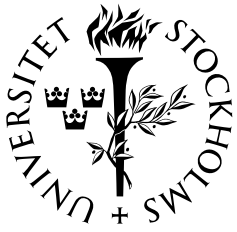

SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Turings maskin, beräkningsbarhet och avgörbarhetsproblemet

av

Simon Wikander

2009 - No 9

Turings maskin, beräkningsbarhet och avgörbarhetsproblemet

Simon Wikander

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Rikard Bøgvad

2009

Sammanfattning

Detta är en beskrivning och förklaring av Alan Turings arbete om sin maskin med utgångspunkt i hans artikel ”On computable numbers with an application to the Entscheidungsproblem” från 1936. Jag visar hur Turingmaskiner fungerar och arbetar och hur vi med hjälp av dessa kan definiera de beräkningsbara talen och mer generellt även beräkningsbarhet. Efter detta konstrueras den universella Turingmaskinen, en maskin som kan ta en annan Turingmaskin som indata. Med hjälp av denna maskin visas det att det finns en sats i först ordningens predikatlogik som är oavgörbar, och därigenom löser vi avgörbarhetsproblemet med en negativ lösning.

Innehåll

1 Introduktion

Inspirationen och frågeställningarna som ligger bakom en stor del av 1900-talets arbete i logik började under den andra internationella matematikkongressen i Paris år 1900. Under denna kongress var den tyske matematikern och logikern David Hilbert inbjuden att föreläsa, och han bestämde sig för att dra upp riktlinjer för i vilken riktning han tyckte att 1900-talets matematik borde utvecklas. Hans numera berömda föreläsning innehöll 23 frågor, obesvarade problem inom spridda matematiska områden, varav speciellt en har legat till grunden för mycket studier i matematisk logik och först fick sin lösning under 30-talet. Det Hilbert frågade efter i sin 10:e frågeställning var en generell process för att bestämma bevisbarheten av godtyckliga påståenden i matematisk logik, ett problem som brukar kallas Entscheidungsproblemet eller avgörbarhetsproblemet. Det var denna frågeställning Alan Turing (1912-1954) ville besvara i sin uppsats ”On computable numbers, with an application to the Entscheidungsproblem” som han publicerade år 1936.

Turings arbete var ett av många framsteg i matematisk logik under 30-talet (varav många med samma eller liknande frågeställning), men trots detta och att Alonzo Church (1903-1995) publicerade ett likvärdigt arbete innan Turing, lyckades Turing uppfinna en ny och smidigare teori för att lösa avgörbarhetsproblemet (med ett negativt svar). I sin uppsats försöker han definiera vad som är beräkningsbart, och för att göra detta introducerar han en imaginär och ytterst teoretisk maskin för att simulera vad en människa kan beräkna, och med hjälp av denna dra gränsen för vad som är beräkningsbart. Denna maskin, som senare kommit att kallas för Turingmaskinen, var en mycket smidig lösning som inte bara användes som ett verktyg i hans uppsatts utan också var starten till modern datorvetenskap, och är än idag är en bra modell att arbeta med. När han väl definierat vad som är beräkningsbart, det som en Turingmaskin kan beräkna, använder han denna teori för att lösa avgörbarhetsproblemet med en negativ lösning, och därmed visa att det inte finns någon generell ändlig metod för att avgöra en sats sanningsvärde i första ordningens predikatlogik. Detta sätter en gräns för vad vi kan beräkna, och än mer anmärkningsvärt, en gräns för vad varje tänkbar dator någonsin kommer att kunna beräkna. Detta, med utgångspunkt i Alan Turings uppsats, kommer de kommande sidorna att handla om, och vi börjar med en beskrivning av vad en Turingmaskin egentligen är för något.

2 Turingmaskiner

I sin uppsats ”On computable numbers with an application to the Entscheidungsproblem” var som titeln implicerar Turings huvuduppgift att försöka definiera de beräkningsbara talen (computable numbers). Med att något är beräkningsbart, speciellt ett tal, menas att man kan beräkna detta med

ett ändligt antal olika steg, att det finns en ändlig algoritm som visar hur beräkningen ska göras. Meningen med att göra en definition av vad som är beräkningsbart är att försöka dra en gräns för vad en människa effektivt kan beräkna, och genom detta få kännedom om vad vi kan veta och vilka problem som är värda att försöka reda ut. För att lyckas göra denna definition uppfann Turing sin imaginära maskin, en mycket enkel uppsättning grundläggande delar som är enkel att överskåda men som samtidigt anses beräkna allt en människa skulle klara av. Efter detta är det enkelt att definiera de beräkningsbara talen. De beräkningsbara talen är de tal som kan beräknas av en sådan maskin.

2.1 Komponenter

En Turingmaskin, förkortat TM, är i grunden en uppsättning regler som i varje enskilt tillstånd som maskinen kan befinna sig i definierar vad maskinen ska göra och vilket nästa tillstånd är. Maskinen befinner sig alltid i exakt ett väldefinierat tillstånd och antalet olika tillstånd är alltid ändligt.

Vi ska nu formalisera maskinens olika komponenter, men kom ihåg att dessa komponenter endast är en teoretisk modell, de begrepp som tas upp har inte nödvändigtvis någon fysisk form. Maskinen har tillgång till en *remsa* bestående av oändligt många på varandra följande rutor, där varje ruta har möjlighet att innehålla en symbol ur ett ändligt alfabet. Maskinen har ett *läshuvud* som har möjlighet att läsa innehållet i precis en av dessa rutor i taget, och symbolen som läses i den i:e rutan betecknar vi med $S(i)$. Det som styr maskinen och som ger den dess förmåga att göra beräkningar kallar vi *lägen* och betecknas med $q_1, q_2, \ldots, q_n$, dessa är alltid ändligt många. I varje ögonblick är maskinen i precis ett läge och ser precis en symbol på remsan, och denna kombination av en läst symbol och ett läge kallar vi för maskinens *konfiguration*. För varje konfiguration har maskinen möjlighet att skriva över eller radera symbolen i rutan den befinner sig på (den ruta som läshuvudet kan se) och ändra den lästa rutan ett steg till höger eller ett steg till vänster. Varje konfiguration kan också låta maskinen övergå till ett nytt läge.

Remsan kan vi tänka på som en potentiellt oändlig rad med symboler i en bok, det vill säga vi orienterar oss på remsan från vänster till höger. När vi bara har blanka (tomma) rutor till vänster om läshuvudet kan vi säga att vi är i början av remsan (även om den är oändligt lång så innehåller den alltid bara ändligt antal symboler) och på motsvarande sätt kan vi säga att vi är i slutet av remsan när det bara är tomma rutor till höger om läshuvudet.

Det kanske inte verkar så enkelt på en sån här abstrakt nivå, men faktum är att dessa tre olika förändringar av maskinens och remsans tillstånd (flytta, skriv/radera och byta läge) är i princip allt som behövs för att beräkna ett beräkningsbart tal. Snart ska vi få se några konkreta exempel.

En viktig del i skapandet av sin maskin var för Turing att lyckas bygga in allt det man kan förvänta sig att en människa kan göra vid en beräkn-

ing i maskinen. Det är omöjligt att bevisa detta på ett strikt matematiskt sätt, vilket inte är så konstigt eftersom det är väldigt svårt att dra gränser och strikt definiera så dynamiska saker som människor, människor är inte matematiska objekt. Påståendet att allt som är beräkningsbart (av en människa) också är beräkningsbart av en Turingmaskin kallas Church–Turings tes, eller ibland men tyvärr inte lika ofta bara Turings tes, och är alltså inte bevisad. Eftersom Turings resonemang är mer av ett troliggörande än ett bevis av denna ekvivalens, är han noggrann att dra analogier mellan sin maskin och sättet som människor gör sina beräkningar på. Remsan som maskinen använder sig av motsvaras av papper, och har alltså som funktion att på något sätt hålla ordning och minnas vad maskinen gör. En människas ”state of mind”, en delberäkning som människan gör, en tanke, motsvaras i maskinen av lägen eller möjligtvis flera lägen som gör en deluppgift av hela maskinens uppgift.

2.2 Exempel på ett beräkningsbart tal

För att bättre förstå och få en känsla för vad en Turingmaskin gör ska vi titta på ett exempel som beräknar sekvensen 001100110011 . . .. De maskiner vi ska titta på är maskiner som skriver två sorters symboler. Den första sortens symboler är 0 och 1, det är dessa symboler vi läser av från remsan som maskinens resultat. Den andra sorten är alla andra symboler i maskinens alfabet. Den andra typen av symboler kommer att fungera som en slags uppmärkning och minnesfunktion för att utföra mer komplexa beräkningar, och det är bara dessa symboler vi får radera från remsan. Symbolerna av första sorten är maskinens resultat, om vi läser av remsan från vår startpunkt åt höger kommer vi att ha en sekvens av 1:or och 0:or, denna sekvens är maskinens resultat. Om vi sätter ett binärkomma framför denna sekvens får vi ett reellt tal (på binär form), och detta tal är det tal som maskinen beräknar. I vårt första exempel beräknas sekvensen 001100110011 . . ., och denna sekvens motsvarar talet 0,001100110011 . . . vilket med tio som bas är 11/15. Låt oss börja med exemplet. Turingmaskinen kan vi representera med följande tabell, som visar de olika övergångarna och beteenden vid respektive konfiguration, och kallas för *övergångstabell*:

Konfiguration		Beteende	
b	blank	P0, R	c
c	blank	R	d
d	blank	P0, R	e
e	blank	R	f
f	blank	P1, R	g

g	blank	R	h
h	blank	P1, R	i
i	blank	R	b

Denna tabell innehåller alla delar av Turingmaskinen som vi gått igenom. I den första kolumnen finns lägena, i detta fall 8 stycken namngivna med början på b i alfabetisk ordning. Observera att lägena är namngivna med gemener ur början på alfabetet med start på b. Vi kommer att använda namn på lägen så att övergångstabellerna blir så lättlästa som möjligt och bara använda vår beteckning $q_1, q_2, \ldots, q_n$ när vi blir tvungna att formalisera vårt resonemang. I den andra kolumnen har vi den lästa symbolen, symbolen i den ruta som är under läshuvudet. De två första kolumnerna på varje rad bildar tillsammans en konfiguration. I den tredje kolumnen ser man vad som ska göras med remsan. P visar att vi vill skriva (från engelskans print) och P:t följs sedan av symbolen vi vill skriva. I stället för P kan vi också använda E (Erase), som visar att vi vill radera symbolen i aktuell ruta. R (Right) visar att vi vill flytta läshuvudet ett steg till höger och L (Left) att vi vill flytta läshuvudet ett steg till vänster. I den sista kolumnen finns det läge vi ska övergå till. Exempelvis: om vi är i läge f och läser en ruta som är blank så ska vi skriva 1 i rutan, gå ett steg till höger och sedan övergå till läge g. När maskinen börjar sin beräkning är remsan helt tom och vi är i läge b (från engelskans begin). De två sista kolumnerna på varje rad (som kallas beteende) är de regler som något oprecist nämndes i början av det föregående stycket, dessa visar vad maskinen ska göra och till vilket läge maskinen övergår i vid varje konfiguration.

För att åskådliggöra maskinens olika steg kan vi göra en tabell över maskinens *fullständiga konfiguration*, vilket är konfigurationen tillsammans med de symboler som redan finns på remsan. Vi anger den fullständiga konfigurationen genom att först ange läget följt av ett kolon, och sedan sekvensen av symbolerna som finns på remsan med den lästa symbolen understruken. I vårt exempel blir den första fullständiga konfigurationen b:_. Här läses en blank ruta och alltså gör vi vad som står i tredje kolumnen för b, nämligen skriver en 0:a och går ett steg till höger, sedan övergår vi till läge c som den fjärde kolumnen visar. Nu har vi den fullständiga konfigurationen c:0_. Om vi fortsätter på samma sätt är det enkelt att se vad resultat kommer att bli, vår serie av fullständiga konfigurationer fortsätter:

d:0 _	e:0 0_	f:0 0 _
g:0 0 1_	h:0 0 1 _	i:0 0 1 1_
b:0 0 1 1 _		

Vi ser att efter en cykel av 8 byten av läget så är vi i läge b igen, och sekvensen 0011 har skrivits på remsan. Denna process kommer att börja om

och sedan fortsätta på samma sätt i oändlighet. Resultatet blir den sekvens vi ville beräkna: 001100110011 . . .

Lägg märke till att maskinen aldrig kommer att sluta skriva till remsan. Denna typ av maskin, som skriver ett oändligt antal symboler av den första sorten på remsan, kallas för cirkelfria maskiner. Motsatsen är en cirkulär maskin och en sådan skriver bara ett ändligt antal symboler av den första sorten till remsan. En cirkulär maskin kommer till en konfiguration som inte är definierad eller alternativt hamnar i en cykel där inga nya symboler av den första typen skrivs. De maskiner vi är intresserade av och som ger oss någon kunskap om de beräkningsbara talen är de cirkelfria, de som beräknar våra oändliga sekvenser. Vi ska begränsa oss till cirkelfria maskiner i denna text precis som Turing gjorde, trots att man brukar använda varianter av Turingmaskiner som är cirkulära i andra viktiga samanhang.

Lägg också märke till att maskinen mellan varje symbol av första typen som skrivs hoppar över en blank ruta. Detta är ett medvetet trick för att lättare kunna utnyttja remsans uppgift som minne, vars resurs först framgår i mer komplexa maskiner senare. Idén är att dela upp remsan i två olika sorters rutor. Varannan ruta är en ruta som innehåller symboler av första eller andra sorten och kallas F-rutor. Det är från dessa rutor vi läser av sekvensen av symboler av första sorten som vårt resultat, maskinens beräkning. De andra rutorna kallas E-rutor och får innehålla symboler av den andra typen. Det är dessa rutor som fungerar som ett slags minne för maskinen, eller kanske ett kladdpapper, och motsvarar det mänskliga minnet. Det är bara symboler i E-rutorna som maskinen får ändra och radera, F-rutornas innehåll ska vara statiska så fort de har blivit skrivna till.

2.3 Ett mer komplext exempel

För att skapa en Turingmaskin som är något mer sofistikerad, och som i någon mening av ordet kanske är mer intelligent, måste vi använda oss av E-rutorna som vi ignorerade i föregående exempel. Vi måste även utöka vårt alfabet så att det inte bara innehåller 0 och 1, utan även symboler av den andra sorten som vi använder för att styra beräkningen. Vi ska konstruera en TM som beräknar sekvensen 1011011101111 . . ., alltså en sekvens med successivt ökande antal 1:or separerade med 0:or. Vi låter vårt alfabet bestå av 0, 1, ə och x. ə symbolen kommer vi bara använda till att markera början på maskinens utskrift, i övrigt används den inte. För att spara utrymme och förenkla läsningen av övergångstabellen för maskinen ska vi göra en förenkling, men denna påverkar inte vilka beräkningar maskinen kan göra. I varje konfiguration ska vi tillåta att flera förflyttningar (L/R) görs och att vi för varje förflyttning får skriva en symbol eller radera en symbol. Detta gör ingen skillnad i vad maskinen kan beräkna, snarare drar man ihop en serie lägen till ett läge. Exempelvis kan vi om vi flyttar två steg till höger i konfigurationen $(b, S(i))$ och sedan hoppa till lägen c, alltid bara hoppa ett

steg i konfigurationen $(b, S(i))$ och hoppa till ett nytt läge d som när $S(i)$ läses också flyttar ett steg till höger och sedan hoppar till c. Samma sak gäller om vi har fler förflyttningar än 2, och i varje delkonfiguration kan vi självklart alltid skriva eller radera en symbol, så det är klart att det inte blir någon teoretisk skillnad mellan maskinerna. Två förflyttningar till höger anger vi genom R2, tre förflyttningar till vänster med L3 och så vidare. Nu är vi redo att titta på övergångstabellen för maskinen:

Konfiguration		Beteende	
b	blank	Pə, R, Pə, R, P1, R	c
c	ə	L	e
	x	E, R	d
	blank	$L2$	c
d	något	R2	d
	blank	P1, L	c
e	1, ə	R2	e
	0	R, Px, R	e
	blank	P0, R, Px, R	d

Maskinen har fyra olika lägen, men denna gång är det inte lika uppenbart hur de olika delarna samarbetar och vad de har för uppgift. Läge b anropas som vanligt när maskinen startar, men till skillnad från förra maskinen så återkommer den aldrig till detta läge, det är bara en initiering av de tre första symbolerna på remsan som kommer att vara əə1. Från b övergår vi till c som har tre olika sätt att agera på beroende av konfigurationen, och vars uppgift är att flytta läshuvudet till den första ə symbolen. Om maskinen stöter på ett x på vägen så raderas detta, ett steg till höger tas och maskinen övergår till läge d. Läge d har som uppgift att gå längst bak i maskinens utskrift och skriva en 1:a och sedan övergå till läge c igen. ”Något” i andra kolumnen vid läge d betyder någon symbol men inte blank, när vi vill inkludera blank i detta kommer vi att skriva ”allt”. När vi i läge c har kommit till början av utskriften, när vi står på den första ə symbolen, övergår maskinen till läge e. Läget e:s uppgift är att gå längst bak i maskinens utskrift och skriva en 0:a, och på vägen markera alla nollor med ett x, inklusive 0:an som skrivs. En markering görs genom att sätta ett x direkt till höger om 0:an som ska markeras. Mer allmänt: när vi sätter någon symbol α av den andra typen på en E-ruta direkt till höger om en ruta som innehåller symbolen A av den första typen, säger vi att vi har markerat rutan som innehåller A och att A är markerad med symbolen α. Syftet med att markera alla nollor när vi stegar oss upp med e är att hålla reda på hur många 1:or vi ska skriva efter 0:an som e skriver sist. För att se att tekniken fungerar tittar vi på några av de första fullständiga konfigurationerna (en längre tabell finns i appendix A):

b:_	c:əə1_	c:əə̲1
e:ə̲ə1	e:əə1̲	e:əə1 _
d:əə1 0x_	c:əə1 0x̲1	d:əə1 0 1̲
d:əə1 0 1 _	c:əə1 0 1_1	c:əə1 0_1 1
c:əə1_0 1 1	c:əə̲1 0 1 1	e:ə̲ə1 0 1 1

I den sista fullständiga konfigurationen är vi tillbaka i läge e för tredje gången, och om vi skulle fortsätta så skulle en 0:a skrivas ut sist i utskriften och alla 0:or bli markerade med x. Men vi ser redan hur maskinen arbetar, och vi kan enkelt övertyga oss om att nästa cykel kommer att skriva ut tre 1:or.

3 Den universella Turingmaskinen

Vi har redan sett två exempel på vad en TM kan göra, båda exemplen blir ganska omständiga att konstruera och är väldigt begränsade i det de gör. Varje TM definierar en enda algoritm och det finns uppenbart oändligt många Turingmaskiner och alltså verkar detta som en ganska dålig modell för att försöka definiera beräkningsbarhet. Det vi vill undvika är specialbyggda maskiner för enskilda beräkningar, utan vi vill ha en generell maskin som kan ta itu med alla problem vi kan tänka oss kan lösas med en algoritm. En sådan maskin kommer vi att kalla den universella Turingmaskinen (förkortat UTM). En UTM är programmerbar med andra Turingmaskiner, och kan ses mer som en modern dator där vi i samma arkitektur programmerar diverse olika program för att uppfylla våra krav för stunden och lösa våra problem. Den universella Turingmaskinen tar alltså en annan Turingmaskin som indata och gör sedan de beräkningar denna Turingmaskin skulle gjort. Hur vi ger en godtycklig Turingmaskin som indata till den universella Turingmaskinen ska vi titta på senare, först ska vi ta itu med lite förberedelser.

3.1 Förkortningar och subrutiner

Om vi konstruerar ett par maskiner för relativt enkla beräkningar ser vi ganska snabbt att samma eller liknande delsteg är inblandade i beräkningarna. Exempel på sådana delsteg som vi använder i vårt andra exempel är att gå till början av utskriften, att markera alla 0:or med x och att söka upp första förekomsten (bakifrån i vårt fall) av x och radera det. För att göra vår konstruktion av den universella Turingmaskinen både tydligare, kortare och enklare ska vi nu introducera en notation för att återanvända ofta återkommande (del)algoritmer. Dessa kan liknas vid metoder eller funktioner i moderna programmeringsspråk, de utför en mindre deluppgift som ofta kommer upprepas. Vi kallar våra förkortningar för *m-funktioner* (för maskin-funktioner), och kan se dessa som vanliga matematiska funktioner där vi substituerar varje förekomst av variablerna.

En vanlig uppgift en Turingmaskin gör är att leta upp den första (den längst till vänster) förekomsten av en symbol α. Vi kan enkelt skapa en m-funktion som gör denna uppgift och som om α hittas övergår till konfigurationen A och annars till konfigurationen B. Observera att α, A och B är variabler som kommer att substitueras när vi använder funktionen. Om vi namnger vår m-funktion till *hitta* får vi en m-funktion $hitta(A, B, \alpha)$ som definieras av övergångstabellen:

Konfiguration		Beteende	
$hitta(A, B, \alpha)$	ə	L	$hitta_1(A, B, \alpha)$
	inte ə	L	$hitta(A, B, \alpha)$
	blank	L	$hitta(A, B, \alpha)$
$hitta_1(A, B, \alpha)$	α		A
	inte α	R	$hitta_1(A, B, \alpha)$
	blank	R	$hitta_2(A, B, \alpha)$
$hitta_2(A, B, \alpha)$	något		$hitta_1(A, B, \alpha)$
	blank	R	B

M-funktionen *hitta* är som du ser uppbyggd med hjälp av två andra m-funktioner. *hitta* går till början av utskriften, till det första ə. $hitta_1$ och $hitta_2$ söker efter α, om $hitta_1$ hittar ett α är vi klara och går till A och om vi läser en blank ruta går vi till $hitta_2$ för att se om det är två blanka rutor på rad, i så fall har vi inte hittat α och vi går till B, annars tillbaka till $hitta_1$. Om vi substituerar A, B och α med exempelvis c, d och x så får vi en fullständig övergångstabell med tre lägen, och denna övergångstabell kan vi ”anropa” genom att i sista kolumnen på någon rad i en annan övergångstabell skriva $hitta(c, d, x)$. Detta skulle få samma effekt som om vi i övergångstabellen direkt skrev in de tre lägen vi just definierat, men nu har vi ett kortare och enklare sätt att göra detta på.

På liknande sätt kan vi definiera vilka andra m-funktioner vi vill och på så sätt konstruera ett bibliotek av ”bra-att-ha-metoder”. Observera att vi kan variera antalet parametrar till m-funktionerna och att vi även kan ha funktioner med samma namn men med olika antal parametrar. I det senare fallet så kan vi tänka på funktionerna som olika funktioner (funktioner med olika uppgift) och vilken vi ska använda förstår vi av hur många parametrar vi använder funktionen med. De m-funktioner vi behöver för att konstruera den universella Turingmaskinen ska jag beskriva, men inte noga definiera då jag tror att du börjar förstå konstruktionen och sidorna är begränsade. Följande m-funktioner kommer vi att använda oss av utöver den vi redan tittat på:

- $V(A)$, läshuvudet tar ett steg till vänster och övergår sedan till läge A.

- $H(A)$, läshuvudet tar ett steg till höger och övergår sedan till läge A.
- $hittaH(A, B, \alpha)$, fungerar som *hitta* förutom att den placerar läshuvudet ett steg till höger om det α som hittas.
- $hittaV(A, B, \alpha)$, fungerar som *hitta* förutom att den placerar läshuvudet ett steg till vänster om det α som hittas.
- $tabort(A, B, \alpha)$, första förekomsten av α raderas och maskinen övergår till läge A. Om det inte finns något α hamnar vi i B. Denna m-funktion kan definieras med hjälp av *hitta*.
- $tabort(A, \alpha)$, som föregående men denna tar bort alla förekomster av α. Beroende på antalet parametrar man anropar m-funktionen med så tas endera första förekomsten eller alla förekomster bort. Denna version av *tabort* kan definieras med hjälp av den förra.
- $skriv(A, \beta)$, skriver β på den första tomma F-rutan från början av utskriften sett, och övergår sedan till A. Använder sig av *hitta*. Mer generellt kan vi låta $skriv_n(A, \alpha_1, \alpha_2, \ldots, \alpha_n)$ vara en m-funktion som först skriver α_1 sist på remsan, sedan α_2 och så vidare till alla n symbolerna skrivits.
- $kopieraMarkerad(A, B, \alpha)$, skriver den första symbolen markerad med α sist på remsan och övergår till A. Om ingen symbol är markerad med α övergår maskinen till B. Använder sig av *hitta*.
- $kopieraMarkeradTabort(A, B, \alpha)$, som föregående men tar bort markeringen.
- $kopieraMarkeradTabort(A, \alpha)$, som föregående men kopierar alla markerade symboler (i ordning) och raderar alla markeringar. Vi låter även $kopieraMarkeradTabort_n(A, \alpha_1, \alpha_2, \ldots, \alpha_n)$ vara den generella (rekursiva) m-funktionen som kopierar de symboler markerade med α_1 till α_n till slutet av utskriften (i ordning) och raderar alla förekomster av markeringarna α_1 till α_n.
- $ersatt(A, B, \alpha, \beta)$ och $ersatt(A, \alpha, \beta)$, Ersätter den första förekomsten respektive alla förekomster av α med β och övergår sedan till A. Om det inte finns någon förekomst av α vid $ersatt(A, B, \alpha, \beta)$ övergår maskinen till B. Använder sig av *hitta*.
- $jamfor(A, B, C, \alpha, \beta)$, den första symbolen som är markerad med α och den första symbolen som är markerad med β jämförs. Om inga symboler är markerade med varken α eller β så övergår maskinen till C. Om det finns symboler markerade med α och β och de är lika övergår maskinen till A. I alla andra fall övergår maskinen till B.

- $jamforTabort(A, B, C, \alpha, \beta)$, Som föregående men innan vi övergår till A så raderas markeringarna (α och β).
- $jamforTabort(A, B, \alpha, \beta)$, som föregående men här fortsätter jämförelsen rekursivt, det vill säga vi jämför en sekvens av markerade symboler. Om sekvenserna är lika (innehåller lika många symboler och lika symboler i samma ordning) övergår maskinen till B, annars A. De markeringar för symboler som är lika (även om inte hela sekvenserna är lika) raderas.
- $hittaSista(A, B, \alpha)$, hittar den sista förekomsten av α om det finns någon och övergår sedan till A, annars övergår maskinen till B.
- $tabort(A)$, tar bort alla markeringar, det vill säga raderar innehållet i alla F-rutor.

Dessa är alla m-funktioner vi behöver för att skapa vår UTM, bortsett från en som vi visar senare.

3.2 De beräkningsbara talen är uppräkneliga

Alla beräkningsbara sekvenser, och därmed alla beräkningsbara tal, är bestämda av de Turingmaskiner som beräknar sekvensen. Samma sekvens kan bli beräknad av flera olika Turingmaskiner, exempelvis kan vi skapa flera Turingmaskiner som beräknar samma sak genom att lägga till lägen som inte har någon funktion, som inte gör något. Men observera att varje cirkelfri TM av vår typ är designad för att beräkna exakt en sekvens.

3.2.1 Kodning av Turingmaskiner

Vi vill visa att de beräkningsbara talen är uppräkneliga, och detta kan vi göra genom att visa att antalet Turingmaskiner är uppräkneliga. För att visa att antalet Turingmaskiner är uppräkneliga visar vi att det finns en bijektion mellan dessa och en delmängd av de naturliga talen, som vi vet är uppräknelig.

För att göra detta behöver vi ett nytt sätt att beskriva varje Turingmaskin, ett som är kompaktare och lättare att jämföra med heltalen än våra övergångstabeller. Vi kommer att koda Turingmaskinerna, och detta kommer vi även att använda oss av senare när vi konstruerar den universella Turingmaskinen. Låt oss gå tillbaka till den allra enklaste och mest avskalade övergångstabellen. I den tillåter vi bara maskinen ta ett steg vid varje konfiguration. Alfabetet till Turingmaskinen i fråga låter vi symboliseras av $S_0, S_1, S_2, \ldots, S_n$, där vi bestämmer att $S_0 = blank$, $S_1 = 0$ och $S_2 = 1$. Vi låter även våra lägen symboliseras av $q_1, q_2, \ldots, q_n$, där q_1 alltid är det läge vi börjar på. Med denna notation kommer varje rad i en övergångstabell se ut på ett av tre olika standardiserade sätt. Varje sådan rad kan skrivas på

en kompakt form som en femtuppel, som består av vad vi hittar på en rad i övergångstabellen, möjligtvis bortsett från P, och ser ut som $q_i S_j S_k L q_m$, $q_i S_j S_k R q_m$ eller $q_i S_j S_k N q_m$ där N representerar att inget steg tas, där vi för att inte skriva någon symbol skriver samma symbol som är under läshuvudet och för att radera en symbol skriver S_0 (symbolen för blank). Om vi låter varje femtuppel börja med ett semikolon och skriver alla rader i en övergångstabell på detta sätt får vi ett ord som på ett entydigt sätt beskriver en TM. För att förenkla denna beskrivning ytterligare låter vi nu (som Turing gjorde) q_i beskrivas av ett D följt av i stycken A och på samma sätt S_i av D följt av i stycken C. Nu består vår beskrivning av maskinen enbart av symbolerna ”D”, ”A”, ”C”, ”L”, ”R”, ”N” och ”;”. Denna beskrivning kallar vi *standardbeskrivningen* av en Turingmaskin.

Som exempel kan vi hitta standardbeskrivningen av maskinen i exempel 1 (se sida 3, sektion 2.2). Övergångstabellen kan beskrivas enlingt vår metod som:

$;q_1 S_0 S_1 \mathrm{R} q_2 ; q_2 S_0 S_0 \mathrm{R} q_3 ; q_3 S_0 S_1 \mathrm{R} q_4 ; q_4 S_0 S_0 \mathrm{R} q_5 ; q_5 S_0 S_2 \mathrm{R} q_6 ; q_6 S_0 S_0 \mathrm{R} q_7$
$; q_7 S_0 S_2 \mathrm{R} q_8 ; q_8 S_0 S_0 \mathrm{R} q_1$

Och när vi skriver detta som en standardbeskrivning blir det som väntat men lite långt och krångligt:

;DADDCRDAA;DAADDRDAAA;DAAADDCRDAAAA;DAAAADDR
DAAAAA;DAAAAADDCCRDAAAAAA;DAAAAAADDRDAAAAAA
A;DAAAAAAADDCCRDAAAAAAAA;DAAAAAAAADDRDA

Varför krångla till det på detta sättet? Anledningen är att på grund av Turingmaskinernas enkla arkitektur vill vi ha en beskrivning som representeras av så få symboler som möjligt samt är endimensionell så att den enkelt kan undersökas av en annan Turingmaskin. Du kanske redan misstänker vad som kommer, att det är detta den universella Turingmaskinen ska ”analysera”. Mer om det senare, just nu använder vi standardbeskrivningen i beviset av sats 1.

3.2.2 Beskrivningstalet av en Turingmaskin

Sats 1. *Mängden av alla Turingmaskiner är uppräknelig.*

Bevis. Vi behöver visa att det finns en bijektion mellan alla standardbeskrivningar och en delmängd av de naturliga talen. I kodningen av standardbeskrivningen av en Turingmaskin gör vi substitutionen $A = 1$, $C = 2$, $D = 3$, $L = 4$, $R = 5$, $N = 6$ och $; = 7$. Om vi gör denna substitution för alla standardbeskrivningar får vi uppenbart en mängd av naturliga tal i vilken varje tal beskriver en Turingmaskin. Självklart har vi en bijektion mellan de naturliga talen och en delmängd av de naturliga talen och alltså finns det uppräkneligt många Turingmaskiner. □

Det naturliga talet vi konstruerar i beviset av sats 1 kallar Turing för *beskrivningstalet* av en Turingmaskin. Beskrivningstalet av maskinen i exempel 1 blir:

```
731332531173113353111731113325311117311113353111117311111332
531111117311111133531111111731111111332253111111117311111111
3531
```

Ett stort tal som vi inte kommer se något mer av i denna text. Mängden av alla dessa beskrivningstal är en uppräkning av alla cirkelfria Turingmaskiner, och vi har nu alltså en uppräkning av alla beräkningsbara sekvenser, något vi kommer att använda oss av när vi applicerar diagonalprocessen på de beräkningsbara talen senare.

En intressant sak att lägga märke till är att vi nu har ett sätt att med heltal representera alla beräkningsbara tal och med dessa inkluderas många irrationella tal. Som vi vet finns det ändliga algoritmer för att räkna ut decimalerna i π, och alltså kan vi skapa en Turingmaskin som beräknar π. Vi kan använda beskrivningstalet till denna maskin för att representera π på ett exakt sätt. Om vi har ett naturligt tal som vi vet är beskrivningstalet till en Turingmaskin som beräknar π har vi också ett sätt att beräkna π.

3.2.3 Kodning av fullständiga konfigurationer

På samma sätt ska vi koda våra fullständiga konfigurationer som vi använde oss av tidigare. De fullständiga konfigurationerna gav information om vilka symboler som var på remsan, samt vilken ruta som blir läst och vilket läge maskinen befinner sig i. Denna notation är inte endimensionell vilket ger oss problem om vi ska koda den, men gör de fullständiga konfigurationerna enklare att läsa vid exempel. Vi ska göra en liten manipulation av de fullständiga konfigurationerna genom att, istället för att ange läget innan remsan och sedan stryka under den lästa symbolen, ska vi skriva det aktuella läget precis innan den lästa symbolen. På samma sätt som för standardbeskrivningarna ska vi använda oss av ”D” i kombination med ”A” för att representera lägen och ”D” i kombination med ”C” för symboler. Observera att blanka rutor i slutet av remsan inte skrivs ut i de fullständiga konfigurationerna eftersom de är oändligt många. Om vi följer denna kodning blir sekvensen av de fem första fullständiga konfigurationerna i exempel 1 (med kolon framför varje fullständig konfiguration för att separera dem) som följer:

```
:DA:DCDAA:DCDDAAA:DCDDCDAAAA:DCDDCDDAAAAA
```

3.3 Den universella maskinen

Med allt förberedelsearbete bakom oss kan vi nu börja titta på hur vi kan göra en UTM, som vi ska kalla $\mathcal{U}$. Vi vill att $\mathcal{U}$ ska kunna programmeras

med en godtycklig cirkelfri TM, $\mathcal{M}$. För att representera $\mathcal{M}$ har vi standardbeskrivningen, och denna ger vi till $\mathcal{U}$ genom att skriva den på F-rutorna direkt efter två ə-symboler (den första på en F-ruta och den andra på en E-ruta) och låter detta följas av en symbol "::". Alltså är remsan inte tom när $\mathcal{U}$ börjar sin beräkning, utan innehåller en beskrivning av $\mathcal{M}$ på remsan. Standardbeskrivningen kan ses som en uppsättning instruktioner till $\mathcal{U}$, vilka visar vad $\mathcal{U}$ ska göra i varje steg. $\mathcal{U}$ har sedan som huvuduppgift att beräkna alla $\mathcal{M}$:s fullständiga konfigurationer, för att sedan beräkna det tal som $\mathcal{M}$ beräknar.

Vid en första anblick av problemet kan detta kanske tyckas som en svår uppgift, men kom ihåg att alla Turingmaskiner börjar i q_1 med en tom remsa vilket innebär att vår första fullständiga konfiguration alltid kommer att vara DA. Från en fullständig konfiguration till nästa är det ett litet steg. Det vi behöver göra är att i en fullständig konfiguration hitta ett "D" följt av ett eller flera "A" tillsammans med den efterföljande "D"/"C" kombinationen, detta vet vi är en konfiguration som vi kan hitta i standardbeskrivningen av $\mathcal{M}$. När vi har hittat rätt 5-tuppel i standardbeskrivningen vet vi vad som ska göras för att konstruera nästa fullständiga konfiguration. På detta sätt, som vi snart ska se, skriver $\mathcal{U}$ ut en sekvens av fullständiga konfigurationer med tillägget att mellan varje fullständiga konfiguration skrivs de symboler av första typen (1 eller 0) som finns med i den nya (senare) fullständiga konfigurationen men inte i den gamla (tidigare). Vi ser att det som behöver göras är att söka, jämföra och kopiera olika kombinationer av ett ändligt antal tecken, vilket inte blir allt för svårt när vi redan har vårt bibliotek av m-fuktioner.

Vi ska lägga till en sista m-funktion till vårt bibliotek nu när vi vet hur vi ska koda våra instruktioner och fullständiga konfigurationer. Denna är $markKonf(A, \alpha)$ och markerar den konfiguration närmast till höger om läshuvudet med symbolen α och går sedan till A, läshuvudet slutar fyra steg till höger om sista rutan i den aktuella konfigurationen. Om vi anropar denna funktion utan en andra parameter, $markKonf(A,)$, innebär det av vi inte gör någon markering utan bara en förflyttning. I specialfallet när vi bara har ett läge kodat till höger om läshuvudet och inte någon läst symbol efter detta läge, kommer $markKonf$ att skriva ett D efter läget, markera detta och sedan flytta tre steg till höger. Detta görs för att utvidga remsan. Vi har inget sätt att representera oändligt många på varandra följande tomma rutor, men på detta sätt lägger vi till tomma rutor (D) allt eftersom det behövs, och det räcker.

3.4 Övergångstabell för $\mathcal{U}$

För att få en överblick av vad den universella Turingmaskinen gör ska vi nu successivt gå igenom övergångstabellen för denna med en standardbeskrivning av en enkel TM, $\mathcal{M}$. De symboler den universella Turingmaskinen, $\mathcal{U}$,

ska skriva är ”A”, ”C”, ”D”, ”:” för att representera de fullständiga konfigurationerna, ”u”, ”v”, ”w”, ”x”, ”y” och ”z” för markeringar samt ”0” och ”1” för att skriva resultatet. $\mathcal{U}$ måste även kunna läsa ”L”, ”R”, ”N” och ”;” i de kodade standardbeskivningarna, samt ett ”::” för att separera standardbeskrivningen, som kommer att finnas på remsan, från det som $\mathcal{U}$ kommer att skriva med hjälp av detta.

När maskinen börjar arbeta är maskinens remsa inte tom utan innehåller en standardbeskrivning av en annan maskin, denna standardbeskrivning finns på de första F-rutorna efter de två ə-symbolerna. Remsan börjar som vanligt med två ə-symboler, och ett ”::” avgränsar det som finns på remsan från start från det som beräknas efter det att maskinen börjat arbeta. När maskinen börjar i läge b kommer maskinens fullständiga konfiguration alltså i detta fall att se ut på följande sätt:

b: <u>ə</u>ə; D A D D C R D A A ; D A A D D C C R D A ::

Till höger om ”::”-symbolen kommer $\mathcal{U}$ att skriva de fullständiga konfigurationerna för varje steg som $\mathcal{M}$ skulle gjort. Eftersom vi vet att $\mathcal{M}$ börjar på läge q_1, som representeras av DA, och vi också vet att remsan är tom när $\mathcal{M}$ börjar, kommer $\mathcal{M}$:s första fullständiga konfiguration att vara DA (oberoende av $\mathcal{M}$, detta är lika för alla maskiner). Vi låter varje fullständig konfiguration som $\mathcal{U}$ skriver börja på ”:”. $\mathcal{U}$:s uppgift är nu till att börja med att skriva ut denna fullständiga konfiguration. Början av övergångstabellen för $\mathcal{U}$ ser ut som följer:

Konfiguration		Beteende	
b	allt		$hitta(b_1, b_1, ::)$
b_1	allt	R2, P:, R2, PD, R2, PA	*start*

b ställer läshuvudet över den första rutan som innehåller ”::” med hjälp av *hitta* och övergår sedan till b_1. I vårt fall innebär det att vi ställer läshuvudet i början av vår representation av de fullständiga konfigurationerna av $\mathcal{M}$. b_1 skriver sedan :DA på de tre följande F-rutorna och går sedan över till läget som heter *start*. Efter denna initiering kommer $\mathcal{U}$:s fullständiga konfiguration att vara:

start: əə; D A D D C R D A A ; D A A D D C C R D A :: : D <u>A</u>

3.4.1 Markera den aktuella konfigurationen

Nästa del av övergångstabellen, *start*, är:

Konfiguration		Beteende
start	allt	$hittaSista(start_1, :)$
$start_1$	allt	$markKonf(hittaKonf, y)$

start kommer här att ställa läshuvudet på det ":" som är längst till höger på remsan med hjälp av *hittaSista*, vilket är det kolon som föregår den senast skrivna fullständiga konfigurationen. Denna fullständiga konfiguration, den längst till höger på remsan, kommer vi att kalla den aktuella fullständiga konfigurationen eftersom denna motsvarar $\mathcal{M}$:s fullständiga konfiguration i varje läge. På samma sätt kommer vi kalla konfigurationen för den aktuella fullständiga konfigurationen för den aktuella konfigurationen. $start_1$ markerar konfigurationen till höger om läshuvudet. Eftersom vår fullständiga konfiguration bara är DA nu, kommer specialfallet för $markKonf$ att användas här, vilket innebär att vi även kommer att skriva ett D efter DA. Den fullständiga konfigurationen för $\mathcal{U}$ är nu:

$hittaKonf$: əə; D A D D C R D A A ; D A A D D C C R D
A :: : DyAyDy _

3.4.2 Matcha den aktuella konfigurationen

Nu är den aktuella konfiguration markerad. Nu behöver vi veta vad som ska göras vid denna konfiguration, eller mer precist vad $\mathcal{M}$ skulle gjort vid denna konfiguration. Detta kan vi hitta efter motsvarande konfiguration i standardbeskrivningen av $\mathcal{M}$ som finns på remsan. De följande lägena letar reda på denna konfiguration i standardbeskrivningen:

Konfiguration		Beteende	
	;	R, Pz, L	$markKonf(jamforKonf, x)$
$hittaKonf$	z	L2	$hittaKonf$
	inte z/;	L	$hittaKonf$
$jamforKonf$	allt	$jamT(tabort(tabort(start, x), y), lika, x, y)$	

$hittaKonf$ letar reda på det semikolon närmast till vänster om läshuvudet som inte är markerat med ett "z" och markerar detta med ett "z" samt markerar den konfiguration som är efter detta semikolon med "x". När denna konfiguration är markerad använder sig $jamforKonf$ av $jamforTabort$ (som är förkortat till $jamT$ i denna tabell) för att jämföra det på remsan som

är markerat med ”x” med det på remsan som är markerat med ”y”, alltså våra två markerade konfigurationer. Om dessa är lika har vi hittat rätt konfiguration i standardbeskrivningen och övergår till läget *lika* efter att x- och y-markeringarna tagits bort. Om de är olika måste vi leta vidare efter rätt konfiguration, då tar vi bort x- och y-markeringarna på våra konfigurationer med hjälp av *tabort* och börjar om i läget *start*. Lägg märke till att de konfigurationer vi redan har testat kommer att vara markerade med ”z” och kommer att hoppas över i nästa jämförelse. I vårt fall kommer första jämförelsen att misslyckas, men andra kommer att lyckas, så när vi övergår till *lika* kommer $\mathcal{U}$:s fullständiga konfiguration att vara:

lika: əə;zD A D D C R D A A ;zD A A D D C C R D A :: :
D A <u>D</u>

3.4.3 Markera operationen

Nu vet vi att den konfiguration efter det semikolon som är märkt med ”z” och är längst till vänster är lika med den aktuella konfiguration. Vi ska nu ta reda på vad som ska göras för denna konfiguration, vi ska markera operationen (det som ska skrivas och hur maskinen ska ta ett steg) som ska göras för denna konfiguration med ”u” och det läge vi sedan ska övergå till med ”y”. Detta görs med följande lägen:

Konfiguration			Beteende
lika	allt		$hittaV(lika_1, lika_1, \text{z})$
$lika_1$	allt		$markKonf(lika_2,)$
$lika_2$	A		$lika_3$
	inte A	L, Pu, R3	$lika_2$
$lika_3$	A	L, Py, R3	$lika_3$
	inte A	L, Py	$tabort(markFKonf, \text{z})$

I *lika* kommer först $HittaV$ att ställa läshuvudet på det första semikolonet markerat med ”z”, sedan kommer $lika_1$ att ställa läshuvudet fyra steg till höger om konfigurationen som följer efter detta semikolon. Detta kommer i vårt fall att resultera i att det C som är längst till vänster på remsan kommer att vara under läshuvudet. Från denna position kommer det som representerar den symbol som $\mathcal{M}$ skulle skrivit för den aktuella konfigurationen (DC) samt den symbol som representerar hur maskinen ska stega att markeras med ”u”, detta tar $lika_2$ hand om. När detta är klart tar $lika_3$ över och markerar det läge som $\mathcal{M}$ skulle ha övergått till från den aktuella konfigurationen (DAA) med ”y”. När $lika_3$ är klar raderas alla z-markeringar och maskinen

övergår till $markFKonf$. Den fullständiga konfigurationen när detta är gjort kommer att vara:

$markFKonf$: əə; D A D DuCuRuDyAyAy;_D A A D D C C R D A :: : D A D

3.4.4 Markera den fullständiga konfigurationen

$markFkonf$ står för markera-Fullständig-Konfiguration, och markerar den aktuella fullständiga konfigurationen på remsan. Det som markeras är det som ska kopieras över till nästa fullständiga konfiguration, det som är konstant, vilket är det som representerar symbolerna på remsan, förutom den lästa symbolen. Lägena som hanterar denna markering är:

Konfiguration			Beteende
$markFKonf$	allt		$hittaSista(markFKonf_1, :)$
$markFKonf_1$	A	L4	$markFKonf_2$
	inte A	R2	$markFKonf_1$
$markFKonf_2$	C	R, Px, L3	$markFKonf_2$
	:		$markFKonf_4$
	D	R, Px, L3	$markFKonf_3$
$markFKonf_3$	:		$markFKonf_4$
	inte :	R, Pv, L3	$markFKonf_3$
$markFKonf_4$	allt		$markKonf(V(V(markFKonf_5)),)$
$markFKonf_5$	något	R, Pw, R	$markFKonf_5$
	blank	P:	$skrivResultat$

För den fullständiga konfigurationen vi har nu så kommer ingenting att markeras, utan det enda som kommer att hända är att det skrivs ett kolon efter denna fullständiga konfiguration. Det krävs en något längre fullständig konfiguration, en där det finns symboler både före och efter den aktuella konfigurationen. Låt oss anta att maskinen har gjort några andra beräkningar så att den aktuella fullständiga konfigurationen på remsan ser ut så här:

: D C D C C D A A D D C C

I detta fall kommer den markering som $markFKonf$ gör att bli:

: DvCvDxCxCxD A A D DwCwCw:

Det som representerar symbolen i rutan innan rutan som blir läst markeras med ”x”, allt innan denna med ”v” och de symboler efter den aktuella konfigurationen markeras med ”w”. Allt som är markerat kommer att finnas med i

nästa fullständiga konfiguration som skrivs på remsan. Läget och den lästa symbolen kommer att ändras till nästa fullständiga konfiguration, och det som detta ska ändras till är det som är markerat med ”u” och ”y” sedan tidigare. Anledningen till att det som är direkt innan läget är markerat med ”x” och inte ”v” är att om $\mathcal{M}$ flyttar vänster i den aktuella konfigurationen så måste det nya läget (det läge maskinen övergår till) att stå framför det som är markerat med ”x”. Om maskinen i stället går till höger behöver vi bara byta ordning på den lästa symbolen och konfigurationen, så att den första symbol som är markerad med ”w” är den nya lästa symbolen. Innan nästa fullständiga konfiguration börjar skrivas till remsan måste $\mathcal{U}$ skriva ut en ”1” eller en ”0” på remsan om $\mathcal{M}$ skulle gjort det för den aktuella konfigurationen. Kom ihåg att vi ibland skriver en 1:a eller 0:a över en annan 1:a eller 0:a i stället för att inte skriva någonting, i dessa fall ska $\mathcal{U}$ inte skriva något på remsan. Det är alltså bara när $\mathcal{M}$ har läst en tom ruta som $\mathcal{U}$ ska skriva något i detta steg.

3.4.5 Skriv resultatet

Följande tabell skriver ut $\mathcal{U}$:s resultat av den aktuella fullständiga konfigurationen:

Konfiguration		Beteende	
$skrivResultat$	allt		$hitta(skrivResultat_1, skrivFKonf, \text{u})$
$skrivResultat_1$	allt	L3	$skrivResultat_2$
$skrivResultat_2$	D	R4	$skrivResultat_3$
	inte D		$skrivFKonf$
$skrivResultat_3$	C	R2	$skrivResultat_4$
	inte C		$skrivFKonf$
$skrivResultat_4$	C	R2	$skrivResultat_5$
	inte C		$skriv_2(skrivFKonf, 0, :)$
$skrivResultat_5$	C		$skrivFKonf$
	inte C		$skriv_2(skrivFKonf, 1, :)$

De två första lägena här placerar läshuvudet på den sista symbolen i det som representerar den lästa symbolen vid den aktuella konfigurationen i $\mathcal{M}$:s beskrivning. Om denna symbol är D innebär det att det är en blank ruta som har blivit läst, då måste vi gå vidare med detta steg, för då vill vi skriva en 1:a eller en 0:a. Om det inte är ett D vi läser här så hoppar vi direkt till $skrivFKonf$ eftersom vi då inte ska skriva ut någon 1:a eller 0:a på remsan. Lägena $skrivResultat_3$, $skrivResultat_4$ och $skrivResultat_5$ tar reda på om det är en 1:a (DCC) eller en 0:a (DC) som ska skrivas. Korrekt symbol skrivs

sedan till remsan följt av ett kolon, och $\mathcal{U}$ övergår till $skrivFKonf$. I vårt fall hittar $skrivResultat$ en kombination DC som representerar 0 vilket ska skrivas på remsan. Den fullständiga konfigurationen av $\mathcal{U}$ efter dessa steg blir:

$skrivFKonf$: əə; D A D DuCuRuDyAyAy; D A A D D C C R D A :: : D A D : 0 :

3.4.6 Skriv nästa fullständiga konfiguration

Det enda som återstår nu är att skriva nästa fullständiga konfiguration till remsan och sedan börja om från början. I nästa genomgång av tabellen kommer den nya fullständiga konfigurationen undersökas, om det är aktuellt kommer en 1:a eller en 0:a skrivas efter den fullständiga konfigurationen, och sedan kommer ytterligare en fullständig konfiguration att skrivas på remsan. Denna loop är allt den universella Turingmaskinen gör. För att fullborda detta visar vi de allra sista lägena för $\mathcal{U}$:

Konfiguration		Beteende	
$skrivFKonf$	allt		hittaSista(L($skrivFKonf_1$), u)
$skrivFKonf_1$	L	R, E	kopieraMarkeradTabort(slut, v, y, x, u, w)
	R	R, E	kopieraMarkeradTabort(slut, v, x, u, y, w)
	N	R, E	kopieraMarkeradTabort(slut, v, x, y, u, w)
$slut$	allt		tabort(start)

$skrivFKonf$ (förkortning av skriv-Fullständig-Konfiguration) sätter läshuvudet över den sista symbolen som är markerad med ”u”, detta är symbolen som visar om $\mathcal{M}$ ska ta ett steg och i så fall också hur och kommer att vara ”R”, ”L” eller ”N”. Beroende på denna symbol kommer sekvenserna markerade med ”u”, ”y”, ”v”, ”x” och ”w” att kopieras till slutet av remsan i lite olika ordning. Alla sekvenserna som vi har markerat med dessa markeringar är väldefinierade delar av den nya fullständiga konfiguration vi är på väg att skriva på remsan. Den sekvens markerad med ”u” representerar den symbol som $\mathcal{M}$ ska skriva i den aktuella rutan. Den sekvens markerad med ”y” representerar det läge som $\mathcal{M}$ ska övergå till, detta ska stå framför den symbol som är den lästa i den nya fullständiga konfigurationen. Den sekvens markerad med ”v” är det som representerar den sekvens av symboler från början av remsan fram till symbolen innan det aktuella läget. Det som är markerat med ”x” är symbolen precis innan det aktuella läget. Den sekvens av symboler markerade med ”w” är de symboler som representerar de symboler på remsan efter den lästa symbolen. I den nya fullständiga konfigurationen kommer alltid det som är markerat med ”v” först och det markerat med ”w” sist. Ordningen på ”u”, ”y” och ”x” däremellan är beroende av hur läshuvudet

ska röra sig i $\mathcal{M}$. $skrivFKonf_1$ tar hand om kopieringen av de markerade sekvenserna, men innan den kopierar något så tas markeringen bort för den ruta som läshuvudet står på så att inte något ”L”, ”R” eller ”N” följer med i kopieringen. Om $skrivFKonf_1$ läser ”N” innebär det att $\mathcal{M}$ inte tar något steg, då ska de markerade sekvenserna kopieras utan att ändra inbördes ordning. Om $skrivFKonf_1$ läser ”L” innebär det att $\mathcal{M}$ flyttar vänster, då ska det nya läget (markerat med y) vara placerat en ruta till vänster om den aktuella rutan, alltså ska då det som är markerat med ”y” kopieras före det som är markerat med ”x”, annars samma ordning som för ”N”. Om $skrivFKonf_1$ läser ”R” innebär det att $\mathcal{M}$ flyttar höger, då ska det nya läget(sekvensen markerad med ”y”) och den symbol som ska skrivas i den aktuella rutan (sekvensen markerad med ”u”) byta plats i förhållande till varandra, i övrigt ska delarna ha samma inbördes ordning. Efter att de markerade sekvenserna har kopierats i rätt ordning till slutet av remsan så övergår $\mathcal{U}$ till läget *slut* som raderar alla markeringar och sedan börjar om på *start* igen. $\mathcal{U}$:s fullständiga konfiguration efter hela första loopen kommer att vara:

start: əə; D A D D C R D A A ; D A A D D C C R D A :: : D A D : 0 : D C D A A _

Efter hela andra loopen kommer $\mathcal{U}$:s fullständiga konfiguration att vara:

start: əə; D A D D C R D A A ; D A A D D C C R D A :: : D A D : 0 : D C D A A : 1 : D C D C C D A _

Detta är i princip samma läge som innan första loopen, och om vi fortsätter ser vi enkelt att det mellan varje fullständig konfiguration som $\mathcal{U}$ skriver kommer att vara en sekvens av alternerande 0:or och 1:or. Detta är det vi läser av som $\mathcal{U}$:s beräkning, den sekvens $\mathcal{U}$ beräknar med maskinen $\mathcal{M}$ kodad på remsan är 010101 . . .

Nu har vi definierat den universella Turingmaskinen, en maskin som kan simulera alla de cirkelfria Turingmaskiner som vi kan tänka oss konstruera med teorin vi här har byggt upp. Detta är en stor bedrift av Turing och en viktig insikt för utvecklingen av datorer och modern datorvetenskap, att det är onödigt att designa olika datorer för olika ändamål: Det räcker att programmera en och samma dator att utföra alla specifika uppgifter. Detta insåg Turing var de universella maskinernas styrka. Detta uttrycker Turing tydligt i sin kontroversiella artikel ”Computing machinery and intelligence” från 1950. Vi avslutar detta avsnitt med ett citat från denna text:

> This special property of digital computers, that they can mimic any discrete-state machine, is described by saying that they are universal machines. The existence of machines with this property has the important consequence that, considerations of speed

> apart, it is unnecessary to design various new machines to do various computing processes. They can all be done with one digital computer, suitably programmed for each case. It will be seen that as a consequence of this all digital computers are in a sense equivalent.

4 Avgörbarhetsproblemet

I detta avsnitt ska vi se på mer teoretiska egenskaper hos Turingmaskiner och genom detta se vilka begränsningar de har och även lösa avgörbarhetsproblemet. Vi vet redan att de beräkningsbara talen är uppräkneliga och detta ska vi använda för att dra slutsatser kring Turingmaskiner, främst genom att applicera Georg Cantors diagonalprocess (från 1891) på de beräkningsbara talen.

4.1 Diagonalprocessen

Diagonalprocessen är en metod som den tyske matematikern Georg Cantor använde för att visa att det finns överuppräkneliga mängder, mängder som inte går att sätta i ett 1-1 förhållande med de naturliga talen. Denna process kan användas för att visa att de reella talen är överuppräkneliga, men har använts av många till mycket, och som sagt också av Turing i sitt resonemang om de beräkningsbara talen som vi snart ska titta noggrannare på. Vi börjar med att först göra oss bekväma med metoden genom att se hur processen kan appliceras på en mängd reella tal.

4.1.1 Diagonalprocessen för de reella talen

Sats 2. *Mängden av alla reella tal mellan 0 och 1 är överuppräknelig.*

Bevis. Antag att vi har en uppräkning av alla reella tal mellan 0 och 1, där varje tal skrivs med oändligt många nollskilda siffror i sin decimalutveckling. Exempelvis skrivs $0{,}10000\ldots$ som $0{,}99999\ldots$ och båda dessa får inte förekomma i uppräkningen. Låt α_n vara det n:e talet i uppräkningen och $\phi_n(m)$ vara den m:e decimalen i α_n. Vi kan nu konstruera ett tal β genom att låta $\beta(m) = \alpha_m(m) - 1 \pmod{10}$ (vi låter -1 vara 9). Detta tal kommer att vara skillt från alla de reella talen i uppräkningen, β är ”diagonalen” i uppräkningen, men är självt uppenbart ett reellt tal. Alltså finns det ett reellt tal som inte finns med i uppräkningen av alla de reella talen mellan 0 och 1, vilket motsäger vårt antagande. Detta innebär att vi inte kan hitta en uppräkning av alla de reella talen och alltså är denna mängd överuppräknelig. □

4.1.2 Diagonalprocessen för de beräkningsbara talen

Vad händer om vi applicerar samma resonemang på de beräkningsbara talen, kan vi på samma sätt skapa ett beräkningsbart tal utifrån en uppräkning av alla beräkningsbara tal?

Låt oss applicera diagonalprocessen på våra beräkningsbara sekvenser. Låt α_n vara den n:e sekvensen i uppräkningen (som vi diskuterade i avsnitt 3.2) av alla beräkningsbara sekvenser och låt $\phi_n(m)$ vara den m:e symbolen (1 eller 0) i den n:e sekvensen. Då kan vi skapa en sekvens där den n:e symbolen i β är $1-\phi_n(n)$. Antag att β är en beräkningsbar sekvens. Då måste $\beta = \alpha_N$ för något N, eller med andra ord att $1 - \phi_n(n) = \phi_N(n)$ gäller för alla n. Om vi sätter $n = N$ får vi att $1-\phi_N(N) = \phi_N(N)$ eller $1 = 2\phi_N(N)$. Detta är omöjligt, 1 är inte ett jämt tal, vi har kommit fram till en motsägelse som visar att någon av våra premisser är fel. Eftersom vi sedan tidigare vet att mängden av alla beräkningsbara sekvenser är uppräknelig kan vi inte dra slutsatsen att denna mängd är överuppräknelig. Det är något annat som är fel i vårt resonemang, och det som är fel måste vara vårt antagande att β är beräkningsbart.

4.1.3 Varför β inte är beräkningsbart

Varför är inte β beräkningsbart? Vi har ju givit en algoritm för att beräkna β. Kom ihåg att vi vet att de cirkelfria maskinerna som definierar de beräkningsbara talen är uppräkneligt många. Vi har däremot inte gjort någon explicit uppräkning av dem, och det är här problemet med att beräkna β finns. För att göra denna uppräkning måste vi först för varje naturligt tal avgöra om det är ett beskrivningstal för en cirkelfri Turingmaskin, vilket vi ska visa med ett direkt argument, inte går att göra med en ändlig algoritm och alltså inte kan göras av en Turingmaskin.

Det vi måste göra för att lyckas skapa ett nytt tal på samma sätt som β som är beräkningsbart, är att konstruera en Turingmaskin som analyserar beskrivningstalet av en annan Turingmaskin (inte nödvändigtvis cirkelfri) och bestämmer denna maskins resultat, och därmed om den är cirkelfri eller inte. För att visa att β inte är beräkningsbart ska vi försöka konstruera en maskin $\mathcal{H}$ som beräknar detta tal.

Antag att vi har en maskin $\mathcal{D}$ som avgör om ett givet naturligt tal är ett beskrivningstal för en cirkelfri TM eller inte och som gör detta i ett ändligt antal steg. Det $\mathcal{H}$ kommer att göra är att för varje naturligt tal N (med början på 0) med hjälp av maskinen $\mathcal{D}$ bestämma om detta tal är beskrivningstalet till en cirkelfri TM. Om så är fallet låter vi $R(N)$ (ett tal $\mathcal{H}$ håller reda på), antalet beskrivningstal som representerar cirkelfria maskiner av de N första talen, öka med 1. Sedan gör vi om beskrivningstalet till en standardbeskrivning och låter $\mathcal{U}$ (den universella Turingmaskinen) beräkna denna maskins sekvens upp till den $R(N)$:e siffran, säg α. Ett minus

α lägger vi till i sekvensen som är β, det vill säga den $R(N)$:e symbolen i β är $1 - \alpha$ där α är den $R(N)$:e siffran som $\mathcal{U}$ beräknar för det $R(N)$:e beskrivningstalet. Sedan gör vi om samma sak med nästa tal, $N + 1$. Om N inte är ett beskrivningstal för en cirkelfri TM behöver vi inte göra några vidare beräkningar utan går direkt vidare för att testa $N + 1$ med $\mathcal{D}$. $\mathcal{H}$ gör för varje naturligt tal en loop på detta sätt. $\mathcal{H}$ är en cirkelfri maskin eftersom både $\mathcal{D}$ och $\mathcal{U}$ gör sina beräkningar i ett ändligt antal steg och kommer fram till ett korrekt resultat efter ett ändligt antal steg, resten av det $\mathcal{H}$ gör är triviala beräkningar som också görs i ett ändligt antal steg. Lägg märke till att kontrollen $\mathcal{D}$ gör av beskrivningstalen garanterar att $\mathcal{U}$ gör en cirkelfri beräkning.

Eftersom $\mathcal{H}$ är en Turingmaskin har $\mathcal{H}$ ett beskrivningstal associerat med sig, låt detta beskrivningstal vara K. Efter $K - 1$ loopar måste $\mathcal{H}$ hantera K, beskrivningstalet av sig själv. Eftersom $\mathcal{H}$ är en cirkelfri maskin måste $\mathcal{D}$ godkänna K som beskrivningstalet för en cirkelfri maskin. Alltså kommer $\mathcal{U}$ att ta hand om standardbeskrivningen som K representerar för att beräkna den $R(K)$:e symbolen som maskinen som K representerar beräknar. Men denna maskin är ju $\mathcal{H}$, och den sekvens $\mathcal{H}$ beräknar är β. De första $R(K-1)$ symbolerna kommer gå bra att beräkna, det har vi redan gjort, men när den $R(K)$:e symbolen måste beräknas av $\mathcal{U}$ måste vi först beräkna de $R(K-1)$ första symbolerna och sedan den $R(K)$:e. Med andra ord, för att räkna ut den $R(K)$:e symbolen i β måste vi känna till den $R(K)$:e symbolen i β, vilket är en paradox, vi hamnar i en oändlig rekursion. Detta innebär att när $\mathcal{H}$ skall beräkna den $R(K)$:e symbolen i β kommer $\mathcal{H}$ inte skriva ut något resultat utan forsätta i oändlighet med en loop som inte går att komma ut ur, alltså är $\mathcal{H}$ cirkulär. $\mathcal{H}$ är både cirkelfri och cirkulär samtidigt vilket är en uppenbar motsägelse, alltså drar vi slutsatsen att vårt antagande om att det existerar en maskin $\mathcal{D}$ är felaktigt. Vi har på ett nytt sätt visat att β inte kan vara beräkningsbart och alltså att de beräkningsbara talen (fortfarande) är uppräkneliga. Dock kan vi inte göra någon uppräkning av dem i ett ändligt antal steg.

4.2 Maskinen $\mathcal{E}$

Med hjälp av ovanstående resonemang kan vi också visa att det inte finns en maskin $\mathcal{E}$ som tar reda på om en godtycklig Turingmaskin $\mathcal{M}$ någonsin skriver en viss symbol i sitt resultat. Antag att det finns en sådan maskin $\mathcal{E}$ och låt $\mathcal{M}_1$ vara en maskin som beräknar samma sekvens som $\mathcal{M}$ fast med den första 0:an utbytt mot $\overline{0}$, som är en symbol i $\mathcal{M}_1$:s alfabet utöver de symboler i $\mathcal{M}$:s alfabet. Mer generellt låter vi $\mathcal{M}_n$ vara en maskin som skriver samma sekvens som $\mathcal{M}$ fast med de n första 0:orna utbytta mot $\overline{0}$-symboler. Vi kan enkelt ändra standardbeskrivningen av $\mathcal{M}$ till den för $\mathcal{M}_1$, vi behöver bara lägga till ett läge som ser ut som det läge där 0 skrivs första gången men istället för att skriva 0 skriver $\overline{0}$, samt se till att detta läge bara används i det

fall då 0:an som skulle skrivits var den första. Vi kan självklart på samma enkla sätt ändra standardbeskrivningen av $\mathcal{M}_{n-1}$ till att vara $\mathcal{M}_n$ för alla n. Alltså kan vi skapa en maskin $\mathcal{F}$ som för en standardbeskrivning av $\mathcal{M}$ successivt skriver ner standardbeskrivningarna till $\mathcal{M}, \mathcal{M}_1, \mathcal{M}_2, \ldots$ på remsan. Vi kan nu skapa ytterligare en maskin $\mathcal{G}$. $\mathcal{G}$ använder $\mathcal{F}$ för att skriva ner standardbeskrivnignarna av en maskin $\mathcal{M}$ och maskinerna $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \ldots$ associerade med denna. Efter varje standardbeskrivning skriver $\mathcal{G}$:0: om det är så att maskinen associerad med denna standardbeskrivning aldrig skriver 0, och detta bestäms med hjälp av $\mathcal{E}$. På detta sätt kommer $\mathcal{G}$ att skriva 0 oändligt många gånger om $\mathcal{M}$ bara skriver 0 ändligt många gånger, och $\mathcal{G}$ kommer inte att skriva några 0:or alls om $\mathcal{M}$ skriver oändligt många 0:or. Vi kan nu testa $\mathcal{G}$, för en standardbeskrivning av en godtycklig maskin $\mathcal{M}$, med $\mathcal{E}$. Om $\mathcal{E}$ ger resultatet att $\mathcal{G}$ aldrig skriver en 0:a vet vi att $\mathcal{M}$ skriver 0 oändligt ofta. På samma sätt kan vi skapa maskiner som avgör om 1 (eller någon annan godtycklig symbol) skrivs oändligt ofta av en maskin. Om vi skulle ha en maskin $\mathcal{E}$ skulle vi enkelt kunna ta reda på om ett naturligt tal är beskrivningstalet för en cirkelfri Turingmaskin, vilket vi redan har bevisat är omöjligt, alltså går det inte att konstruera någon maskin $\mathcal{E}$.

4.3 Lösningen av avgörbarhetsproblemet

Det är nu dags att ta itu med avgörbarhetsproblemet, vi skall visa att det inte finns någon generell metod för att bestämma om en godtycklig formel F i första ordningens predikatlogik går att bevisa.

För att lyckas med detta ska vi för varje TM $\mathcal{M}$ konstruera en formel i första ordningens predikatlogik, som vi kallar $Un(\mathcal{M})$. Sedan visar vi att om det finns en generell metod för att avgöra om $Un(\mathcal{M})$ är bevisbar så måste det också finnas en generell metod som bestämmer om $\mathcal{M}$ någonsin skriver 0 (en maskin $\mathcal{E}$), vilket vi har visat inte finns.

4.3.1 Konstruktion av formeln $Un(\mathcal{M})$

Vi börjar vår konstruktion av denna formel med att definiera några predikat som vi behöver. Vi låter rutorna på remsan till en Turingmaskin vara numrerade från 0 (den första, den maskinen läser först) och vidare uppåt. Vi låter även de fullständiga konfigurationerna vara numrerade med början på 0. Då kan vi identifiera ruta nummer m vid den fullständiga konfiguration n med paret (n, m). Predikaten och deras tolkning är som följer:

- $R_{S_l}(x, y)$ tolkas som ”I den fullständiga konfigurationen x är symbolen i ruta y S_l”.
- $I(x, y)$ tolkas som ”I den fullständiga konfigurationen x blir ruta y läst”.
- $K_{q_m}(x)$ tolkas som ”I den fullständiga konfigurationen x är det aktuella läget q_m”.

- $F(n, m)$ tolkas som ”m är succesorn till n”. Successorn till n är det minsta naturliga tal m som är större än n.
- F^n förkortar en konjunktion $F(0,1) \wedge F(1,2) \wedge \ldots \wedge F(n-1,n)$, vilket tolkas som att alla tal m mindre eller lika med n är den m:e successorn till 0.
- $G(n, m)$ tolkas som ”m är större än n”.

Med dessa predikat kan vi till varje rad i övergångstabellen av $\mathcal{M}$ associera en formel. Kom ihåg (se avsnitt 3.2) att vi med övergångstabellen av $\mathcal{M}$ på vår standardform kan beskriva varje rad av denna som en femtuppel (q_i, S_j, S_k, L, q_l) (eller med R eller N istället för L). Med varje sådan femtuppel ur en övergångstabell associerar vi en formell $Inst(q_i, S_j, S_k, L, q_l)$ som definieras som:

$$\forall x, y, x', y' \Bigg(\Big(R_{S_j}(x,y) \wedge I(x,y) \wedge K_{q_i}(x) \wedge F(x,x') \wedge F(y',y) \Big) \rightarrow \Bigg(I(x',y') \wedge R_{S_k}(x',y) \wedge K_{q_l}(x') \wedge \forall z \Big(F(y',z) \vee \big((R_{s_0}(x,z) \rightarrow R_{s_0}(x',z)) \wedge \ldots \wedge (R_{s_M}(x,z) \rightarrow R_{s_M}(x',z)) \big) \Big) \Bigg) \Bigg)$$

Denna formel beskriver en rad ur en övergångstabell på formen (q_i, S_j, S_k, L, q_l). x och y är den fullständiga konfigurationen respektive den lästa rutan innan vi övergår till nästa läge (q_l). x' och y' är den fullständiga konfigurationen respektive den lästa rutan efter övergången. Antecedenten i implikationen beskriver under vilken förutsättning vi ska använda denna konfiguration samt förhållandet mellan x och x' samt y och y'. För den aktuella fullständiga konfigurationen gäller att symbolen i ruta y är S_j (R_{S_j}), ruta y blir läst $(I(x,y))$ och det aktuella läget är q_i (K_{q_i}). De två F-predikaten säger att x' är nästa fullständiga konfiguration och att rutan y' ligger direkt till vänster om rutan y. Konsekventen i implikationen visar hur nästa fullständiga konfiguration ser ut, vad som händer när vi använder den aktuella konfigurationen. I nästa fullständiga konfiguration, x', blir ruta y' läst (den till vänster om y) och symbolen i ruta y har nu ändrats till S_k. Läget är nu q_l. Den sista konjunktionen, den som börjar med en all-kvantor, säger att för alla rutor z på remsan i denna fullständiga konfiguration så är antingen $z = y$ (och då har innehållet i den ändrats till S_k) eller så är innehållet i rutan oförändrat sedan föregående fullständiga konfiguration. S_1 till S_m i uttrycken vid ellipsen står för alla symboler i alfabetet till $\mathcal{M}$, och precis en av försatserna i de m stycken implikationerna kommer att vara sann för varje $z \neq y$. På samma sätt kan vi skapa formlerna $Inst(q_i, S_j, S_k, R, q_l)$ och $Inst(q_i, S_j, S_k, N, q_l)$ för de fall då $\mathcal{M}$ flyttar höger eller inte alls i den med

$Inst$ associerade raden i en övergångstabell. Dessa kan du se i appendix A, de är konstruerade enligt samma princip så vi tar inte med dem här. Nu kan vi beskriva varje rad i en övergångstabell med en $Inst$-formel och därmed också hela övergångstabellen för $\mathcal{M}$ med en konjunktion av alla $Inst$-formler associerade med raderna i övergångstabell för $\mathcal{M}$, denna formel kallar vi $Des(\mathcal{M})$. $Des(\mathcal{M})$ är sann för en Turingmaskin, inte nödvändigtvis cirkelfri, det är därför vi säger att $Des(\mathcal{M})$ beskriver en Turingmaskin $\mathcal{M}$.

För att konstruera vår formel $Un(\mathcal{M})$, som $Des(\mathcal{M})$ kommer att vara en del av, måste vi i vår formel säga något om de naturliga talen och deras egenskaper, detta eftersom de naturliga talen inte finns med i första ordningens predikatlogik. Jag kommer inte göra mer än att visa vad vi behöver i form av axiom för de naturliga talen, för en mer genomgående diskusion hänvisar jag till ”The annotated Turing” av Charles Petzold eller någon bok som tar upp Peanoaxiomen mer grundläggande. Jag kommer inte heller behandla de naturliga talen som Turing gjorde, utan istället anta att alla problem kring formalisering av de naturliga talen är lösta, och använda de naturliga talen precis som vi är vana vid, även i formlerna skrivna i predikatlogik, detta för att jag tror att resonemanget blir lättare att förstå trots att det inte blir fullständigt korrekt. Det vi behöver är följande, och vi förkortar denna formel med Q:

$$\forall x\,\exists w\,\forall y,z\left(F(x,w)\wedge\Big(F(x,y)\rightarrow G(x,y)\Big)\wedge\Big(F(x,Z)\wedge G(z,y)\rightarrow G(x,y)\Big)\right.$$
$$\left.\wedge\Big(G(z,x)\vee\Big(G(x,y)\wedge F(y,z)\Big)\vee\Big(F(x,y)\wedge F(z,y)\Big)\rightarrow\neg F(x,z)\Big)\right)$$

Vad denna formel säger är att det finns en entydig efterföljare till varje tal, det vill säga att det finns en successorfunktion.

För att beskriva en maskin $\mathcal{M}$ fullständigt behöver vi beskriva hur maskinen och remsan ser ut när den startar. Som vi vet är remsan tom (blank ruta representeras av S_0), maskinen läser ruta 0 och läget är q_1 när en godtycklig maskin börjar sin beräkning (bortsett från vår UTM). Detta kan vi beskriva med våra redan definierade predikat som $\forall y R_{S_0}(u,y)\wedge I(u,u)\wedge K_{q_1}(u)$, där u är en obunden variabel som självklart ska tolkas som 0, den första symbolen i uppräkningen av de naturliga talen. Om vi kopplar ihop det vi nu gjort får vi ett uttryck som beskriver en godtycklig Turingmaskin $\mathcal{M}$ och dess läge och remsa vid start, samt säger något om de naturliga talens egenskaper. Vi förkortar detta uttryck med $A(\mathcal{M})$, och i sin helhet blir det:

$$Q\wedge\forall y\,R_{S_0}(u,y)\wedge I(u,u)\wedge K_{q_1}(u)\wedge Des(\mathcal{M})$$

Nu är vi nästan klara med konstruktionen av $Un(\mathcal{M})$. Det vi vill uttrycka är att om det existerar ett u så att $A(\mathcal{M})$ är sann så finns det en fullständig

konfiguration s till $\mathcal{M}$ där symbolen 0 (som representeras med S_1) är skriven i ruta t. Eller annorlunda uttryckt, om det finns en maskin $\mathcal{M}$ som beskrivs av $A(\mathcal{M})$ så kan vi bestämma om denna maskin någonsin skriver en 0:a, vilket vi redan har visat är omöjligt. $Un(\mathcal{M})$ ska alltså vara:

$$\exists u\, A(\mathcal{M}) \rightarrow \exists s, t\, R_{S_1}(s, t)$$

Nu är den stora frågan om vi kan avgöra om $Un(\mathcal{M})$ är bevisbar. Vi ska visa att om det finns en algoritm för att avgöra om $Un(\mathcal{M})$ är bevisbar så måste det även finnas en algoritm som avgör om $\mathcal{M}$ någonsin skriver 0. Eftersom det inte finns någon algoritm som avgör om en godtycklig maskin någonsin skriver en viss symbol så kan vi heller inte avgöra om $Un(\mathcal{M})$ är bevisbar, alltså är $Un(\mathcal{M})$ ett oavgörbart problem och detta visar att avgörbarhetsproblemet har en negativ lösning. Detta kommer snart bli tydligare. Den sista delen av beviset delar vi upp i två hjälpsatser, där vi visar påståendet "$Un(\mathcal{M})$ är bevisbar om och endast om S_1 finns på remsan vid någon fullständig konfiguration av $\mathcal{M}$".

4.3.2 Två hjälpsatser

Lemma 1. *Om symbolen S_1 finns på remsan vid någon fullständig konfiguration av $\mathcal{M}$ så är $Un(\mathcal{M})$ bevisbar.*

Bevis. Vi beskriver sekvensen av symboler på remsan för $\mathcal{M}$ vid den n:e fullständiga konfigurationen med hjälp av en funktion $r(n, m)$ som $S_{r(n,0)}, S_{r(n,1)}, \ldots, S_{r(n,n)}$. Funktionen $r(n, m)$ är alltså en funktion som ger indexet till symbolen i den m:e rutan vid den fullständiga konfigurationen n. Eftersom en TM endast skriver en symbol (eller ingen) vid varje fullständig konfiguration kommer det aldrig finnas fler än n symboler på remsan vid den n:e fullständiga konfigurationen, alla övriga rutor är blanka. Den lästa rutan vid den n:e fullständiga konfigurationen låter vi vara den $i(n)$:e och läget låter vi vara $q_{k(n)}$. Då kan vi beskriva den n:e fullständiga konfigurationen med formeln:

$$\begin{aligned} &R_{S_{r(n,0)}}(n, 0) \wedge R_{S_{r(n,1)}}(n, 1) \wedge \ldots \wedge R_{S_{r(n,n)}}(n, n) \\ &\wedge I(n, i(n)) \wedge K_{q_{k(n)}}(n) \wedge \forall y G(n, y) \rightarrow R_{S_0}(n, y) \end{aligned}$$

Detta förkortar vi FK_n, den n:e fullständiga konfigurationen.

Vi ska fortsätta genom att bevisa att formeln $A(\mathcal{M}) \wedge F^n \rightarrow FK_n$ (förkortat CF_n) går att bevisa för varje n. Antecedenten i denna formel beskriver en godtycklig maskin $\mathcal{M}$, dess startposition och hela dess övergångstabell. Konsekventen beskriver den n:e fullständiga konfigurationen till denna maskin. Varje fullständig konfiguration n har en formel CF_n associerad med sig, där numreringen av dessa börjar på 0. Det borde med andra ord kännas rimligt att denna formel går att bevisa för alla n.

Vi bevisar formeln med hjälp av induktion. Den första fullständiga konfigurationen av en maskin är alltid väldigt enkel som vi vet. Alla rutor är tomma, ruta 0 blir läst och läget är som vanligt q_1. Större delen av det som beskriver den fullständiga konfigurationen FK_0 faller bort eftersom det inte behövs för att beskriva vår första enkla fullständiga konfiguration. FK_0 blir $I(0,0) \wedge K_{q_1}(1) \wedge \forall y\, R_{S_0}(0,y)$. Om vi minns hur $A(\mathcal{M})$ såg ut så fanns denna del med, det var precis det här som beskrev startpositionen för $\mathcal{M}$, förutom att vi här har gjort det lite enklare för oss (och tydligare) genom att representera u med 0, de representerar samma tal. Alltså har vi ett bevis för $A(\mathcal{M}) \rightarrow FK_n$. Nu visar vi att, för alla n, om CF_n är sann så är även CF_{n+1} sann, det vill säga vi ska bevisa $CF_n \rightarrow CF_{n+1}$. Eftersom varje femtuppel som vi beskriver med en *inst*-formel kan se ut på tre olika sätt beroende på om maskinen flyttar vänster, höger eller inte alls vid den aktuella fullständiga konfigurationen, kommer vi att få tre olika men liknande fall att ta hand om. Vi antar att $\mathcal{M}$ flyttar vänster vid den n:e fullständiga konfigurationen.

När $\mathcal{M}$ är i den fullständiga konfigurationen n är den aktuella femtuppeln (den femtuppel som representerar den rad i övergångstabellen som maskinen ska ”använda” i den n:e fullständiga konfigurationen)

$$(q_{k(n)}, S_{r(n,i(n))}, S_{r(n+1,i(n))}, L, q_{k(n+1)})$$

enligt vår definition av funktionerna r, k och i. Vi förkortar vår formel $Inst(q_{k(n)}, S_{r(n,i(n))}, S_{r(n+1,i(n))}, L, q_{k(n+1)})$ associerad med den aktuella femtuppeln vid den fullständiga konfigurationen n med $Inst_n$. Eftersom $Des(\mathcal{M})$ är en konjunktion av alla beskrivningar av $\mathcal{M}$:s femtupplar så får vi på ett trivialt sätt att

$$Des(\mathcal{M}) \rightarrow Inst_n$$

Eftersom $Des(\mathcal{M})$ är en del av konjunktionen $A(\mathcal{M})$ så har vi lika trivialt att

$$A(\mathcal{M}) \rightarrow Inst_n$$

och även

$$A(\mathcal{M}) \wedge F^{n+1} \rightarrow Inst_n \wedge F^{n+1}$$

Eftersom $Inst_n$ är det uttryck som beskriver hur vi ska ta oss från den fullständiga konfigurationen n till den efter, $n+1$, kan vi enkelt övertyga oss om att

$$Inst_n \wedge Q \wedge F^{n+1} \rightarrow (FK_n \rightarrow FK_{n+1})$$

gäller. Här dyker Q upp igen, som är en del av $A(\mathcal{M})$, för att fastställa entydigheten av successorer. Beviset av detta är mekaniskt men ganska trassligt och långt så vi hoppar över det här. Vidare får vi att

$$A(\mathcal{M}) \wedge F^{n+1} \rightarrow (FK_n \rightarrow FK_{n+1})$$

vilket är ekvivalent med

$$(A(\mathcal{M}) \wedge F^{n+1} \rightarrow FK_n) \rightarrow (A(\mathcal{M}) \wedge F^{n+1} \rightarrow FK_{n+1})$$

Eftersom $F^{n+1} \rightarrow F^n$ enligt konstruktionen av F^n så får vi även

$$(A(\mathcal{M}) \wedge F^n \rightarrow FK_n) \rightarrow (A(\mathcal{M}) \wedge F^{n+1} \rightarrow FK_{n+1})$$

vilket är precis $CF_n \rightarrow CF_{n+1}$. För fallet när $\mathcal{M}$ flyttar höger eller inte alls går vi tillväga på ett analogt sätt. Vi har nu bevisat att CF_n gäller för alla n.

Enligt vårt antagande i detta lemma så finns symbolen S_1, det vill säga en 0:a, i någon ruta K vid någon fullständig konfiguration N av $\mathcal{M}$. Detta innebär att FK_N är en konjunktion som innehåller $R_{S_1}(N, K)$, och vi har att

$$FK_N \rightarrow R_{S_1}(N, K)$$

och på grund av det vi redan fastställt även

$$A(\mathcal{M}) \wedge F^n \rightarrow R_{S_1}(N, K)$$

På grund av Q i $A(\mathcal{M})$ vet vi att det finns successorer till ett tal u och därför har vi att

$$\exists u\, A(\mathcal{M}) \rightarrow \exists u\, \exists u' \ldots \exists u^{N'}\, (A(\mathcal{M}) \wedge F^N)$$

och sammanfattningsvis

$$\exists u\, A(\mathcal{M}) \rightarrow \exists N \exists K\, R_{S_1}(N, K)$$

Bortsett från namnet på variablerna, som vi enkelt kan ändra, är detta precis vårt uttryck för $Um(\mathcal{M})$ som vi skrev som $\exists u\, A(\mathcal{M}) \rightarrow \exists s \exists t\, R_{S_1}(s, t)$. Alltså är $Um(\mathcal{M})$ bevisbar och vi är klara. □

Lemma 2. *Om $Un(\mathcal{M})$ är bevisbar så finns symbolen S_1 på remsan vid någon fullständig konfiguration av $\mathcal{M}$.*

Bevis. Som vi redan har nämnt har vi tolkat predikaten vi använt för att konstruera $Un(\mathcal{M})$ så att tolkningen av $Un(\mathcal{M})$ blir ”symbolen S_1 finns i ruta t vid den fullständiga konfigurationen s av $\mathcal{M}$”. Eftersom det är en premiss att $Un(\mathcal{M})$ är sann/bevisbar så är därmed också detta påstående sant, och därmed är lemma 2 också visat. □

4.3.3 Avslutning

Vi har nu visat att $Un(\mathcal{M})$ är bevisbar om och endast $\mathcal{M}$ skriver en 0:a någon gång. Allt arbete är gjort för att enkelt bevisa avgörbarhetsproblemet för första ordningens predikatlogik med en negativ lösning.

Sats 3. *Det finns ingen generell algoritm för att avgöra om en godtycklig sats i första ordningens predikatlogik är bevisbar.*

Bevis. Antag att det finns en generell algoritm för att avgöra om $Un(\mathcal{M})$ är bevisbar för en godtycklig Turingmaskin $\mathcal{M}$. Då kan vi också ta reda på om $\mathcal{M}$ någonsin skriver en 0:a (enligt ovanstående argument). Detta är omöjligt enligt vad vi tidigare visade, att det inte finns någon generell algoritm för att ta reda på om en godtycklig Turingmaskin någonsin skriver en viss symbol. Vi har en motsägelse och därmed är avgörbarhetsproblemet för första ordningens predikatlogik bevisat.

□

Appendix

A Tabeller och formler

Tabell 1: Fullständiga konfigurationer för exempel 2

b:_	c:əə1_
c:ə$\underline{ə}$1	e:$\underline{ə}$ə1
e:əə$\underline{1}$	e:əə1 _
d:əə1 0x_	c:əə1 0$\underline{x}$1
d:əə1 0 $\underline{1}$	d:əə1 0 1 _
c:əə1 0 1_1	c:əə1 0_1 1
c:əə1_0 1 1	c:ə$\underline{ə}$1 0 1 1
e:$\underline{ə}$ə1 0 1 1	e:əə$\underline{1}$ 0 1 1
e:əə1 $\underline{0}$ 1 1	e:əə1 0x$\underline{1}$ 1
e:əə1 0x1 $\underline{1}$	e:əə1 0x1 1 _
d:əə1 0x1 1 0x_	c:əə1 0x1 1 0$\underline{x}$1
d:əə1 0x1 1 0 $\underline{1}$	d:əə1 0x1 1 0 1 _
c:əə1 0x1 1 0 1_1	c:əə1 0x1 1 0_1 1
c:əə1 0x1 1_0 1 1	c:əə1 0x1_1 0 1 1
c:əə1 0$\underline{x}$1 1 0 1 1	d:əə1 0 $\underline{1}$ 1 0 1 1
d:əə1 0 1 $\underline{1}$ 0 1 1	d:əə1 0 1 1 $\underline{0}$ 1 1
d:əə1 0 1 1 0 $\underline{1}$ 1	d:əə1 0 1 1 0 1 $\underline{1}$
d:əə1 0 1 1 0 1 1 _	c:əə1 0 1 1 0 1 1_1
c:əə1 0 1 1 0 1_1 1	c:əə1 0 1 1 0_1 1 1
c:əə1 0 1 1_0 1 1 1	c:əə1 0 1_1 0 1 1 1
c:əə1 0_1 1 0 1 1 1	c:əə1_0 1 1 0 1 1 1
c:ə$\underline{ə}$1 0 1 1 0 1 1 1	e:$\underline{ə}$ə1 0 1 1 0 1 1 1

Tabell 2: Övergångstabell för den universella Turingmaskinen, del 1

Konfiguration		Beteende	
b	allt		$hitta(b_1, b_1, ::)$
b_1	allt	R2, P:, R2, PD, R2, PA	$start$
$start$	allt		$hittaSista(start_1, :)$
$start_1$	allt		$markKonf(hittaKonf, y)$
$hittaKonf$	;	R, Pz, L	$markKonf(jamforKonf, x)$
	z	L2	$hittaKonf$
	inte z/;	L	$hittaKonf$
$jamforKonf$	allt	$jamforTabort(tabort(tabort(start, x), y), lika, x, y)$	
$lika$	allt		$hittaV(lika_1, lika_1, z)$
$lika_1$	allt		$markKonf(lika_2,)$
$lika_2$	A		$lika_3$
	inte A	L, Pu, R3	$lika_2$
$lika_3$	A	L, Py, R3	$lika_3$
	inte A	L, Py	$tabort(markFKonf, z)$

Tabell 3: Övergångstabell för den universella Turingmaskinen, del 2

Konfiguration			Beteende
$markFKonf$	allt		$hittaSista(markFKonf_1, :)$
$markFKonf_1$	A	L4	$markFKonf_2$
	inte A	R2	$markFKonf_1$
$markFKonf_2$	C	R, Px, L3	$markFKonf_2$
	:		$markFKonf_4$
	D	R, Px, L3	$markFKonf_3$
$markFKonf_3$	:		$markFKonf_4$
	inte :	R, Pv, L3	$markFKonf_3$
$markFKonf_4$	allt		$markKonf(V(V(markFKonf_5)),)$
$markFKonf_5$	något	R, Pw, R	$markFKonf_5$
	blank	P:	$skrivResultat$
$skrivResultat$	allt		$hitta(skrivResultat_1, skrivFKonf, \text{u})$
$skrivResultat_1$	allt	L3	$skrivResultat_2$

Tabell 4: Övergångstabell för den universella Turingmaskinen, del 3

Konfiguration			Beteende
$skrivResultat_2$	D	R4	$skrivResultat_3$
	inte D		$skrivFKonf$
$skrivResultat_3$	C	R2	$skrivResultat_4$
	inte C		$skrivFKonf$
$skrivResultat_4$	C	R2	$skrivResultat_5$
	inte C		$skriv_2(skrivFKonf, 0, :)$
$skrivResultat_5$	C		$skrivFKonf$
	inte C		$skriv_2(skrivFKonf, 1, :)$
$skrivFKonf$	allt		$hittaSista(V(skrivFKonf_1)$, u)
$skrivFKonf_1$	L	R, E	$kopieraMarkeradTabort(slut$, v, y, x, u, w)
	R	R, E	$kopieraMarkeradTabort(slut$, v, x, u, y, w)
	N	R, E	$kopieraMarkeradTabort(slut$, v, x, y, u, w)
$slut$	allt		$tabort(start)$

Formeln $Inst(q_i, S_j, S_k, N, q_l)$

$$\forall x, y, x' \Bigg(\Big(R_{S_j}(x, y) \wedge I(x, y) \wedge K_{q_i}(x) \wedge F(x, x') \Big) \rightarrow \Bigg(I(x', y) \wedge R_{S_k}(x', y) \wedge K_{q_l}(x') \wedge \forall z \Big(\neg (F(z, y) \vee F(y, z)) \vee \big((R_{s_0}(x, z) \rightarrow R_{s_0}(x', z)) \wedge \ldots \wedge (R_{s_M}(x, z) \rightarrow R_{s_M}(x', z)) \big) \Big) \Bigg) \Bigg)$$

Formeln $Inst(q_i, S_j, S_k, R, q_l)$

$$\forall x, y, x' y' \Bigg(\Big(R_{S_j}(x, y) \wedge I(x, y) \wedge K_{q_i}(x) \wedge F(x, x') \wedge F(y, y') \Big) \rightarrow \Bigg(I(x', y') \wedge R_{S_k}(x', y) \wedge K_{q_l}(x') \wedge \forall z \Big(F(z, y') \vee \big((R_{s_0}(x, z) \rightarrow R_{s_0}(x', z)) \wedge \ldots \wedge (R_{s_M}(x, z) \rightarrow R_{s_M}(x', z)) \big) \Big) \Bigg) \Bigg)$$

B En Turingmaskinsinterpretator

Här finns källkoden till ett javaprogram som interpreterar Turingmaskiner (eller snarare övergångstabeller) definierade i en textfil. Programmet är gjort för att på ett tydligt och pedagogiskt sätt visa hur en Turingmaskin arbetar. När du laddar in en textfil (skriven på rätt form) i programmet så kommer övergångstabellen till maskinen visas som en tabell överst i programmet. Under denna visas maskinens resultat samt de fullständiga konfigurationerna för varje steg maskinen tar. Stega fram maskinen gör du med en knapp längst ner i programmet markerad med ”»”. Textfilen måste vara skriven med en speciell syntax för att programmet ska klara av att tolka den korrekt. På första raden i textfilen får endast läget som maskinen befinner sig på vid start vara skrivet. Resten av raderna i filen innehåller information som vi annars skriver i en övergångstabell. Varje sådan rad måste ha formen ”Läge;Läst symbol;Handling;Nästa Läge”. *Läge* och *Näst läge* är namnet på två lägen, exempelvis ”q1” eller ”b”. *Läst symbol* är här begränsad till att vara ett tecken, för *blank* skriver vi ett mellanslag och för *något* skriver vi tecknet ”:”. För symbolen ə skriver vi tecknet ”$”. *Handling* skrivs in som vi är vana vid. Skriv ”P” följt av ett tecken för att skriva en symbol. Skriv ”L” eller ”R” för att ta ett steg till vänster respektive höger. Skriv ”E” för att ta bort symbolen i den aktuella rutan. De olika operationerna måste separeras med ett komma. Observera att ”;” används för att separera olika delar av filen och kan därför inte användas på något annat sätt i filen. Ett exempel på en textfil på rätt form, en som programmet kan interpretera korrekt, och som representerar vårt exempel 2 (se sid 6) ser ut så här:

```
b
b; ;P$, R, P$, R, P1, R;c
c;$;L;e; c;x;E, R;d
c; ;L, L;c
d;:;R, R;d
d; ;P1, L;c
e;1;R, R;e
e;$;R, R;e
e;0;R, Px, R;e
e; ;P0, R, Px, R;d
```

B.1 Javakoden till programmet

Nedan kan du se javakoden till programmet. Programmet är uppdelat i två klasser, *tMachine* och *turing*. I den senare ligger vår main-funktion som startar ett grafiskt gränssnitt för att hantera funktionaliteten som finns i *tMachine*. Koden och programmet finns att hämta på: `http://people.su.se/~siwi1387/turing/`.

```
//*** tMachine.java ***
import java.util.*;

class tMachine {
  private StringBuffer tape;
  private int currentPosition;
  private StringBuffer currentState;
  private String startState;
  private boolean stop;
  private HashMap<String, String> transitions;

  /* Konstruktor som initierar en ny Turingmaskin.
   * tape - (remsan) sätts till att vara tom från början
   * currentPosition - maskinens position är definierad som
   *           0 vid start.
   * startState - Det läge maskinen börjar på sätts till q0.
   *        Kan ändras senare med setStartState().
   * currentState - Maskinens läge sätts att vara samma som
   *          startläget vid start.
   * transitions - Lagrar maskinens konfigurationer, tom från start.
   *         Sätt in en ny transaktion med newTransition().
   */
  public tMachine() {
    tape = new StringBuffer("␣");
    currentPosition = 0;
    stop = false;
    startState = "q0";
    currentState = new StringBuffer(startState);
    transitions = new HashMap<String, String>();
  }

  //Funktion som stegar Turingmaskinen ett steg frammåt.
  public void step() {
    String conf = currentState.toString() + tape.substring(currentPosition
        , currentPosition + 1);
    String todo = transitions.get(conf);
    if (todo == null) {
      todo = transitions.get(currentState.toString() + ":");
    }
    if (todo != null) {
      int i = 0;
      while (i < todo.length()) {
        if (currentPosition == tape.length()-1) {
          tape.append("␣");
        }
        if (todo.charAt(i) == 'P') {
          i++;
          tape.setCharAt(currentPosition, todo.charAt(i));
        } else if (todo.charAt(i) == 'L') {
          currentPosition--;
        } else if (todo.charAt(i) == 'E') {
          tape.setCharAt(currentPosition, '␣');
        } else if (todo.charAt(i) == 'R') {
          currentPosition++;
        } else if (todo.charAt(i) == ';') {
          i++;
        currentState = new StringBuffer(todo.substring(i));
          i = todo.length();
        }
        i++;
      }
```

```
    }
    else {
      stop = true; //I detta fall finns det odefinierade konfigurationer
          och maskinen kraschar.
    }
  }

  //Returnerar den aktuella fullständiga konfiguration som en sträng
      Strängen är formaterad med html.
  public String getCompleteConf() {
    if (stop == true)
      return "Odefinierad konfiguration, maskinen kraschar!";
    else {
      StringBuffer result = new StringBuffer("");
      result.append("<b>" + currentState + "</b> :");
      result.append(tape.substring(0,currentPosition));
      String readSymbol = tape.substring(currentPosition, currentPosition
          + 1);
      if (readSymbol.equals(" "))
        readSymbol = " ";
      result.append("<u>" + readSymbol + "</u>");
      result.append(tape.substring(currentPosition + 1, tape.length()) + "
          <br>");
      return result.toString().replace("$", "\u0259");
    }
  }

  public void newTransition(String conf, String behavior) {
    transitions.put(conf, behavior);
  }

  public String getResult() {
    StringBuffer result = new StringBuffer("");
    for(int i = 0; i < tape.length(); i++) {
      char symbol = tape.charAt(i);
      if (symbol == '1' || symbol == '0')
        result.append(symbol);
    }
    return result.toString();
  }

  public String getStartState() {
    return startState;
  }
  public void setStartState(String state) {
    startState = state;
  }
  public void setTape(String t) {
    tape = new StringBuffer(t);
  }
  public void setCurrentState(String currState) {
    currentState = new StringBuffer(currState);
  }
  public void setCurrentPosition(int currPos) {
    currentPosition = currPos;
  }
}
```

```
//*** turing.java ***
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.util.*;
import java.io.*;

public class turing extends JFrame implements ActionListener {
  private static final long serialVersionUID = 1L;
  private JEditorPane resultArea;
  private StringBuffer content;
  private JTable transitionTable;
  private DefaultTableModel tableModel;
  private tMachine machine;
  private JLabel resultLabel;

  //Konstruktor, initierar all fönsterhantering.
  public turing() {
    super("The Turing machine");
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (UnsupportedLookAndFeelException e) {
      System.err.print("Sorry, the look and feel the program are trying to
          load is unsupported." + e.getMessage());
      System.exit(1);
    }
    catch (InstantiationException e) {
      System.err.print("Sorry, the look and feel could not be loaded." + e
          .getMessage());
      System.exit(1);
    }
    catch (IllegalAccessException e) {
      System.err.print("Sorry, you don't have access to the look and feel
          you're trying to load." + e.getMessage());
      System.exit(1);
    }
    catch (ClassNotFoundException e) {
      System.err.print("Sorry, the look and feel you are trying to load
          does not exist" + e.getMessage());
      System.exit(1);
    }

    content = new StringBuffer();
    final JPanel mainPanel = new JPanel (new BorderLayout( ));
    add(mainPanel);

    JPanel centerPane = new JPanel();
    //Här skapas en panel som innehåller övergångstabellen och
        resultatarean.
    centerPane.setLayout(new GridLayout(2, 1, 5, 5));

    tableModel = new DefaultTableModel();
    tableModel.addColumn("State");
    tableModel.addColumn("Symbol");
    tableModel.addColumn("Operation");
    tableModel.addColumn("Next state");

    transitionTable = new JTable(tableModel);
    transitionTable.setRowMargin(5);
    transitionTable.setRowHeight(25);
```

```
transitionTable.setShowVerticalLines(false);
transitionTable.setShowHorizontalLines(false);
transitionTable.setPreferredScrollableViewportSize(new Dimension(500,
    70));
transitionTable.setFillsViewportHeight(true);
transitionTable.setBorder(BorderFactory.createLineBorder(Color.black))
    ;
JScrollPane scrollTransitionTable = new JScrollPane(transitionTable);
scrollTransitionTable.setBorder(BorderFactory.createEmptyBorder(10,
    10, 10, 10));
centerPane.add(scrollTransitionTable);

resultArea = new JEditorPane("text/html", content.toString());
resultArea.setBorder(BorderFactory.createLineBorder(Color.black));
JScrollPane scrollResultArea = new JScrollPane(resultArea);
setPreferredSize(new Dimension(450, 110));
scrollResultArea.setBorder(BorderFactory.createEmptyBorder(10, 10, 10,
     10));
centerPane.add(scrollResultArea);

mainPanel.add(centerPane, BorderLayout.CENTER);

//Här skapas panelen längst ner, som innehåller navigationsknapparna
    och maskinens resultat.
JPanel bottom = new JPanel();
bottom.setLayout(new GridLayout(2, 1, 5, 5));
bottom.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
resultLabel = new JLabel("Result: ");
bottom.add(resultLabel);

JPanel navigation = new JPanel();
navigation.setLayout(new BoxLayout(navigation, BoxLayout.LINE_AXIS));

JButton restartButton = new JButton ( "Restart" ) ;
restartButton.setActionCommand("restart");
restartButton.addActionListener(this);

JButton forwardButton = new JButton ( ">>" ) ;
forwardButton.setActionCommand("forward");
forwardButton.addActionListener(this);

getRootPane().setDefaultButton(forwardButton);

navigation.add(Box.createHorizontalGlue());
navigation.add(restartButton);
navigation.add(Box.createRigidArea(new Dimension(10, 0)));
navigation.add(forwardButton);
navigation.add(Box.createHorizontalGlue());

bottom.add(navigation);
mainPanel.add(bottom, BorderLayout.PAGE_END);

//Här skapas en meny med två alternativ, Load och Exit.
final JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);
final JMenu menu = new JMenu("File");
JMenuItem load = new JMenuItem("Load...");
load.setActionCommand("load");
load.addActionListener(this);
JMenuItem exit = new JMenuItem("Exit");
exit.setActionCommand("exit");
exit.addActionListener(this);
```

```
    menu.add(load);
    menu.add(exit);
    menuBar.add(menu);

    setDefaultCloseOperation ( JFrame.EXIT_ON_CLOSE ) ;
    pack ( )  ;
    setSize ( 500,500 ) ;
    setResizable(true);
    setVisible(true) ;
  }

  public void actionPerformed(ActionEvent e) {
    if ("load".equals(e.getActionCommand())) {
      final JFileChooser fc = new JFileChooser();
      int returnVal = fc.showOpenDialog(this);

      if (returnVal == JFileChooser.APPROVE_OPTION) {
        File file = fc.getSelectedFile();
        init(file);
      }
      else {

      }
    }
    if ("forward".equals(e.getActionCommand())) {
      content.append(machine.getCompleteConf());
      resultArea.setText(content.toString());
      resultLabel.setText("Result: " + machine.getResult());
      machine.step();
    }
    if ("exit".equals(e.getActionCommand())) {
      System.exit(0);
    }
    if ("restart".equals(e.getActionCommand())) {
      resultArea.setText("");
      resultLabel.setText("Result: ");
      machine.setCurrentState(machine.getStartState());
      machine.setCurrentPosition(0);
      machine.setTape(" ");
      content = new StringBuffer(new String(""));
    }
  }

  //Initierar en ny maskin när man laddar en fil.
  private void init(File f) {
    machine = new tMachine();
    try {
      resultArea.setText("");
      BufferedReader in = new BufferedReader(new FileReader(f));
      Scanner s = new Scanner(in);
      String line = s.nextLine();
      machine.setStartState(line);
      machine.setCurrentState(line);
      while(s.hasNext()) {
        line = s.nextLine();
        StringTokenizer token = new StringTokenizer(line, ";");
        String state = token.nextToken();
        String symbol = token.nextToken();
        String action = token.nextToken();
        String nextState = token.nextToken();
        machine.newTransition(state + symbol, action + ";" + nextState);
        if (symbol.equals(" "))
```

```
            symbol = "blank";
          if (symbol.equals(":"))
            symbol = "anny";
          if (symbol.equals("$"))
            symbol = "\u0259";
          action = action.replace("$", "\u0259");
          tableModel = (DefaultTableModel) transitionTable.getModel();
          String[] row = {state, symbol, action, nextState};
          tableModel.addRow(row);
          transitionTable.setModel(tableModel);
        }
      }
      catch (FileNotFoundException e) {
        System.err.print("Kunde inte hitta filen, programmet avslutas!");
        System.exit(1);
      }
    }

    //main-funktionen, detta drar igång hela programmet.
    public static void main ( final String[] args ) {
      SwingUtilities.invokeLater ( new Runnable ( ) {
        public void run ( ) { new turing( ) ; }
      } ) ;
    }
}
```

C Referenser

Referenser

[1] Bilaniuk, Stefan A problem Course in Mathematical Logic, version 1.6. http://euclid.trentu.ca/math/sb/pclm/

[2] Hopcroft, John E., Rajeev Motwani och Jeffrey D. Ullman Introduction to automata Theory, Languages, and Computation, Addison-Wesley 2001, andra upplagan.

[3] MathWorld: Turingmaskiner till Mathematica. http://demonstrations.wolfram.com/search.html?query=turing+machine&start=1&limit=25

[4] Mendelson, Elliot Introduction to Mathematical Logic, Chapman & Hall, 1997.

[5] Petzold, Charles The annotated Turing, Wiley Publishing 2008.

[6] Turing, Alan M. On computable numbers, with an application to the Entscheidungsproblem, från Proceedings of the London Mathematical Society, (Ser. 2, Vol. 42, 1937). http://www.turingarchive.org/browse.php/B/12 eller http://www.thocp.net/biographies/papers/turing_oncomputablenumbers_1936.pdf

[7] Turing, Alan M. Computing machinery and intelligence. Mind, 1950. http://www.loebner.net/Prizef/TuringArticle.html