Konstruktiv logik

Erik Palmgren
Uppsala universitet

september 2001

Konstruktiv logik bygger på den ledande principen att

(I) *bevis är program.*

Denna har sedan utökats med principen att

(II) *påståenden är datatyper.*

inom (konstruktiv) typteori. Den senare benämns även påståenden-som-typer principen (engelska: *propositions-as-types*). Innebörden i dessa principer är långt ifrån självklar. Syftet med dessa föreläsningar att stegvis förklara den och dess konsekvenser. Ett viktigt mål är att ge grundkunskaper för att kunna studera Martin-Löfs typteori och använda dess implementerade versioner.

Bevis utförda inom konstruktiv logik kan exekveras som funktionella program (närmast att likna vid ML-program). Fördelen med detta är att man ur ett existensbevis (t.ex. av att det finns godtyckligt stora primtal) kan utvinna dels ett program som hittar eller konstruerar objektet, och dels en verifikation att programmet terminerar (hittar något tal) och är korrekt (hittar bara tillräckligt stora primtal). Denna kombination av konstruktion och verifikation av program har väckt stort intresse inom datalogisk forskning, och benämns ibland *integrerad programlogik*. En avläggare till denna idé är den om *bevisbärande kod* (eng. *Proof-carrying code*) där programkod levereras med ett bevis för att programmet inte gör något otillbörligt när det exekveras, något som kan vara viktigt för nätbaserad programmering (t.ex. Java).

Den konstruktiva logiken har en historia långt före datorernas tillkomst. Kring sekelskiftet 1900 hade man börjat tvivla på de axiomatiska grundvalarna för den abstrakta matematik som just hade börjat utvecklas med hjälp av olika mängdkonstruktioner. Betrand Russell hade genom sin paradox visat att oinskränkt mängdbildning kan leda till en motsägelse. Även andra principer, såsom urvalsaxiomet, hade visat sig ha förbryllande och onaturliga konsekvenser, t.ex. att de reella talen kan välordnas, även om det inte ledde till rena motsägelser. I denna tidsstämning inledde den framstående holländske matematikern Luitzen E.J. Brouwer en kritik och omarbetning av grundvalarna för matematiken

som gick djupare än tidigare, till själva logiken snarare än axiomen. Genom sin s.k. *intuitionistiska matematik* ville han ställa matematiken på en tillförlitlig och intuitiv grund. Hans idé var att varje bevis måste bygga på en s.k. *mental konstruktion.* Vid denna tid (1910-talet) fanns varken programmeringsspråk, eller något exakt algoritmbegrepp, men det visade sig senare att begreppet mental konstruktion kunde tolkas som algoritmisk konstruktion. Detta krav ledde Brouwer till att avvisa en logisk princip som tagits för given sedan Aristoteles, nämligen *lagen om det uteslutna tredje.* Denna utsäger att för varje påstående A gäller A eller icke-A. För konkreta, ändligt verifierbara påståenden fanns det inget skäl att betvivla denna lag. Det problematiska var enligt Brouwer när A var ett påstående som innefattade kvantifikation över en oändlig mängd, t.ex. mängden av heltal. Brouwer visade att det var möjligt att utveckla matematiska teorier även utan antagande av denna lag. Det skall nämnas att Brouwer aldrig fick särskilt många anhängare till sin konstruktiva matematiska filosofi, men den utvecklas alltjämt (se Bishop-Bridges 1985, Bridges-Richman 1987). Däremot fick Brouwers filosofiska idéer stor betydelse inom logiken och via denna även en plats inom datalogin.

1 Icke-konstruktiva bevis

Vi skall här illustrera hur lagen om det uteslutna tredje används i några existensbevis, och hur detta kan leda till att det algoritmiska innehållet försvinner.

Sats 1.1 *Det finns irrationella tal a och b sådana att a^b är ett rationellt tal.*

Bevis. Som bekant är $\sqrt{2}$ ett irrationellt tal. Betrakta talet $\sqrt{2}^{\sqrt{2}}$. Enligt lagen om det uteslutna tredje är talet rationellt eller irrationellt. Om det är rationellt, så blir beviset färdigt genom att sätta $a = b = \sqrt{2}$. Om det är irrationellt, låt $a = \sqrt{2}^{\sqrt{2}}$ och $b = \sqrt{2}$. Då är a^b rationellt, eftersom det till och med är

$$a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = (\sqrt{2})^2 = 2. \blacksquare$$

I detta existensbevis så har vi inte avgjort vilket av talen $a = \sqrt{2}$ eller $a = \sqrt{2}^{\sqrt{2}}$ med $b = \sqrt{2}$ som ger det avsedda exemplet. (Man kan faktiskt visa att $\sqrt{2}^{\sqrt{2}}$ är irrationellt men detta kräver avsevärt mer matematisk teori och beviset upptar ett par boksidor. Ett alternativt exempel fås av talen $a = e$ och $b = \ln 2$, men beviset för deras irrationalitet är svårare än det för $\sqrt{2}$.)

Många klassiska existensbevis börjar "Antag att det inte finns något objekt x sådant att ...", och resten av beviset ägnas åt att härleda en motsägelse. Här är ett enkelt exempel.

Sats 1.2 *Varje oändlig följd $(a_k)_{k=0}^{\infty}$ av naturliga tal har en minsta term, dvs. det finns ett n sådant att $a_k \geq a_n$ för alla k.*

Bevis. Låt $(a_k)_{k=0}^{\infty}$ vara en följd av naturliga tal. Antag att det inte finns någon minsta term i följden, dvs. att för varje k finns $k' > k$ med $a_k > a_{k'}$. Låt $k_0 = 0$. Enligt antagande finns $k_1 > k_0$ med $a_{k_0} > a_{k_1}$. Igen, enligt antagandet, finns $k_2 > k_1$ sådant att $a_{k_1} > a_{k_2}$. Fortsätter vi på detta sätt får vi en följd $k_0 < k_1 < k_2 < \cdots$ sådan att

$$a_{k_0} > a_{k_1} > a_{k_2} > \cdots$$

Eftersom denna delföljd av $(a_k)_k$ minskar åtminstone med 1 för varje steg, har vi $a_{k_{a_0+1}} < 0$. Detta är motsägelse, ty vi antog att talen i följden var icke-negativa. ∎

Detta existensbevis ger ingen som helst information om var minimum antas. Denna typ av informationslösa existensbevis är inte ovanliga i matematisk analys. Exempelvis bygger vissa resultat om existens av lösningar till differentialekvationer på sådana bevis (Cauchy-Peanos existenssats). Det finns relativt enkla exempel på ordinära differentialekvationer som ej har någon effektivt beräknebar lösning, men som dock har en teoretisk lösning (se Beeson 1985).

Konstruktiv matematik och logik bygger på att existensbegreppet tas på större allvar än i klassisk matematik: att bevisa att ett visst objekt existerar är det samma som att ge en metod för att konstruera det.

Övningar

1.1. (För den som är bekant med algebraiska och transcendentala tal.) Talen e och π är transcendentala. Visa att $e + \pi$ eller $e - \pi$ är transcendental. (Detta har ett klassiskt enradsbevis, men det är inte känt vilket av talen som är transcendentalt!)

2 Typad lambdakalkyl

Vi skall här ge en kort inledning till den lambdakalkyl som behövs för att precisera begreppet (mental) konstruktion. För en fullständig framställning av kalkylen, se Hindley och Seldin 1986 eller Barendregt 1992.

Lambdakalkylen introducerades 1932 av den amerikanske logikern Alonzo Church, ursprungligen för att ge en formalisering av matematiken. Den kom snart att få en användning för att precisera begreppet beräknebarhet. John McCarthy utgick från denna kalkyl när han konstruerade programmeringspråket LISP, vilket kom att bli utgångspunkten för senare generationer av funktionella programmeringsspråk såsom (S)ML och Haskell.

Vi inför och studerar en typad version av lambdakalkyl. Först definierar vi begreppet typ induktivt.

Definition 2.1 *Typer.*

(i) $\mathbb{N}$ är en typ (typen av naturliga tal).

(ii) Om A och B är typer, så är $(A \times B)$ en typ (produkttypen av A och B).

(iii) Om A och B är typer, så är $(A \to B)$ en typ (typen av funktioner från A till B).

(iv) Om A och B är typer, så är $(A + B)$ en typ (den disjunkta unionen av A och B).

Exempel på typer är

$$(\mathbb{N} \to (\mathbb{N} \times \mathbb{N})) \qquad ((\mathbb{N} + (\mathbb{N} \times \mathbb{N})) \to \mathbb{N}).$$

För att nedbringa antalet parenteser inför vi konventionen att $\times$ binder starkare än $+$, vilken i sin tur binder starkare än $\to$. Ovanstående typer skrivs härmed

$$\mathbb{N} \to \mathbb{N} \times \mathbb{N} \qquad \mathbb{N} + \mathbb{N} \times \mathbb{N} \to \mathbb{N}.$$

Dessutom använder vi ibland konventionen att $\to$ associerar till höger, så att $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ skall läsas $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ och inte $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$.

Anmärkning 2.2 Typkonstruktionerna (i) – (iii) är välbekanta från SML, men betecknas där `nat`, `A * B` respektive `A -> B`.

Vi skall nu definiera termerna, eller "programmen", i lambdakalkylen. De viktigaste konstruktionerna är *abstraktion,* $\lambda x \to b$, och *applikation,* $\mathsf{apply}(f, a)$. Abstraktionen bildar från b en ny funktion $\lambda x \to b$ av en utvald variabel x. Om b har typen B och x har typen A, så får den nya funktionen typen $A \to B$. I $\lambda x \to b$ är x en bunden variabel. Applikationen $\mathsf{apply}(f, a)$ tillämpar en funktion f på ett argument a. Sambandet mellan dessa konstruktioner ges av *β-regeln*

$$\mathsf{apply}(\lambda x \to b, a) = b[a/x], \tag{1}$$

där högerledet betyder att a har substituerats för x i b. En förutsättning för att regeln skall få tillämpas är att inga variabler i a blir bundna vid substitutionen. Liksom i första ordningens logik har de bundna variablernas namn har ingen betydelse. Detta uttrycks med α-regeln:

$$\lambda x \to b = \lambda y \to b[y/x], \tag{2}$$

där y är en variabel som ej förekommer i b. Genom att använda denna regel kan vi alltid tillse att β-regeln är tillämpbar. (I SML skrivs $\lambda x \to b$ som `(fn x => b)`. Observera att standardbeteckning i de flesta framställningar av lambdakalkylen är $\lambda x.b$ istället för $\lambda x \to b$.)

En annan viktig konstruktion är *parbildning:* om a har typ A och b har typ B, så kan vi bilda paret av dessa, $\langle a, b \rangle$, som har typ $A \times B$. Det finns två projektionsfunktioner, eller *selektorer,* som plockar ut första och andra komponent ur detta par

$$\#_1(\langle a, b \rangle) = a, \qquad \#_2(\langle a, b \rangle) = b. \tag{3}$$

Den tredje typkonstruktionen är disjunkt union $A + B$ av två typer. Denna är intuitivt av unionen av A och B, men där man markerar från vilken av mängderna elementet kommit, den vänstra eller den högra. (Detta kan behövas om $A = B$.) Typen har två konstruktorer, inl och inr. Om a har typ A, så har $\mathsf{inl}(a)$ typ $A + B$. Om b har typ B, så har $\mathsf{inr}(b)$ typ $A + B$. För att uttrycka att varje element har någon av dessa former $\mathsf{inl}(a)$ eller $\mathsf{inr}(b)$, inför vi en fallåtskiljare when. Om f har typ $A \to C$ och g har typ $B \to C$, så gäller att $\mathsf{when}(d, f, g)$ har typ C. Denna har beräkningsreglerna

$$\mathsf{when}(\mathsf{inl}(a), f, g) = \mathsf{apply}(f, a), \qquad \mathsf{when}(\mathsf{inr}(b), f, g) = \mathsf{apply}(g, b). \tag{4}$$

I SML finns ingen primitiv typbildare som fungerar som $+$. Däremot kan man definiera en ny typkonstruktion

```
datatype ('a,'b) sum = inl of 'a | inr of 'b
```

De naturliga talen representeras enklast med unär notation: 0 har typ $\mathbb{N}$, om n har typ $\mathbb{N}$, så har det efterföljande talet, $\mathsf{S}(n)$, typ $\mathbb{N}$. Talet 3 representeras således av termen $\mathsf{S}(\mathsf{S}(\mathsf{S}(0)))$. En sådan term benämns *numeral* och betecknas $\overline{3}$. Vi introducerar en operator rec som medger en generaliserad form av primitiv rekursion (jämför Salling 1999): för f av typ A och g av typ $\mathbb{N} \to A \to A$

$$\begin{aligned} \mathsf{rec}(0, f, g) &= f, & (5)\\ \mathsf{rec}(\mathsf{S}(n), f, g) &= g(n)(\mathsf{rec}(n, f, g)). & (6) \end{aligned}$$

Vi definierar nu termerna formellt. Vi använder förkortningen $a : A$ för att a är en term av typ A.

Definition 2.3 *Termer.*

Till varje typ A hör oändligt många variabler $x^A, y^A, z^A, \ldots$ som är av typ A.
Om $f : (A \to B)$ och $a : A$, så $\mathsf{apply}(f, a) : B$.
Om $b : B$ och x^A är en variabel, så $\lambda x^A \to b : (A \to B)$, och variabeln x^A binds.
Om $a : A$ och $b : B$, så $\langle a, b\rangle : (A \times B)$.
Om $c : (A \times B)$, så $\#_1(c) : A$ och $\#_2(c) : B$.
Om $a : A$ och B är en typ, så $\mathsf{inl}(a) : (A + B)$.
Om $b : B$ och A är en typ, så $\mathsf{inr}(b) : (A + B)$.
Om $d : (A + B)$, $f : (A \to C)$ och $g : (B \to C)$, så $\mathsf{when}(d, f, g) : C$.
Om $a : \mathbb{N}$, $f : A$ och $g : (\mathbb{N} \to (A \to A))$, så $\mathsf{rec}(a, f, g) : A$.

Vi utelämnar ofta typinformationen A från variabler x^A (eller andra termer) när den kan härledas från sammanhanget. Uttrycket $\mathsf{apply}(f, a)$ förkortas $f(a)$. Pilen $\to$ binder svagast varför uttrycket $\lambda x \to f(a)$ skall läsas $\lambda x \to \mathsf{apply}(f, a)$, medan $\mathsf{apply}(\lambda x \to f, a)$ skrivs $(\lambda x \to f)(a)$.

Lambdakalkylen är en *ekvationell teori,* dvs. en teori där bara likheter mellan termer förekommer. "Formlerna" är således uttryck av formen $s = t$, där s och t är termer av samma typ. Som axiom antar vi alla instanser av likheterna (1), (2), (3), (4), (5), (6) och $t = t$. Härledningsreglerna är

$$\frac{t_1 = t_2}{t_2 = t_1} \qquad \frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$$

$$\frac{t_1 = t_2}{t_1[t/x] = t_2[t/x]} \qquad \frac{t_1 = t_2}{t[t_1/y] = t[t_2/y]}$$

och syftar enbart till att administrera kalkyler i deluttryck. Med hjälp av dessa kan man kalkylera som förväntat.

Exempel 2.4 *Komposition av funktioner.* Låt

$$\mathsf{comp} =_{\text{def}} \lambda f^{B\to C} \to \lambda g^{A\to B} \to \lambda x^{A} \to f(g(x)).$$

Denna term har typen $(B \to C) \to (A \to B) \to A \to C$. Vi har $\mathsf{comp}(f)(g)(x) = f(g(x))$.

Exempel 2.5 Föregångarfunktionen $\mathsf{pd} : \mathbb{N} \to \mathbb{N}$ definieras av

$$\mathsf{pd} =_{\text{def}} \lambda x \to \mathsf{rec}(x, 0, \lambda n \to \lambda y \to n).$$

Vi har

$$\mathsf{pd}(0) = \mathsf{rec}(0, 0, \lambda n \to \lambda y \to n) = 0$$

och

$$\begin{aligned} \mathsf{pd}(\mathsf{S}(x)) &= \mathsf{rec}(\mathsf{S}(x), 0, \lambda n \to \lambda y \to n) \\ &= (\lambda n \to \lambda y \to n)(x)(\mathsf{rec}(x, 0, \lambda n \to \lambda y \to n)) \\ &= (\lambda y \to x)(\mathsf{rec}(x, 0, \lambda n \to \lambda y \to n)) \\ &= x \end{aligned}$$

Exempel 2.6 *Multiplikation med 2.* Definiera

$$\mathsf{double} =_{\text{def}} \lambda x \to \mathsf{rec}(x, 0, \lambda n \to \lambda y \to \mathsf{S}(\mathsf{S}(y))).$$

Vi har $\mathsf{double}(0) = 0$ och $\mathsf{double}(\mathsf{S}(x)) = \mathsf{S}(\mathsf{S}(\mathsf{double}(x)))$.

Exempel 2.7 *En representation av hela tal.* Låt $\mathbb{Z} = \mathbb{N} + \mathbb{N}$ och låt $\mathsf{inl}(m)$ stå för det negativa talet $-(m + 1)$ medan $\mathsf{inr}(n)$ står för det icke-negativa talet n. Man verifierar enkelt att denna term byter tecken på ett heltal:

$$\mathsf{neg} =_{\text{def}} \lambda z \to \mathsf{when}(z, \lambda u \to \mathsf{inr}(\mathsf{S}(u)), \lambda v \to \mathsf{rec}(v, \mathsf{inr}(0), \lambda n \to \lambda y \to \mathsf{inl}(n))).$$

Dvs. att vi har $\mathsf{neg}(\mathsf{inl}(m)) = \mathsf{inr}(\mathsf{S}(m))$, $\mathsf{neg}(\mathsf{inr}(0)) = \mathsf{inr}(0)$ och $\mathsf{neg}(\mathsf{inr}(\mathsf{S}(m))) = \mathsf{inl}(m)$.

Exempel 2.8 Definiera en lambda-term I av typ $(A \to A) \to \mathbb{N} \to (A \to A)$ sådan att när den tillämpas på en numeral $\overline{n}$, som följer, så

$$I(f)(\overline{n}) = \lambda x \to f(f(\cdots f(x)\cdots)))$$

där antalet f i högerledet är n. Vi ser att högerledet svarar mot att sammansätta f med sig själv n gånger. För att göra detta använder vi rekursionsoperatorn.

$$I =_{\text{def}} \lambda f \to \lambda n \to \mathsf{rec}(n, \lambda x \to x, \lambda m \to \lambda g \to \mathsf{comp}(f, g))$$

Anmärkning 2.9 Det är naturligt att betrakta likheterna (=) som beräkningsrelationer riktade från vänster till höger, så att exempelvis $\#_1(\langle a, b\rangle) = a$ blir $\#_1(\langle a, b\rangle)$ *beräknar till* a. På så sätt kan vi betrakta den typade lambda-kalkylen som ett programmeringsspråk. Den har den ovanliga egenskapen att *alla program terminerar*. Att visa detta är mycket komplicerat, se exempelvis (Hindley och Seldin 1986) för ett bevis för en förenklad kalkyl. Att alla program terminerar medför också att programmeringsspråket inte är Turingfullständigt, dvs. det finns en Turing-beräknebar funktion som inte kan representeras som en lambdaterm. (Jämför Övning 2.5 nedan.) Dock innehåller det alla program som kan *bevisas terminera* i teorin för Peano-aritmetik.

Anmärkning 2.10 I (ren) otypad lambdakalkyl används endast abstraktion och applikation. Den har, som namnet anger, inte några typer så varje term kan appliceras på vilken som helst term, till och med sig själv. Exempelvis är $(\lambda x \to x(x))(\lambda x \to x(x))$ en välbildad term. Om man tillämpar β-regeln får man tillbaka samma term! Detta innebär att det finns icke-terminerande program i kalkylen. Trots den extrema enkelheten hos den otypade kalkylen är den Turingfullständig. Det är dock relativt komplicerat att visa detta, och gjordes först av Turing 1936, byggandes på resultat av S.C. Kleene.

Övningar

2.1 Verifiera likheterna i Exempel 2.6, 2.7, 2.8.

2.2 Konstruera en lambdaterm $e : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ sådan att

$$\begin{aligned} e(0)(0) &= \mathsf{S}(0) \\ e(0)(\mathsf{S}(y)) &= 0 \\ e(\mathsf{S}(x))(0) &= 0 \\ e(\mathsf{S}(x))(\mathsf{S}(y)) &= e(x)(y). \end{aligned}$$

Detta är alltså en funktion som kan användas för att avgöra om två tal är lika.

2.3 Visa att varje primitivt rekursiv funktion kan simuleras av en lambdaterm.

2.4 Det är känt att Ackermann-funktionen $a : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, definierad av

$$\begin{aligned} a(0, n) &= \mathsf{S}(n) \\ a(\mathsf{S}(m), 0) &= a(m, \mathsf{S}(0)) \\ a(\mathsf{S}(m), \mathsf{S}(n)) &= a(m, a(\mathsf{S}(m), n)), \end{aligned}$$

växer alltför fort för att kunna vara en primitivt rekursiv funktion. Visa att den kan definieras i den typade lambdkalkylen med hjälp av rekursionsoperatorn rec. [Ledning 1: expandera definitionen av $a(\mathsf{S}(m), n)$ i högerledet. Ledning 2: definiera en funktion $b : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ så att $a(m, n) = b(m)(n)$. Använd eventuellt också Exempel 2.8]

2.5 (a) Låt $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ och låt $g(n) = f(n, n) + 1$. Visa att det inte finns något m så att $f(m, n) = g(n)$ för alla n.

(b) Visa att g är beräknebar, om f är beräknebar.

(c) Antag att lambdatermerna innehåller tillräcklig information för att bestämma deras typ på ett entydigt sätt. Detta kan göras genom förse $\mathsf{apply}, \mathsf{when}, \mathsf{rec}, \mathsf{inl}, \mathsf{inr}$ med typinformation: $\mathsf{apply}_{A,B}, \mathsf{when}_{A,B,C}, \mathsf{rec}_A, \mathsf{inl}_{A,B}, \mathsf{inr}_{A,B}$ (se Definition 2.3). Argumentera informellt för att följande funktion är totalt definierad och beräknebar

$$f(m, n) = \begin{cases} k & \text{om } m \text{ i basen 2 är Ascii-kod för en lambdaterm } t \text{ av typ } \mathbb{N} \to \mathbb{N}, \\ & \text{och } k \text{ är värdet av } t(\overline{n}). \\ 0 & \text{annars.} \end{cases}$$

Använd det faktum att $t(x)$ alltid terminerar med en numeral som värde då x är en numeral. (Vi förutsätter att specialtecknen $\lambda, \to$ kodas på lämpligt sätt.)

(d) Visa med hjälp av (a) – (c) att det finns en beräknebar funktion som inte kan beräknas av någon term i den typade lambdakalkylen (som presenterats i detta kapitel).

3 Konstruktiv tolkning av de logiska konnektiven

Enligt Brouwer måste varje bevisat påstående A bygga på någon (mental) konstruktion a. Konstruktionen a kan betraktas som ett *certifikat* för A:s sanning. Vi skall använda de lambdatermer som introducerades i kapitel 2 som konstruktioner. Denna konstruktiva tolkning förtydligades senare av Arend Heyting (en student till Brouwer) och av A.N. Kolmogorov (för övrigt den moderna sannolikhetsteorins grundare) och kallas därför *Brouwer-Heyting-Kolmogorov-tolkningen* (BHK-tolkningen).

Det skall påpekas att klassen av konstruktioner inte är begränsad till de lambdatermer vi införde, utan kan behöva utvidgas när man upptäcker nya konstruktionsmetoder (se dock Anmärkning 3.5). Det finns emellertid begränsningar. Man skulle kunna tro att detta är en konstruktion

$$f(n) = \begin{cases} 1 & \text{det finns } n \text{ konsekutiva 7:or i decimalutveckligen av talet } \pi, \\ 0 & \text{annars.} \end{cases}$$

Detta är, a priori, inte en konstruktiv funktion eftersom ingen har (ännu) åstadkommit en algoritm som kan avgöra om det finns n 7:or i decimalutveckligen av π. Fallvis definition är tillåten bara om vi kan effektivt avgöra vilket fall som gäller, för givna parametrar.

BHK-tolkningen. Vi förklarar vad det betyder att vara *certifikat* för ett påstående A, med induktion på formen av A.

- $\bot$ har inget certifikat.
- p är ett certifikat för $s = t$ omm $p = 0$ och s och t beräknas till samma sak.
- p är ett certifikat för $A \wedge B$ omm p är ett par $\langle a, b\rangle$ där a är certifikat för A och b är certifikat för B.
- p är ett certifikat för $A \to B$ omm p är en funktion som till varje certifikat a för A tilldelar ett certifikat $p(a)$ för B.
- p är ett certifikat för $A \vee B$ omm p har formen $\mathsf{inl}(a)$, i vilket fall a är ett certifikat för A eller p har formen $\mathsf{inr}(b)$, i vilket fall b är ett certifikat för B
- p är ett certifikat för $(\forall x \in S)A(x)$ omm p är en funktion som till varje element $d \in S$, tilldelar ett certifikat $p(d)$ av $A(d)$.
- p är ett certifikat för $(\exists x \in S)A(x)$ omm p är ett par $\langle d, q\rangle$ som består av $d \in S$ och ett certifikat q för $A(d)$.

Man säger att A är *giltig under BHK-tolkningen* om man kan finna en konstruktion p sådan att p är ett certifikat för A. (Ofta säger man att p är ett *bevis för A,* men för att inte blanda ihop detta med formella härledningar använder vi här ordet *certifikat.* Konstruktionen p kallas i den alternativa terminologin *bevisobjekt.*)

Anmärkningar 3.1 I fallet för $=$ krävs det att vi kan avgöra om två element är lika. Detta är möjligt för naturliga tal, heltal, rationella tal och ändliga datastrukturer som listor och träd. Likhet för t.ex. funktioner kan definieras argumentvis. I $\vee$-fallet, indikerar $\mathsf{inl}(a)$ eller $\mathsf{inr}(b)$ om det är den vänstra eller högra disjunkten vi har ett certifikat för.

Här är några exempel på BHK-tolkningar. Vi använder konstruktioner från typad lambdakalkyl.

Exempel 3.2 1. $p = \lambda x \to x$ är ett certifikat för påståendet $A \to A$. Detta är klart, ty om a är ett certifikat för A, så $p(a) = (\lambda x \to x)(a) = a$ och detta är enligt antagande ett certifikat för A.

2. Ett certifikat för $A \wedge B \to B \wedge A$ ges av konstruktionen $f = \lambda x \to \langle \#_2(x), \#_1(x)\rangle$.

3. Betrakta påståendet $\bot \to A$. Ett certifikat för detta är en godtycklig funktion f såsom $f(x) = 42$: Antag att a är ett certifikat för $\bot$. Men enligt BHK-tolkningen har $\bot$ inget certifikat, så vi har nått en motsägelse. Vad som helst följer då, speciellt att 42 är ett bevis för A.

Negation definieras genom $\neg A =_{\text{def}} (A \to \bot)$. Att bevisa $\neg A$ är alltså att bevisa att A kan leda till en motsägelse. Som vanligt definierar vi $A \leftrightarrow B$ att vara $(A \to B) \wedge (B \to A)$.

Exempel 3.3 Kontrapositionslagen $(A \to B) \to (\neg B \to \neg A)$ är giltig under BHK-tolkningen. Antag att f är ett certifikat för $A \to B$. Vi vill finna ett certifikat för $(\neg B \to \neg A)$, dvs. $(B \to \bot) \to (A \to \bot)$. Antag följaktligen att g certifierar $\neg B$ och a certifierar A. Därmed är $f(a)$ ett certifikat för B, och således är $g(f(a))$ ett certifikat för $\bot$. Konstruktionen $\lambda a \to g(f(a))$ certifierar således $\neg A$. Abstraherar vi på g är det klart att $\lambda g \to \lambda a \to g(f(a))$ certifierar $\neg B \to \neg A$. Konstruktionen

$$\lambda f \to \lambda g \to \lambda a \to g(f(a))$$

blir slutligen certifikatet för kontrapositionslagen. ∎

Antag att vi har ett certifikat p för påståendet

$$(\forall x \in S)\,(\exists y \in T)\,A(x, y). \tag{7}$$

Då gäller för varje $a \in S$ att $p(a)$ certifierar $(\exists y \in T)\,A(a, y)$. Men $p(a)$ är ett par $\langle c, q\rangle$ så att q certifierar $A(a, c)$. Det följer att $\#_2(p(a))$ certifierar $A(a, \#_1(p(a)))$. Således definierar $f(x) = \#_1(p(x))$ en funktion sådan att $\lambda x \to \#_2(p(x))$ certifierar $(\forall x \in S)\,A(x, f(x))$. Vi har alltså en metod att räkna ut y från x.

Påståenden av formen (7) kan exempelvis uttrycka en programspecifikation, där S är typen för invärdena, T är typen för utvärdena och $A(x, y)$ beskriver de villkor programmet skall uppfylla. Certifikatet p ger nu ett program f som uppfyller specifikationen A.

Anmärkning 3.4 *Lagen om det uteslutna tredje* (LUT)

$$A \vee \neg A$$

har ingen en uppenbart giltig BHK-tolkning, eftersom vi måste finna en metod som givet parametrarna i A avgör om A gäller eller inte. Om man begränsar konstruktionerna till beräkningsbara funktioner kan man visa att (LUT) ej har en BHK-tolkning. Det är känt att det finns en primitivt rekursiv funktion T sådan att $T(e, x, t) = 1$ om t beskriver en terminerande beräkning (t en "programspårning") för Turing maskinen e med input x, och sådan $T(e, x, t) = 0$ i annat fall. Man kan genom lämplig kodning betrakta argumenten till T som naturliga tal. Stopproblemet för e och x kan nu uttryckas med formeln

$$H(e, x) =_{\text{def}} (\exists t \in \mathbb{N})\,T(e, x, t) = 1.$$

Enligt lagen om det uteslutna tredje

$$(\forall e \in \mathbb{N})(\forall x \in \mathbb{N})\,H(e, x) \vee \neg H(e, x).$$

Om detta påstående skulle ha ett beräknebart certifikat, så skulle vi kunna avgöra stopproblemet, i strid med Turings resultat att detta är algoritmiskt oavgörbart. Principen om indirekta bevis, *reductio ad absurdum* (RAA) $\neg\neg A \to A$ kan visas vara ekvivalent med LUT inom intutionistisk logik, så denna saknar också en BHK-tolkning. ∎

Vi har sett att ett certifikat för ett påstående kan betraktas som ett program, genom att låta konstruktionerna vara lambda-termer. I nästa kapitel visas hur ett bevis av ett påstående A utfört med s.k. *intuitionistisk logik* ger upphov till ett certifikat a för A.

Den andra ledande idéen, att betrakta påståenden som datatyper, är förverkligad i följande inskränkta mening. Om certifikaten för påståendet A har typen S, och certifikaten för påståendet B har typen T, så gäller

- certifikaten för $A \wedge B$ har typen $S \times T$,
- certifikaten för $A \to B$ har typen $S \to T$
- certifikaten för $A \vee B$ har typen $S + T$,
- certifikaten för $(\forall x \in \mathbb{N})A(x)$ har typen $\mathbb{N} \to S$
- certifikaten för $(\exists x \in \mathbb{N})A(x)$ har typen $\mathbb{N} \times S$

Certifikaten för $s = t$ har typen $\mathbb{N}$. Eftersom $\bot$ inte har några certifikat, kunde vi formellt låta dessa ha typen $\mathbb{N}$ likaså. Det är dock fördelaktigt att införa en särskild *tom typ*. Utöka Definition 2.1 med

> $\emptyset$ är en typ.

Denna typ har inga konstruktorer. Eftersom vi är säkra på att så är fallet, inför vi för varje typ A, och varje term c av typ $\emptyset$, en term $!_A(c)$ av typ A.

I det ovanstående är det klart att påståendet A:s motsvarande typ S kan innehålla termer som inte är certifikat för A. Exempelvis, för $A_1 = (\forall n \in \mathbb{N})\, n = n^2$ är motsvarande typ $\mathbb{N} \to \mathbb{N}$. Denna innehåller termen $\lambda x \to 0$, vilken inte är ett certifikat för A_1.

Det vore önskvärt att kunna identifiera påståenden med datatyper på ett sådant sätt att ett påstående A har ett certifikat om, och endast om, den betraktad som datatyp är icketom. Logikern H.B. Curry insåg att man för (intuitionistisk) satslogik kunde låta en formel exakt svara mot en typ genom att införa typvariabler X som kan stå för godtyckliga typer, tomma eller icke-tomma (se Curry och Feys 1958). W.A. Howard utvidgade 1969 Currys idé till allkvantifierade påståenden. Påstående-som-typer principen kallas ibland även *Curry-Howard isomorfin* (särskilt om ytterligare samband gäller mellan bevisens och programmens reduktionsrelationer; se Simmons (2000)). En fullfjädrad version av påståenden-som-typer principen kom först i och med den svenske logikern Per Martin-Löfs typteori (1971). I detta och senare arbeten framgick hur denna princip kan generaliseras till många andra logiska konstruktioner, till och med sådana som hör till den högre mängdteorin.

För påståenden innehållande individvariabler blir motsvarande typer betydligt mer komplicerade. I exemplet A_1 ovan skall $n = n^2$ svara mot en tom typ $S_n = \emptyset$ då $n \geq 2$ och en typ innehållande 0, t.ex. $S_n = \{0\}$, då $n = 0, 1$. Detta kräver att man måste kunna hantera s.k. *beroende typer*. Typen S_n beror på $n \in \mathbb{N}$, ett element i en annan typ. Vi återkommer till typteorin i Kapitel 7.

Anmärkning 3.5 Det är också möjligt att använda otypade termer som konstruktioner. En viktig metod är Kleenes rekursiva realiserbarhetstolkning, där konstruktionerna är index för partiella rekursiva funktioner (se t.ex. Troelstra och van Dalen 1988). Dessa index kan förstås som program för Turingmaskiner. Det kan hävdas att inga ytterligare konstruktionsmetoder är nödvändiga om man godtar Church-Turing tesen. Emedan konstruktionerna i typad lambda-kalkyl, eller Martin-Löf typteori, kan förstås direkt och oberoende av annan teori, med s.k. *meningsförklaringar* (Martin-Löf 1984), behöver de otypade konstruktionerna förstås inom någon metateori.

Övningar

3.1 Ge BHK-tolkningar för följande påståenden:

(a) $A \wedge B \to A$ och $A \wedge B \to B$

(b) $A \to (B \to A \wedge B)$

(c) $A \to A \vee B$ och $B \to A \vee B$

(d) $(A \to C) \to ((B \to C) \to (A \vee B \to C))$

(e) $A \to (B \to A)$

(f) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

(g) $\bot \to A$

(h) $A(t) \to (\exists x \in D)A(x)$

(i) $(\forall x \in D)(A(x) \to B) \to ((\exists x \in D)A(x) \to B))$, där x ej förkommer fri i B

(j) $(\forall x \in D)A(x) \to A(t)$

(k) $(\forall x \in D)(B \to A(x)) \to (B \to (\forall x \in D)A(x))$, där x ej förkommer fri i B.

Dessa är de logiska *axiomen* för (första ordningens) intuitionistisk logik. Dessutom tillkommer modus ponens regeln och generaliseringsregeln: från A, härled $(\forall x \in D)A$.

3.2 Ge BHK-tolkningar för följande påståenden:

(a) $\neg\neg\neg A \to \neg A$

(b) $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$.

(c) $\neg(\exists x \in D)A(x) \leftrightarrow (\forall x \in D)\neg A(x)$

3.3 Har någon av följande, klassiskt giltiga, formler BHK-tolkningar. Diskutera!

(a) $\neg(\forall x \in D)A(x) \rightarrow (\exists x \in D)\neg A(x)$.

(b) $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$.

4 Intuitionistisk logik

Vi förutsätter att naturlig deduktion för predikatlogik är välbekant (se t.ex. van Dalen (1997) eller Hansen (1997)). Ett system för naturlig deduktion är följande.

Härledningsregler:

$$\frac{A \qquad B}{A \wedge B}\,(\wedge I) \qquad \frac{A \wedge B}{A}\,(\wedge E1) \qquad \frac{A \wedge B}{B}\,(\wedge E2)$$

$$\frac{\begin{array}{c}\overline{A}^{\,h}\\ \vdots\\ B\end{array}}{A \rightarrow B}(\rightarrow I, h) \qquad \frac{A \rightarrow B \qquad A}{B}\,(\rightarrow E)$$

$$\frac{A}{A \vee B}\,(\vee I1) \qquad \frac{B}{A \vee B}\,(\vee I2) \qquad \frac{A \vee B \qquad \begin{array}{c}\overline{A}^{\,h_1}\\ \vdots\\ C\end{array} \qquad \begin{array}{c}\overline{B}^{\,h_2}\\ \vdots\\ C\end{array}}{C}\,(\vee E, h_1, h_2)$$

$$\frac{\bot}{A}\,(\bot E) \qquad \frac{\begin{array}{c}\overline{\neg A}^{\,h}\\ \vdots\\ \bot\end{array}}{A}(RAA, h)$$

Detta är den satslogiska delen av reglerna. Notera att ett avslutat antagande A betecknas $\overline{A}^{\,h}$ i härledningen, där h är en symbol som identifierar vilka antaganden som avslutas på samma gång. Symbolen noteras vid den regel som avslutar och måste vara unik.

$$\frac{A}{(\forall x)A}\,(\forall I) \qquad \frac{(\forall x)A}{A[t/x]}\,(\forall E)$$

$$\frac{A[t/x]}{(\exists x)A}\ (\exists I) \qquad\qquad \frac{(\exists x)A \quad \begin{array}{c}\overline{A}^{h}\\ \vdots \\ C\end{array}}{C}\ (\exists E, h)$$

Reglerna för kvantifikatorerna har vissa begränsningar. För $(\forall I)$ gäller att x ej får vara fri i oavslutade antaganden ovanför A. För $(\exists E)$ gäller att x ej får vara fri i C eller i ostrukna antaganden andra de markerade med h.

De ovanstående reglerna utgör ett system för *klassisk predikatlogik* (utan likhet). Om regeln RAA utesluts, fås ett system för *intuitionistisk predikatlogik*.

Giltighet under BHK-tolkningen. Vi visar att reglerna för det *intuitionistiska* systemet är giltiga under BHK-tolkningen, dvs att om vi har certifikat för påståendena ovanför härledningsstrecket så kan vi konstruera ett certifikat för påståendet under strecket. Notationen $a : A$ används för uttrycka att a är ett certifikat för A. Vi förutsätter att kvantifikationsdomänen är D. Att göra ett antagande A tolkas som att anta att någon variabel x är ett certifikat för A, dvs $x : A$.

$$\frac{a : A \qquad b : B}{\langle a, b\rangle : A \wedge B}\ (\wedge I) \qquad \frac{c : A \wedge B}{\#_1(c) : A}\ (\wedge E1) \qquad \frac{c : A \wedge B}{\#_2(c) : B}\ (\wedge E2)$$

$$\frac{\begin{array}{c}\overline{x : A}^{h}\\ \vdots \\ b : B\end{array}}{\lambda x \to b : A \to B}(\to I, h) \qquad \frac{c : A \to B \qquad a : A}{\mathsf{apply}(c, a) : B}\ (\to E)$$

$$\frac{a : A}{\mathsf{inl}(a) : A \vee B}\ (\vee I1) \qquad \frac{b : B}{\mathsf{inr}(b) : A \vee B}\ (\vee I2)$$

$$\frac{c : A \vee B \qquad \begin{array}{c}\overline{x : A}^{h_1}\\ \vdots \\ d : C\end{array} \qquad \begin{array}{c}\overline{y : B}^{h_2}\\ \vdots \\ e : C\end{array}}{\mathsf{when}(c, \lambda x \to d, \lambda y \to e) : C}\ (\vee E, h_1, h_2)$$

$$\frac{c : \bot}{!(c) : A}\ (\bot E)$$

$$\frac{a : A}{\lambda x \to a : (\forall x)\, A}\ (\forall I) \qquad \frac{c : (\forall x)\, A}{\mathsf{apply}(c, t) : A[t/x]}\ (\forall E)$$

$$\frac{a : A[t/x]}{\langle t, a\rangle : (\exists x)\, A}\ (\exists I) \qquad \frac{c : (\exists x)A \quad \begin{array}{c}\overline{y : A}^{\,h}\\ \vdots \\ d : C\end{array}}{d[\#_1(c), \#_2(c)/x, y] : C}\ (\exists E, h)$$

Vi har visat

Sats 4.1 *Reglerna för intuitionistisk logik är giltiga under BHK-tolkningen.*

Exempel 4.2 Vi härleder kontrapositionslagen i intuitionistisk logik och gör sedan BHK-tolkningen:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{g : \neg B}^{\,h_2} \qquad \dfrac{\overline{f : A \to B}^{\,h_3} \qquad \overline{a : A}^{\,h_1}}{f(a) : B}\ (\to E)}{g(f(a)) : \bot}\ (\to E)}{\lambda a \to g(f(a)) : \neg A}\ (\to I, h_1)}{\lambda g \to \lambda a \to g(f(a)) : \neg B \to \neg A}\ (\to I, h_2)}{\lambda f \to \lambda g \to \lambda a \to g(f(a)) : (A \to B) \to (\neg B \to \neg A)}\ (\to I, h_3)$$

Exempel 4.3 Vi härleder och BHK-tolkar $(\exists x)\,(A \wedge B) \to A \wedge (\exists x)\,B$, där x ej är fri i A.

$$\dfrac{\dfrac{\overline{z : (\exists x)(A \wedge B)}^{\,h_2} \qquad \dfrac{\dfrac{\overline{y : A \wedge B}^{\,h_1}}{\#_1(y) : A}\ (\wedge E1) \qquad \dfrac{\dfrac{\overline{y : A \wedge B}^{\,h_1}}{\#_2(y) : B}\ (\wedge E2)}{\langle x, \#_2(y)\rangle : (\exists x)B}\ (\exists I)}{\langle \#_1(y), \langle x, \#_2(y)\rangle\rangle : A \wedge (\exists x)B}\ (\wedge I)}{\langle \#_1(\#_2(z)), \langle \#_1(z), \#_2(\#_2(z))\rangle\rangle : A \wedge (\exists x)B}\ (\exists E, h_1)}{\lambda z \to \langle \#_1(\#_2(z)), \langle \#_1(z), \#_2(\#_2(z))\rangle\rangle : (\exists x)(A \wedge B) \to A \wedge (\exists x)B}\ (\to I, h_2)$$

(Vad gör certifikatet på sista raden?)

Anmärkning 4.4 * För tydliggöra vilka öppna antaganden som finns i en given position i ett bevisträd brukar man använda en särskild notation, ibland kallad *sekventnotation.* Uttrycket

$$A_1^{h_1}, \ldots, A_n^{h_n} \vdash B.$$

säger att B är bevisat under de öppna antagandena $A_1, \ldots, A_n$. Ordningen mellan antagandena spelar ingen roll. Samma formel får förekomma flera gånger, men markörerna $h_1, \ldots, h_n$ måste vara distinkta. Se (Troelstra och Schwichtenberg 1996) för en formulering av naturlig deduktion i denna notation. Under BHK-tolkningen blir markörerna överflödiga och kan ersättas av variabler

$$x_1 : A_1, \ldots, x_n : A_n \vdash b : B. \tag{8}$$

Modellteori för intuitionistisk logik.* Vi har sett att de intuitionistiska härledningsreglerna är giltiga under BHK-tolkningen. Emellertid finns för denna typ av tolkning ingen fullständighetssats (se van Dalen och Troelstra 1988). Den är däremot fullständig för en annan typ av semantik, s.k. Beth-Kripke-Joyal semantik. Den enklaste formen är Kripke-semantik, och vi hänvisar till (van Dalen 1997) för en introduktion till denna. Denna fyller samma funktion som semantik för klassisk logik och är mycket användbar för att bevisa att ett logiskt påstående ej kan bevisas med intuitionistisk logik. (Man kan exempelvis enkelt ge negativa lösningar till övningarna i 3.3.)

Ickelogiska axiom: induktion. BHK-tolkningar har ovan endast använts för ren logik. Det går även att verifiera vissa icke-logiska axiom med hjälp av BHK-tolkningen, t.ex. induktionsschemat för naturliga tal. Det har vissa begreppsliga fördelar att skriva detta schema som en eliminationsregel. Vi antar nu att kvantifikationsdomänen är $\mathbb{N}$. Induktionsregeln

$$\dfrac{\begin{array}{c}\overline{A(x)}^{\,h}\\ \vdots\\ A(0) \qquad A(\mathsf{S}(x))\end{array}}{A(t)}\ (\text{induktion}, h)$$

kan enkelt ges en BHK-tolkning med hjälp av rekursionsoperatorn:

$$\dfrac{\begin{array}{c}\overline{y : A(x)}^{\,h}\\ \vdots\\ b : A(0) \qquad c : A(\mathsf{S}(x))\end{array}}{\mathsf{rec}(t, b, \lambda x \to \lambda y \to c) : A(t)}\ (\text{induktion}, h).$$

Övningar

4.1 Härled 3.1 (a) - (k) i intuitionistisk logik. (Genomför gärna BHK-tolkningar av härledningarna och jämför med konstruktionerna som erhölls i 3.1.)

4.2 Härled 3.2 i intuitionistisk logik.

4.3 Bevisa $\neg\neg\neg A \leftrightarrow \neg A$ i intuitionistisk logik.

5 Brouwerska motexempel*

I detta avsnitt diskuteras ett vanligt sätt att visa att ett påstående är konstruktivt obevisbart. Avsnittet kan med fördel förbigås vid en första genomläsning.

Det finns vissa enkla logiska principer, som är trivialt sanna i klassisk logik, men som inte kan bevisas konstruktivt. Dessa kallas *allvetenhetsprinciper,* (eng. *principles of omniscience*) och uttrycks i regel i termer av oändliga binära följder. En oändlig binär följd är en funktion $\alpha : \mathbb{N} \to \{0, 1\}$. Vi skall ge två exempel på sådana principer

(LPO) För alla oändliga binära följder α gäller $\exists n\, \alpha_n = 1$ eller $\forall n\, \alpha_n = 0$.

(LLPO) Låt α vara en oändlig binär följd med högst en 1:a. Då gäller $\forall n\, \alpha_{2n} = 0$ eller $\forall n\, \alpha_{2n+1} = 0$.

Det kan lätt bevisas att följande implikation gäller

$$\text{(LPO)} \Rightarrow \text{(LLPO)}.$$

Under en rekursiv realiserbarhetstolkning kan man visa att (LLPO) är falsk, och således är även (LPO) falsk (se t.ex. Troelstra och van Dalen 1988). Det går också att intuitivt övertyga sig om att (LLPO) ej kan ha något konstruktivt bevis. Man kan låta α vara en följd som är definierad av $\alpha_n = 1$ om n är den första position i decimalutvecklingen av π som inleder en svit av 100 7:or, och $\alpha_n = 0$ i annat fall. En konstruktivt bevis av (LLPO) skulle omedelbart ge oss en metod att konstatera att sviten aldrig inleds på en udda position, eller att sviten aldrig inleds på en jämn position.

Om P är ett påstående vars konstruktiva giltighet vi betvivlar, och man kan visa att P medför någon av principerna LPO eller LLPO, så kan vi upphöra att leta efter ett konstruktivt bevis för P.

Övningar

5.1 Visa implikationen LPO $\Rightarrow$ LLPO utan att använda LUT eller RAA.

5.2 Visa konstruktivt att Sats 1.2 implicerar LPO.

6 Klassiska och intuitionistiska bevis

Vi har sett att bevis i intuitionistisk logik har den intressanta egenskapen att program kan extraheras ur dem. De flesta naturligt förekommande bevis (i läroböcker eller matematiska tidskrifter) använder klassisk logik. En naturlig fråga är om det finns någon systematisk metod för att översätta klassiska bevis till konstruktiva bevis. Det är klart att det måste finnas begränsningar i sådan metod med tanke på motexemplen i föregående kapitel. Kurt Gödel och Gerhard Gentzen visade att det finns en sådan metod för rent logiska bevis,

och vissa enkla teorier, under förutsättning att utsagan A som bevisas får ersättas av en (klassiskt) ekvivalent utsaga A^*.

I klassisk logik är de logiska konstanterna (konnektiv och kvantifikatorer) $\exists$ och $\vee$ egentligen onödiga eftersom vi har följande bevisbara ekvivalenser

$$A \vee B \quad \leftrightarrow \quad \neg(\neg A \wedge \neg B) \tag{9}$$
$$(\exists x)C \quad \leftrightarrow \quad \neg(\forall x)\neg C \tag{10}$$

för godtyckliga formler A, B, C. Faktum är att om vi tar högerleden som definitioner av motsvarande logiska konstanter, och endast använder RAA, $(\bot E)$ och introduktions- och eliminationsreglerna för $\wedge, \rightarrow$ och $\forall$, så har vi ett fullständigt system för klassisk predikatlogik. En predikatlogisk formel där de enda logiska konstanterna är $\wedge, \rightarrow$ och $\forall$ kallas $\wedge, \rightarrow, \forall$-*formel.*

Definiera *Gödel-Gentzens negativa översättning* $(\cdot)^*$ med rekursion på $\wedge, \rightarrow, \forall$-formler:

- $\bot^* = \bot$,
- $R(t_1, \ldots, t_n)^* = \neg\neg R(t_1, \ldots, t_n)$, om R är en predikatsymbol,
- $(A \wedge B)^* = A^* \wedge B^*$,
- $(A \rightarrow B)^* = A^* \rightarrow B^*$,
- $((\forall x)C)^* = (\forall x)C^*$.

Det är tydligt att det enda denna översättning åstadkommer är att sätta in två negationstecken framför varje predikatsymbol. Uppenbarligen är bevisbart A ekvivalent med A^* i klassisk logik.

Exempel 6.1 Låt R vara en binär predikatsymbol. En formel $A = (\forall x)(\exists y)(R(x, y) \vee R(y, x))$ är klassiskt ekvivalent med $B = (\forall x)\neg(\forall y)\neg\neg(\neg R(x, y) \wedge \neg R(y, x))$. Denna formels Gödel-Gentzen översättning B^* är

$$(\forall x)\neg(\forall y)\neg\neg(\neg\neg\neg R(x, y) \wedge \neg\neg\neg R(y, x))$$

Sats 6.2 *Låt A vara en $\wedge, \rightarrow, \forall$-formel. Om A kan bevisas i klassisk predikatlogik, så kan A^* bevisas i intuitionistisk predikatlogik.*

Bevis. Ett formellt bevis sker med induktion på härledningar. Eftersom bevisreglerna är identitiska, med undantag av RAA, behöver man endast visa att

$$\neg\neg A^* \rightarrow A^* \tag{11}$$

kan bevisas i intuitionistisk predikatlogik, för varje $\wedge, \rightarrow, \forall$-formel A. Detta kan bevisas med induktion på formeln A. Man behöver härleda följande i intuitionistisk logik

$\vdash \neg\neg\bot \rightarrow \bot$,
$\vdash \neg\neg\neg\neg B \rightarrow \neg\neg B$,
$\neg\neg A \rightarrow A, \neg\neg B \rightarrow B \vdash \neg\neg(A \wedge B) \rightarrow A \wedge B$,
$\neg\neg B \rightarrow B \vdash \neg\neg(A \rightarrow B) \rightarrow (A \rightarrow B)$,
$(\forall x)(\neg\neg A \rightarrow A) \vdash \neg\neg(\forall x)A \rightarrow (\forall x)A$.

Vi lämnar beviset av dessa som en övning. ∎

En $\wedge, \rightarrow, \forall$-formel A kallas *negativ* om varje predikatsymbol i A föregås av en negation. I en sådan formel kommer varje predikatsymbol i A^* föregås av tre negationer. Intuitionistiskt, gäller $\neg\neg\neg B \leftrightarrow \neg B$. Följaktligen gäller att en negativ formel är ekvivalent med sin egen Gödel-Gentzen tolkning. Vi har

Korollarium 6.3 *Låt A vara en negativ $\wedge, \rightarrow, \forall$-formel. Om A kan bevisas i klassisk predikatlogik, så kan den också bevisas i intuitionistisk predikatlogik.*

Sats 6.2 kan ibland utvidgas till vissa teorier T, så att om A kan bevisas i klassisk predikatlogik med axiomen från T, så kan A^* bevisas i intuitionistisk predikatlogik från axiomen T. Ett exempel är när $T = PA$, första ordningens teori för aritmetik, *Peano-aritmetiken,* med följande axiom:

$$
\begin{aligned}
&(\forall x)\, x = x \\
&(\forall x)(\forall y)[x = y \rightarrow y = x] \\
&(\forall x)(\forall y)(\forall z)[x = y \wedge y = z \rightarrow x = z] \\
&(\forall x)(\forall y)[x = y \rightarrow \mathsf{S}(x) = \mathsf{S}(y)] \\
&(\forall x)(\forall y)(\forall z)(\forall u)[x = z \wedge y = u \rightarrow x + y = z + u] \\
&(\forall x)(\forall y)(\forall z)(\forall u)[x = z \wedge y = u \rightarrow x \cdot y = z \cdot u] \\
&(\forall x)\, \neg\mathsf{S}(x) = 0 \\
&(\forall x)(\forall y)[\mathsf{S}(x) = \mathsf{S}(y) \rightarrow x = y] \\
&(\forall x)\, x + 0 = x \\
&(\forall x)(\forall y)\, x + \mathsf{S}(y) = \mathsf{S}(x + y) \\
&(\forall x)\, x \cdot 0 = 0 \\
&(\forall x)(\forall y)\, x \cdot \mathsf{S}(y) = x \cdot y + x \\
&A(0) \wedge (\forall x)[A(x) \rightarrow A(\mathsf{S}(x))] \rightarrow (\forall x)\, A(x),
\end{aligned}
$$

där $A(x)$ är en godtycklig formel i språket $\{=, 0, \mathsf{S}, +, \cdot\}$. För P kvantorfri gäller till och med att: om

$$A = (\forall x)(\exists y)P(x, y)$$

är bevisbar från axiomen i PA med klassisk logik, så är A redan bevisbar med intuitionistisk logik från axiomen (se Troelstra och van Dalen 1988). Eftersom A har formen av en programspecifikation, kan man ibland använda detta, och liknande, resultat för att extrahera program från klassiska bevis (se Schwichtenberg 1999).

Övningar

6.1 Låt $P(x)$ vara en predikatsymbol. Låt $A = (\exists x)\, P(x) \vee (\forall x) \neg P(x)$. Eliminera förekomster av $\vee$ och $\exists$ med hjälp av (9). Bevisa sedan Gödel-Gentzen översättningen av den resulterande formeln i intuitionistisk logik.

7 Martin-Löfs Typteori

I Martin-Löfs typteori (Martin-Löf 1984) tillkommer flera nya begrepp såsom, beroende typer, $\Pi-$ och Σ-typer, för att förverkliga påståenden-som-typer principen.

7.1 Mängdteoretiska summor och produkter.

För att förklara $\Pi-$ och Σ-typer går vi igenom ett par mindre välkända mängdkonstruktioner. Låt I vara en mängd, och låt A_i vara en mängd för varje $i \in I$. Vi säger att A_i $(i \in I)$ är en *familj av mängder*. Den *disjunkta unionen* av denna familj är en mängd av par

$$(\Sigma i \in I)A_i = \{\langle i, a\rangle \mid i \in I, a \in A_i\}.$$

(Vanliga alternativa beteckningar: $\sum_{i\in I} A_i$, $\coprod_{i\in I} A_i$ och $\dot{\cup}_{i\in I} A_i$ — notera pricken.)

Exempel 7.1 Låt $J = \{1, 2, 3\}$ och $B_1 = \{0\}$, $B_2 = \{0, 1\}$, $B_3 = \{1, 2\}$. Då

$$(\Sigma j \in J)B_j = \{\langle 1, 0\rangle, \langle 2, 0\rangle, \langle 2, 1\rangle, \langle 3, 1\rangle, \langle 3, 2\rangle\}.$$

Exempel 7.2 Låt $C_n = \{m \in \mathbb{N} : m \leq n\}$, för $n \in \mathbb{N}$. Då

$$(\Sigma n \in \mathbb{N})C_n = \{\langle n, m\rangle \in \mathbb{N} \times \mathbb{N} \mid m \leq n\}.$$

Betrakta återigen en godtycklig familj A_i $(i \in I)$ av mängder. Den *kartesiska produkten* av denna familj är en mängd av funktioner

$$(\Pi i \in I)A_i = \{f : I \to \cup_{i\in I} A_i \mid (\forall i \in I) f(i) \in A_i\}$$

(Alternativ beteckning: $\prod_{i\in I} A_i$.) Detta är alltså mängden av funktioner f definierade på I, sådana att för varje $i \in I$ tillhör värdet $f(i)$ mängden A_i. Värdemängden A_i beror alltså på argumentet i. Därför kallas ibland konstruktionen för *beroende funktionsmängd* (funktionstyp).

Exempel 7.3 Låt B_j $(j \in J)$ vara en familj av mängder som i Exempel 7.1. Då består $(\Pi j \in J)B_j$ av fyra olika funktioner f, g, h, k, där

$$\begin{array}{llll} f(1)=0 & g(1)=0 & h(1)=0 & k(1)=0 \\ f(2)=0 & g(2)=0 & h(2)=1 & k(2)=1 \\ f(3)=1 & g(3)=2 & h(3)=1 & k(3)=2 \end{array}$$

Exempel 7.4 Låt C_n $(n \in \mathbb{N})$ vara som i Exempel 7.2. Notera att $\cup_{n\in\mathbb{N}} C_n = \mathbb{N}$. Så $(\Pi \in \mathbb{N})C_n = \{f : \mathbb{N} \to \mathbb{N} \mid (\forall n \in \mathbb{N})\, f(n) \in C_n\} = \{f : \mathbb{N} \to \mathbb{N} \mid (\forall n \in \mathbb{N})\, f(n) \leq n\}$.

Anmärkning 7.5 (Binära summor och produkter) Låt $I = \{0, 1\}$ och $C_0 = A$, $C_1 = B$. Då

$$(\Sigma i \in I)C_i = \{\langle 0, a\rangle : a \in A\} \cup \{\langle 1, b\rangle : b \in B\},$$

dvs en disjunkt union av A och B. Vi betecknar denna mängd med $A + B$. Vidare

$$(\Pi i \in I)C_i = \{f : I \to C_0 \cup C_1 \mid f(0) \in C_0, f(1) \in C_1\}.$$

Eftersom funktioner med definitionsområdet I kan betraktas som par, ser vi att $(\Pi i \in I)C_i$ är väsentligen $A \times B$.

Anmärkning 7.6 Antag att $A_i = A$ för alla $i \in I$. Då gäller $(\Sigma i \in I)A_i = (I \times A)$ och $(\Pi i \in I)A_i = (I \to A)$. Vi kan alltså betrakta Σ och Π som generaliseringar av konstruktionerna $\times$ och $\to$.

7.2 Påståenden som mängder

Låt oss nu se hur dessa konstruktioner kan användas för att ge tolkningar av påståenden som *mängder*. Definiera en mängd $E_{m,n}$ som beror på $m, n \in \mathbb{N}$ genom:

$$E_{m,n} = \begin{cases} \{0\} & \text{om } m = n \\ \emptyset & \text{om } m \neq n. \end{cases}$$

Vi har att $E_{m,n}$ är icketom exakt då $m = n$.

Exempel 7.7 Påståendet $(\exists n \in \mathbb{N})\, m = 2n$ är sant om, och endast om, m är ett jämnt naturligt tal. Betrakta nu mängden $S_m = (\Sigma n \in \mathbb{N})\, E_{m,2n}$. Den enda möjligheten för denna att vara icketom är att $(n, 0) \in S_m$ för något n, vilket gäller då $m = 2n$. Vi har

$$S_m \neq \emptyset \Leftrightarrow (\exists n \in \mathbb{N})\, m = 2n.$$

Exempel 7.8 Påståendet $(\forall n \in \mathbb{N})(n+k)^2 = n^2 + 4n + 4$ är sant om, och endast om, $k = 2$. Bilda mängden $T_k = (\Pi n \in \mathbb{N})\, E_{(n+k)^2, n^2+4n+4}$. Denna mängd är icketom omm funktionen $f(n) = 0$ tillhör mängden, dvs. om för alla $n \in \mathbb{N}$: $E_{(n+k)^2, n^2+4n+4} = \{0\}$, dvs. $(n+k)^2 = n^2 + 4n + 4$ gäller. Alltså:

$$(\forall n \in \mathbb{N})(n+k)^2 = n^2 + 4n + 4 \Leftrightarrow T_k \neq \emptyset.$$

Exempel 7.9 Påståendet $(\forall n \in \mathbb{N})[n = 0 \vee (\exists m \in \mathbb{N})n = m + 1]$ uttrycker att varje naturligt tal är 0 eller en efterföljare. Motsvarande mängd blir

$$(\Pi n \in \mathbb{N})\, [E_{n,0} + (\Sigma m \in \mathbb{N}) E_{n,m+1}].$$

Övning: visa att denna innehåller (exakt) ett element.

7.3 Typteorin

Vi har i föregående avsnitt endast givit en *mängdteoretisk beskrivning* av beroende typkonstruktioner. Det är därmed inte säkert att ett element i en mängd svarar mot en algoritmisk konstruktion. För att göra både princip (I) och (II) giltiga måste man åstadkomma algoritmiska motsvarigheter till beroende typer. Detta görs i *Martin-Löfs typteori,* som är en typad lambda-kalkyl som generaliserar kalkylen från Kapitel 2 genom att införa beroende typer. Systemet liknar en BHK-tolkad version av intuitionistisk naturlig deduktion (se Kapitel 4), där det som härleds är *omdömen* av formen $a : A$, vilka kan (valfritt) läsas som (i) *a har typen A* eller (ii) *a är ett certifikat för påståendet A*. Det visar sig att denna idé att identifiera påståenden med typer också leder till begreppsliga förenklingar: $\wedge$ och $\exists$ kan sammanföras till konstruktionen Σ, medan $\rightarrow$ och $\forall$ kan sammanföras till Π-konstruktionen. Disjunktion $\vee$ ges av typkonstruktionen $+$.

En nyhet i detta system är att typer kan bero på andra typer. Att en typ B beror på $z : A$ betyder väsentligen att variabeln z förekommer fri i typuttrycket B. Man kan nu tänka sig ännu en typ C som beror på $y : B$ och $z : A$, osv. Vi skall här utelämna vissa aspekter av den formella hanteringen av beroende typer (se dock avsnitt 7.4 nedan).

Π-typen. Låt B vara en typ som beror på $x : A$. Introduktionsregeln för Π är denna:

$$\frac{\begin{array}{c}\overline{x : A} \\ \vdots \\ b : B\end{array}}{\lambda x \rightarrow b : (\Pi x : A)B}(\Pi I)$$

Eliminationsregeln är

$$\frac{f : (\Pi x : A)B \qquad a : A}{\mathsf{apply}(f, a) : B[a/x]}\ (\Pi E)$$

Tillhörande beräkningsregel är β-regeln $\mathsf{apply}(\lambda x \rightarrow b, a) = b[a/x] : B[a/x]$.

För en typ B som är oberoende av x, ser vi att detta blir BHK-tolkningen av reglerna $(\to I)$ respektive $(\to E)$ för intuitionistisk logik.

Om A betraktas som kvantifikationsdomänen, ser vi att reglerna liknar dem för $\forall$. I intuitionistisk logik är dock omdömena $x : A$ och $a : A$ dolda, eftersom alla variabler och termer automatiskt har typen A. Om vi gör dessa omdömen explicita överensstämmer reglerna.

Σ-typen. Låt B vara en typ som beror på $x : A$. Introduktionsregeln för Σ är

$$\frac{a : A \qquad b : B[a/x]}{\langle a, b\rangle : (\Sigma x : A)B}\ (\Sigma I)$$

Om A förstås som kvantifikationsdomänen och B förstås som ett påstående, ser vi att (ΣI) kan läsas som regeln $(\exists I)$.

Om B ej beror på x (s.a. $B[a/x] = B$), ser vi att regeln (ΣI) har samma form som $(\wedge I)$. (Mängdteoretiskt gäller i detta fall: $(\Sigma x \in A)B = A \times B$, se Anmärkning 7.6.)

Eliminationsregeln för Σ är följande: antag att C är en typ som beror på $z : (\Sigma x : A)B$.

$$\frac{\begin{array}{cc} & \overline{x : A} \quad \overline{y : B} \\ & \vdots \quad \vdots \\ c : (\Sigma x : A)B & d : C[\langle x, y\rangle/z] \end{array}}{\mathsf{split}(c, \lambda x \to \lambda y \to d) : C[c/z]}\qquad (\Sigma E)$$

Tillhörande beräkningsregel är $\mathsf{split}(\langle a, b\rangle, g) = g(a)(b) : C(\langle a, b\rangle)$.

Om C ej beror på z, och man tar hänsyn till dolda omdömen för kvantifikationsdomänen, inser man att regeln (ΣE) överensstämmer med $(\exists E)$.

Om B ej beror på x, $C = A$, $d = x$ har vi att

$$\mathsf{split}(\langle a, b\rangle, \lambda x \to \lambda y \to x) = (\lambda x \to \lambda y \to x)(a)(b) = a.$$

Vi kan betrakta $\mathsf{split}(z, \lambda x \to \lambda y \to x)$ som första projektionen $\#_1(z)$, och därmed har vi generaliserat $(\wedge E1)$. (Övning: hur definieras $\#_2(z)$ i termer av split?)

Anmärkning 7.10 I formaliseringar av typteorin lämpade för implementation, t.ex. Alfa/Agda, förser man termerna med fullständig typinformation. Till exempel skrivs uttrycket $\mathsf{split}(c, \lambda x \to \lambda y \to d)$ fullständigt som

$$\mathsf{split}(A, \lambda x \to B, \lambda z \to C, c, \lambda x \to \lambda y \to d),$$

där $\lambda x \to B$ anger att B är en familj av typer som beror på x (i A).

$+$-typen. Introduktionsregeln för den binära summatypen $(+)$ är

$$\frac{a : A}{\mathsf{inl}(a) : A + B}\ (+I1) \qquad \frac{b : B}{\mathsf{inr}(b) : A + B}\ (+I2)$$

Låt nu C vara en typ som beror på $z : A + B$. Eliminationsregeln ges av

$$\frac{\begin{array}{ccc} & \overline{x : A} & \qquad\qquad \overline{y : B} \\ & \vdots & \qquad\qquad \vdots \\ c : A + B & d : C[\mathsf{inl}(x)/z] & e : C[\mathsf{inr}(x)/z] \end{array}}{\mathsf{when}(c, \lambda x \to d, \lambda y \to e) : C[c/z]}\ (+E)$$

Beräkningsregeln är identisk med (4).

Vi överlåter åt läsaren att visa hur dessa regler generaliserar reglerna för $\vee$.

Anmärkning 7.11 *Funktionell programmering och beroende typer.* Som bekant har typdisciplinen i ett typat programmeringspråk den nyttan att många programmeringsfel kan upptäckas redan vid kompileringstillfället. Beroende typer gör att denna disciplin kan skärpas ytterligare, och att än fler programmeringsfel kan upptäckas. (Ett exempel på ett funktionellt språk som utnyttjar sådana typer är Cayenne (se Augustsson 1998).)

Antag att vi vill skriva ett program f som multiplicerar matriser A och B. För att matrisprodukten AB skall vara väldefinierad måste antalet kolonner i A vara det samma som antalet rader i B. Beteckna med $M(r, k)$ typen av $r \times k$-matriser. Vi kommer alltså att ha $AB : M(r, k)$ när $A : M(r, n)$ och $B : M(n, k)$. Man kan tänka sig att $f(A, B)$ antar ett speciellt felvärde när matrisernas dimensioner är felaktiga. Detta innebär dock att dimensionsfel ej upptäcks vid kompilering. Med beroende typer kan man däremot låta f ha typen

$$(\Pi r : \mathbb{N})(\Pi n : \mathbb{N})(\Pi k : \mathbb{N})[M(r, n) \times M(n, k) \to M(r, k)],$$

varvid det blir omöjligt att skriva ett program som använder f och gör ett dimensionsfel! En viktig fördel med Σ-typer är man inte behöver ett särskilt begrepp för moduler, som t.ex. i SML. En modulspecifikation är en typ av formen

$$(\Sigma f_1 : A_1) \cdots (\Sigma f_n : A_n)\, P(f_1, \ldots, f_n).$$

där $f_1, \ldots, f_n$ är funktioner eller operationer, och $P(f_1, \ldots, f_n)$ ses som ett påstående som beskriver deras inbördes sammanhang och verkningssätt. Ett element $\langle g_1, \ldots, \langle g_n, q\rangle \cdots\rangle$ i denna typ är en implementation av modulen.

Bastyper och rekursiva typer. Vi ger reglerna för typerna $\emptyset$ och $\mathbb{N}$. Det finns ingen introduktionsregel för $\emptyset$, däremot finns eliminationsregeln

$$\frac{c : \emptyset}{!_A(c) : A}\quad (\emptyset E)$$

Den rekursiva typen $\mathbb{N}$ har introduktionsreglerna

$$0 : \mathbb{N} \qquad \frac{a : \mathbb{N}}{\mathsf{S}(a) : \mathbb{N}} \qquad (\mathbb{N}I).$$

Eliminationsregeln är en sammansmältning av rekursionsoperatorn rec och induktionsregeln. Låt C vara en typ som beror på $z : \mathbb{N}$.

$$\frac{\begin{array}{ccc} & & \overline{x : \mathbb{N}} \quad \overline{y : C[x/z]} \\ & & \vdots \quad \vdots \\ t : \mathbb{N} & b : C[0/z] & c : C[\mathsf{S}(x)/z] \end{array}}{\mathsf{rec}(t, b, \lambda x \to \lambda y \to c) : C[t/z]}\ (\mathbb{N}E).$$

Beräkningsregeln är den samma som för rekursionsoperatorn i Kapitel 2.

Vi introducerar en grundläggande beroende typ $L(z)$ som beror på $z : \mathbb{N}$.

$$L(0) = \emptyset \qquad L(\mathsf{S}(x)) = \mathbb{N}.$$

Kombinerar vi denna med funktionen e i Övning 2.4 har vi en beroende typ $L(e(m)(n))$ som är tom exakt då $m \neq n$ (jämför $E_{m,n}$ i ovan). Vi kan nu skriva ner typerna som i Exempel 7.7 – 7.9 i typteorin genom att byta tecknet "$\in$" mot ":".

Man kan enkelt utöka typteorin med regler för olika sorters uppräkningstyper och rekursiva typer av den form som finns i ML eller Haskell. Typen $\mathbb{B}$ av booleska värden är en uppräkningstyp med följande introduktions- och eliminationsregler:

$$\mathsf{tt} : \mathbb{B} \qquad \mathsf{ff} : \mathbb{B} \qquad (\mathbb{B}I)$$

För en typ C som beror på $z : \mathbb{B}$

$$\frac{c : \mathbb{B} \qquad d : C[\mathsf{tt}/z] \qquad e : C[\mathsf{ff}/z]}{\mathsf{if}(c, d, e) : C[c/z]} \qquad (\mathbb{B}E)$$

Beräkningsreglerna är $\mathsf{if}(\mathsf{tt}, c, d) = c$ och $\mathsf{if}(\mathsf{ff}, c, d) = d$.

Som ännu ett exempel på en rekursiv datatyp ger vi introduktionsregler för listor av element av typ A

$$\mathsf{nil} : \mathrm{List}(A) \qquad \frac{a : A \qquad \ell : \mathrm{List}(A)}{\mathsf{cons}(a, \ell) : \mathrm{List}(A)}.$$

Eliminationsregel: formuleringen lämnas som övning.

Varning. En viss försiktighet måste iakttas när man definierar nya rekursiva typer, så att inte egenskapen att alla program terminerar går förlorad. Teorin kan också bli inkonsistent. (Om det exempelvis finns en rekursiv typ A så att $A = A \to \emptyset$, så är det lätt att härleda en motsägelse.) Det finns syntaktiska kriterier som garanterar terminering, se (Dybjer 2000). I system som Agda (se Kapitel 8) har sådana kontrollfunktioner implementerats i bevismaskinen.

Typuniversum. Ett *typuniversum* är en typ U som består av andra typer, s.a. A är en typ för alla $A : U$. Syftet med ett typuniversum är att medge kvantifikation över typer, och familjer av typer. Givet en typ X och $B : X \to U$ så är $\mathsf{apply}(B, x) = B(x)$ en typ som beror av $x : X$. $X \to U$ kan alltså ses som typen av alla typer i U som beror på X.

Exempel 7.12 Man kan införa ett litet universum U' som bara består av typerna $\emptyset$ och $\mathbb{N}$. Då kan man *definiera* L genom rekursion: låt $C = U'$

$$L(z) =_{\text{def}} \mathsf{rec}(z, \emptyset, \lambda x \to \lambda y \to \mathbb{N})$$

Beräkningsreglerna ger $L(0) = \emptyset$ och $L(\mathsf{S}(x)) = \mathbb{N}$. ∎

I standardversioner av Martin-Löfs typteori finns ett universum Set som innehåller bastyperna $\emptyset$ och $\mathbb{N}$ och som dessutom är slutet under bildande av Π-, Σ- och $+$-typer. Det senare betyder att vi har

- $(\Pi x : A)B(x) : \mathsf{Set}$, om $A : \mathsf{Set}$ och $B : A \to \mathsf{Set}$,
- $(\Sigma x : A)B(x) : \mathsf{Set}$, om $A : \mathsf{Set}$ och $B : A \to \mathsf{Set}$,
- $A + B : \mathsf{Set}$, om $A : \mathsf{Set}$ och $B : \mathsf{Set}$.

Exempel 7.13 Med hjälp av ett universum kan man definiera typer som saknar motsvarighet i (S)ML. T.ex. för $n : \mathbb{N}$ och $A : \mathsf{Set}$,

$$\begin{aligned} F(0)(A) &= A \\ F(\mathsf{S}(n))(A) &= A \to F(n)(A) \end{aligned}$$

Vi har $F(n)(A) = A \to A \to \cdots \to A$ (n pilar), så det antal argument som en funktion av denna typ tar *beror* på n. (Jämför exemplet med matriser ovan.) Formellt kan vi definiera F som

$$\lambda n \to \mathsf{rec}(n, \lambda A \to A, \lambda x \to \lambda y \to \lambda A \to (A \to y(A)),$$

där $F : \mathbb{N} \to \mathsf{Set} \to \mathsf{Set}$. (Övning: kontrollera detta med hjälp av beräkningsreglerna.) ∎

Anmärkning 7.14 Typuniversum gör det möjligt att typa polymorfa funktioner (sådana som verkar på samma sätt oavsett typ), t.ex. konkatenering av listor

$$\mathsf{append} : (\Pi A : U)\,[\mathit{List}(A) \to \mathit{List}(A) \to \mathit{List}(A)].$$

Då är $\mathsf{append}(\mathbb{N})$ en konkateneringsfunktion för listor av naturliga tal. ∎

Det kan ibland vara nödvändigt att stapla universum på varandra. Typen $B = (\Sigma A : \mathsf{Set})(\Sigma n : \mathbb{N})F(n)(A)$ tillhör inte Set, eftersom vi inte har $\mathsf{Set} : \mathsf{Set}$. Vi kan således inte bilda typen $F(n)(B)$, eftersom F kräver att B skall vara i Set. För kunna konstruera ett F som kan ta B som argument kan vi införa ett större universum Set_2 som är slutet under Π-, Σ- och $+$-konstruktioner och sådant att

- $\mathsf{Set} : \mathsf{Set}_2$,
- $A : \mathsf{Set}_2$ för alla $A : \mathsf{Set}$.

Samma problem uppstår igen om vi i B byter Set mot Set_2.

Anmärkning 7.15 Den enkla och radikala lösningen att anta $\mathsf{Set} : \mathsf{Set}$ leder dessvärre till en inkonsistent teori (se Martin-Löf 1971). Denna motsägelse kallas *Girards paradox*. I praktiken kan man ofta klara sig med en eller två nivåer av universum.

7.4 Typteorin som formellt system*

I det formella systemet för Martin-Löfs typteori behöver man inte bara regler som styr hur omdömen av formen $a : A$ får fällas. Typerna kan nämligen inte definieras lika grammatiskt enkelt som för lambdakalkylen i Kapitel 2. Att ett uttryck $B(a)$ är en typ kan bero på att vi tidigare konstaterat att $a : A$. Därför behövs en särskild omdömesform för att något uttryck är en typ. T.ex. behöver man regler av formen

$$\dfrac{\begin{array}{c}\overline{x : A}\\ \vdots\\ A \text{ type} \quad B \text{ type}\end{array}}{(\Pi x : A)B \text{ type}} \qquad \dfrac{z : \mathbb{N}}{L(z) \text{ type}}$$

Dessutom skall även beräkningsreglerna ges särskild omdömesform $a = b : A$, som står för att a och b beräknar till samma sak av typ A. På grund konstruktioner som $L(\cdot)$ ovan måste man även kunna uttrycka att två typer beräknar till samma typ, detta görs med $A = B$. Vi har exempelvis $L(e(0)(1)) = L(0) = \emptyset$.

Man använder antagandelistor (se Anmärkning 4.4 ovan) för att administrera öppna antaganden

$$\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n.$$

En sådan lista kallas även *kontext*. De är för typteorin komplicerade eftersom ordningen mellan antagandena har betydelse: A_n kan bero på samtliga variabler $x_1 : A_1, \ldots, x_{n-1} : A_{n-1}$. Detta betyder att, i allmänhet, kan ett antagande tidigt i listan ej strykas förrän de som kommer senare i listan strukits. För att administrera kontext har man ännu en särskild omdömesform. Exempelvis måste man först konstatera att något är en typ innan ett antagande att något variabel har denna typ får läggas till kontext:

$$\frac{\Gamma \vdash B \text{ type}}{\Gamma, y : B \text{ context}}.$$

Antaganderegeln är

$$\frac{x_1 : A_1, \ldots, x_n : A_n \text{ context}}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i}.$$

I implementationer av typteorin hanteras kontext och likheter av systemet, så att regler för dessa behöver man inte bekymra sig om för att kunna använda teorin.

Övningar

7.1. Gör övningen i Exempel 7.9.

7.2. Formulera en eliminationsregel för den rekursiva datatypen $List(A)$.

7.3. Definiera $\#_2(\cdot)$ i termer av split.

8 Implementationer av typteori

En modern implementation av Martin-Löfs typteori är Agda (C. Coquand 1998). Till detta finns ett grafiskt gränssnitt Alfa (Hallgren 1998). Vi beskriver syntaxen för Agda, som är en "osockrad" version av Alfas syntax. Man kan även arbeta på denna nivå i Alfa genom att begagna kommandot "Edit as Text".

- Typtillhörighet betecknas i Agda med dubbla kolontecken $a :: A$.
- Funktionsapplikation $f(a)$ skrivs genom sammanställning $f\ a$ med mellanslag emellan.
- En beroende funktionstyp $(\Pi x : A)B$ skrivs $(x::A)$ `->` B. I motsvarande lambdaabstraktion $\lambda x \to b$ skrivs alltid argumenttypen ut, och blir $\backslash(x::A)$ `->` b. (Då B ej beror på x kan man skriva A `->` B.)
- Den tvåställiga Σ-typen är ett specialfall av en mer allmän *posttyp* (eng. *record type*) eller *generell* Σ*-typ,* där komponenterna är namngivna. Exempelvis svarar $(\Sigma x : A)B$ mot

  $$\texttt{sig \{ fst::}A\texttt{; snd::}B[\texttt{fst}/x]\texttt{ \}}$$

  Låt z vara ett element av denna. För att komma åt komponenten med namnet `fst` skrivs z.`fst`. Paret $\langle a, b\rangle : (\Sigma x : A)B$ skrivs `struct { fst =` a`; snd =` b `}`
- Uppräkningstyper och rekursiva datatyper kan definieras på liknande sätt som i SML, fast än mer flexibelt, med hjälp av konstruktionen `data`. Ur en sådan definition härleds sedan konstruktorer och fallåtskiljare `case`. Vi ger ett par enkla exempel:

  Typen $\mathbb{B}$ definieras av `data tt | ff`

  Konstruktorerna kallas `tt@_` respektive `ff@_`. Svarande mot $\mathsf{if}(c, d, e)$ har vi `case` c `of { tt ->` d`; ff ->` e`}`

  Den tomma typen är `data` utan konstruktorer, och dess fallåtskiljare $!(c)$ blir trivial: `case` c `of { }`

  Naturliga tal $\mathbb{N}$ definieras rekursivt av `Nat :: Set = data zero | S (n::Nat)`. (I detta uttryck betyder `Nat :: Set` att den nya typen tillhör typuniversumet `Set`.)

  Konstruktorerna i denna typ kallas `zero@_` respektive `S@_`. Fallåtskiljaren som svarar mot den rekursiva datatypen `Nat` är `case` c `of {zero ->` u_1`; S n ->` u_2`}`, där n kan vara fri i u_2.
- I Agda finns två typuniversum kallade `Set` och `Type` (närmast svarande mot Set_2 ovan). Dessa universum är slutna under Π-typer, generaliserade Σ-typer och bildande av rekursiva datatyper.

Definitioner i Agda har formen

$$f\ \ (v_1\ \texttt{::}\ \ intyp_1)\ \cdots\ (v_n\ \texttt{::}\ \ intyp_n)\ \texttt{::}\ \ uttyp\ \texttt{=}\ def$$

där f är en identifierare för den funktion som skall definieras, $v_1, \ldots, v_n$ är variabler för argumenten, *uttyp* är funktionsvärdets typ och *def* är den definierande termen.

Exempel 8.1 +-konstruktionen från Kapitel 2 definieras i Agda så här

```
Plus (A::Set)(B::Set) :: Set
  = data inl (x::A) | inr (y::B)
```

`Plus` är alltså en typbildare som tar två typer från universumet `Set` och bildar den disjunkta unionen av dem i samma universum. Fallåtskiljaren `when` kan definieras genom

```
when (A::Set)(B::Set)(C::Set)
     (c:Plus A B)(d::A->C)(e::B->C) :: C
  = case c of {(inl x) -> d x;
               (inr y) -> e y}
```

`when` har alltså sex argument varav de tre första endast är typinformation (vilken undertrycktes i Kapitel 2; se dock Övning 2.3). ■

Exempel 8.2 Rekursionsoperatorn `rec` (Kapitel 2) definieras som

```
rec (A::Set)(a::Nat)(f::A)(g::Nat->A->A) :: A
  = case a of { zero     -> f;
                (S x)    -> g x (rec A x f g)}
```

■

I dessa exempel har vi inte utnyttjat beroende typer på något väsentligt sätt. Typuniversumen gör att vi kan hantera dessa på ett bekvämt sätt. Typen `A -> Set` består av alla alla familjer av typer i `Set` som beror av `A`. Vi kan införa förkortningen `Fam A` för denna typ, i Agda:

```
Fam (A::Set) :: Type
  = A -> Set
```

Exempel 8.3 Den generella rekursionsoperatorn (Kapitel 7) ger även induktionsprincipen, och uttrycks så här

```
rec (A::Fam Nat)
    (a::Nat)(f::A zero@_)(g::(x::Nat)->(y::A x)->A (S@_ x)) :: A a
  = case a of { zero    -> f;
                (S x)   -> g x (rec A x f g)}
```

■

Exempel 8.4 Den beroende typen L definieras genom

```
L (z::Nat) :: Set
  = case z of {zero -> Empty;
               (S x) -> Nat}
```

där `Empty::Set = data`.

Exempel 8.5 En modulspecifikation för funktioner som hittar godtyckligt stora primtal kan skrivas så här:

```
Primefinder :: Set
   = sig {f:: Nat -> Nat;
          correct1 :: (n::Nat) -> Prime (f n);
          correct2 :: (n::Nat) -> LessEq n (f n)}
```

där `Prime` är ett predikat som beskriver när ett tal är primtal, och `LessEq` är relationen $\leq$ på $\mathbb{N}$. För `z::Primefinder` gäller att `z.f` är en funktion som givet n ger ett primtal $p \geq n$. (Övning: definiera `Prime` och `LessEq`.)

9 Modallogik och Kripke-semantik

Modal satslogik kan betraktas som en logik som ligger emellan satslogik och predikatlogik i uttryckskraft. Satslogiken utökas med *modala operatorer* som kan uttrycka exempelvis tidsaspekter av påståendens giltighet: alltid P, någon gång P. Detta görs i sk. *tidslogik* (eng. temporal logic) som fått omfattande användning inom verifikation av system och program. Modala operatorer kan även uttrycka kunskapläget hos olika agenter i ett system: 007 vet P, 008 vet inte att 007 vet att P etc. *Epistemisk logik* behandlar sådan operatorer, och har tillämpning inom verifikation av behörighetssystem m.m.

Vi påminner först om semantiken för satslogik. Låt $\mathcal{P}$ vara en fixerad oändlig mängd av satslogiska variabler. En *värdering* av dessa variabler är en funktion $V : \mathcal{P} \to \{0, 1\}$, där $V(Q) = 1$ betyder att Q hålls för sann, medan $V(Q) = 0$ betyder att Q hålls för falsk. Denna värdering utvidgas som vanligt (se exempelvis Hansen 1997) till satslogiska formler över $\mathcal{P}$ genom en rekursiv definition

$$\begin{aligned} V(A \wedge B) &= \min(V(A), V(B)), \\ V(A \vee B) &= \max(V(A), V(B)), \\ V(\neg A) &= 1 - V(A), \\ V(\bot) &= 0. \end{aligned}$$

Vi definierar $A \to B$ som $\neg A \vee B$. Notera att $V(A \to B) = 1$ omm $V(A) \leq V(B)$.

En *modallogisk modell* $\mathcal{M}$ är en trippel $(W, R, \{V_t\}_{t \in W})$ där W är en icketom mängd, kallad mängden av *möjliga världar*, och där vi för varje värld $t \in W$, har en värdering V_t av de satslogiska variablerna $\mathcal{P}$, och där R är en binär relation på W. Relationen $R(s, t)$ skall uttrycka att världen t är tillgänglig från världen s, och kallas *nåbarhetsrelationen.*

En formel A *är sann i en modallogisk modell* $\mathcal{M}$ omm $V_t(A) = 1$ för alla $t \in W$, dvs. om den är sann i alla möjliga världar. Isåfall skriver vi $\mathcal{M} \models A$.

Exempel 9.1 En modallogisk modell $\mathcal{M} = (W, R, \{V_t\}_{t \in W})$ kan användas för att beskriva hur sanningsvärden varierar över tiden. Ett typiskt exempel uppstår när $W = \mathbb{N}$ och $R(x, y)$ är relationen $x \leq y$. Vi kan då tolka W som en mängd av tidpunkter och R som relationen att ligga före i tiden. Då beskriver V_t de satslogiska variablernas sanningsvärden vid tidpunkten t. ∎

I modallogiken betraktas nya logiska operatorer $\Box$ och $\Diamond$, sk. *modala operatorer. Modallogiska formler* över $\mathcal{P}$ definieras induktivt:

- $\bot$ och varje satslogisk variabel i $\mathcal{P}$ är en modallogisk formel,
- om A är en modallogisk formel, så är $\neg A$, $\Box A$ och $\Diamond A$ modallogiska formler,
- om A och B är modallogiska formler, så är $A \wedge B$, $A \vee B$ är modallogiska formler.

Formlerna $\Box A$ och $\Diamond A$ brukar i allmänhet utläsas *A är nödvändig* respektive *A är möjlig*. De modala operatorerna ges en precis mening i en modallogisk modell $\mathcal{M} = (W, R, \{V_t\}_{t\in W})$: utvidga varje V_t till operatorerna genom

$$\begin{aligned} V_t(\Box A) = 1 &\quad\Leftrightarrow\quad (\forall s \in W)[R(t,s) \to V_s(A) = 1] \\ V_t(\Diamond A) = 1 &\quad\Leftrightarrow\quad (\exists s \in W)[R(t,s) \wedge V_s(A) = 1]. \end{aligned}$$

Tanken med dessa definitioner är i första fallet att A är nödvändig i världen t om A är sann i alla världar s som kan nås från t. I det andra fallet är A möjlig i världen t om A är sann i någon värld s som kan nås från t. Man kallar ibland tolkningen i modallogiska modeller för *möjliga världars-semantik* (eng. *possible world semantics*).

Exempel 9.2 Vi återknyter till modellen $\mathcal{M} = (\mathbb{N}, \leq, \{V_t\}_{t\in\mathbb{N}})$ i Exempel 9.1. I denna modell betyder $V_t(\Box P) = 1$ att $V_s(P) = 1$ för alla $s \geq t$, dvs att P är sann från tidpunkten t och framåt, medan $V_t(\Diamond P) = 1$ betyder att P är sann vid någon tidpunkt $s \geq t$. Antag att för de satslogiska variablerna H_s $(s = 0, 1, 2, \ldots)$ gäller

$$V_t(H_s) = \begin{cases} 1 & \text{om } t \geq s \\ 0 & \text{om } t < s \end{cases}$$

Då gäller $\mathcal{M} \models H_{19} \to \Box P$ omm $V_t(P) = 1$ för alla $t \geq 19$, dvs att P är sann från och med tidpunkten 19. Medan $\mathcal{M} \models \Box(P \vee H_{19})$ betyder att P gäller till och med tidpunkten $18 = 19\text{-}1$. Kombinationen $V_t(\Box\Diamond P) = 1$ betyder att för alla $r \geq t$ finns det $s \geq r$ så att $V_s(P) = 1$, med andra ord, P är sann vid oändligt många tidpunkter efter t. På detta sätt kan modaloperatorerna användas för att beskriva olika tidsaspekter av ett systems tillstånd. ∎

Sats 9.3 *Låt $\mathcal{M} = (W, R, \{V_t\}_{t\in W})$ vara en modallogisk modell. Då gäller för alla världar $t \in W$ och alla modallogiska formler A och B:*

(a) $V_t(\Diamond A) = V_t(\neg\Box\neg A)$,

(b) $V_t(\Box(A \wedge B)) = V_t(\Box A \wedge \Box B)$,

(c) $V_t(\Box(A \to B) \wedge \Box A) \leq V_t(\Box B)$.

Bevis. (a): $V_t(\neg\Box\neg A) = 1$ omm $\neg(\forall s \in W)[R(t,s) \to V_s(\neg A) = 1]$ omm $(\exists s \in W)[R(t,s) \wedge V_s(A) = 1]$.

(b): Övning.

(c): Antag $V_t(\Box(A \to B) \wedge \Box A) = 1$. Då gäller $V_t(\Box(A \to B)) = 1$ och $V_t(\Box A) = 1$, så för alla $s \in W$ med $R(t,s)$: $V_s(A \to B) = 1$ och $V_s(A) = 1$. Följaktligen, $V_s(B) = 1$ för alla s sådana att $R(t,s)$. Alltså $V_t(\Box B) = 1$. ∎

Denna sats medför att följande modallogiska formler är sanna i varje modallogisk modell $\mathcal{M}$:

$$\Diamond A \leftrightarrow \neg\Box\neg A$$
$$\Box(A \wedge B) \leftrightarrow \Box A \wedge \Box B$$
$$\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$$

Allmäna egenskaper hos nåbarhetsrelationen bestämmer till stor del vilka modallogiska axiom som är giltiga. Några egenskaper hos relationen R som brukar betraktas:

- R *reflexiv* omm $(\forall x \in W)\, R(x, x)$.
- R *symmetrisk* omm $(\forall x, y \in W)[R(x, y) \rightarrow R(y, x)]$.
- R *antisymmetrisk* omm $(\forall x, y \in W)[R(x, y) \wedge R(y, x) \rightarrow x = y]$.
- R *transitiv* omm $(\forall x, y, z \in W)[R(x, y) \wedge R(y, z) \rightarrow R(x, z)]$.
- R är en *ekvivalenrelation* omm R reflexiv, transitiv och symmetrisk.
- R är en *partiell ordning* omm R reflexiv, transitiv och antisymmetrisk.
- R är en *linjär ordning* omm R partiell ordning och $(\forall x, y \in W)[R(x, y) \vee R(y, x)]$.

I modallogiska modeller med reflexiv nåbarhetsrelation gäller

$$\Box A \rightarrow A.$$

Är nåbarhetsrelationen transitiv gäller

$$\Box A \rightarrow \Box\Box A.$$

Detta är en direkt konsekvens av nedanstående sats

Sats 9.4 *Låt $\mathcal{M} = (W, R, \{V_t\}_{t \in W})$ vara en modallogisk modell. Då gäller för alla världar $t \in W$ och alla modallogiska formler A:*

(a) Om R är reflexiv, så $V_t(\Box A) \leq V_t(A)$.

(b) Om R är transitiv, så $V_t(\Box A) \leq V_t(\Box\Box A)$.

Bevis. (a): $V_t(\Box A) = 1$ medför speciellt att $R(t, t) \rightarrow V_t(A) = 1$. Om R är reflexiv, så $R(t, t)$ och följaktligen $V_t(A) = 1$.
(b): Övning. ∎

Anmärkning 9.5 Man kan även betrakta modallogiska modeller med flera nåbarhetsrelationer $\mathcal{M} = (W, R_1, \ldots, R_n, \{V_t\}_{t \in W})$. Till varje relation R_i hör ett par av operatorer $\Box_i$ och $\Diamond_i$ definierade på samma sätt som $\Box$ och $\Diamond$ är i termer av R. (Se övning 9.8 nedan.)

Exempel på sådana modeller uppstår i *epistemisk logik,* där $\Box_k A$ betyder att agent k vet A. De sk. *introspektionsaxiomen* för varje agent k är

$$\Box_k(A \to B) \wedge \Box_k A \to \Box_k B$$
$$\Box_k A \to A$$
$$\Box_k A \to \Box_k \Box_k A$$
$$\neg \Box_k A \to \Box_k \neg \Box_k A.$$

Man kan visa att dessa axiom är uppfyllda om varje R_k är en ekvivalensrelation (Övning 9.4).

Modeller för intuitionistisk logik. En modallogisk modell $\mathcal{M} = (T, \leq, \{V_t\}_{t \in T})$ kallas *Kripke-modell* om $\leq$ är en partiell ordning och för alla satslogiska variabler Q:

$$s \leq t \Rightarrow V_s(Q) \leq V_t(Q). \tag{12}$$

Denna *monotonitetsegenskap* (12) säger att satslogiska variabler som är en sanna i en värld fortfar att vara sanna i senare världar. Vi skall visa att monotonitetsegenskapen kan utvidgas till en mängd formler. Definiera *intuitionistisk implikation* genom

$$(A \supset B) \equiv \Box(A \to B).$$

Intuitionistisk negation definieras av $\sim A \equiv (A \supset \bot)$. Vi definierar induktivt en delmängd av de modallogiska formlerna som vi kallar *intuitionistiska formler*

- $\bot$ och varje satslogisk variabel är intuitionistiska formler,
- Om A och B är intuitionistiska formler, så är $A \wedge B$, $A \vee B$ och $A \supset B$ är intuitionistiska formler.

Notera att i en Kripke-modell gäller $V_t(A \supset B) = 1$ omm $V_t(\Box(A \to B)) = 1$ omm för alla $s \geq t$:

$$V_s(A) = 1 \Rightarrow V_s(B) = 1.$$

Lemma 9.6 *Låt $\mathcal{M}$ vara en Kripke-modell. Då gäller för alla intuitionistiska formler A:*

$$s \leq t \Rightarrow V_s(A) \leq V_t(A).$$

Bevis. Induktion på A. Basfallen är klara enligt definitionen av en Kripke-modell. Fallen då $A = B \wedge C$ och $A = B \vee C$ följer eftersom max och min är monotona funktioner. Betrakta nu $A = B \supset C$ och antag att $s \leq t$ och $V_s(A) = 1$. Detta betyder att för att alla $r \geq s$,

$$V_r(B) = 1 \Rightarrow V_r(C) = 1.$$

Speciellt gäller detta för alla $r \geq t$, eftersom $\geq$ är transitiv. Vi har alltså visat $V_t(A) = 1$. Induktionen är fullbordad. ∎

Vi beskriver ett naturlig deduktionssystem för intuitionistisk satslogik. Man betecknar med $\Gamma \vdash A$ att A är bevisbar i intuitionistisk logik under de öppna antagandena $\Gamma = A_1, A_2, \ldots, A_n$.

$A_1, \ldots, A_n \vdash A_i$ (antaganderegeln)

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \qquad \frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}$$

Sats 9.7 *Låt $\mathcal{M}$ vara en Kripke-modell. Om A är bevisbar i intuitionistisk satslogik, så gäller $\mathcal{M} \models A$.*

Bevis. Om $\Gamma = A_1, \ldots, A_n$, låt $\Gamma \to B$ beteckna $A_1 \wedge \cdots \wedge A_n \to B$. För $n = 0$ är $\Gamma \to B$ helt enkelt B. Vi visar med induktion på härledningar att för intuitionistiska formler Γ, B

$$\Gamma \vdash B \Longrightarrow \mathcal{M} \models \Gamma \to B \tag{13}$$

för alla Γ, B. (För $\vdash A$ följer därmed att $\mathcal{M} \models A$.)

($\supset I$)-regeln: Antag $\mathcal{M} \models \Gamma, A \to B$, där $\Gamma = A_1, \ldots, A_n$. Vi har att bevisa att $\mathcal{M} \models \Gamma \to \Box(A \to B)$. Låt t vara godtycklig sådan att $V_t(A_1 \wedge \cdots \wedge A_n) = 1$. Låt nu $s \geq t$ och antag $V_s(A) = 1$. Enligt monotonitetsegenskapen (Lemma 9.6) hos intutionistiska formler gäller $V_s(A_1 \wedge \cdots \wedge A_n) = 1$. Därmed följer att $V_s(B) = 1$ från det första antagandet. Vi har visat att $V_t(\Box(A \to B)) = 1$.

Vi lämnar verifikationen av de övriga reglerna som en övning. ∎

Denna sats kan användas för att visa att en formel A inte är bevisbar i intuitionistisk satslogik: det räcker att finna en Kripke-modell $\mathcal{M}$ med någon värld t så att $V_t(A) = 0$.

Exempel 9.8 Vi visar att $Q \vee {\sim}Q$ (RAA) inte är bevisbar i intuitionistisk satslogik. Vi betraktar Q som satslogisk variabel. Konstruera en modallogisk modell $\mathcal{M} = (W, \leq, \{V_t\}_{t \in W})$ med två möjliga världar $W = \{0, 1\}$ (tidpunkter), där $0 \leq 1$, och värderingar där $V_0(Q) = 0$ och $V_1(Q) = 1$. (Vi behöver ej bry oss om andra variabler än Q.) Då gäller

$$V_0(Q \vee {\sim}Q) = \max(V_0(Q), V_0({\sim}Q)) = V_0({\sim}Q) = V_0(\Box\neg Q)$$

Detta uttryck är 1 omm för alla $t \geq 0$: $V_t(\neg Q) = 1$. Men $V_1(\neg Q) = 1 - V_1(Q) = 0$, så $V_0(Q \vee {\sim}Q) = 0$. Alltså $\mathcal{M} \not\models Q \vee {\sim}Q$.

Anmärkning 9.9 När en Kripke-modell $\mathcal{M} = (W, \leq, \{V_t\}_{t \in W})$ är given, brukar utsagan att A är sann i en värld s ofta skrivas med så kallad *forcing-notation:*

$$s \Vdash A \Longleftrightarrow_{\text{def}} V_s(A) = 1$$

($s \Vdash A$ utläses s *tvingar* A.) Med denna notation

$$\begin{aligned}
&s \nVdash \bot \\
&s \Vdash A \wedge B \Leftrightarrow s \Vdash A \text{ and } s \Vdash B \\
&s \Vdash A \vee B \Leftrightarrow s \Vdash A \text{ or } s \Vdash B \\
&s \Vdash A \supset B \Leftrightarrow (\forall t \geq s)\,[t \Vdash A \Rightarrow t \Vdash B] \\
&s \Vdash \sim A \Leftrightarrow (\forall t \geq s)\,[t \nVdash A].
\end{aligned}$$

Det är möjligt att utvidga Kripke-semantiken till intuitionistisk predikatlogik, och bevisa en sundhets- och fullständighetssats för denna. Vi hänvisar till Troelstra och van Dalen (1988) eller van Dalen (1997) för en utförlig framställning.

Övningar

9.1 Vad betyder $V_t(\Diamond\Box P) = 1$ i en modell som i Exempel 9.1?

9.2

(a) Visa att $\Diamond(A \vee B) \leftrightarrow \Diamond A \vee \Diamond B$ gäller i varje modallogisk modell för alla A och B

(b) Visa att $\Box(A \vee B) \leftrightarrow \Box A \vee \Box B$ är falsk i någon modallogisk modell för några lämpliga formler A och B.

9.3 Låt P och Q vara satslogiska variabler.

(a) Visa att $\sim\sim P \supset P$ inte är bevisbar i intuitionistisk logik.

(b) Visa att $(P \supset Q) \vee (Q \supset P)$ inte är bevisbar i intuitionistisk logik.

9.4 Antag att $\mathcal{M} = (W, R, \{V_t\}_{t \in W})$ är en modallogisk modell. Låt A vara en godtycklig modallogisk formel.

(a) Visa att om R är reflexiv, så gäller $\mathcal{M} \models \Box A \rightarrow A$.

(b) Visa att om R är transitiv, så gäller $\mathcal{M} \models \Box A \rightarrow \Box\Box A$.

(c) Visa att om R är symmetrisk, så gäller $\mathcal{M} \models \Diamond\Box A \rightarrow A$.

(d) Visa att om R är en ekvivalensrelation, så uppfyller $\Box$ introspektionsaxiomen i $\mathcal{M}$.

(e) Visa att om R är en linjär ordning, så gäller $\mathcal{M} \models \Diamond\Box\Diamond A \leftrightarrow \Box\Diamond A$

9.5 Låt $\mathcal{M}$ vara en modallogisk modell där nåbarhetsrelationen är en linjär ordning. Låt $u \in \{\Box, \Diamond\}^*$ beteckna en godtycklig sträng av modaloperatorer. Om A är en modallogisk formel, så är $u\,A$ en modallogisk formel. Visa att $u\,A$ är ekvivalent med någon av följande formler

$$A \qquad \Box A \qquad \Diamond A \qquad \Box\Diamond A \qquad \Diamond\Box A.$$

(Ledning: Använd 9.4.)

9.6* Låt $\mathcal{M} = (W, R, \{V_t\}_{t\in W})$ vara en modallogisk modell. Låt P vara en satslogisk variabel. Visa att om $\mathcal{M} \models \Box P \to P$, för alla val av värderingarna V_t, så måste R vara reflexiv. Detta är omvändningen till 9.4(a). Bevisa liknande omvändningar till 9.4(b) och 9.4(c).

9.7 Antag att $\mathcal{M} = (W, R, \{V_t\}_{t\in W})$ är en modallogisk modell, där mängden av de möjliga världarna är $W = \mathbb{Q}$ och där nåbarhetsrelationen $R(x, y)$ är $x < y$. Observera att R ej är reflexiv. Visa att trots detta gäller för alla modallogiska formler A:

(a) $\mathcal{M} \models \Box\Box A \leftrightarrow \Box A$ (Ledning: använd $\mathbb{Q}$:s egenskap att $x < y \to (\exists z)x < z < y$)

(b) $\mathcal{M} \models \Diamond\Box\Diamond A \leftrightarrow \Box\Diamond A$.

9.8 Betrakta en modallogisk modell $\mathcal{M} = (\mathbb{N}, \leq, \{V_t\}_{t\in\mathbb{N}})$ där H_t $(t = 0, 1, 2, \ldots)$ är satslogiska variabler med

$$V_t(H_s) = \begin{cases} 1 & \text{om } t \geq s, \\ 0 & \text{om } t < s. \end{cases}$$

(Jämför Exempel 9.2.) Uttryck följande med modallogiska formler som använder fixt antal H-variabler (oberoende av m och n).

(a) P gäller vid alla tidpunkter i intervallet $[m, n] = \{m, m + 1, m + 2, \ldots, n\}$.

(b) P gäller vid någon tidpunkt i intervallet $[m, n]$.

9.9* Antag att $\mathcal{M} = (\mathbb{Q}, <, >, \{V_t\}_{t\in\mathbb{Q}})$ är en modallogisk modell med två nåbarhetsrelationer $<$ och $>$. Definiera från relationen $<$ modaloperatorerna $\Box_+$ och $\Diamond_+$, och från $>$ modaloperatorerna $\Box_-$ och $\Diamond_-$. Undersök vilka förenklingsregler finns det för dessa operatorer. För vilka formler $B = u\,A$, där prefixet $u \in \{\Box_+, \Diamond_+, \Box_-, \Diamond_-\}^*$, går det att skriva om B till en satslogisk kombination av formler med kortare prefix? (Jämför Övning 9.5.)

.

Litteraturförteckning

En bra inledning till lambdakalkyl är Hindley och Seldin (1986), som främst behandlar den otypade och enkelt typade kalkylen. För ett djupare studium av otypad kalkyl är Barendregt (1977) standardreferensen. Troelstra och Schwichtenberg (1996) är en grundlig introduktion till både typad lambdakalkyl och bevisteori för intuitionistisk predikatlogik. Barendregt (1992) behandlar typad lambdakalkyl med beroende typer. Martin-Löf (1984) ger en elegant framställning av sin egen typteori med meningsförklaringar och filosofiska motiveringar. Originaluppsatsen (Martin-Löf 1971) kan också rekommenderas starkt. För en introduktion till typteorin och dess datalogiska tillämpningar, se Coquand mfl. (1994), Nordström mfl. (1990) eller Thompson (1991). En kort inledning till intuitionistisk logik och dess semantik finns i (van Dalen 1997). Standardverket om konstruktiva logiska system är (van Dalen och Troelstra 1988).

Augustsson, Lennart (1998), Cayenne - a language with dependent types. *Proc. of the International Conference on Functional Programming (ICFP'98).* ACM Press, September 1998. [Se även URL: `www.cs.chalmers.se/~augustss/`.]

Barendregt, H.P. (1977), *The Lambda Calculus.* North-Holland.

Barendregt, H.P. (1992), Lambda calculi with types. *Handbook of Logic in Computer Science*, vol 2. sid. 118 – 279. Oxford University Press.

Beeson, Michael (1985), *Foundations of Constructive Mathematics.* Springer-Verlag.

Bishop, Errett och Bridges, Douglas S. (1985), *Constructive Analysis.* Springer-Verlag.

Bridges, Douglas S. och Richman, Fred (1987), *Varieties of Constructive Mathematics.* London Mathematical Society Lecture Notes, Vol. 97. Cambridge University Press.

Coquand, Catarina. *Agda.* URL: `www.cs.chalmers.se/~catarina/agda/`

Coquand, Thierry, Nordström, Bengt, Smith, Jan och von Sydow, Björn (1994), Type theory and Programming. *The EATCS bulletin,* February 1994. [Även tillgänglig på URL: `www.cs.chalmers.se/~smith/`.]

Curry, H.B. och Feys, R. (1958), *Combinatory Logic.* North-Holland.

Dybjer, P. (2000), A general formulation of simultaneous inductive-recursive definitions in type theory, *Journal of Symbolic Logic* 65(2000).

Hallgren, Thomas. *The Proof Editor Alfa.* URL: `www.cs.chalmers.se/~hallgren/Alfa/`

Hansen, Kaj B. (1997), *Grundläggande Logik.* Studentlitteratur.

Hansen, Michael R. och Rischel, Hans (1999), *Introduction to Programming Using SML.* Addison-Wesley.

Heyting, Arend (1971), *Intuitionism.* North-Holland.

Hindley, J.R. och Seldin, J.P. (1986), *Introduction to Combinators and Lambda Calculus.* Cambridge University Press.

Howard, W.A. (1980), The Formulae-as-Types Notion of Construction. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* J.P. Seldin and J.R. Hindley (red.), Academic Press, sid. 479 - 490.

Martin-Löf, Per (1971), *An intuitionistic theory of types.* Inst. rapport, Stockholms universitet. Publicerad i (Sambin och Smith 1998).

Martin-Löf, Per (1984), *Intuitionistic Type Theory.* Bibliopolis.

Martin-Löf, Per (1985), Constructive Mathematics and Computer Programming. *Mathematical Logic and Computer Languages.* C.A.R. Hoare and J.C. Shepherdson (red.). Prentice-Hall.

Mints, Grigori (2000), *A Short Introduction to Intuitionistic Logic.* Kluwer Academic/Plenum Publishers.

Nordström, Bengt, Peterson, Kent och Smith, Jan (1990), *Programming in Martin-Löf's Type Theory.* Oxford University Press. [Boken är ur tryck, men finns tillgänglig på URL: `www.cs.chalmers.se/Cs/Research/Logic/book/` .]

Nordström, Bengt, Peterson, Kent och Smith, Jan (2000), Martin-Löf's type theory. *Handbook of Logic in Computer Science*, Vol. 5. Oxford University Press.

Salling, Lennart (1999), *Formella språk, automater och beräkningar.* Eget förlag, Uppsala.

Sambin, Giovanni och Smith, Jan (1998), *Twenty-Five Years of Type Theory.* Oxford University Press.

Schwichtenberg, Helmut (1999), *Classical Proofs and Programs.* Marktoberdorf Summer School '99. Tillgänglig på URL: `www.mathematik.uni-muenchen.de/~schwicht/`

Simmons, Harold (2000), *Derivation and Computation.* Cambridge University Press.

Thompson, Simon (1991), *Type Theory and Functional Programming.* Addison-Wesley.

Troelstra, Anne S. och Schwichtenberg, Helmut (1996), *Basic Proof Theory,* Cambridge University Press.

Troelstra, Anne S. och van Dalen, Dirk (1988), *Constructivism in Mathematics, Vol. I & II.* North-Holland.

van Dalen, Dirk (1997), *Logic and Structure,* 3:e upplagan. Springer.