

Induktive og rekursive definitioner

Denne note omhandler matematiske objekter, som formelt er opbygget fra et antal basale byggesten, kaldet *basistilfælde* eller blot *basis*, ved gentagen brug af et antal *regler*. Vi starter med nogle eksempler:

- *Sandhedsværdierne* er opbygget fra de to basistilfælde TRUE og FALSE – uden brug af regler. Derfor er der kun to sandhedsværdier, nemlig TRUE og FALSE selv. De skrives ofte også *true* og *false*.
- *De naturlige tal* er opbygget fra basis ZERO ved brug af én regel: hvis n er et naturligt tal, så er $\text{SUCC}(n)$ det også. De naturlige tal ser derfor således ud:

$$\begin{aligned} & \text{ZERO} \\ & \text{SUCC}(\text{ZERO}) \\ & \text{SUCC}(\text{SUCC}(\text{ZERO})) \\ & \text{SUCC}(\text{SUCC}(\text{SUCC}(\text{ZERO}))) \\ & \text{SUCC}(\text{SUCC}(\text{SUCC}(\text{SUCC}(\text{ZERO})))) \\ & \vdots \end{aligned} \tag{1}$$

Standard-notationen er velkendt: ZERO skrives 0 og $\text{SUCC}(n)$ skrives $1 + n$, ligesom vi normalt bruger decimal notation for hvert af tallene. $\text{SUCC}(\text{SUCC}(\text{SUCC}(\text{ZERO})))$ skrives f.eks. 3.

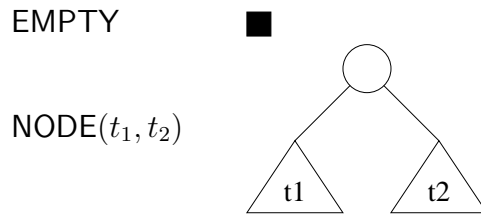
- *Sekvenser over en mængde A* er opbygget fra basis NIL, den tomme sekvens, ved brug af én regel: hvis $a \in A$, og s er en sekvens over A , så er $\text{CONS}(a, s)$, dvs. a efterfulgt af s , også en sekvens over A . Hvis $A = \{a, b\}$ ser sekvenserne over A således ud:

$$\begin{aligned} & \text{NIL} \\ & \text{CONS}(a, \text{NIL}) \\ & \text{CONS}(b, \text{NIL}) \\ & \text{CONS}(a, \text{CONS}(a, \text{NIL})) \\ & \text{CONS}(a, \text{CONS}(b, \text{NIL})) \\ & \text{CONS}(b, \text{CONS}(a, \text{NIL})) \\ & \text{CONS}(b, \text{CONS}(b, \text{NIL})) \\ & \text{CONS}(a, \text{CONS}(a, \text{CONS}(a, \text{NIL}))) \\ & \text{CONS}(a, \text{CONS}(a, \text{CONS}(b, \text{NIL}))) \\ & \vdots \end{aligned} \tag{2}$$

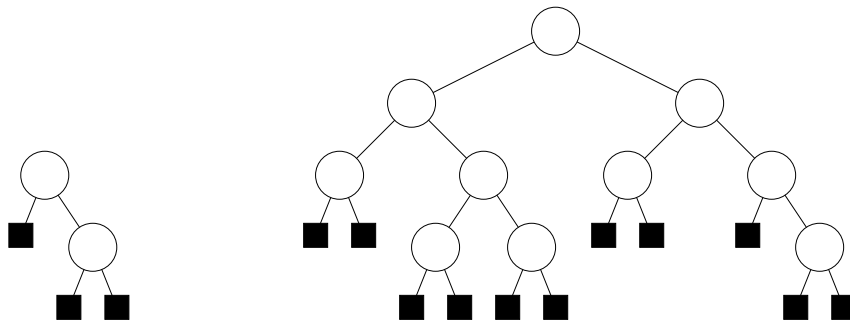
For den tomme sekvens bruges også notationen Λ (stort lambda), λ (lille lambda) eller ϵ (epsilon) – kært barn har mange navne. Sekvensen $\text{CONS}(a, s)$ skrives normalt blot as . Den sidste sekvens ovenfor skrives normalt aab , idet også det afsluttende NIL udelades.

- *Binære træer* er opbygget fra basis **EMPTY**, det tomme træ, ved brug af én regel: hvis t_1 og t_2 er binære træer, så er $\text{NODE}(t_1, t_2)$ det også. Dette træ består af en knude, kaldet *roden*, og har t_1 som *venstre undertræ* og t_2 som *højre undertræ*.

Normalt beskrives binære træer med figurer så som



Det binære træ $\text{NODE}(\text{EMPTY}, \text{NODE}(\text{EMPTY}, \text{EMPTY}))$ afbildes som på figuren forneden til venstre.



Det større træ til højre på figuren skrives formelt

$$\begin{aligned}
 &\text{NODE}(\text{NODE}(\text{NODE}(\text{EMPTY}, \\
 &\quad \text{EMPTY}), \\
 &\quad \text{NODE}(\text{NODE}(\text{EMPTY}, \\
 &\quad \quad \text{EMPTY}), \\
 &\quad \quad \text{NODE}(\text{EMPTY}, \\
 &\quad \quad \quad \text{EMPTY}))), \\
 &\quad \text{NODE}(\text{NODE}(\text{EMPTY}, \\
 &\quad \quad \text{EMPTY}), \\
 &\quad \quad \text{NODE}(\text{EMPTY}, \\
 &\quad \quad \quad \text{NODE}(\text{EMPTY}, \\
 &\quad \quad \quad \quad \text{EMPTY}))))
 \end{aligned} \tag{3}$$

Bemærk, hvorledes de 10 forekomster af **NODE** svarer til de 10 hvide cirkler i figuren, mens de 11 forekomster af **EMPTY** svarer til de 11 sorte firkanter.

Sandhedsværdier

I programmeringssproget SML kan sandhedsværdierne defineres som følger:

$$\mathbf{datatype} \textit{ bool} = \text{TRUE} \mid \text{FALSE} \quad (4)$$

Datotypen *bool* har nu to værdier og svarer til mængden $\mathcal{B} = \{\textit{true}, \textit{false}\}$. I SML kan datotypen bruges til at definere de logiske operationer:

$$\begin{aligned} \mathbf{fun} \textit{ Not}(\text{TRUE}) &= \text{FALSE} \\ &| \textit{ Not}(\text{FALSE}) = \text{TRUE} \\ \mathbf{fun} \textit{ Or}(\text{TRUE}, \text{TRUE}) &= \text{TRUE} \\ &| \textit{ Or}(\text{TRUE}, \text{FALSE}) = \text{TRUE} \\ &| \textit{ Or}(\text{FALSE}, \text{TRUE}) = \text{TRUE} \\ &| \textit{ Or}(\text{FALSE}, \text{FALSE}) = \text{FALSE} \end{aligned} \quad (5)$$

Dette ligner jo til forveksling sandhedstabellerne fra Martin. SML svarer:

$$\begin{aligned} \mathbf{val} \textit{ Not} &= \mathbf{fn} : \textit{ bool} \rightarrow \textit{ bool} \\ \mathbf{val} \textit{ Or} &= \mathbf{fn} : \textit{ bool} \times \textit{ bool} \rightarrow \textit{ bool} \end{aligned} \quad (6)$$

Dvs. *Not* er en funktion $\textit{ bool} \rightarrow \textit{ bool}$, mens *Or* er en funktion $\textit{ bool} \times \textit{ bool} \rightarrow \textit{ bool}$.

Som en sidebemærkning kan vi notere os at funktionen *Or* kan defineres mere kortfattet således:

$$\begin{aligned} \mathbf{fun} \textit{ Or}(\text{TRUE}, b) &= \text{TRUE} \\ &| \textit{ Or}(\text{FALSE}, b) = b \end{aligned} \quad (7)$$

I denne definition splittes der kun op i de to tilfælde, som første argument giver anledning til. Som vi vil se nedenfor er denne form typisk.

Opgave 1 Definer operationerne *And* (\wedge) og *Implies* (\rightarrow). Begge skal være funktioner af samme type som *Or*. Afprøv funktionerne, f.eks. ved at udføre kaldet *Implies*(FALSE, FALSE);.

Naturlige tal

De naturlige tal defineres i SML som følger:

$$\mathbf{datatype} \text{ nat} = \mathbf{ZERO} \mid \mathbf{SUCC} \text{ of } \text{nat} \quad (8)$$

Datatypesen *nat* omfatter nu værdien **ZERO** og alle værdier på formen “**SUCCESSOR** of a *nat*”, dvs. **SUCC(ZERO)**, **SUCC(SUCC(ZERO))**, osv. Dermed svarer værdierne i *nat* til mængden \mathcal{N} af naturlige tal.

Opgave 2 Definer funktionen $Iszero : nat \rightarrow bool$, der returnerer **TRUE** på input **ZERO** og **FALSE** på input, der ikke er **ZERO**. Afprøv definitionen.

Vi definerer fordobling således:

$$\begin{aligned} \mathbf{fun} \text{ Double}(\mathbf{ZERO}) &= \mathbf{ZERO} \\ &| \text{ Double}(\mathbf{SUCC}(n)) = \mathbf{SUCC}(\mathbf{SUCC}(\text{Double}(n))) \end{aligned} \quad (9)$$

Dette skal forstås som følger: Vi vil gerne fordoble et naturligt tal. Hvis vores naturlige tal er 0, bliver resultatet 0. Hvis vores naturlige tal er efterfølger til tallet n , altså $1 + n$, bliver resultatet efterfølgeren til efterfølgeren til det dobbelte af n , altså $2 + 2 \cdot n$. Med almindelig matematisk notation ville definitionen se således ud:

$$\begin{aligned} \text{Double}(0) &= 0 \\ \text{Double}(1 + n) &= 2 + (\text{Double}(n)) \end{aligned} \quad (10)$$

Bemærk at funktionen $\text{Double} : \mathcal{N} \rightarrow \mathcal{N}$, som vi er ved at definere, forekommer igen (“it recurs”) på højre-siden af lighedstegnet. Derfor kaldes definitionen *rekursiv*. Her skal man naturligvis være varsom. Ikke alle rekursive forskrifter som den ovenfor giver anledning til veldefinerede funktioner. Det er afgørende, at

- (i) Funktionen defineres direkte, dvs. uden rekursive kald, på basis (her: **ZERO** eller 0).
- (ii) Når funktionen defineres på en værdi, der er opstået ved brug af en regel (her: **SUCC**(n) for n af type *nat* eller $1 + n$ for $n \in \mathcal{N}$), så sker eventuelle rekursive kald på n . Dette sikrer, at rekursionen i hvert trin nærmer sig basis, hvor den stopper pga. punkt (i).

F.eks. kaldes *Double* rekursivt på værdien 2, så på 1 og til sidst på 0 under udregning af at $\text{Double}(3) = 6$. For datatypesen ser beregningen sådan ud:

$$\begin{aligned} &\text{Double}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{ZERO})))) \\ &= \mathbf{SUCC}(\mathbf{SUCC}(\text{Double}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{ZERO})))))) \\ &= \mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\text{Double}(\mathbf{SUCC}(\mathbf{ZERO})))))) \\ &= \mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\text{Double}(\mathbf{ZERO}))))))) \\ &= \mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{SUCC}(\mathbf{ZERO})))))) \end{aligned}$$

Sammenlign med hvad der sker, hvis vi forsøger at definere *Double* som følger

$$\begin{aligned} Double(0) &= 0 \\ Double(1 + n) &= Double(1 + n) \end{aligned} \tag{11}$$

Disse ligninger er også sande, men som en definition går det ikke:

Opgave 3 Skriv den tilsvarende definition i SML og prøv at udføre kaldet

$$Double(SUCC(SUCC(SUCC(ZERO)))); \tag{12}$$

Hvad sker der og hvorfor? Ret definitionen til den korrekte (9) og afprøv den.

Operationer som addition og multiplikation kan defineres på samme måde som *Or* (jf. (7)):

$$\begin{aligned} \mathbf{fun} \textit{Add}(\mathbf{ZERO}, m) &= m \\ &| \textit{Add}(\mathbf{SUCC}(n), m) = \mathbf{SUCC}(\textit{Add}(n, m)) \\ \mathbf{fun} \textit{Mult}(\mathbf{ZERO}, m) &= \mathbf{ZERO} \\ &| \textit{Mult}(\mathbf{SUCC}(n), m) = \textit{Add}(m, \textit{Mult}(n, m)) \end{aligned} \tag{13}$$

Vi siger, at *Add* og *Mult* er definerede rekursivt på første argument, fordi vi splitter op i to tilfælde alt efter formen på første argument.

Opgave 4 Definer fakultetsfunktionen *Fac* rekursivt, således at $Fac(n) = n!$. Afprøv definitionen.

Opgave 5 Definer potensopløftning *Pow* rekursivt på andet argument, således at $Pow(m, n) = m^n$. Afprøv definitionen.

Opgave 6 Definer funktionen $Equal : nat \times nat \rightarrow bool$, der på input (n, m) returnerer TRUE præcis hvis m og n er det samme tal. Hint: brug rekursion på *begge* argumenter. Afprøv definitionen.

Opgave 7 Betragt definitionen af de såkaldte *Fibonacci-tal*:

$$\begin{aligned} Fib(0) &= 1 \\ Fib(1) &= 1 \\ Fib(2 + n) &= Fib(1 + n) + Fib(n) \end{aligned} \tag{14}$$

Giv en tilsvarende definition i SML. Kontroller at $Fib(6) = 13$.

Sekvenser

Sekvenser af naturlige tal kan i SML defineres som følger:

$$\mathbf{datatype} \text{ seq} = \mathbf{NIL} \mid \mathbf{CONS} \text{ of } \mathit{nat} \times \text{seq} \quad (15)$$

Datotypen *seq* omfatter nu værdien **NIL** og alle værdier der er “CONS-structured from a *nat* and a *seq*”, f.eks:

$$\mathbf{CONS}(\mathbf{ZERO}, \mathbf{NIL}) \quad \text{og} \quad \mathbf{CONS}(\mathbf{ZERO}, \mathbf{CONS}(\mathbf{ZERO}, \mathbf{NIL})) \quad . \quad (16)$$

Dermed svarer værdierne i *seq* til mængden \mathcal{N}^* af sekvenser af naturlige tal. Vi definerer længde og sammensætning som følger:

$$\begin{aligned} \mathbf{fun} \text{ Length}(\mathbf{NIL}) &= \mathbf{ZERO} \\ &| \text{ Length}(\mathbf{CONS}(n, s)) = \mathbf{SUCC}(\text{Length}(s)) \\ \mathbf{fun} \text{ Concat}(\mathbf{NIL}, s') &= s' \\ &| \text{ Concat}(\mathbf{CONS}(n, s), s') = \mathbf{CONS}(n, \text{Concat}(s, s')) \end{aligned} \quad (17)$$

Opgave 8 Definer funktionen $\text{Sum} : \text{seq} \rightarrow \text{nat}$ rekursivt, således at $\text{Sum}(s)$ returnerer summen af tallene i *s*. Afprøv definitionen.

Opgave 9 Definer funktionen Rev rekursivt, således at $\text{Rev}(s)$ returnerer sekvensen *s* vendt om (reverseret). Afprøv definitionen.

Opgave 10 Definer funktionen $\text{Contains} : \text{nat} \times \text{seq} \rightarrow \text{bool}$ rekursivt på andet argument. Funktionen skal på input (n, s) returnere **TRUE** præcis hvis *s* indeholder *n*. Afprøv definitionen.

Opgave 11 Definer funktionen $\text{Prefix} : \text{seq} \times \text{seq} \rightarrow \text{bool}$ rekursivt på begge argumenter. Funktionen skal på input (s, s') returnere **TRUE** præcis hvis *s* er et præfiks af *s'*. Afprøv definitionen.

Opgave 12 Definer funktionen $\text{SeqEqual} : \text{seq} \times \text{seq} \rightarrow \text{bool}$, der på input (s, s') returnerer **TRUE** præcis hvis *s* og *s'* er den samme sekvens. Afprøv definitionen.

Opgave 13 Definer en datatype *bseq* svarende til mængden \mathcal{B}^* af sekvenser af sandhedsværdier. Definer også funktionerne $\text{SeqOr} : \text{bseq} \rightarrow \text{bool}$ og $\text{SeqAnd} : \text{bseq} \rightarrow \text{bool}$. Argumenter for værdien af disse to funktioner på den tomme sekvens af sandhedsværdier.

Binære træer

Binære træer kan i SML defineres som følger:

$$\mathbf{datatype} \text{ tree} = \mathbf{EMPTY} \mid \mathbf{NODE} \text{ of } \text{tree} \times \text{tree} \quad (18)$$

Værdierne af typen *tree* svarer til mængden af binære træer. Da reglen for konstruktion af det binære træ $\mathbf{NODE}(t_1, t_2)$ fra de binære træer t_1 og t_2 involverer *to* mindre værdier i stedet for kun en som hidtil, vil rekursive funktioner på binære træer normalt involvere *to* rekursive kald. Vi definerer størrelse og spejlvending som følger:

$$\begin{aligned} \mathbf{fun} \text{ Size}(\mathbf{EMPTY}) &= \mathbf{ZERO} \\ &| \text{ Size}(\mathbf{NODE}(t_1, t_2)) = \mathbf{SUCC}(\mathbf{Add}(\text{Size}(t_1), \text{Size}(t_2))) \end{aligned} \quad (19)$$

$$\begin{aligned} \mathbf{fun} \text{ Mirror}(\mathbf{EMPTY}) &= \mathbf{EMPTY} \\ &| \text{ Mirror}(\mathbf{NODE}(t_1, t_2)) = \mathbf{NODE}(\text{Mirror}(t_2), \text{Mirror}(t_1)) \end{aligned}$$

Opgave 14 Betragt definitionen af højden af et binært træ:

$$\begin{aligned} \text{Height}(\mathbf{EMPTY}) &= 0 \\ \text{Height}(\mathbf{NODE}(t_1, t_2)) &= 1 + \mathbf{Max}(\text{Height}(t_1), \text{Height}(t_2)) \end{aligned} \quad (20)$$

Giv en tilsvarende definition i SML. Du får brug for at definere operationen *Max* på naturlige tal.

Binære træer bliver mere interessante, hvis deres knuder indeholder information. Vi vil derfor ændre vores definition af binære træer, så de får naturlige tal i knuderne:

$$\mathbf{datatype} \text{ tree} = \mathbf{EMPTY} \mid \mathbf{NODE} \text{ of } \text{tree} \times \mathbf{nat} \times \text{tree} \quad (21)$$

Opgave 15 Ret i definitionerne af *Size*, *Mirror* og *Height* så de passer til den nye version af *tree*.

Opgave 16 Definer funktionen $\text{TreeSum} : \text{tree} \rightarrow \mathbf{nat}$ således at $\text{TreeSum}(t)$ returnerer summen af tallene i t . Afprøv definitionen.

Opgave 17 Definer funktionen $\text{TreeContains} : \mathbf{nat} \times \text{tree} \rightarrow \mathbf{bool}$ så den på input (n, t) returnerer **TRUE** præcis hvis t indeholder n . Afprøv definitionen.

Opgave 18 Definer funktionen $\text{Flatten} : \text{tree} \rightarrow \mathbf{seq}$, der returnerer en sekvens indeholdende de naturlige tal, der forekommer i t . Hvordan skal din definition ændres for at opnå forskellige rækkefølger af tallene i t ? (Hint/udfordring: der er 6 “naturlige” rækkefølger).

Opgave 19 Skriv en funktion $\text{TreeAdd} : \mathbf{nat} \times \text{tree} \rightarrow \text{tree}$ der på input (n, t) returnerer træet, der opnås ved at lægge n til tallene i alle t 's knuder.

Udtryk

Udsagnslogiske udtryk kan i SML defineres som datatypen

```
datatype prop = BOOL of bool
                | NOT of prop
                | OR of prop × prop
                | AND of prop × prop
                | IMPLIES of prop × prop
```

 (22)

Vi kan beregne sandhedsværdien af et udsagnslogisk udtryk med funktionen $Value : prop \rightarrow bool$. Denne funktion defineres ved hjælp af vores operationer på sandhedsværdier:

```
fun Value(BOOL(b)) = b
    | Value(NOT(p)) = Not(Value(p))
    | Value(OR(p, q)) = Or(Value(p), Value(q))
    | Value(AND(p, q)) = And(Value(p), Value(q))
    | Value(IMPLIES(p, q)) = Implies(Value(p), Value(q))
```

 (23)

Opgave 20 Definer en funktion $RemoveImp : prop \rightarrow prop$, der omskriver et logisk udtryk vha. ækvivalensen $p \rightarrow q \iff \neg p \vee q$ sådan at resultatet bliver et logisk udtryk uden implikation. Kontroller at funktionen på input $(false \rightarrow false) \rightarrow false$ returnerer udtrykket $\neg(\neg false \vee false) \vee false$.

Opgave 21 Definer en datatype, der beskriver aritmetiske udtryk over naturlige tal, fordobling, addition og multiplikation. Definer også en rekursiv funktion, der beregner værdien af et aritmetisk udtryk.