Sprawdzian z WDI, rozwiązanie zadania 1

Antoni Kościelski

15 grudnia 2015

1 Sformułowanie zadania

Dany jest następujący program w kodzie RAM:

```
        READ   1
        LOAD   1
        STORE  3
        READ   2
pętla:  LOAD   1
        SUB    2
        JZERO  wynik0
        JGTZ   wynik1
        LOAD   1
        ADD    3
        STORE  1
        JUMP   pętla
wynik1: WRITE  =1
        HALT
wynik0: WRITE  =0
```

1) Podaj wartość wypisywaną na taśmie wyjściowej dla podanych zawartości taśmy wejściowej (konkretne zawartości, a także inne – zbędne teraz – szczegóły, zostaną podane później).

2) Uzupełnij specyfikację powyższego programu. **Odpowiedź uzasadnij** – najlepiej poprzez podanie schematu blokowego, uzasadnienie, w tym schemat blokowy, możesz zamieścić na odwrocie kartki.

3) Rozważmy modyfikację powyższego programu polegającą na zamianie instrukcji `"write =0"` na `"write ^0"`. Uzupełnij specyfikację tak uzyskanego programu. **Odpowiedź uzasadnij.**

2 Coś w rodzaju schematu blokowego

Rozwiązując takie zadanie, najpierw trzeba zrozumieć dany program. Może w tym pomóc narysowanie schematu blokowego. Tutaj dany program będzie stopniowo przekształcany do prostszej, bardziej zrozumiałej postaci. Najpierw zapiszemy ten program używając instrukcji przypisania i zmiennych zamiast poleceń dla maszyny RAM. Użyjemy czterech zmiennych `Acc`, `X`, `A` i `B`, które będą odpowiadać kolejno komórkom maszyny RAM o numerach od 0 do 3. W pierwszym rzędzie

polecenia dla maszyny RAM wyrazimy za pomocą instrukcji przypisania, następnie usuniemy z programu instrukcje przypisania z udziałem zmiennej `Acc`, odpowiednika akumulatora. Przekształcając tak dany otrzymamy następujące programy:

```
        READ   1          |      Czytaj(X)          |      Czytaj(X)
        LOAD   1          |      Acc ← X            |
        STORE  3          |      A ← Acc            |      A ← X
        READ   2          |      Czytaj(B)          |      Czytaj(B)
pętla:  LOAD   1          |  pp: Acc ← X            |  pp:
        SUB    2          |      Acc ← Acc - B      |
        JZERO  wynik0     |      Jeśli Acc = 0,     |      Jeśli X - B = 0,
                          |         to Idź do w0    |         to Idź do w0
        JGTZ   wynik1     |      Jeśli Acc > 0,     |      Jeśli X - B > 0,
                          |         to Idź do w1    |         to Idź do w1
        LOAD   1          |      Acc ← X            |
        ADD    3          |      Acc ← Acc + A      |
        STORE  1          |      X ← Acc            |      X ← X + A
        JUMP   pętla      |      Idź do pp          |      Idź do pp
wynik1: WRITE  =1         |  w1: Pisz(1)            |  w1: Pisz(1)
        HALT              |      Stop               |      Stop
wynik0: WRITE  =0         |  w0: Pisz(0)            |  w0: Pisz(0)
```

Ostatni z powyższych programów ma już postać, która bez trudu może zostać przekształcona w schemat blokowy. Nie zostanie to zrobione, ale zainteresowane osoby mogą zrobić to same. Ten program można już wykorzystać w dalszych rozważaniach. Można go też jeszcze bardziej uprościć. Możemy na przykład wyjść z pętli tylko w jeden sposób, trochę wcześniej niż w podanym programie, przed ustaleniem wyniku, a wynik obliczeń określić poza pętlą. W końcu możemy do programu wprowadzić instrukcję `dopóki`. W ten sposób otrzymamy dwie kolejne wersje naszego programu:

```
     Czytaj(X)            |      Czytaj(X)          | Czytaj(X)
     A ← X                |      A ← X              | A ← X
     Czytaj(B)            |      Czytaj(B)          | Czytaj(B)
pp:  Jeśli X - B = 0      |  pp: Jeśli X ⩾ B,       | dopóki X < B wykonuj
        to Idź do w0      |         to Idź do w0    |
     Jeśli X - B > 0,     |                         |
        to Idź do w1      |                         |
     X ← X + A            |      X ← X + A          |    X ← X + A
     Idź do pp            |      Idź do pp          |
w1:  Pisz(1)              |  w0: Jeśli X > B,       | Jeśli X > B,
     Stop                 |         to Pisz(1)      |    to Pisz(1)
w0:  Pisz(0)              |         wpp Pisz(0)     |    wpp Pisz(0)
```

Szczególnie prosta jest ostatnia wersja. Można ją łatwo zapisać w języku C lub w Pascalu.

Podczas sprawdzianu kilka osób zauważyło, że to samo, co dany, robi także inny, nieco prostszy program:

```
          READ   1
          READ   2
          LOAD   1
          SUB    2
pętla:    JZERO  wynik0
          JGTZ   wynik1
          ADD    1
          JUMP   pętla
wynik1:   WRITE  =1
          HALT
wynik0:   WRITE  =0
```

Proszę sprawdzić, że tak jest w rzeczywistości.

3 Specyfikacja

Wiedząc już, co robi nasz program, możemy pokusić się o odtworzenie jego specyfikacji. Wśród pozytywnie ocenianych rozwiązań najczęściej pojawiała się taka oto:

Specyfikacja:

Wejście: liczby naturalne a i b wczytane odpowiednio do 1 i 2 komórki,

Wyjście: 0, jeżeli liczba a dzieli liczbę b, oraz 1 w przeciwnym przypadku.

Niektóre osoby zamiast *a dzieli liczbę b* pisały równoważny zwrot *b jest wielokrotnością a*. Po podaniu specyfikacji należało ją uzasadnić.

3.1 Krótkie uzasadnienie przytoczonej specyfikacji

Po początkowych przypisaniach wartości zmiennych `A` oraz `B` nie ulegają zmianie i są stale równe odpowiednio pierwszej wczytanej liczbie a oraz drugiej – b. Stąd w szczególności wynika, że wartościami zmiennej `X` są kolejne wielokrotności liczby a i są one porównywane z liczbą b.

Przed zakończeniem działania programu wartość zmiennej `X` jest większa lub równa b. Wobec tego, jeżeli liczba b jest wielokrotnością a, to w pewnym momencie wartość zmiennej X stanie się równa b, nastąpi skok do etykiety `wynik0`, na taśmie wyjściowej pojawi się 0 i program zatrzyma się. W przeciwnym razie, wartość zmiennej `X` nigdy nie będzie równa b, a gdy przekroczy b nastąpi skok do etykiety `wynik1`, na taśmie wyjściowej zostanie zapisana liczba 1 i program zakończy pracę.

Tak więc rozważany program działa zgodnie z powyższą specyfikacją.

3.2 Jeszcze krótsze uzasadnienie

Oczywiście na sprawdzianie takie uzasadnienia można pisać krótszymi zdaniami, na przykład takimi:

Wartości `A` oraz `B` nie ulegają zmianie i są stale równe odpowiednio a oraz b. Stąd wartościami `X` są kolejne wielokrotności a. Każda obliczona wielokrotność jest porównywana z b.

Przed zakończeniem programu mamy `X` $\geqslant b$. Jeżeli b jest wielokrotnością a, to w pewnym momencie `X` $= b$, program skoczy do etykiety `wynik0`, wypisze 0 i zatrzyma się. W przeciwnym razie, wartość `X` nigdy nie będzie $= b$, a gdy przekroczy b, to program napisze 1 i zakończy pracę.

3.3 O specyfikacjach – teoretycznie

Specyfikacja powinna dostarczać wszystkich istotnych z punktu widzenia użytkownika informacji o programie. Użytkownik zapewne chciałby wiedzieć, jak zakończy się działanie programu dla konkretnych danych. Chciałby też wiedzieć, co można sądzić o danych znając wynik działania programu. Wobec tego uzasadnienie następującej specyfikacji:

Specyfikacja:

Wejście: liczby naturalne a i b wczytane odpowiednio do 1 i 2 komórki,

Wyjście: 0, jeżeli $\varphi(a, b)$
oraz 1, w przeciwnym przypadku.

powinno polegać na wykazaniu o słuszności dwóch równoważności

$$\varphi(a, b) \Leftrightarrow \text{wynikiem obliczeń jest } 0$$

oraz

$$\neg\varphi(a, b) \Leftrightarrow \text{wynikiem obliczeń jest } 1.$$

Zwroty *wynikiem obliczeń jest 0* i podobne są tutaj skrótami myślowymi oznaczającym, że tak jest po uruchomieniu programu z danymi a i b.

Obie podane równoważności są bardzo podobne i często wystarczy wykazać tylko jedną z nich. Tak jest w przypadku, gdy zachodzi alternatywa

$$\text{wynikiem obliczeń jest } 0 \ \vee \ \text{wynikiem obliczeń jest } 1.$$

Implikuje ona właściwie równoważność

$$\text{wynikiem obliczeń nie jest } 0 \Leftrightarrow \ \text{wynikiem obliczeń jest } 1,$$

ponieważ obliczenia są deterministyczne i 0 oraz 1 nie mogą być jednocześnie wynikami obliczeń. Z kolei ta równoważność oznacza, że obie równoważności uzasadniające specyfikację stwierdzają to samo.

W sytuacji jak wyżej można też uzasadnić specyfikację pokazując tylko dwie implikacje

$$\varphi(a, b) \Rightarrow \text{wynikiem obliczeń jest } 0$$

oraz

$$\neg\varphi(a, b) \Rightarrow \text{wynikiem obliczeń jest } 1.$$

Właśnie taki jest ogólny plan uzasadnienia przytoczonego w rozdziale 3.1. To samo da się uzyskać wykazując dwie implikacje odwrotne.

Zdarza się jednak, że podana alternatywa nie jest prawdziwa, a zachodzi dopiero alternatywa

$$\text{wynikiem obliczeń jest } 0 \ \vee \ \text{wynikiem obliczeń jest } 1 \ \vee \ \text{program nie zatrzymuje się.}$$

Wtedy działanie programu „kończy się” na trzy sposoby. Bywa też, że jest więcej dopuszczalnych wyników obliczeń. W takich przypadkach specyfikacja powinna podawać warunki kończenia działania programu na wszystkie możliwe sposoby. Wtedy możemy ją uzasadniać analogicznie, a większość poczynionych spostrzeżeń pozostaje prawdziwa.

3.4 Kilka uwag

Teraz musimy zauważyć, że uzasadnienie przedstawione w punkcie 3.1 nie jest w pełni poprawne. Staranne prześledzenie podanej argumentacji pozwala znaleźć w niej kilka usterek. Stawia to też pod znakiem zapytania słuszność specyfikacji przytoczonej na początku rozdziału.

Szukanie dziury w całym to podstawowy obowiązek studentów. Dlatego proponuję, aby przed dalszym czytaniem tekstu wrócić do podanej specyfikacji, a zwłaszcza jej uzasadnienia, i ponownie spróbować ocenić ich poprawność.

Wiąże się to z problemem, czy zero jest liczbą naturalną. Na ten temat pojawiają się sprzeczne opinie. Liczby naturalne służą do liczenia rzeczy i może być trudne do wyobrażenia, jak można policzyć coś, czego nie ma. Zgódźmy się z tym na chwilę i nie zaliczajmy 0 do liczb naturalnych. W takim przypadku specyfikacja jest w pełni poprawna, a jej uzasadnienie z punktu 3.1 jest właściwie poprawne i można w nim znaleźć najwyżej kilka luk. Inaczej jest w przeciwnym przypadku.

3.5 Lepsza specyfikacja

Jest kilka powodów, dla których zero jednak powinno być liczbą naturalną. Zdarza się, liczymy przedmioty spełniające warunek, który okazał się sprzeczny. Tak więc ostatnią specyfikację trzeba poprawić. Lepsza jest następująca:

Specyfikacja:

Wejście: liczby naturalne a i b wczytane odpowiednio do 1 i 2 komórki,

Wyjście: 0, jeżeli liczba b jest niezerową wielokrotnością liczby a
oraz 1, jeżeli $a > 0$ i b nie jest niezerową wielokrotnością a.

Zamiast mówić, że nie ma 0, mówimy, że interesują nas tylko liczby większe od 0, a także wielokrotności większe od 0 (albo „więcej niż 0 razy"), czyli niezerowe[1]. Po takich zmianach uzasadnienie 3.1 staje się właściwie poprawne. Może warto je uzupełnić w dwóch miejscach.

Nasz program analizuje wielokrotności począwszy od $1 \cdot a$. Błędem pierwszej specyfikacji było traktowanie wielokrotności $0 \cdot a$ jak każdej innej, mimo że zwykle znajduje się poza rozważanym zakresem. Teraz specyfikacja stwierdza, że program bada mniej, tylko sprawdza, czy dana liczba jest wielokrotnością z zakresu, który faktycznie jest badany przez program.

Podobny jest charakter drugiej usterki: program robi mniej, niż stwierdzała to pierwotna specyfikacja. Warunkiem wydrukowania liczby 1 było znalezienie wielokrotności a większej niż b. Takiej wielokrotności może nie być dla $a = 0$. Jeżeli wymienionym w specyfikacji warunkiem wydrukowania 1 będzie dodatniość a, to zagwarantuje on istnienie dostatecznie dużych wielokrotności a i pozwoli na udowodnienie poprawności specyfikacji.

Rozważana teraz specyfikacja także ma wadę: nie opisuje działania programu we wszystkich przypadkach. Jest to jednak poprawna specyfikacja w tym sensie, że podaje poprawne warunki wyprowadzenia na taśmę wyjściową wartości 0 i 1. Z drugiej strony, uzasadnienie tego faktu chyba nie jest do końca jasne. Skupialiśmy się na dowodzeniu pewnych implikacji, a z nich potrzebne równoważności – zgodnie z treścią punktu 3.3 – wynikają tylko niektórych sytuacjach.

[1]Precyzyjna definicja niezerowej wielokrotności jest następująca: liczba b jest niezerową wielokrotnością liczby a wtedy i tylko wtedy, gdy $b = k \cdot a$ dla pewnej liczby naturalnej $k > 0$.

3.6 Najlepsza specyfikacja

Po usunięciu ostatniej usterki otrzymujemy następującą specyfikację:

Specyfikacja:

Wejście: liczby naturalne a i b wczytane odpowiednio do 1 i 2 komórki,

Wyjście: 0, jeżeli liczba b jest niezerową wielokrotnością liczby a,
oraz 1, jeżeli $a > 0$ i b nie jest niezerową wielokrotnością a.
W pozostałych przypadkach program liczy bez końca (i na taśmie wyjściowej nic nie zostaje napisane).

Jest w niej dość trudny do zrozumienia zwrot „*w pozostałych przypadkach*". Zauważmy, że napis

$$(\exists k > 0\ b = k \cdot a)\ \vee\ (a > 0 \wedge \forall k > 0\ b \neq k \cdot a)$$

mówi to samo, co

$$a > 0 \vee b = 0.$$

Wobec tego zwrot „*w pozostałych przypadkach*" oznacza tu „*gdy* $a = 0$ *i* $b > 0$". Użycie tego zwrotu dałoby nieco bardziej zrozumiałą specyfikację.

Rzeczą niepokojącą jest to, że o możliwości zapętlenia analizowanego programu wspomniała na sprawdzianie chyba tylko jedna osoba.

3.7 Niezmienniki

Uzasadniając taką specyfikację, być może należy się posłużyć niezmiennikami pętli. Analizowany program ma jedną pętlę, która ma wiele niezmienników. Dla tej pętli niezmiennikiem jest własność $\psi(X, A, B)$ wartości zmiennych X, A, B oraz różnych stałych (np. liczb a i b), dla której zachodzi implikacja

jeżeli $X < B$ oraz przed wykonaniem instrukcji $X \leftarrow X + A$ zachodzi $\psi(X, A, B)$,

to własność $\psi(X, A, B)$ zachodzi także po jej wykonaniu.

Śledząc dalsze wyjaśnienie dobrze mieć przed oczami jakaś wersję rozważanego programu. Może to być dowolna ostatnia z trzech podawanych w rozdziale 2 obok siebie. W pierwszej mamy pętlę zaczynającą się etykietą `pp` i zakończoną instrukcją `Idź do pp`, w drugiej – pętlę `dopóki`. Wewnątrz tych dwóch pętli jest wykonywana tylko jedna instrukcja $X \leftarrow X + A$.

Mówiąc w skrócie, własność $\psi(X, A, B)$ jest niezmiennikiem pętli, jeżeli w sytuacji, gdy nie wychodzimy z pętli (np. w wyniku skoku), jej zachodzenie przed wykonaniem instrukcji wewnętrznych pętli gwarantuje też jej prawdziwość po wykonaniu tych instrukcji.

Z teorii niezmienników wynika, że niezmienniki prawdziwe przed przystąpieniem do wykonywania pętli są prawdziwe podczas jej wykonywania, a zwłaszcza **po jej zakończeniu**. Po zakończeniu pętli dodatkowo zachodzi też warunek powodujący jej zakończenie. W rozważanym teraz programie jest to nierówność $X \geqslant B$.

3.8 Dokładniejsze uzasadnienie specyfikacji

W rozważanym tutaj programie, następujące własności są niezmiennikami tej pętli, która w nim występuje:

1) $A = a$ oraz $B = b$,

2) X jest niezerową wielokrotnością A,

3) $X < A + B$,

Trzy pierwsze własności oczywiście są niezmiennikami. Dla przykładu pokażemy, że niezmiennikiem jest także ostatnia własność.

Załóżmy więc, że przed wykonaniem przypisania $X \leftarrow X + A$ zachodzą nierówności $X < B$ oraz $X < A+B$. Z pierwszej nierówności wynika, że zachodzi także $X + A < A + B$. Po wykonaniu przypisania, nową wartością X staje się dotychczasowe $X + A$, a więc znowu zachodzi nierówność $X < A + B$.

Aby wykazać poprawność najlepszej specyfikacji, wystarczy uzasadnić trzy implikacje:

1) jeżeli b jest niezerową wielokrotnością a, to na taśmie wyjściowej zostaje wypisane 0,

2) jeżeli $a > 0$ oraz b nie jest niezerową wielokrotnością a, to na taśmie wyjściowej zostaje wypisane 1,

3) jeżeli $a = 0$ i $b > 0$, to program nie kończy pracy.

Implikacje odwrotne do wymienionych też będą prawdziwe. Tak jest ponieważ każda para liczb naturalnych spełnia jeden z wymienionych w specyfikacji warunków, a także dlatego, że żadne wymienione w specyfikacji efekty działania programu nie mogą zajść jednocześnie. Na przykład, jeżeli na taśmie wyjściowej pojawi się liczba 0, to ani nie pojawi się liczba 1, ani program nie zapętli się, gdyż instrukcja przekazująca liczbę 0 znajduje się na końcu programu i po jej wykonaniu program nic już nie zrobi, w szczególności nie spowoduje zapisania na taśmie wyjściowej liczby 1.

Najpierw pokażemy, że w dwóch sytuacjach program zatrzymuje się.

Tak jest wtedy, gdy $b = 0$. W tym przypadku, przed przystąpieniem do wykonywania pętli spełniony jest warunek $B \leqslant X$ przerywający jej wykonywanie. Zależnie od śledzonej wersji programu, w tej sytuacji albo następuje skok do instrukcji kończących program, albo zostaje zakończone wykonywanie instrukcji `dopóki`.

Jeżeli natomiast $a > 0$, to w pętli jest wielokrotnie zmieniana wartość zmiennej X, a ciąg kolejno przyjmowanych wartości zmiennej X jest w tym przypadku rosnący. Dostatecznie długi rosnący ciąg liczb naturalnych zawiera liczby przekraczające z góry zadaną wartość. Stąd podczas wykonywania pętli w pewnym momencie staje się prawdziwa nierówność $B \leqslant X$, która w pierwszym rzędzie powoduje zakończenie wykonywania pętli, a następnie – całego programu.

Spróbujmy teraz pokazać pierwszą z wymaganych implikacji. Załóżmy więc, że b jest niezerową wielokrotnością a. Jeżeli $a = 0$, to także $b = 0$ i od razu zachodzi warunek $X = B$. Jest oczywiste, że wtedy na taśmie wyjściowej pojawia się wartość 0. Jeżeli zaś $a > 0$, to także $b > 0$ i tuż przed pętlą staje się prawdziwa nierówność $X < A + B$. Ponieważ jest ona niezmiennikiem, więc również zachodzi po (ewentualnym) zakończeniu wykonywania pętli. Wiemy już, że ta pętla nie jest wykonywana „w nieskończoność” (ponieważ $a > 0$). Po jej zakończeniu zachodzi też warunek zakończenia pętli, czyli $B \leqslant X$. Mamy więc dwie liczby: wartość zmiennej X po zakończeniu pętli oraz b spełniające nierówność

$$X - a < b \leqslant X.$$

Obie te liczby są niezerowymi wielokrotnościami a, a w przedziale $(X - a, X]$ jedyną taką liczbą jest X. Stąd $X = b = B$. W tej sytuacji, program zapisuje na taśmie wyjściowej 0 i kończy pracę.

W bardzo podobny sposób pokazujemy drugą implikację. Z założenia z tej implikacji mamy, że $a > 0$. Jak wyżej pokazujemy, że $B \leqslant X$. Ponieważ X jest, a b nie jest niezerową wielokrotnością a, więc $B < X$, program zapisuje na taśmie wyjściowej 1 i kończy pracę.

Pozostaje do udowodnienia trzecia implikacja. Załóżmy więc, że $a = 0$, $b > 0$ i – dla dowodu nie wprost – że program jednak zatrzymuje się. W tej sytuacji program wcześniej zakończy wykonywanie pętli. Wtedy wartość zmiennej X będzie niezerową wielokrotnością a spełniającą nierówność $b \leqslant X$, czyli b będzie równe 0. To jednak przeczy warunkowi $b > 0$ i kończy dowód trzeciej implikacji, a także uzasadnienie ostatniej z rozważanych specyfikacji.

4 Pierwsza część zadania

Znając dobrą specyfikację bardzo łatwo odpowiedzieć na pytanie, co zostanie zapisane na taśmie wyjściowej po uruchomieniu naszego programu z taśmą wejściową, na której znajdują się liczby 5 i 4. Wtedy $a = 5$ i $b = 4$. Oczywiście, b nie jest niezerową wielokrotnością $a > 0$. Zgodnie ze specyfikacją, w tej sytuacji program spowoduje zapisanie na taśmie wyjściowej liczby 1 i zakończy pracę.

Być może jednak jest uzasadnione szukanie odpowiedzi na takie pytanie przed znalezieniem specyfikacji. Wtedy takie zadanie można rozwiązać w inny sposób. Spróbujmy znaleźć odpowiedź na postawione pytanie przy założeniu, że na taśmie wejściowej mamy zapisane liczby 5 i 10. Można to zrobić poprzez utworzenie i wypełnienie takiej oto tabelki, symulując w ten sposób działanie danego programu:

Instrukcja	Wejście	Akum.	K. nr 1	K. nr 2	K. nr 3	Wyjście
	5, 10					
...						

Możemy nawet uzupełnić tę tabelę o wskaźnik (licznik) rozkazów. Spróbujmy to zrobić. Przyjmijmy jako zasadę, że zapisujemy liczbę w odpowiednim miejscu tabelki, jeżeli w wyniku wykonania instrukcji następuje zmiana zawartości pewnej komórki. Tak więc zawartość komórki to ostatnia liczba zapisana w odpowiadającej jej kolumnie. Umówmy się także, że instrukcje danego programu zostały ponumerowane w naturalny sposób i wskaźnik rozkazu zawiera tak rozumiany numer instrukcji, która ma zostać wykonana.

Instrukcja	Wejście	Akum.	K. nr 1	K. nr 2	K. nr 3	Wyjście	Wsk. r.
	5, 10	?	?	?	?	`puste`	1
`Read 1`	10		5				2
`Load 1`		5					3
`Store 3`					5		4
`Read 2`	`pusta`			10			5
`Load 1`		5					6
`Sub 2`		-5					7
`Jzero w0`							8
`Jgtz w1`							9
`Load 1`		5					10
`Add 3`		10					11
`Store 1`			10				12
`Jump pp`							5
`Load 1`		10					6
`Sub 2`		0					7
`Jzero w0`							15
`Write =0`						0	16

5 Trzecia część zadania

5.1 Adresowanie pośrednie

Pozostała do omówienia ostatnia część zadania. Aby ją zrobić trzeba wiedzieć, co to jest adresowanie pośrednie, a właściwie w jaki sposób jest wykonywana instrukcja `Write ^n`. Zacznijmy więc od przypomnienia semantyki tej instrukcji. Aby o niej opowiadać, dobrze jest wprowadzić symbol oznaczający zawartość n-tej komórki pamięci maszyny RAM. Przyjmijmy więc, że jest to wartość oznaczana wzorem $m[n]$. W szczególności, napis $m[0]$ oznacza zawartość akumulatora.

Wykonanie instrukcji `Write ^n` polega na

1) pobraniu z komórki pamięci o numerze (adresie) n numeru (adresu) adr komórki z interesującą nas wartością; można myśleć, że wykonujemy przypisanie $adr \leftarrow m[n]$,

2) pobraniu z komórki o adresie adr jej zawartości, czyli na przykład wykonaniu przypisania $liczba \leftarrow m[adr]$,

3) przekazaniu na taśmę wyjściową pobranej wartości $liczba$.

Pisząc w skrócie, wykonanie instrukcji `Write ^n` polega na przekazaniu na taśmę wyjściową wartości $m[m[n]]$.

5.2 Analiza sytuacji

Teraz powinniśmy zastanowić się, jakie konsekwencje będzie miało zastąpienie w rozważanym programie instrukcji `Write =0` poleceniem `Write ^0`.

Zwykle instrukcja maszyny RAM może zostać wykonana w dwóch sytuacjach: bezpośrednio po wykonaniu poprzedniej instrukcji lub po wykonaniu odpowiedniego skoku. Instrukcja `Write =0` może zostać wykonana tylko w wyniku skoku, ponieważ znajduje się za instrukcją `Halt` przerywającą wykonywanie programu. Co więcej, odpowiedni skok wykonuje tylko jedna instrukcja i jest ona postaci `Jzero wynik0`. Tak więc skok do instrukcji `Write =0` jest wykonywany po stwierdzeniu, że w akumulatorze znajduje się 0, a więc gdy $m[0] = 0$. Wykonanie skoku nie zmienia zawartości akumulatora, czyli podana równość będzie też zachodzić tuż przed wykonaniem instrukcji `Write =0`. Wykonując tę instrukcję maszyna RAM wysyła na taśmę wyjściową liczbę 0, a następnie przerywa wykonywanie programu (wykonała ostatnią instrukcję).

Jeżeli zamiast `Write =0` pojawi się instukcja `Write ^0`, to maszyna RAM zrobi prawie to samo. Zamiast przesyłać na taśmę wyjściową wartość podaną w programie, wyliczy i prześle na taśmę wyjściową wartość

$$m[m[0]] = m[0] = 0,$$

a następnie zakończy wykonywanie programu. Zmieniona część programu działa więc tak samo, jak pierwotna, i dokonana zmiana nie ma wpływu na specyfikację.

5.3 Rozwiązanie

Aby rozwiązać ostatnią część zadania, należało więc swierdzić, że po zmianie ostatniej instrukcji program ma dokładnie taką samą specyfikację, jak przed zmianą, i przytoczyć – może w nieco skróconej formie – kilka z wyżej przytoczonych argumentów.