Algoritmer, datastrukturer och komplexitet, hösten 2016

Uppgifter till övning 8

Oavgörbarhet

Övningens ena timme ägnas åt genomgång av mästarprov 1.

Oavgörbarhet 1 Problemet *orm i kakel* tar som indata en mängd T av kakelplattstyper och två punkter p_1 och p_2 i planet. Frågan är om det går att använda endast kakeltyper i T kakla en orm som ringlar sig från p_1 till p_2, som inte bryter kakelmönstret någonstans och som hela tiden håller sig i det övre halvplanet.

Är följande varianter av ormproblemet avgörbara eller oavgörbara?

a) Indata utvidgas med en fjärde parameter, en kakeltyp $t \in T$. Första kakelplattan (den som täcker p_1) måste vara av typ t.

b) Indata utvidgas med ett tal N och frågan utvidgas med bivillkoret att ormen måste bestå av högst N plattor.

c) Indata utvidgas med ett tal N och frågan utvidgas med bivillkoret att ormen måste bestå av åtminstone N plattor.

Oavgörbarhet 2 Finns det något explicit program P så att det givet y är avgörbart huruvida P stannar på indata y?

Oavgörbarhet 3 Finns det något explicit program P så att det givet y är oavgörbart huruvida P stannar på indata y?

Oavgörbarhet 4 Om programmet x stannar på tomt indata så låter vi $f(x)$ vara antalet steg innan det stannar. Annars sätter vi $f(x) = 0$. Definiera nu $MR(y)$, den maximala körtiden över alla program vars binära kodning är mindre än y.

$$MR(y) = \max_{x \leq y} f(x),$$

Är MR beräkningsbar?

Oavgörbarhet 5 Visa att funktionen MR i föregående uppgift växer snabbare än varje rekursiv funktion. Visa närmare bestämt att för varje rekursiv funktion g finns ett y så att $MR(y) > g(y)$.

Oavgörbarhet 6 Anta att en turingmaskin M, indata X (som står på bandet från början) och en heltalskonstant K är givna. Är följande problem avgörbart eller oavgörbart?

Stannar M på indata X efter att ha använt högst K rutor på bandet (en använd ruta får skrivas och läsas flera gånger)?

Oavgörbarhet 7 En *rekursivt uppräknelig mängd* definierades på föreläsningen som ett språk som kan kännas igen av en funktion vars ja-lösningar kan verifieras ändligt. En alternativ definition är en mängd som är uppräknelig (elementen kan numreras med naturliga tal) och kan produceras av en algoritm. Använd den senare definitionen och visa att varje rekursiv mängd också är rekursivt uppräknelig.

Oavgörbarhet 8 Den *diagonaliserade stoppmängden* består av alla program p som stannar på indata p. Visa att denna mängd är rekursivt uppräknelig.

Lösningar

Lösning till Oavgörbarhet 1

a) Oavgörbart. Vi reducerar *orm-i-kakel* till *orm-med-speciell-början* på följande sätt.

orm-i-kakel$(T, p_1, p_2) =$
    for $t \in T$ **do**
        if *orm-med-speciell-början*(T, p_1, p_2, t) **then**
            return true
    return false

b) Avgörbart. Vi behöver ”bara” testa alla tänkbara ormar av längd högst N som börjar i p_1 och se om dom duger. Eftersom detta är ett ändligt antal (begränsat av $(3|T|)^N$) så är algoritmen ändlig och problemet avgörbart.

c) Oavgörbart. Vi reducerar *orm-i-kakel* till *orm-med-minsta-längd* på följande sätt.

orm-i-kakel$(T, p_1, p_2) =$
    return *orm-med-minsta-längd*$(T, p_1, p_2, 1)$

□

Lösning till Oavgörbarhet 2
Ja, det finns massor av sådana program. Ta till exempel det enkla programmet $P(y) =$ **return**. Det är lätt att se att P stannar på alla indata. □

Lösning till Oavgörbarhet 3
Ja. Låt till exempel P vara en interpretator och indata y vara ett program x följt av indata y' till det programmet. Det betyder att $P(y)$ beter sig precis som programmet x skulle göra på indata y', och det är ju oavgörbart huruvida ett program x stannar på ett visst indata y'. □

Lösning till Oavgörbarhet 4
Nej, *MR* är inte beräkningsbar. Vi vet att det är oavgörbart huruvida ett program stannar på blankt indata (tomma strängen). Reducera därför detta problem till *MR*. Notera att *MR*(y) är det maximala antalet steg som varje program av ”storlek” högst y behöver innan det stannar om det startas med blankt indata.

StopBlank(y) =
    $s \leftarrow MR(y)$
    if $s = 0$ **then return false**
    else
        simulera y på blankt indata i s steg (eller tills y stannar)
        if y stannade **then return true**
        else return false

Om inte y har stannat inom s steg så vet vi att den aldrig stannar. Eftersom StopBlank inte är beräkningsbar så kan inte MR vara det heller. □

Lösning till Oavgörbarhet 5
Anta motsatsen, dvs att det finns en rekursiv funktion g så att $g(y) \geq MR(y)$ för alla y. Då skulle vi kunna byta ut första satsen $s \leftarrow MR(y)$ i programmet i lösningen till föregående uppgift mot satsen $s \leftarrow g(y)$ och programmet skulle ändå fungera. Dessutom skulle programmet då bli beräkningsbart vilket vi vet att det inte kan vara. Därför har vi en motsägelse och vårt antagande var alltså falskt. □

Lösning till Oavgörbarhet 6
Problemet är avgörbart. Eftersom turingmaskinen måste hålla sig inom K rutor på bandet så finns det bara ett ändligt antal konfigurationer som turingmaskinen kan befinna sig i. Om maskinen har m tillstånd så är antalet konfigurationer $m \cdot K \cdot 3^K$ (antalet tillstånd gånger antalet möjliga positioner för läs- och skrivhuvudet gånger antalet möjliga bandkonfigurationer). Simulera därför turingmaskinen i $m \cdot K \cdot 3^K + 1$ steg och kontrollera hela tiden att den inte rör sig utanför dom K rutorna på bandet. Om turingmaskinen stannar inom denna tid svarar vi *ja*. Om turingmaskinen inte stannat efter denna tid så måste den ha återkommit till en konfiguration som den tidigare varit i, och därmed är den inne i en oändlig slinga och vi kan tryggt svara *nej*. □

Lösning till Oavgörbarhet 7
Anta att S är en rekursiv mängd. Det betyder att det finns en algoritm $A(x)$ som talar om ifall $x \in S$. Elementen i S lagras naturligtvis som allt annat som binära strängar. Följande program genererar mängden S:

for $i \leftarrow 0$ **to** ∞ **do**
    if $A(i)$ **then write** i

□

Lösning till Oavgörbarhet 8

for $i \leftarrow 0$ **to** ∞ **do**
    for $p \leftarrow 0$ **to** i **do**
        simulera beräkningen $p(p)$ under i steg
        if $p(p)$ stannar inom i steg **then write** p

Samma p kommer att skrivas ut många gånger. Om man vill undvika det får man se till att bara skriva ut p om antingen $p(p)$ stannar på exakt i steg eller $p = i$. □