ jest większa od k . Ewentualne późniejsze dodanie jej do zbioru A_n nie zmieni więc odpowiedzi wycroczni na pytania o liczby mniejsze niż k .

¹³Przez „odrzucanie” rozumiemy w tym dowodzie zatrzymanie maszyny w stanie odrzucającym.

Przypadek nieparzysty. Niech teraz $n = \langle 2e + 1, \text{coś} \rangle$. Jeżeli zachodzą warunki

1. $F_n(2e + 1) \notin A_n$;
2. maszyna $M_d^{B_e}$ odrzuca wejście $F_n(2e + 1)$,

to $B_{n+1} = B_n$, $A_{n+1} = A_n \cup \{F_n(2e + 1)\}$, oraz

$$F_{n+1}(x) = \begin{cases} 3^k \cdot F_n(x), & \text{jeśli } x > 2e + 1 \text{ i } x \text{ jest parzyste;} \\ F_n(x), & \text{w przeciwnym przypadku,} \end{cases}$$

gdzie k jest takie, że obliczenie $M_e^{B_n}$ na wejściu $F_n(2e + 1)$ odwołuje się do wyrocni B_n tylko dla argumentów mniejszych od k .

Przypadek trywialny. Jeśli nie zachodzi żaden z powyższych przypadków (warunki 1–2 nie są spełnione) to wszystko zostaje bez zmian: $A_{n+1} = A_n$, $B_{n+1} = B_n$ i $F_{n+1} = F_n$.

Przypadek parzysty i nieparzysty są do siebie całkowicie dualne. W dalszym ciągu zwykle rozważamy tylko jeden z nich. Ponieważ ciągi zbiorów A_n i B_n są wstępujące, więc możemy od razu zdefiniować

$$A = \bigcup_{n \in \mathbb{N}} A_n \quad \text{oraz} \quad B = \bigcup_{n \in \mathbb{N}} B_n.$$

Procedura opisana powyżej jest efektywna, więc zbiory A i B są rekurencyjnie przeliczalne.

Lemat 6.3 Dla dowolnego x istnieje takie n_x , że $F_n(x) = F_{n_x}(x)$ dla wszystkich $n \geq n_x$.

Dowód: Indukcja ze względu na x . Przypuśćmy, że x jest parzyste, oraz $F_{n+1}(x) \neq F_n(x)$. Wtedy $\ell(n)$ jest nieparzyste i mniejsze od x , oraz $F_n(\ell(n)) \in A_{n+1} - A_n$. Jeśli ponadto $F_{m+1}(x) \neq F_m(x)$ dla pewnego $m \neq n$ to mamy też $F_m(\ell(m)) \in A_{m+1} - A_m$. Oznacza to, że $F_n(\ell(n)) \neq F_m(\ell(m))$. W ten sposób każdemu n , takiemu że $F_{n+1}(x) \neq F_n(x)$, przypisujemy inny element zbioru $\{F_m(y) \mid m \in \mathbb{N} \wedge y < x\}$. Z założenia indukcyjnego wynika, że ten zbiór jest skończony, a więc x wartość $F_n(x)$ może się zmieniać tylko skończenie wiele razy. \square

Z powyższego lematu wynika, że dla dowolnego x wartość $F_n(x)$ „ustala się” od pewnego miejsca, możemy więc zdefiniować:

$$F(x) = F_n(x) \text{ dla dostatecznie dużych } n.$$

Lemat 6.4 Jeżeli $F_n(x) = F_m(y)$ to $x = y$. W szczególności F jest różnowartościowa.

Dowód: Mamy $F_n(x) = 2^x \cdot 3^{\text{coś}}$, a to jednoznacznie określa x . \square

Lemat 6.5 Jeśli M_e^A odrzuca wejście $F(2e)$, to $F(2e) \in B$.

Dowód: Obliczenie maszyny M_e^A dla wejścia $F(2e)$ używa wyrocni A tylko dla argumentów mniejszych od pewnego k . Weźmy takie $n > k$, że $F(2e) = F_n(2e)$ oraz

$$A \cap \{0, 1, \dots, k\} = A_n \cap \{0, 1, \dots, k\}.$$

Wtedy z punktu widzenia naszego obliczenia wyroczenia A_n jest równie dobra jak wyroczenia A , a więc maszyna $M_e^{A_n}$ też odrzuca wejście $F(2e)$. Jeśli $F(2e) = F_n(2e) \in B_n$ to już jest dobrze, bo $B_n \subseteq B$. W przeciwnym razie stosuje się przypadek parzysty i mamy $F(2e) \in B_{n+1}$. \square

Lemat 6.6 *I na odwrót: jeśli $F(2e) \in B$ to maszyna M_e^A odrzuca $F(2e)$.*

Dowód: Załóżmy, że $F(2e) \in B$ i niech n będzie takie, że $F(2e) \in B_{n+1} - B_n$. Dla n zachodzi wtedy przypadek parzysty, a z Lematu 6.4 wynika, że $\ell(n) = 2e$. Maszyna $M_e^{A_n}$ odrzuca więc $F(2e)$ używając wyroczeni A_n dla argumentów mniejszych od pewnego k . Aby stwierdzić, że maszyna M_e^A też odrzuca $F(2e)$, wystarczy sprawdzić, że

$$A \cap \{0, 1, \dots, k\} = A_n \cap \{0, 1, \dots, k\}.$$

Przypuśćmy, że $z \in A - A_n$, czyli istnieje takie $m > n$, że $z \in A_{m+1} - A_m$. Wtedy $\ell(m) = 2d+1$ dla pewnego d i dla m zachodzi przypadek nieparzysty. Ponieważ wartość $F(2e) = F_n(2e)$ „już się ustaliła”, więc $F_m(2e) = F_n(2e)$. To znaczy, że $2e < 2d+1$, bo inaczej mielibyśmy $F_{m+1}(2e) > F_m(2e)$.

A zatem $z = F_m(2d+1) \geq F_{n+1}(2d+1) = 3^k \cdot F_n(2d+1) > k$, zbiory A i A_n mogą się więc różnić tylko powyżej k . \square

Twierdzenie 6.7 (A.A. Muchnik, R.M. Friedberg) *Istnieją nieporównywalne rekurencyjnie przeliczalne T-stopnie.*

Dowód: Jeśli $B \leq_T A$, to pewna maszyna M_e^A rozstrzyga zbiór B . Z lematów 6.5 i 6.6 otrzymujemy sprzeczność: maszyna M_e^A odrzuca $F(2e)$ wtedy i tylko wtedy, gdy akceptuje $F(2e)$. Analogicznie dowodzimy, że $A \not\leq_T B$. \square

Z twierdzenia 6.7 wynika, że istnieją nierozstrzygalne problemy decyzyjne, których nierozstrzygalności nie można jednak udowodnić metodą redukcji z problemu stopu. Autorowi niniejszego nie jest jednak znany żaden nierozstrzygalny problem „matematyczny” (np. kombinatoryczny), który nie jest zupełny w sensie Turinga.

Ćwiczenia

1. Uogólnić Lemat 6.1(2): Jeśli B jest produktywny i $B \leq_m A$ to A jest produktywny.
2. Funkcje i zbiory *rekurencyjne względem* (całkowitej) funkcji g definiujemy tak, jak funkcje rekurencyjne, ale dodając g do funkcji bazowych. Pokazać, że $A \leq_T B$ wtedy i tylko wtedy, gdy zbiór A jest rekurencyjny względem funkcji charakterystycznej c_B zbioru B .
3. Zbiór A jest *tablicowo sprowadzalny* do zbioru B , co zapisujemy $A \leq_{tt} B$, gdy istnieje rekurencyjna funkcja f , która każdemu $x \in \mathbb{N}$ przypisuje „warunek”, czyli skończony zbiór A_x i kombinację boolowską pytań postaci „ $a \in B$?”, gdzie $a \in A$, oraz dla dowolnego x ,

$$x \in A \quad \text{wtedy i tylko wtedy, gdy} \quad \text{warunek } f(x) \text{ jest spełniony.}$$

Niech teraz $K^* = \{n \in \mathbb{N} \mid n \in K \text{ oraz warunek } f(x) \text{ nie jest spełniony dla } K\}$.

Udowodnić, że $K^* \leq_T K$ ale $K^* \not\leq_{tt} K$. (A zatem tt-stopnie są „drobniejsze” niż T-stopnie.)

7 Hierarchia arytmetyczna

Problem stopu jest wprawdzie nierozstrzygalny, ale jest rekurencyjnie przeliczalny, tj. rozstrzygalny częściowo. Może jednak być gorzej.

Fakt 7.1 *Zbiór $A = \{n \mid \varphi_n \text{ jest funkcją całkowitą}\}$ i jego dopełnienie nie są rekurencyjnie przeliczalne.*

Dowód: Mamy już Wniosek 5.4, więc wystarczy pokazać, że A nie jest rekurencyjnie przeliczalny. W przeciwnym razie, A byłby obrazem pewnej rekurencyjnej funkcji g . Wtedy funkcja $H(n, x) = \Phi(g(n), x) = \varphi_{g(n)}(x)$ byłaby uniwersalna w klasie funkcji rekurencyjnych, co jest niemożliwe (Fakt 4.1). \square

Zbiory, które nie są rekurencyjne, można klasyfikować ze względu na stopień skomplikowania formuł wyrażających ich definicje w języku formalnej arytmetyki. Przez *arytmetykę* rozumiemy tu teorię w języku pierwszego rzędu zawierającym dwuargumentowe symbole funkcyjne $+$ i \cdot , jednoargumentowy symbol \mathbf{s} dla następnika i stałą 0 . Jedynym symbolem relacyjnym jest znak równości. Dla dowolnej liczby $n \in \mathbb{N}$, skrót \underline{n} oznacza term $\mathbf{s}(\mathbf{s}(\dots \mathbf{s}(0) \dots))$, gdzie symbol \mathbf{s} występuje n razy. Strukturę $\mathcal{N} = \langle \mathbb{N}, +, \cdot, \mathbf{s}, 0 \rangle$, w której symbole arytmetyki są interpretowane „jak zwykle”, nazywamy *standardowym modelem arytmetyki*.

Mówimy, że k -argumentowa relacja nad \mathbb{N} jest *arytmetyczna*, gdy istnieje formuła $\varphi(\vec{x})$, która ma (co najwyżej) k zmiennych wolnych \vec{x} , i spełnia dla dowolnego $\vec{n} \in \mathbb{N}^k$ warunek

$$\vec{n} \in r \quad \text{wtedy i tylko wtedy, gdy} \quad \mathcal{N} \models \varphi(\underline{\vec{n}}).$$

Funkcja jest *arytmetyczna*, gdy jest arytmetyczna jako relacja. Inaczej można powiedzieć, że funkcje (relacje) arytmetyczne, to te funkcje (relacje), które można zdefiniować za pomocą formuł arytmetyki pierwszego rzędu.

Następujące twierdzenie jest słabą wersją tzw. twierdzenia Gödla o reprezentacji.

Twierdzenie 7.2 *Każda funkcja rekurencyjna jest arytmetyczna.*

Dowód: Nietrudno sprawdzić, że funkcje bazowe (stała zero, rzuty, następnik) są arytmetyczne, oraz że funkcja otrzymana przez złożenie funkcji arytmetycznych musi być arytmetyczna. Dla funkcji f zdefiniowanej przez minimum efektywne

$$f(\vec{x}) = \mu y (g(\vec{x}, y) = 0),$$

gdzie g jest definiowalna formułą $\psi(\vec{x}, y, z)$, możemy napisać formułę

$$\varphi(\vec{x}, y) \equiv \psi(\vec{x}, y, 0) \wedge \forall z (z < y \rightarrow \neg \psi(\vec{x}, y, 0)),$$

w której $z < y$ jest skrótem wyrażenia $\exists u (z + u = y \wedge u \neq 0)$. Pozostaje przypadek funkcji określonej przez rekursję prostą:

$$\begin{aligned} f(0, \vec{x}) &= g(\vec{x}); \\ f(s(m), \vec{x}) &= h(\vec{x}, m, f(m, \vec{x})). \end{aligned}$$

Tutaj musimy się odwołać pewnego tricku z teorii liczb, a mianowicie do *funkcji beta* Gödla:

$$\beta(x, y, i) = x \bmod(y(i+1) + 1).$$

Najważniejsza własność funkcji beta jest znana jako „chińskie twierdzenie o resztach”.

Dla dowolnego skończonego ciągu k_0, k_1, \dots, k_n istnieją takie liczby a, b , że $\beta(a, b, i) = k_i$ dla $i = 0, \dots, n$.

Funkcja beta, jak łatwo widzieć, jest arytmetyczna. Niech $B(x, y, i, z)$ będzie odpowiednią formułą i niech f będzie funkcją określoną przez rekursję prostą, jak powyżej. Załóżmy, że $\varphi(\vec{x}, y)$ i $\psi(\vec{x}, y, z, u)$ są formułami definiującymi odpowiednio g i h . Wtedy formuła definiująca funkcję f jest taka:

$$\begin{aligned} \exists a, b [\exists w (B(a, b, 0, w) \wedge \varphi(\vec{x}, w)) \wedge B(a, b, y, z) \\ \wedge \forall i (i < y \rightarrow \exists u, v (B(a, b, i, u) \wedge B(a, b, s(i), v) \wedge \psi(\vec{x}, i, u, v)))]]. \end{aligned}$$

Powyższa formuła wyraża następującą równoważność: $f(\vec{x}, y) = z$ wtedy i tylko wtedy, gdy istnieją takie liczby k_0, k_1, \dots, k_y , że $k_0 = g(\vec{x})$ i $k_y = z$, a dla dowolnego $i = 0, \dots, y-1$ zachodzi $k_{i+1} = h(\vec{x}, i, k_i)$. \square

Z powyższego wynika, że każdy zbiór rekurencyjny jest arytmetyczny. Na mocy Faktu 1.10(3), wystarczy jeden kwantyfikator egzystencjalny, aby przejść od zbiorów rekurencyjnych do rekurencyjnie przeliczalnych. A zatem mamy:

Wniosek 7.3 *Wszystkie funkcje częściowo rekurencyjne też są arytmetyczne.*

Zdefiniowanie innych zbiorów arytmetycznych może wymagać większej liczby kwantyfikatorów. Na przykład przynależność liczby n do zbioru A z Faktu 7.1 można wyrazić tak:

Dla dowolnego wejścia x istnieje takie y , że obliczenie $\varphi_n(x)$ kończy się po y krokach.

Jak wiadomo, każdą formułę pierwszego rzędu można równoważnie przedstawić w *preneksowej postaci normalnej* $Q_1 x_1 Q_2 x_2 \dots Q_n x_n \psi$, gdzie każde Q_i to \forall albo \exists , a formuła ψ jest otwarta (nie zawiera kwantyfikatorów). Nam wystarcza, aby definiowała relację obliczalną. Wtedy możliwe jest dalsze uproszczenie, mamy bowiem takie równoważności:

$$\begin{aligned} \mathcal{N} &\models \forall x_1 \forall x_2 \varphi(x_1, x_2) \leftrightarrow \forall x \varphi(\ell(x), r(x)); \\ \mathcal{N} &\models \exists x_1 \exists x_2 \varphi(x_1, x_2) \leftrightarrow \exists x \varphi(\ell(x), r(x)), \end{aligned}$$

a zatem istotne są tylko te prefiksy kwantyfikatorowe, w których kwantyfikatory ogólne występują na przemian ze szczegółowymi. Prowadzi to do następującej klasyfikacji zbiorów, którą nazywamy *hierarchią arytmetyczną* lub *hierarchią Kleene'go-Mostowskiego*.

Definicja 7.4 Niech $A \subseteq \mathbb{N}$. Zbiór A należy do klasy Σ_n (ozn. też Σ_n^0), wtedy i tylko wtedy, gdy dla pewnego rekurencyjnego zbioru $B \subseteq \mathbb{N}^{n+1}$ i wszystkich $y \in \mathbb{N}$ zachodzi

$$y \in A \quad \text{wtedy i tylko wtedy, gdy} \quad \exists x_1 \forall x_2 \exists x_3 \dots Q x_n \cdot \langle x_1, x_2, \dots, x_n, y \rangle \in B,$$

gdzie Q powyżej to \forall lub \exists , zależnie od parzystości n . Dualnie, A jest zbiorem klasy Π_n (ozn. też Π_n^0), gdy dla pewnego rekurencyjnego $B \subseteq \mathbb{N}^{n+1}$ (i odpowiedniego Q) mamy

$y \in \mathcal{A}$ wtedy i tylko wtedy, gdy $\forall x_1 \exists x_2 \forall x_3 \dots Qx_n \cdot \langle x_1, x_2, \dots, x_n, y \rangle \in B$.

Dodatkowo definiujemy $\Delta_n = \Sigma_n \cap \Pi_n$ (piszemy też Δ_n^0).

Nietrudno teraz zauważyć, że

- Każdy zbiór arytmetyczny należy do jednej z klas Σ_n i Π_n .
- Zbiory rekurencyjne tworzą klasę $\Delta_0 = \Sigma_0 = \Pi_0$.
- Klasa Σ_1 to klasa zbiorów rekurencyjnie przeliczalnych.
- Do klasy Π_1 należą dopełnienia zbiorów rekurencyjnie przeliczalnych.
- A zatem $\Delta_1 = \Delta_0$, na mocy Faktu 1.4.
- Zbiór A z Faktu 7.1 jest elementem klasy Π_2 , a jego dopełnienie należy do Σ_2 .

Zbiory arytmetyczne można numerować w podobny sposób jak numeruje się zbiory rekurencyjnie przeliczalne. W numeracji określonej poniżej, symbol $V_{n,m}$ oznacza zbiór z klasy Σ_n o numerze m , a symbol $\Lambda_{n,m}$ oznacza zbiór z klasy Π_n o numerze m .

$$\begin{aligned} V_{1,m} &= W_m; \\ \Lambda_{n,m} &= -V_{n,m}; \\ V_{n+1,m} &= \{x \in \mathbb{N} \mid \exists y \langle x, y \rangle \in \Lambda_{n,m}\} \end{aligned}$$

Odpowiednikiem problemu stopu w klasie Σ_n jest zbiór $S_n = \{\langle x, y \rangle \mid x \in V_{n,y}\}$.

Fakt 7.5 Dla dowolnego n zbiór S_n jest zupełny w klasie Σ_n , tj. $S_n \in \Sigma_n$ oraz $A \leq S_n$ dla dowolnego $A \in \Sigma_n$.

Dowód: Zauważmy, że $\langle x, y \rangle \in S_{n+1}$ wtedy i tylko wtedy, gdy $\langle \langle x, z \rangle, y \rangle \notin S_n$ dla pewnego z . Przez indukcję można więc łatwo pokazać, że $S_n \in \Sigma_n$ dla wszystkich n . Dalej, jeśli $A = V_{n,m}$, to $x \in A$ jest równoważne $\langle x, m \rangle \in S_n$. \square

Hierarchia arytmetyczna jest ostra, w następującym sensie:

Fakt 7.6 Dla każdego $n \geq 1$ zachodzą inkluzje $\Sigma_n, \Pi_n \subsetneq \Sigma_n \cup \Pi_n \subsetneq \Delta_{n+1}$.

Dowód: Rozpatrzmy zbiór $K_n = \{m \in \mathbb{N} \mid \langle m, m \rangle \in S_n\}$. Wówczas oczywiście $K_n \in \Sigma_n$. Mamy jednak $K_n \notin \Pi_n$, w przeciwnym razie mielibyśmy bowiem $-K_n \in \Sigma_n$ skąd $-K_n = V_{n,m}$ dla pewnego m . Wtedy $m \in K_n$ byłoby równoważne temu, że $m \notin K_n$.

A zatem $\Sigma_n \not\subseteq \Pi_n$, co implikuje $\Pi_n \subsetneq \Sigma_n \cup \Pi_n$. Podobnie jest dla Σ_n . Inkluzja $\Sigma_n \cup \Pi_n \subseteq \Delta_{n+1}$ wynika natychmiast z tego, że dostawienie dodatkowego kwantyfikatora (nie wiążącego żadnej zmiennej) nie zmienia znaczenia formuły. Pozostaje pokazać, że i ta inkluzja jest ostra.

Zacznijmy od tego, że klasy Σ_n i Π_n są zamknięte ze względu na sumę zbiorów. Mamy bowiem takie tautologie pierwszego rzędu:

$$\begin{aligned} \exists x \varphi(x) \vee \exists x \psi(x) &\leftrightarrow \exists x (\varphi(x) \vee \psi(x)); \\ \forall x \varphi(x) \vee \forall x \psi(x) &\leftrightarrow \forall x \forall y (\varphi(x) \vee \psi(y)). \end{aligned}$$

Zatem np. alternatywę dwóch formuł postaci $\exists x_1 \forall x_2 \dots \varphi(x_1, x_2, \dots)$ i $\exists x_1 \forall x_2 \dots \psi(x_1, x_2, \dots)$ można zapisać jako jedną formułę $\exists x_1 \forall x_2 \forall y_2 \dots (\varphi(x_1, x_2, \dots) \vee \psi(x_1, y_1, \dots))$. Podobnie jest z iloczynem zbiorów, a więc klasy Δ_n są ciałami zbiorów (algebrami Boole'a). Tymczasem suma $\Sigma_n \cup \Pi_n$ ciałem zbiorów nie jest i z tego wynika nasza teza.

Aby to udowodnić, rozpatrzmy dwa zbiory $X = \{2k \mid k \in K_n\}$ oraz $Y = \{2k + 1 \mid k \notin K_n\}$. Wtedy $X \leq K_n$ oraz $Y \leq -K_n$, skąd $X \in \Sigma_n$ i $Y \in \Pi_n$ (zob. Ćwiczenie 3). Z drugiej strony zarówno $K_n \leq X \cup Y$ jak i $-K_n \leq X \cup Y$, gdyby więc $X \cup Y$ należało do Σ_n to mielibyśmy $-K_n \in \Sigma_n$, czyli sprzeczność. Podobnie, gdyby $X \cup Y \in \Pi_n$ to $K_n \in \Pi_n$ i też byłoby źle. Zatem $X \cup Y \notin \Sigma_n \cup \Pi_n$, a więc klasa $\Sigma_n \cup \Pi_n$ nie jest zamknięta ze względu na sumę. \square

Przykłady

Fakt 7.7 Zbiór $A = \{n \mid \varphi_n \text{ jest funkcją całkowitą}\} = \{n \mid W_n = \mathbb{N}\}$ jest zupełny w klasie Π_2 .

Dowód: Że nasz zbiór jest w klasie Π_2 , to już wiemy. Załóżmy więc, że $X \in \Pi_2$. Wtedy istnieje takie $B \in \Sigma_1$, że $x \in X$ wtedy i tylko wtedy, gdy wszystkie pary postaci $\langle x, y \rangle$ są w zbiorze B . Jeśli teraz $\varphi_{f(x)} = \lambda y. \chi_B(x, y)$ to $x \in X$ wtedy i tylko wtedy, gdy $\varphi_{f(x)}$ jest funkcją całkowitą. Mamy więc redukcję X do A . \square

Fakt 7.8 Zbiór $B = \{n \mid W_n \text{ jest nieskończony}\}$ jest zupełny w klasie Π_2 .

Dowód: Zbiór B jest w Π_2 , bo własność $n \in B$ można wypowiedzieć tak: *Dla dowolnego x istnieje takie $y > x$, że $y \in W_n$.* Zupełność wynika z następującej redukcji $A \leq B$. Dla danego n niech $Z_n = \{x \mid \{0, \dots, x-1\} \subseteq W_n\}$. Mamy $n \in A$ wtedy i tylko wtedy, gdy Z_n jest nieskończony, co oznacza redukcję $A \leq B$, bo numer zbioru Z_n jest obliczalny. \square

Wniosek 7.9 Zbiór $-B = \{n \mid W_n \text{ jest skończony}\}$ jest zupełny w klasie Σ_2 .

Następny będzie przykład zupełny w klasie Σ_3 , ale to wymaga pewnych przygotowań.

Lemat 7.10 Klasy Π_n i Σ_n są zamknięte ze względu na kwantyfikatory ograniczone.

Dowód: Klasa zbiorów rekurencyjnych jest zamknięta ze względu na kwantyfikatory ograniczone, więc dla $n = 0$ teza zachodzi. Dalej użyjemy indukcji. Przypuśćmy, że klasa Σ_n jest zamknięta ze względu na kwantyfikatory ograniczone i rozpatrzmy własność $P(x, y)$ postaci $\exists v \leq y \forall z \phi(x, y, z, v)$, gdzie formuła $\forall z \phi(x, y, z, v)$ ma $n+1$ kwantyfikatorów (tj. definiuje zbiór klasy Π_{n+1}). Własność taką możemy równoważnie wyrazić formułą $\forall u \exists v \leq y \forall z \leq u \phi(x, y, z, v)$, a na mocy założenia indukcyjnego wyrażenie $\exists v \leq y \forall z \leq u \phi(x, y, z, v)$ definiuje zbiór klasy Σ_n .

Jeszcze łatwiej jest w przypadku własności postaci $\forall v \leq y \forall z \phi(x, y, z, v)$. Zatem pokazaliśmy tezę dla Σ_{n+1} , a dla Π_{n+1} argumentacja jest dualna. \square

Symbol $\forall^\infty x$ czytamy „dla prawie wszystkich x ”. Zastępuje on wyrażenie $\exists y \forall x (x \geq y \rightarrow \dots)$.

Lemat 7.11 *Następujące formuły są prawdziwe w \mathcal{N} :*

$$\begin{aligned}\forall^\infty x \forall y \varphi(x, y) &\leftrightarrow \forall^\infty z (\varphi(\ell(z), r(z)) \vee \exists w < r(z) \neg \varphi(\ell(z), w)); \\ \forall^\infty x (\forall y \varphi(x, y) \vee \exists y \psi(x, y)) &\leftrightarrow \forall^\infty z (\varphi(\ell(z), r(z)) \vee \exists w < r(z) \neg \varphi(\ell(z), w) \vee \exists w \psi(\ell(z), w)).\end{aligned}$$

Dowód: Ćwiczenie. □

Fakt 7.12 *Zbiór $C = \{n \mid W_n \text{ jest ko-skończony}\}$ jest zupełny w klasie Σ_3 .*

Dowód: Ko-skończoność zbioru W_n wyraża zdanie: *Istnieje takie x , że każde $y \geq x$ należy do W_n .* Mamy więc $C \in \Sigma_3$. Dla $X \in \Sigma_3$ określimy teraz redukcję $X \leq C$. Niech $Y \in \Pi_2$ będzie takim zbiorem, że $x \in X$ wtedy i tylko wtedy, gdy $\exists y \langle x, y \rangle \in Y$.

Własność $\exists y \langle x, y \rangle \in Y$ jest równoważna stwierdzeniu $\forall^\infty w \exists y \leq w. \langle x, y \rangle \in Y$, a ponieważ klasa Π_2 jest zamknięta ze względu na kwantyfikatory ograniczone, więc możemy napisać, że

$$x \in X \quad \text{wtedy i tylko wtedy, gdy} \quad \forall^\infty w \forall v. \langle w, v, x \rangle \in Z,$$

dla pewnego rekurencyjnie przeliczalnego Z . Użyjemy teraz Lematu 7.11 do przeformułowania warunku $\forall^\infty w \forall v. \langle w, v, x \rangle \in Z$. Najpierw zastosujemy pierwszą równoważność, zauważając przy tym, że wyrażenie postaci $\exists u \leq \ell(z) \langle \ell(z), u, x \rangle \notin Z$ definiuje zbiór klasy Π_1 (Lemat 7.10). Możemy więc zastosować drugą równoważność i otrzymamy

$$x \in X \quad \text{wtedy i tylko wtedy, gdy} \quad \forall^\infty z (\langle z, x \rangle \in Q_1 \vee \langle z, x \rangle \in Q_2 \vee \exists w \langle z, x, w \rangle \in Q_3),$$

gdzie Q_1, Q_2, Q_3 są rekurencyjne. A zatem:

$$x \in X \quad \text{wtedy i tylko wtedy, gdy} \quad \forall^\infty z (\langle z, x \rangle \in T),$$

gdzie T jest pewnym zbiorem rekurencyjnie przeliczalnym. Jeśli $T_x = \{z \mid \langle z, x \rangle \in T\}$, to

$$x \in X \quad \text{wtedy i tylko wtedy, gdy} \quad T_x \text{ jest ko-skończony,}$$

A więc $X \leq C$, bo numer zbioru T_x zależy w sposób obliczalny od x . □

Wniosek 7.13 *Zbiór $D = \{n \mid W_n \text{ jest rekurencyjny}\}$ jest zupełny w klasie Σ_3 .*

Dowód: Sprawdzenie, że $D \in \Sigma_3$ pozostawiamy Czytelnikowi (Ćwiczenie 6). Dowód zupełności polega na redukcji $C \leq D$. Dla dowolnego $x \in \mathbb{N}$ rozpatrzmy zbiór

$$Z(x) = \{\langle u, v \rangle \mid (u \in W_x \wedge v = 0) \vee (u < v \wedge u \in K) \vee (v \in W_x \wedge v > 0)\}$$

Zbiór $Z(x)$ jest rekurencyjnie przeliczalny, a jego numer zależy w sposób obliczalny od x . Ponadto zbiór W_x jest ko-skończony wtedy i tylko wtedy, gdy zbiór $Z(x)$ jest rekurencyjny. Istotnie, przypuśćmy najpierw, że W_x jest ko-skończony. Wtedy nierozstrzygalny drugi człon alternatywy w definicji $Z(x)$ dotyczy w istocie tylko skończenie wielu par $\langle u, v \rangle$, a każdy zbiór skończony jest rekurencyjny. Zatem $Z(x)$ jest rekurencyjny. Załóżmy więc, że $\neg W_x$ jest zbiorem nieskończonym. Wtedy mamy takie równoważności

$$\begin{aligned}u \in K &\Leftrightarrow \forall v (u < v \rightarrow \langle u, v \rangle \in Z(x)); \\ u \notin K &\Leftrightarrow \exists v (u < v \wedge \langle u, v \rangle \notin Z(x)).\end{aligned}$$

Jeśli $Z(x)$ jest zbiorem rekurencyjnym, to z warunku powyżej wynika rekurencyjna przeliczalność zbioru $\neg K$. \square

Ćwiczenia

1. Udowodnić chińskie twierdzenie o resztach. A jak się nie uda, to znaleźć w książkach.
2. Udowodnić, że zbiór zdań prawdziwych w standardowym modelu arytmetyki jest nierozstrzygalny. *Wskazówka:* Użyć Wniosku 7.3.
3. Udowodnić, że jeśli $A \leq B \in \Sigma_n$ to $A \in \Sigma_n$ oraz, że $A \leq B \in \Pi_n$ implikuje $A \in \Pi_n$.
4. Udowodnić, że zbiór $\{\langle x, y \rangle \mid W_x = W_y\}$ jest zupełny w klasie Π_2 .
5. Udowodnić Lemat 7.11.
6. Udowodnić, że zbiór $D = \{n \mid W_n \text{ jest rekurencyjny}\}$ należy do Σ_3 . *Wskazówka:* Zauważyć, że $x \in D$ wtedy i tylko wtedy, gdy $\exists n \forall m (m \notin W_x \leftrightarrow m \in W_n)$. Następnie sprowadzić tę formułę do postaci normalnej z prefiksem typu $\exists \forall \exists$.

8 Zbiory analityczne

Formuły arytmetyki pierwszego rzędu, w których pod kwantyfikatorami występują zmienne indywidualne (o wartościach liczbowych), definiują zbiory arytmetyczne. O języku *drugiego rzędu* mówimy wtedy, gdy kwantyfikatory wiążą zmienne przebiegające wartości, które są funkcjami lub relacjami. Dla naszych celów wystarczy założyć, że w formułach mogą występować zmienne funkcyjne (których znaczeniem może być dowolna całkowita funkcja z \mathbb{N} do \mathbb{N}) i kwantyfikatory $\forall f$ i $\exists f$, wiążące takie zmienne.

Funkcja $f : \mathbb{N} \rightarrow \mathbb{N}$ może na przykład reprezentować nieskończone obliczenie maszyny Turinga, w ten sposób, że $f(n)$ jest kodem n -tej konfiguracji występującej w takim obliczeniu). Nie trudno napisać formułę stwierdzającą, że tak jest: trzeba napisać, że $f(0)$ to kod konfiguracji początkowej, i że dla dowolnego n , konfiguracja $f(n+1)$ powstaje w jednym kroku z $f(n)$. Dzięki temu możemy np. zdefiniować taki zbiór

$$H = \{k \mid \text{maszyna } M_k \text{ ma nieskończone obliczenie,} \\ \text{w którym stan } q_0 \text{ występuje nieskończenie wiele razy}\}.$$

Definicja 8.1 Zbiór $A \subseteq \mathbb{N}$ jest *analityczny*, wtedy i tylko wtedy, gdy istnieje taka formuła $\varphi(x)$, o co najwyżej jednej zmiennej wolnej x , że dla dowolnego $n \in \mathbb{N}$ zachodzi

$$n \in A \quad \text{wtedy i tylko wtedy, gdy} \quad \mathcal{N} \models \varphi(\underline{n}).$$

Jeśli formuła $\varphi(x)$ ma postać $\exists f_1 \forall f_2 \dots Q_n f_n \psi(f_1, \dots, f_n, x)$, gdzie wszystkie kwantyfikatory funkcyjne (ogólne i szczegółowe na przemian) występują na początku,¹⁴ a w ψ już takich kwantyfikatorów nie ma, to mówimy, że zbiór A jest *klasy* Σ_n^1 . Analogicznie, zbiór klasy Π_n^1 to zbiór, który można zdefiniować formułą postaci $\forall f_1 \exists f_2 \dots Q_n f_n \psi(f_1, \dots, f_n, x)$.

¹⁴Rodzaj kwantyfikatora Q_n zależy od parzystości n .

A zatem zbiór H powyżej jest klasy Σ_1^1 , bo do jego zdefiniowania wystarczy jeden kwantyfikator funkcyjny \exists .

Nazwa „zbiór analityczny” bierze się stąd, że równoważną definicję tego pojęcia można podać, odwołując się do dziedziny \mathbb{R} liczb rzeczywistych, dokładniej do modelu $\langle \mathbb{R}, +, \cdot, 0, 1, int \rangle$, w którym dodatkowy jednoargumentowy predykat int wyróżnia liczby naturalne (tj. $r \in int$ zachodzi wtedy i tylko wtedy, gdy r jest liczbą naturalną). Zbiory analityczne to dokładnie te zbiory, które można w tym modelu zdefiniować formułą pierwszego rzędu postaci $int(x) \wedge \varphi(x)$. Dowód tego faktu, który można znaleźć np. w książce Rogersa, polega na reprezentowaniu liczb rzeczywistych przez ciągi liczb naturalnych, np. przy pomocy tzw. ułamków ciągłych (łańcuchowych).

Fakt 8.2 *Każdy zbiór analityczny jest klasy Σ_n^1 lub Π_n^1 dla pewnego n . Co więcej można go zdefiniować formułą (o jednej zmiennej wolnej x) postaci*

$$Q_1 f_1 Q_2 f_2 \dots Q_n f_n Q' y P(f_1, f_2, \dots, f_n, y, x),$$

gdzie kwantyfikator Q' jest indywidualowy, a warunek $P(f_1, f_2, \dots, f_n, y, x)$ jest rekurencyjny względem¹⁵ funkcji f_1, f_2, \dots, f_n .

Dowód: Oczywiście najpierw sprowadzamy formułę do preneksowej postaci normalnej. Dwa kwantyfikatory tego samego rodzaju możemy „sklejać” stosując funkcję pary, podobnie jak w przypadku arytmetyki pierwszego rzędu. Przy tym kwantyfikatory indywidualowe możemy permutować z funkcyjnymi stosując Ćwiczenie 1. Wreszcie „końcówkę” postaci $\forall f \exists x \forall y$ można skrócić zamieniając ją kolejno na „końcówkę” postaci $\forall f \forall g \exists x$ a potem postaci $\forall F \exists x$. Szczegóły pozostawiamy Czytelnikowi. \square

Klasa $\Sigma_0^1 = \Pi_0^1$ to klasa zbiorów arytmetycznych. Zbiory klasy $\Delta_1^1 = \Sigma_1^1 \cap \Pi_1^1$ nazywamy *hiperarytmetycznymi*. Z poniższego twierdzenia wynika, że klasa zbiorów hiperarytmetycznych jest istotnie szersza niż klasa zbiorów arytmetycznych. Można pokazać np. że zbiór (numerów) zdań prawdziwych w standardowym modelu arytmetyki $\langle \mathbb{N}, +, \cdot, \mathbf{s}, 0 \rangle$ jest hiperarytmetyczny. A z twierdzenia Tarskiego (o niewyrażalności pojęcia prawdy) wiemy, że nie da się go zdefiniować w języku arytmetyki.

Twierdzenie 8.3 (o hierarchii analitycznej)

Dla wszystkich $n \geq 0$ zachodzą ostre inkluzje $\Sigma_n^1 \cup \Pi_n^1 \subsetneq \Sigma_{n+1}^1 \cap \Pi_{n+1}^1$.

Dowód twierdzenia o hierarchii opuszczamy. Pokażemy za to konkretny przykład.

Fakt 8.4 *Zbiór H określony na początku tego rozdziału jest Σ_1^1 -zupelny.*

Dowód: Aby naszkicować dowód posłużymy się Faktem 8.2 i Ćwiczeniem 2 z Rozdziału 6. Wynika z nich, że każdy zbiór $A \in \Sigma_1^1$ można zdefiniować formułą postaci $\exists f \forall y P(f, x, y)$, gdzie warunek $P(f, x, y)$ można rozpoznawać maszyną Turinga z wyrocznią dla funkcji f .

¹⁵Por. Ćwiczenie 2 do Rozdziału 6.

Pokażemy, że dla każdego takiego zbioru A zachodzi $A \leq_m H$. Dla danego $m \in \mathbb{N}$, aby sprawdzić, czy $m \in A$, skonstruujemy maszynę $M(m)$ spełniającą równoważność:

$$m \in A \quad \text{wtedy i tylko wtedy, gdy numer maszyny } M(m) \text{ należy do } H.$$

Maszyna $M(m)$ dla kolejnych liczb $y = 0, 1, 2, \dots$ usiłuje sprawdzać warunek $P(f, m, y)$, „zgadując” niedeterministycznie potrzebną informację o funkcji f (i zapisując to co już zgadła dla ew. ponownego użycia). Po każdej fazie pracy (pozytywnym zweryfikowaniu warunku $P(f, m, y)$, dla kolejnej wartości y) maszyna wchodzi w stan q_0 i dopiero potem rozpoczyna następną fazę. Szczegóły konstrukcji pomijamy. \square

Ćwiczenia

1. Uzupełnić dowód Faktu 8.2. Wskazówka: formułę postaci $\forall x \exists f P(f, x)$ można zastąpić równoważną formułą postaci $\exists F \forall x P(\lambda y. F(x, y), x)$, z formułą o prefiksie $\exists x \forall f$ można postąpić podobnie, używając prawa De Morgana.
2. Niech $\varphi_m(x)$ oznacza m -tą (w pewnej numeracji) formułę o jednej zmiennej wolnej x . Jeżeli $S : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ jest taką funkcją, że $S(m, n) = 0$ wtedy i tylko wtedy, gdy $\mathbb{N} \models \varphi_m(\underline{n})$, to powiemy, że S jest *definicją prawdy*. Udowodnić, że nie istnieje arytmetyczna definicja prawdy.
3. Udowodnić, że zbiór zdań prawdziwych w \mathbb{N} jest hiperarytmetyczny. *Wskazówka:* Istnieje taka formuła $\psi(X)$, że $\mathbb{N} \models \psi(S)$ wtedy i tylko wtedy, gdy S jest definicją prawdy.
4. Udowodnić, że zbiór tautologii klasycznej logiki drugiego rzędu nie jest analityczny. *Wskazówka:* Zdefiniować prawdę drugiego rzędu.

9 Problemy decyzyjne

Oczywiście sens pojęcia obliczalności jest taki: zbiór słów Z jest obliczalny wtedy, gdy istnieje algorytm, który dla danego słowa w zawsze poprawnie odpowiada „tak” lub „nie” na pytanie, czy $w \in Z$.

Takie pytania nazywamy często „problemami decyzyjnymi”. Formalnie, *problem decyzyjny* to po prostu zbiór słów nad ustalonym alfabetem.

W praktyce „problemami decyzyjnymi” nazywamy na przykład takie pytania:

- „Czy dany graf skończony ma ścieżkę Hamiltona?”
- „Czy dana formuła rachunku predykatów jest tautologią?”

gdzie daną wejściową, czyli *instancją* problemu, nie jest słowo, ale jakiś obiekt kombinatoryczny. Oczywiście skończony graf można łatwo przedstawić za pomocą słowa (trzeba po prostu wypisać jego wierzchołki i krawędzie). Podobnie można też postąpić z innymi skończonymi obiektami. Ale trzeba tu zrobić dwie uwagi:

Po pierwsze, nasze pojęcie „(nie)rozstrzygalnego problemu decyzyjnego” ma zastosowanie tylko do tych obiektów matematycznych, które dają się przedstawić w *skończony* sposób. Nie jest więc problemem decyzyjnym zadanie:

- „Czy dany graf nieskończony jest spójny?”

O ile powyższa obserwacja jest dość oczywista, następna uwaga dotyczy pułapki, w którą wpadają nawet doświadczeni badacze. Otóż np. w zadaniu dotyczącym formuł rachunku predykatów¹⁶ instancją problemu nie jest dowolne słowo, ale poprawnie zbudowana formuła. Zatem nie jest to, ściśle biorąc, problem decyzyjny. Ale oczywiście możemy łatwo wybrnąć z sytuacji zadając nieco zmodyfikowane pytanie:

– „Czy dane słowo jest tautologią rachunku predykatów?”

Łatwo, bo istnieje prosty algorytm sprawdzający czy dane słowo jest formułą, i w istocie nie ma znaczenia, które pytanie postawimy.

W praktyce nikt nie formułuje problemów decyzyjnych w taki sposób. Ważne tylko, żeby pytanie:

– „Czy dane słowo jest poprawną instancją problemu?”

(tj. formułą, reprezentacją grafu itp.) samo było rozstrzygalne.¹⁷ Zwykle tak jest i to w oczywisty sposób. Ale może tak nie być. Na przykład to pytanie nie stanowi poprawnego problemu decyzyjnego:

– „Czy dana tautologia (klasycznego) rachunku predykatów jest także tautologią intuicjonistyczną?”

Albo to:

– „Dana jest maszyna \mathcal{M} , zatrzymująca się dla słowa pustego. Czy \mathcal{M} akceptuje słowo puste?”

O tym, że mamy tu tylko pozorną „rozstrzygalność” świadczy to, że dla powyższego problemu nie można sensownie określić złożoności obliczeniowej.

Fakt 9.1 *Niech f będzie dowolną funkcją rekurencyjną. Każdy algorytm odpowiadający poprawnie na pytanie „Czy maszyna \mathcal{M} akceptuje słowo puste?” zawsze gdy \mathcal{M} zatrzymuje się dla słowa pustego, wymaga więcej niż $f(n)$ kroków dla pewnego wejścia rozmiaru n .*

Dowód: Niech L będzie językiem obliczalnym, który nie należy do klasy $\text{DTIME}(2^{f(2n)})$ i niech M_L będzie maszyną rozpoznającą L . Dla danego słowa w , niech M_w będzie maszyną, która zawsze się zatrzymuje i akceptuje dowolne wejście x wtedy i tylko wtedy, gdy $w \in L$ (jej działanie nie zależy od danej wejściowej x). Mając dane w , można skonstruować taką maszynę w czasie równym długości słowa w plus pewna stała — wystarczy użyć maszyny M_L .

Przypuśćmy teraz, że mamy algorytm, który w czasie $f(n)$ rozwiązuje nasz problem. Aby stwierdzić, czy $w \in L$ wystarczy teraz uruchomić ten algorytm dla maszyny M_w . Łącznie z konstrukcją samej maszyny wymaga to czasu rzędu $2n + f(2n)$, gdzie n to długość słowa w , a więc mniej niż $2^{f(2n)}$. \square

Z dotychczasowych naszych rozważań wynika nierozstrzygalność szeregu ważnych problemów decyzyjnych. Pierwszą grupę stanowią własności języków i funkcji obliczanych przez maszyny Turinga (lub też inne równoważne modele obliczeń), zwłaszcza różne warianty problemu stopu. Ich nierozstrzygalność wynika z twierdzenia Rice’a.

¹⁶Nazwa „problem decyzyjny” (Entscheidungsproblem) została po raz pierwszy użyta właśnie w odniesieniu do tego zadania.

¹⁷Jeśli interesuje nas nie tylko rozstrzygalność ale także złożoność problemu, to sprawdzenie poprawności instancji powinno być zadaniem o pomijalnej złożoności.

- Czy dana maszyna Turinga zatrzymuje się dla danego wejścia?
- Czy dana maszyna Turinga zatrzymuje się dla słowa pustego?
- Czy dana maszyna Turinga akceptuje język pusty (nieskończony, ko-skończony, rekurencyjny, itd.)?
- Czy dana maszyna Turinga oblicza funkcję rosnącą?

Uwaga: problem stopu jest nierozstrzygalny także dla pewnych ustalonych maszyn. Tj. można skonstruować konkretną maszynę M dla której pytanie

- Czy dane słowo jest akceptowane przez maszynę M ?

jest nierozstrzygalne. Dalej mamy różne problemy dotyczące maszyn Turinga, których nierozstrzygalność nie wynika wprost z twierdzenia Rice'a, ale z odpowiedniej redukcji:

- Czy dana maszyna Turinga ma obliczenie, w którym występuje dany stan?
- Czy dana niedeterministyczna maszyna Turinga ma obliczenie nieskończone?
- Czy dana niedeterministyczna maszyna Turinga ma obliczenie nieskończone, w którym dany stan powtarza się nieskończenie wiele razy?

Ten ostatni problem jest *wysoce nierozstrzygalny*: nie mieści się w hierarchii arytmetycznej. Każdy model pojęcia obliczalności ma oczywiście swoją wersję problemu stopu, na przykład nierozstrzygalne są problemy:

- Czy dana gramatyka typu zero akceptuje dane słowo?
- Czy w danej gramatyce typu zero można z danego słowa w otrzymać w drodze redukcji dane słowo v ? (Problem osiągalności dla gramatyk typu zero.)
- Czy dany while-program zatrzymuje się dla danej wartości wejściowej?

Rachunek lambda też ma swoje przyjemności, między innymi takie:

- Dane termy M i N . Czy $M \rightarrow_{\beta} N$?
- Dane termy M i N . Czy $M =_{\beta} N$?
- Dany term M . Czy M ma postać normalną?
- Dany term M . Czy M ma własność silnej normalizacji?

Uwaga: nierozstrzygalność trzech pierwszych problemów wynika wprost z Twierdzenia 2.11, ale nierozstrzygalność czwartego problemu wymaga delikatniejszej analizy konstrukcji użytej w dowodzie tego twierdzenia.

Dwa problemy nierozstrzygalne, o których będzie teraz mowa, są szczególnie często używane w dowodach nierozstrzygalności innych problemów.

Automaty dwulicznikowe

Przez *automat dwulicznikowy* rozumiemy krotkę $\mathcal{A} = \langle Q, q_0, q_f, I \rangle$, w której Q jest skończonym zbiorem stanów, q_0 jest stanem początkowym, q_f stanem końcowym, a I jest zbiorem instrukcji, z których każda jest jednej z postaci

1. $(q : c_i := c_i + 1; \text{goto } p);$
2. $(q : c_i := c_i - 1; \text{goto } p);$
3. $(q : \text{if } c_i = 0 \text{ then goto } p \text{ else goto } r);$

gdzie $i = 1, 2$ oraz $q, p, r \in Q$. *Konfiguracja* takiego automatu \mathcal{A} to trójka $\langle q, m, n \rangle$ złożona ze stanu $q \in Q$ i liczb $m, n \in \mathbb{N}$, stanowiących bieżącą zawartość liczników c_1 i c_2 . Konfiguracja *początkowa* ma postać $\mathcal{C}_0 = \langle q_0, 0, 0 \rangle$, a konfiguracje $\langle q_f, m, n \rangle$ są *końcowe*.

Znaczenie instrukcji jest oczywiste. Piszemy $\mathcal{C}_1 \rightarrow_{\mathcal{A}} \mathcal{C}_2$, gdy \mathcal{C}_2 można otrzymać z \mathcal{C}_1 po pewnej liczbie kroków.

Fakt 9.2 *Problem stopu dla automatów dwulicznikowych (czy $\mathcal{C}_0 \rightarrow_{\mathcal{A}} \mathcal{C}$, dla pewnej konfiguracji końcowej?) jest nierozstrzygalny.*

Dowód: Jest to dosyć łatwa konsekwencja nierozstrzygalności problemu stopu dla while-programów. Najpierw łatwo pokazujemy że automat z wieloma licznikami potrafi naśladować obliczenie while-programu. Potem zawartość k liczników reprezentujemy za pomocą jednego, używając triku z Rozdziału 1:

$$\text{kod}(a_0 \dots a_k) = 2^{a_0} 3^{a_1} \dots p_k^{a_k},$$

Polecenie zwiększenia lub zmniejszenia wartości a_i o jeden sprowadza się teraz do pomnożenia lub podzielenia wartości licznika przez p_i . Można to łatwo zrobić, używając drugiego licznika jako pomocniczego. \square

Problem odpowiedniości Posta

Problem odpowiedniości Posta (PCP) jest klasycznym przykładem problemu nierozstrzygalnego. Formułujemy go tak:

Dany jest skończony zbiór par słów $\{(x_i, y_i) \mid i = 1, \dots, n\}$ (system par Posta¹⁸).

Czy istnieje taki niepusty ciąg i_1, i_2, \dots, i_m , że zachodzi równość

$$x_{i_1} x_{i_2} \dots x_{i_m} = y_{i_1} y_{i_2} \dots y_{i_m} ?$$

Poszukiwany ciąg i_1, i_2, \dots, i_m nazywamy *rozwiązaniem* systemu par Posta. Także słowo

$$x_{i_1} x_{i_2} \dots x_{i_m}$$

¹⁸Nazwa „system Posta” używana też jest w innym znaczeniu.

bywa wtedy nazywane rozwiązaniem.

System par Posta $\{(x_i, y_i) \mid i = 1, \dots, n\}$ przedstawia się często w taki sposób:

$$\frac{x_1 \mid x_2 \mid \dots \mid x_n}{y_1 \mid y_2 \mid \dots \mid y_n}$$

Na przykład taki system:

$$\frac{a^2 \mid b^2 \mid ab^2}{a^2b \mid ba \mid b}$$

ma rozwiązanie 1213 (bo $a^2 \cdot b^2 \cdot a^2 \cdot ab^2 = a^2b^2a^3b^2 = a^2b \cdot ba \cdot a^2b \cdot b$), a systemy

$$\frac{a^2b \mid a}{a^2 \mid ba^2} \qquad \frac{ba^2 \mid ba^2}{b \mid a^2b}$$

nie mają rozwiązań.

Twierdzenie 9.3 *Problem odpowiedniości Posta jest nierozstrzygalny.*

Dowód: Redukcja z problemu osiągalności dla gramatyk typu zero (strona 37). Niech $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$ będzie gramatyką typu zero i niech $\Sigma = \mathcal{A} \cup \mathcal{N}$. Ustalmy słowa $w, v \in \Sigma^*$. Skonstruujemy (efektywnie¹⁹) system par Posta P , który będzie miał rozwiązanie wtedy i tylko wtedy, gdy $G \vdash w \rightarrow v$.

Alfabet naszego systemu par będzie taki:

$$\Delta = \Sigma \cup \{\bar{a} \mid a \in \Sigma\} \cup \{*, \bar{*}\} \cup \{[,]\}.$$

Jeśli $u = a_1 \dots a_r$ to symbol \bar{u} oznacza oczywiście słowo $\bar{a}_1 \dots \bar{a}_r$.

Reguły przedstawimy tabelką, w której a oznacza dowolny symbol alfabetu Σ , a $\alpha \Rightarrow \beta$ jest dowolną produkcją gramatyki. Należy to rozumieć tak, że do systemu P należą wszystkie pary zgodne z szablonem w którejś kolumnie tabeli.

$$\frac{[w* \mid a \mid \bar{a} \mid * \mid \bar{*} \mid] \mid] \mid \beta \mid \bar{\beta}}{[\mid \bar{a} \mid a \mid \bar{*} \mid * \mid \bar{*}v \mid *v \mid] \mid \bar{\alpha} \mid \alpha}$$

Na przykład jeśli $w = baca$, $v = owad$, a regułami naszej gramatyki są

$$ba \Rightarrow ow \qquad ca \Rightarrow ad,$$

to mamy taki system par Posta:

$$\frac{[baca* \mid a \mid \bar{a} \mid \dots \mid ow \mid \bar{ow} \mid ad \mid \bar{ad} \mid] \mid]}{[\mid \bar{a} \mid a \mid \dots \mid \bar{ba} \mid ba \mid \bar{ca} \mid ca \mid \bar{*owad} \mid *owad]}$$

¹⁹Tj. opiszemy w istocie algorytm realizujący tę konstrukcję.

Załóżmy teraz, że $w = w_0 \rightarrow_G w_1 \rightarrow_G w_2 \rightarrow_G \cdots \rightarrow_G w_k = v$, i dla ustalenia uwagi przyjmijmy, że k jest parzyste. Wtedy nasz system par Posta można rozwiązać tak:

$$[w_0 * \bar{w}_1 * w_2 * \cdots * \bar{w}_{k-1} * w_k \cdot] = [\cdot w_0 * \bar{w}_1 * w_2 * \cdots * \bar{w}_{k-1} \cdot \bar{w}_k]$$

Rzeczywiście, jeśli mamy np. $w_0 = x\alpha y \rightarrow_G x\beta y = w_1$, przez zastosowanie produkcji $\alpha \Rightarrow \beta$, to słowa w_0* i \bar{w}_1* można przedstawić jako konkatenacje odpowiadających sobie słów z dolnej i górnej części tabelki. Podobnie dla słów \bar{w}_1* i w_2* i tak dalej.

Na dodatek, jeśli mamy jakiegokolwiek rozwiązanie systemu P , to takie rozwiązanie musi wyznaczać pewne wyprowadzenie $w \rightarrow_G v$. Rozwiązanie musi się zaczynać od pary ($[w*$, $[$), jest to bowiem jedyna para, która zaczyna się tak samo (po to są kreski). Jedyńm sposobem zgodnego przedłużenia słów z tej pary jest dopisanie do $[$ słowa $w*$. To jednak oznacza, że do słowa $[w*$ musimy dopisać słowo postaci \bar{u}_1* gdzie $w \rightarrow_G u_1$. Zatem mamy teraz słowa

$$[w * \bar{u}_1 * \quad [w *$$

i musimy do drugiego z nich dopisać \bar{u}_1* . Ale wtedy do pierwszego dopiszemy u_2* , gdzie $u_1 \rightarrow_G u_2$ i tak dalej. To się może skończyć tylko użyciem jednej z par zawierających $]$, co oznacza, że $w \rightarrow v$.

Uwaga: Warunek $u_1 \rightarrow_G u_2$ powyżej nie musi oznaczać $u_1 \rightarrow u_2$. Na przykład przekształcenie bacy w owada może wyglądać nie tylko tak:

$$[baca * \overline{owca*}owad],$$

ale też na przykład tak:

$$[baca * \overline{baca*}owad * \overline{owad}]. \quad \square$$

Przykładem zastosowania PCP jest taki fakt:

Twierdzenie 9.4 *Problem pustości dla języków kontekstowych jest nierozstrzygalny.*

Dowód: Dla danego systemu par Posta P , można łatwo skonstruować maszynę Turinga, rozpoznającą dokładnie te słowa, które są rozwiązaniami tego systemu. Maszyna ta nie używa więcej taśmy, niż zajmowało słowo wejściowe. Modyfikując nieco konstrukcję z dowodu Twierdzenia 2.5, możemy z tej maszyny uzyskać gramatykę monotoniczną generującą zbiór rozwiązań systemu P . W ten sposób zredukujemy problem odpowiedniości Posta do problemu pustości dla języków kontekstowych. \square

Systemy Thuego

Gramatykę typu zero, w której nie rozróżnia się symboli terminalnych i nieterminalnych, oraz nie wyróżnia się symbolu początkowego, nazywamy *systemem pół-thue'owskim*.²⁰ A więc system pół-thue'owski to po prostu skończony zbiór T reguł postaci „ $w \Rightarrow v$ ”, gdzie w, v są słowami nad ustalonym alfabetem. Relacja $w \rightarrow_T v$ zachodzi wtedy i tylko wtedy, gdy $w = uw_1u'$, $v = uw_2u'$, oraz $w_1 \Rightarrow w_2$ jest w T . Jej domknięcie przechodnio-zwrotne oznaczamy jak zwykle przez \rightarrow_T , a najmniejsza relacja równoważności zawierająca \rightarrow_T będzie

²⁰ang.: *semi-Thue system*.

oznaczana przez \leftrightarrow_T . System Thuego²¹ to taki system pół-thue'owski T , w którym wszystkie reguły są odwracalne, tj. jeśli „ $w \Rightarrow v$ ” $\in T$ to także „ $v \Rightarrow w$ ” $\in T$.

Przez *problem słów* dla systemu Thuego T rozumiemy następujący problem decyzyjny: dane słowa w i v , czy $w \leftrightarrow_T v$?

Twierdzenie 9.5 (Post) *Istnieją systemy Thuego o nierozstrzygalnym problemie słów.*

Dowód: Pokażemy najpierw pewien system półthue'owski, dla którego nierozstrzygalny jest problem osiągalności (czasem nazywany problemem słów): dane słowa w i v , czy $w \rightarrow_T v$?

Niech \mathcal{M} będzie deterministyczną maszyną Turinga, dla której nierozstrzygalny jest problem stopu (tj. język $L(\mathcal{M})$). Dla uproszczenia założymy, że \mathcal{M} ma tylko jeden stan końcowy q_a (akceptujący).

Rozpatrzmy alfabet składający się ze stanów i symboli maszyny \mathcal{M} , oraz dodatkowo z nawiasów kwadratowych (na oznaczenie początku i końca konfiguracji). System półthue'owski P składa się z reguł:

- $qa \Rightarrow bp$, gdy $\delta(q, a) = (b, p, +1)$;
- $qa \Rightarrow pb$, gdy $\delta(q, a) = (b, p, 0)$;
- $q] \Rightarrow bp]$, gdy $\delta(q, \mathbf{B}) = (b, p, +1)$;
- $q] \Rightarrow pb]$, gdy $\delta(q, \mathbf{B}) = (b, p, 0)$;
- $cqa \Rightarrow pcb$ i $[qa \Rightarrow [pb$, gdy $\delta(q, a) = (b, p, -1)$;
- $cq] \Rightarrow pcb]$, gdy $\delta(q, \mathbf{B}) = (b, p, -1)$;
- $aq_a \Rightarrow q_a$ i $q_a a \Rightarrow q_a$ (dla *wszystkich* a , także dla $a = [,]$).

Wówczas zachodzi równoważność:

$$[q_0 w] \rightarrow_P q_a \quad \text{wtedy i tylko wtedy gdy } w \in L(\mathcal{M}),$$

bo każdy ciąg postaci $[q_0 w] \rightarrow_P x_1 \rightarrow_P x_2 \rightarrow_P \dots \rightarrow_P q_a$ musi reprezentować obliczenie maszyny dla słowa w , zakończone „wycieraniem” wszystkich symboli oprócz q_a (patrz dowód Twierdzenia 2.5). W istocie mamy jednak lepszą własność:

$$[q_0 w] \leftrightarrow_P q_a \quad \text{wtedy i tylko wtedy gdy } w \in L(\mathcal{M}).$$

Przypuśćmy bowiem, że $[q_0 w] \leftrightarrow_P q_a$. Mamy ciąg:

$$[q_0 w] = x_0 \sim x_1 \sim x_2 \sim \dots \sim x_n = q_a,$$

gdzie każde \sim to albo \rightarrow_P albo $P\leftarrow$. Niech to będzie *najkrótszy* taki ciąg. Łatwo widzieć, że każde x_i zawiera dokładnie jedno wystąpienie pewnego stanu q . Zauważmy, że jeśli

$$x_{i-1} P\leftarrow x_i \rightarrow_P x_{i+1},$$

²¹ang.: *Thue system*.

to w słowie x_i występuje q_a . Inaczej x_i odpowiada niekońcowej konfiguracji maszyny i z powodu determinizmu mamy $x_{i-1} = x_{i+1}$. Ale wtedy nasz ciąg można skrócić. A więc nasza redukcja jest postaci:

$$[q_0w] = x_0 \rightarrow_P x_m \sim x_{m+1} \sim \cdots \sim x_n = q_a,$$

przy czym x_m jest konfiguracją końcową. Najkrótszy sposób na otrzymanie q_a z konfiguracji końcowej to po prostu eliminacja pozostałych symboli (wytarcie każdego symbolu wymaga co najmniej jednego kroku). Skoro więc nasz ciąg jest najkrótszy, to faktycznie mamy redukcje $[q_0w] = x_0 \rightarrow_P x_m \rightarrow_P x_n = q_a$. A zatem $[q_0w] \leftrightarrow_P q_a$ faktycznie oznacza $[q_0w] \rightarrow_P q_a$.

Niech teraz T będzie systemem Thuego otrzymanym przez dodanie odwróconych wersji wszystkich reguł. Następujące trzy warunki są równoważne:

$$[q_0w] \leftrightarrow_T q_a \quad [q_0w] \leftrightarrow_P q_a \quad w \in L(\mathcal{M}).$$

A zatem problem słów dla T jest nierozstrzygalny. \square

Problem słów dla systemów Thuego jest też nazywany *problemem słów w półgrupach*. O systemie Thuego T można bowiem myśleć jak o zbiorze aksjomatów równościowych nad sygnaturą złożoną ze słowa pustego i liter alfabetu (stałych) oraz dwuargumentowej operacji konkatenacji. Ponieważ jednak konkatenacja słów jest łączna, a ε jest jej elementem neutralnym, więc faktycznie mówimy tu o zbiorze zdań T^+ otrzymanym przez dodanie do T aksjomatów *półgrupy z jednością*:

$$\begin{aligned} \forall xyz(x \cdot (y \cdot z) \approx (x \cdot y) \cdot z); \\ \forall x(\varepsilon \cdot x \approx x), \quad \forall x(x \cdot \varepsilon \approx x). \end{aligned}$$

Wniosek 9.6 *Zbiór tautologii klasycznej logiki pierwszego rzędu jest nierozstrzygalny.*

Dowód: Z twierdzenia 9.5 wynika nierozstrzygalność następującego problemu:

Dany zbiór zdań T^+ wyznaczony przez system Thuego T , czy $T^+ \models w = v$?

Ale zbiór T^+ jest skończony; jeśli więc φ jest koniunkcją wszystkich zdań z T^+ , to nasze zadanie możemy sformułować jako pytanie o prawdziwość zdania $\varphi \rightarrow w = v$. \square

Analogicznie jak dla półgrup, rozważa się też *problem słów w grupach*. Tym razem sygnatura zawiera dodatkowo operację unarną $()^{-1}$ i mamy dodatkowe aksjomaty:

$$x^{-1} \cdot x = \varepsilon, \quad x \cdot x^{-1} = \varepsilon.$$

Problem słów dla grup też jest w ogólności nierozstrzygalny, pokazał to Nowikow w 1952 roku. Ale ten dowód jest dużo trudniejszy (problem był znany już 40 lat wcześniej).

Ćwiczenia

1. Udowodnić nierozstrzygalność problemu stopu dla zmodyfikowanych automatów dwulicznikowych, których dozwolone instrukcje są następujące:

- (a) $(q : c_i := c_j + 1; \text{goto } p)$;
- (b) $(q : c_i := 0; \text{goto } p)$;

(c) (q : if $c_i = c_j$ then goto p else goto r);

Wskazówka: Parę liczb $\langle a_0, a_1 \rangle$ można reprezentować nie tylko przez $2^{a_0}3^{a_1}$ ale także przez każdą z liczb postaci $2^{a_0}3^{a_1}5^x$. Aby obniżyć a_0 o jeden, nie trzeba takiej liczby zmniejszać, wystarczy ją pomnożyć przez $5/2$.

2. *Automat kolejkowy* dysponuje kolejką (słowem nad ustalonym alfabetem) na której może wykonywać takie operacje:
 - dodaj literę a na końcu kolejki;
 - usuń pierwszą literę z kolejki;
 - zmień stan w zależności od tego jaka litera znajduje się na początku kolejki.

Udowodnić nierozstrzygalność problemu stopu dla automatów kolejkowych: *czy dany automat uruchomiony z pustą kolejką początkową osiągnie stan końcowy?*

3. Udowodnić nierozstrzygalność problemu stopu dla automatów z dwoma stosami, na których można przechowywać litery ze skończonego alfabetu. Automat może wykonywać operacje *push* i *pop* i zmieniać stan w zależności od tego, co znajduje się na wierzchołku pierwszego stosu.
4. Udowodnić rozstrzygalność problemu stopu dla automatów z jednym stosem.
5. Udowodnić, że dla dowolnego systemu Thuego T relacja $w \leftrightarrow_T v$ zachodzi wtedy i tylko wtedy, gdy równość $w = v$ można wyprowadzić ze aksjomatów półgrupy rozszerzonych o aksjomaty równościowe wyznaczone przez reguły systemu T .

10 Totalność, mortalność i terminalność

Przez *problem totalności* dla maszyn Turinga rozumiemy pytanie:

Dana deterministyczna maszyna \mathcal{M} . Czy \mathcal{M} zatrzymuje się dla dowolnego słowa?

Z Faktu 7.1 natychmiast wynika, że problem totalności nie jest rekurencyjnie przeliczalny ani nie jest dopełnieniem problemu rekurencyjnie przeliczalnego.

Mówimy, że deterministyczna maszyna Turinga \mathcal{M} jest *terminalna* jeżeli dla dowolnej konfiguracji \mathcal{C} (niekoniecznie początkowej), obliczenie maszyny \mathcal{M} rozpoczynające się od \mathcal{C} jest skończone.

Twierdzenie 10.1 *Problem terminalności dla maszyn Turinga (czy dana maszyna jest terminalna?) nie jest rekurencyjnie przeliczalny ani nie jest dopełnieniem problemu rekurencyjnie przeliczalnego.*

Dowód: Dowód opiera się na redukcji problemu totalności do problemu terminalności. Dla danej maszyny \mathcal{M} konstruujemy taką maszynę \mathcal{N} , żeby zachodził warunek:

\mathcal{M} jest totalna wtedy i tylko wtedy, gdy \mathcal{N} jest terminalna.

Oczywiście maszyna \mathcal{N} musi naśladować obliczenia maszyny \mathcal{M} . Trudność przy tej redukcji polega na tym, że konfiguracja, w której uruchamiamy maszynę \mathcal{N} , może być zupełnie dowolna, w szczególności może nie odpowiadać żadnej konfiguracji \mathcal{M} osiągalnej z jakiegokolwiek konfiguracji początkowej. Konstrukcja maszyny \mathcal{N} jest następująca:

Taśma maszyny \mathcal{N} jest podzielona na trzy części. Pierwsza przechowuje słowo wejściowe, druga przeznaczona jest na binarny licznik, a trzecia stanowi obszar roboczy. Maszyna \mathcal{N} w obszarze roboczym symuluje zachowanie maszyny \mathcal{M} , zwiększając licznik o 1 po każdym kroku. W razie wykrycia jakiegokolwiek niezgodności pomiędzy swoimi oczekiwaniami i rzeczywistymi danymi zapisanymi na taśmie, maszyna \mathcal{N} natychmiast się zatrzymuje. Kiedy wyczerpie się miejsce na licznik, maszyna zeruje go, kopiuje słowo wejściowe do obszaru roboczego i zaczyna symulację od początku. W ten sposób maszyna \mathcal{N} po skończonej liczbie kroków musi albo się zatrzymać albo rozpocząć prawidłową symulację maszyny \mathcal{M} , zaczynając się od konfiguracji początkowej. Szczegóły konstrukcji pozostawiamy jako ćwiczenie. \square

Jeśli uważnie przyjrzymy się temu dowodowi, to zobaczymy, że istotnie wykorzystuje on założenie o skończoności konfiguracji maszyny. Obszar roboczy jest skończony i można zawsze odszukać jego lewy koniec (albo pierwszy blank — można zakładać, że konfiguracje maszyny \mathcal{M} nigdy nie zawierają blanków „wewnętrznych”). Jeśli założyć, że konfiguracje maszyny Turinga mogą być nieskończone, to mówimy, że maszyna jest *mortalna*, gdy każde obliczenie maszyny \mathcal{M} rozpoczynające się od dowolnej (także nieskończonej) konfiguracji musi zawsze być skończone.

Podobną własnością jest *ograniczoność* maszyny Turinga. Ma ona miejsce wtedy, gdy istnieje taka stała k , że liczba różnych konfiguracji przyjmowanych przez maszynę w dowolnym ciągu kolejnych kroków jest zawsze mniejsza lub równa k . Zauważmy, że maszyna ograniczona nie musi być mortalna.

Fakt 10.2 *Jeśli maszyna \mathcal{M} jest mortalna, to istnieje taka stała k , że maszyna \mathcal{M} , uruchomiona w dowolnej (być może nieskończonej) konfiguracji, zatrzymuje się po co najwyżej k krokach.*

Dowód: Ćwiczenie. Wskazówka: użyć lematu Königa. \square

Wniosek 10.3 *Problem mortalności (czy dana maszyna jest mortalna?) jest rekurencyjnie przeliczalny.*

Wniosek 10.4 *Każda maszyna mortalna jest ograniczona.*

Twierdzenie 10.5 (Philip K. Hooper) *Problem mortalności jest nierozstrzygalny.*

Pozostała część tego rozdziału to dowód (a właściwie szkic dowodu) twierdzenia Hoopera. Dowód jest w większości po angielsku, bo taki gotowy tekst istnieje, a lepszy taki, niż żaden.

The goal is to reduce the halting problem for two-counter automata (known to be undecidable) to the mortality problem. We start with an arbitrary two-counter automaton M .

Lemat 10.6 *One can assume that*

1. M may execute at most two tests for zero in a row;

2. For each n there is a k , such that if M makes k moves when started on any ID, then the value of the second counter must exceed n at least once during these k moves.

Dowód: An ID of the form $\langle q, c_1, c_2 \rangle$ is represented by $\langle q, 0, 2^{c_1} 3^{c_2} 5^p \rangle$, where $p \geq 0$. A single move of the automaton is simulated in an obvious way, but in addition, after completing each step, the second counter is multiplied by 5. The construction is left to the reader. \square

The next step is to construct a Turing Machine T which simulates the behaviour of M . This construction is quite standard, but essential for the considerations to follow, and thus we describe it in some detail.

The machine T has one tape, infinite in both directions. The set Q of states of T includes the set Q_M of states of M , called *main* states of T . The tape alphabet is $\{0, 1\}$, with 0 used as the blank symbol. An ID of M of the form $\langle q, c_1, c_2 \rangle$ will be represented as²² $\langle 10^{c_1+1}, q, m, 10^{c_2+1}1 \rangle$, and after simulation of a move of M resulting in $\langle p, c'_1, c'_2 \rangle$, the machine T arrives at $\langle 10^{c'_1+1}, q, m - c_1 + c'_1, 10^{c'_2+1}1 \rangle$, i.e., the position of the leftmost “1” does not change. This allows us in a later construction to run a copy of T , using the part of tape to the left to store additional information.

The way T simulates M is as follows. To execute an operation on a counter, T performs the following actions:

- moves the middle “1” by one cell to the left or to the right, if necessary;
- moves the head to the rightmost “1”, and adjusts its position (always by one cell to the right or to the left);
- returns to the middle “1”.

To execute a test for zero, T moves its head two cells to the left or to the right and returns to the initial position. We assume that T halts if it discovers any kind of inconsistency, e.g., when there is no “0” between the middle and the rightmost 1’s.

The machine T is obviously unbounded. The reason is that there are states q of T (called *right search* states), such that whenever T encounters “0” in state q , it moves the head right, and remains in state q . Similarly, there are *left search* states with the dual property. The sets of right search states and left search states are denoted Q_R and Q_L , respectively. The initial state is denoted by q_0 .

Observe that, if T starts in an arbitrary ID, then in at most 10 steps it must enter a (left or right) search state. This is due to (1) in Lemma 10.6.

The crucial part of Hooper’s proof is to construct another machine T^* which will simulate T without actually doing any unbounded search. For this we first define a machine \bar{T} , which is a “mirror copy” of T , with all its left moves replaced by right moves, and conversely. That is, the set of states of \bar{T} is $\bar{Q} = \{\bar{q} \mid q \text{ is a state of } T\}$. Whenever T in a state q and reading

²²Przez $\langle w, q, m, v \rangle$ oznaczamy tu konfigurację maszyny, która znajduje się w stanie q , ma głowicę umieszczoną w pozycji $m \in \mathbb{Z}$, gdzie znajduje się pierwsza litera słowa v zapisanego na taśmie po ostatniej literze słowa w . Zakładamy, że to, co znajduje się na lewo od w i na prawo od v jest dla nas nieistotne.

a symbol a , prints b and moves right, changing state to p , then \bar{T} in state \bar{q} , when reading a , will print b , move *left* and change state to \bar{p} .

The machine T^*

One may think of T^* as a program consisting of two recursive procedures P and \bar{P} , simulating the behaviour of T and \bar{T} , respectively. Each of these procedures may call itself or the other one. (This recursion will be used as a main trick in simulating the searching process.) An activation of P will be represented on the tape as a word of the form $1^x 0^{c_1+1} 10^{c_2+1} 1$, where $x \geq 2$ identifies the context in which the activation has been created. A further call to P will result in locating another word $1^y 0101$ (representing the initial ID of T) inside one of the segments of 0's between the 1's. Similarly, an activation of \bar{P} will be represented by a word of the form $10^{c_2+1} 10^{c_1+1} 1^x$.

Let us describe the behaviour of T^* more precisely. The set S of states of T^* includes $Q \cup \bar{Q}$, and we distinguish the following particular subsets of S :

$S_M = Q_M \cup \bar{Q}_M$, the *main* states;

$S_R = Q_R \cup \bar{Q}_L \cup \{e_R^1, e_R^2\}$, the *right search* states;

$S_L = Q_L \cup \bar{Q}_R \cup \{e_L^1, e_L^2\}$, the *left search* states.

The new states e_R^1, e_R^2 and e_L^1, e_L^2 are called, respectively, right and left *erase* states. We assume that all the right and left search states are identified by integers greater than 1.

Assume now that T^* is in a state $q \in S_R$, numbered x , the head is positioned at the m -th cell containing a "0". Then T^* performs the following action:

- checks whether the nearest "1" to the right is at most $x + 4$ cells away;
- if so, it moves the head to the nearest "1" and enters q again;
- if not, it prints "01^x0101" at the cells $m, m + 1, \dots, m + x + 4$, moves the head to the $(m + x + 2)$ -th cell, and changes the state to q_0 . (This creates a new activation of P .)

Note that to perform the above, the machine uses $\mathcal{O}(x)$ brand new states, for each $q \in S_R$. If $q \in S_L$, then T^* behaves in a dual way, in particular, in step (3) above, it prints $10101^x 0$ at the cells $m - 4 - x, \dots, m$, and enters state \bar{q}_0 , at the $(m - x - 2)$ -th cell.

Now let the current state be $q \in S_R - \{e_R^1, e_R^2\}$, and let the symbol encountered on (m -th cell of) the tape be "1". (This represents a successful search.) Then T^* behaves as follows:

- checks whether the next symbol to the right is "0";
- if so, it executes the appropriate move of T or of \bar{T} (determined by the transition function of T or \bar{T} for q and "1");
- otherwise, it erases the "1" at the m -th cell, and enters the state e_L^1 . (If "1" has been encountered at the $(m + 1)$ -st cell, it must be that the current activation of P has exhausted all the available space.)

The behaviour of T^* in a state $q \in S_L - \{e_L^1, e_L^2\}$ is defined again in a dual way.

If the machine encounters “1”, when in the state e_L^1 , it simply erases it, and enters e_L^2 . When “1” (in the m -th cell) is encountered in state e_L^2 , it should be the case that a whole activation of P has been erased except the initial 01^x . Using brand new states (no more than the largest possible x), the machine erases the 1’s and enters the right search state identified by x at the cell $(m - x + 1)$. Note that the erased activation of P has been created when the tape head was positioned at the “0” preceding “1^x”. Thus, after completing one activation of P , the tape head moved one step to the right. Of course, the behaviour of the left erase states is dual.

We assume that T^* halts whenever it enters a final state of T or \bar{T} , or if it discovers any inconsistency.

Lemat 10.7 *Suppose that T^* is started at $\langle w, q, m, 0^n 1v \rangle$, and that $q \in S_R$. Then, after some number of steps, T^* will either halt or will reach $\langle w0^n, q, m + n, 1v \rangle$. That is, T^* is able to perform a successful right search. Similarly for left search: if T^* starts at $\langle w10^{n-1}, q, m, 0v \rangle$, and $q \in S_L$ then it will either halt or reach $\langle w, q, m - n, 10^n 1v \rangle$.*

Dowód: The proof goes by induction on n (we prove the claim in parallel for all search states). If q is numbered by x and $n \leq 4 + x$ then the claim is obvious. Otherwise, a new activation of P or \bar{P} is created. The space available for this activation consists of $n - x$ cells, thus no search will be required on a distance longer than that. By the induction hypothesis, we may assume that each search was successful, and that our procedure activation is eventually erased. Thus, we arrive at $\langle w0, q, m + 1, 0^{n-1}v \rangle$ in the right search case, or at $\langle w10^{n-2}, q, m - 1, 00v \rangle$ otherwise, and we can use the induction hypothesis again. \square

Clearly, the machine T^* has been constructed in an effective way from the two-counter automaton M . Thus, to complete the proof of Hooper’s theorem, it remains to show:

Lemat 10.8 *M halts on the input $(0, 0)$ iff T^* is mortal.*

Dowód: The “if” part is easy: assume that M does not halt. Then, by Lemma 10.6, the size of ID’s of T^* obtained from $\langle 10, q_0, m, 101 \rangle$, must grow infinitely, whence T^* must be unbounded.

Proving the “only if” part is more delicate. Assume that M halts, and let K_1 be the space needed for T^* to complete a full simulation of the behaviour of M on the input $(0, 0)$.

Oczywiście udana symulacja automatu M musi doprowadzić do zatrzymania maszyny T^* . A taka symulacja będzie miała miejsce zawsze, gdy maszyna T^* znajdzie się w stanie $s \in S_R$, mając na prawo co najmniej $K_1 + K_2$ zer, gdzie K_2 jest ograniczeniem na długość ciągu jedynek kodującego stan s . Wystarczy więc pokazać, że maszyna, prędzej czy później musi wykonać przeszukiwanie w prawo (lub w lewo) na odległość co najmniej $K_1 + K_2$. Maszyna T^* musi oczywiście po skończonej liczbie kroków wejść w stan przeszukujący (lemat 10.6(1)) i wykonać przeszukiwanie na jakiś, być może krótki, dystans.

We shall show that if T^* performs a successful search at a distance n , then after a constant number of steps it must either halt or enter a search at a distance greater than n . Suppose we start in $\langle w, q, m, 0^n 1v \rangle$, where $q \in S_R$. After reaching “1” at the $(m + n)$ -th cell, the machine must enter an ID of one of the following forms (assuming it does not just halt):

1. $\langle w0^n, p, m+n, 01v' \rangle$; here, the rightmost “1” has been moved one more step to the right, and p is a left search state;
2. $\langle w0^{n-2}, p, m+n-1, 010v \rangle$, with $p \in S_L$; if the rightmost “1” has been moved one step to the left;
3. $\langle w0^n, e_L^1, m+n, 0v \rangle$; here, a “1” was found in the $(m+n+1)$ -st cell;
4. $\langle w0^n 0^{x-1}, p, m+n+x, 0v' \rangle$ with $p \in S_L$; this happens when q is e_R^2 ;
5. $\langle w0^{n+1}, e_R^2, m+n+1, v \rangle$; this happens when q is e_R^1 ;

In all cases except (2), it is clear that the next left search is at a distance at least $n+1$. The same holds also for case (2), if the last two symbols in w are 0's. Otherwise, after completing the left search, we enter an ID either of the form $\langle w', s, m-1, 10^{n-1}10v \rangle$ or of the form $\langle w', s, m-2, 10^n 10v \rangle$, where $s \in S_M$. The machine then simulates the behaviour of M with the second counter set to $n-1$, or n , respectively. Assuming it will not halt, then, by Lemma 10.6(2), there is a constant K_3 such that after K_3 steps it must increase c_2 to at least $n+1$, i.e. a left search at a distance $n+1$ will eventually be executed.

A zatem, przedzej czy później maszyna albo się zatrzyma, albo rozpocznie przeszukiwanie na odległość co najmniej $K_1 + K_2$. Ale to przeszukiwanie doprowadzi do pełnej symulacji akceptującego obliczenia automatu M i w konsekwencji do zatrzymania maszyny T^* . \square

Ćwiczenia

1. Uzupełnić dowód Twierdzenia 10.1. W szczególności wyjaśnić w jaki sposób maszyna \mathcal{N} może rozpoznać lewy koniec taśmy. (Przy naszej definicji z rozdziału 1, maszyna mogłaby się zapętlić, próbując przesunąć głowicę poza koniec taśmy.)
2. Udowodnić Fakt 10.2 oraz Wnioski 10.3 i 10.4.
3. Wywnioskować nierozstrzygalność problemu ograniczoności
 - (a) z dowodu Twierdzenia 10.5;
 - (b) z treści Twierdzenia 10.5.

Podziękowania

Za poprawki dziękuję Pani Karolinie Sołtys oraz Panom Danielowi Hansowi, Michałowi Kottowskiemu, Łukaszowi Kożuchowskiemu, Dariuszowi Leniowskiemu i Piotrowi Wilkinowi.