Teori

▼ Term/grunnterm/konstruktørterm

Termer er velformet uttrykk (bygget opp av definerte symboler) EKS 0+X.

En term bygges opp av variable og funksjonssymboler fra spesifikasjonen.

Grunntermer er termer uten variable

Grunnterm bygges opp av funksjonssymbolene på en sort-korrekt måte.

Eksempel grunntermer av sort Nat i NAT-ADD:

I maude er en datatype representert av grunntermene bygd opp av konstruktør-funksjonssymboler

▼ Unifikasjon

En unifikator for t og u er en substitusjon σ slik at tσ = uσ

▼ Prefix og mixfix

Alle ops fra et Maude-system er funksjoner med eller uten argumenter. Kan skrives som prefix f(x, x') eller mix-fix _ + _ .

Concat _ _ : List List → List ([]) .

f(L L) blir f(_ _(L, L'))

▼ Assosiativitet kommtativitet, list og mset

`nil` er identitets-element for lister:

```
sorts List NeList .
subsorts Nat < NeList < List .
op nil : -> List [ctor] .
op __ : List List -> List [assoc id: nil ctor] .
op __ : NeList NeList -> NeList [assoc id: nil ctor] .
```

- Siste deklarasjon kunne også skrives
  `op __ : NeList NeList -> NeList [ctor ditto] .`
- Liste er enten `nil` eller har formen *n l* (eller *l n* eller *l l'*)

Multisett kan sees på som lister der rekkefølgen av elementene ikke spiller rolle: (assoc + id:) + comm:

```
sort Mset .   subsort Nat < Mset .

op none : -> Mset [ctor] .
op __ : Mset Mset -> Mset [ctor assoc comm id: none] .
```

- `2 (3 5)` og `none (3 (5 2))` ekvivalente (ekvivalensklasser av) termer
- 1-til-1 representasjon av multisett

▼ Spesifikasjon

Vi skriver $\{f(x) = a,\ f(g(x,y)) = y\}$ for spesifikasjonen

```
sort S .
op f : S -> S .
op g : S S -> S .
op a : -> S .
vars x y : S .
eq f(x) = a .
eq f(g(x,y)) = y .
```

Vi bruker:

- $a, b, c, a', a_1, \ldots$ for konstanter
- $f, g, h, \ldots$ for andre funksjonsymboler
- $x, y, z, x', x_1, \ldots$ for variable

▼ Relations: Arrows

- $\leftrightsquigarrow$ er den symmetriske tillukningen til $\rightsquigarrow$: $t \leftrightsquigarrow u$ hvis enten $u \rightsquigarrow t$ eller $t \rightsquigarrow u$
- $t \overset{*}{\rightsquigarrow} u$ hvis t kan forenkles til u i null eller flere steg
- $t \overset{+}{\rightsquigarrow} u$ hvis t kan forenkles til u i ett eller flere steg
- $\overset{*}{\leftrightsquigarrow}$ og $\overset{+}{\leftrightsquigarrow}$ defineres på tilsvarende måte

- t er reduserbar hvis det finnes en u slik at $t \rightsquigarrow u$; hvis ikke, så er t irredusibel
- u er en normalform av t hvis $t \overset{*}{\rightsquigarrow} u$ og u er irredusibel
  - hvis t kun har én normalform, skrives normalformen $t!$
- En utledning (eller reduksjonssekvens eller ...) er en endelig eller uendelig sekvens av forenklingssteg

$$t_0 \rightsquigarrow t_1 \rightsquigarrow t_2 \rightsquigarrow \cdots$$

▼ Normalform

Resultatet av computation av et term t er normalformen av t, t! .

Alle kjøringer av et uttryk (i en terminerende spec) skal gi samme resultat. Deterministisk. eq skal være terminerende og konfluent, mens rl (dynamiske tilstandsoverganger) er ofte ikke terminerende og heller ikke konfluent.

- Beregner verdien til et funksjonelt uttrykk
- Burde få samme resultat uansett hvordan Maude bruker ligningene
- Unik "verdi" kalles normalform

ksempel

- Alltid unikt resultat i `NAT-ADD`
- `eq a = b .` `eq a = c .` tilfredsstiller ikke "unikt resultat" egenskapen

▼ Terminering

> **Definisjon (Terminering)**
>
> *Det finnes ingen uendelig utledning i spec'en*
>
> - *ingen red-kommando vil gå i evig løkke, uansett startterm eller hvordan Maude bruker ligningene*

Specs skal alltid terminere (eq equations)

- Likninger/funkjsoner som skal gi ETT svar
- Alle funksjoner skal gi deterministisk verdi

  f : A → B

  hver a |→ én b

Regelmengder trenger ikke å terminere (rl rules)

- Dynamiske tilstandsoverganger som ikke er deterministiske

Unionen av to terminerende systemer

- Se LF oppg 2 eksamen 2003

Oppg 2a Eksamen 2016

Forklar hvorfor man ikke kan definere en funksjon `op knapSackContent : Items NzNat NzNat -> Items .` som returner enten den tomme mengden (hvis et ønsket utplukk ikke finnes), eller et utplukk av varer som tilfredsstiller både vekt- og verdi-grensene, og som tilfredsstiller de vanlige kravene til en funksjon i Maude.

- Fordi Items ikke er et unikt resultat

▼ Partielle ordninger

2.5.1.1 Partiality

Noe er en del av noe annet

Man bør kunne definere utvalg av en type, subsorts, for å utføre effektive funksjoner.

Eks:

- First og last av tom liste
- Divisjon med nat 0

Man bør kunne relatere Nat og Int samtidig som man bruker dem ulikt.

$N \subseteq Z$

Vi skriver R(m1, m2) eller m1 R m2 for (m1, m2) ∈ R

En binær relasjon R er en partiell ordning hvis og bare hvis
• R er refleksiv: mRm holder for enhver m ∈ M

- R er antisymmetrisk: hvis både m1Rm2 og m2Rm1, så er m1 = m2
- R er transitiv: hvis m1Rm2 og m2Rm3,så m1Rm3

Definisjon (Binære relasjon)

En binær relasjon R over en mengde M er en mengde $R \subseteq \{(m_1, m_2) \mid m_1, m_2 \in M\}$

Vi skriver $R(m_1, m_2)$ eller $m_1 \, R \, m_2$ for $(m_1, m_2) \in R$

Definisjon (Partiell ordning)

En binær relasjon R er en partiell ordning hvis og bare hvis

- *R er refleksiv: $m \, R \, m$ holder for enhver $m \in M$*
- *R er antisymmetrisk: hvis både $m_1 \, R \, m_2$ og $m_2 \, R \, m_1$, så er $m_1 = m_2$*
- *R er transitiv: hvis $m_1 \, R \, m_2$ og $m_2 \, R \, m_3$, så $m_1 \, R \, m_3$*

Definisjon (Strikt partiell ordning)

En binær relasjon R er en strikt partiell ordning hviss R er

- *irrefleksiv: $m \, R \, m$ holder ikke for noen m, og*
- *transitiv*

Definisjon (Velfundert strikt partiell ordning)

En strikt partiell order $\succ$ over en mengde S er velfundert hvis det ikke finnes noen uendelig sekvens $s_1 \succ s_2 \succ \cdots$ av S-elementer

Partiell:

≥

Definisjon Strikt partiell:

Transitiv: a > b > c impliserer a > b

Irrefleksiv: a > b betyr at b /> a

Definisjon velfundert:

En strikt partiell ordning > over en mengde S er velfundert hvis det ikke finnes noen uendelig sekvens s1 > s2 > … av S-elementer

Velfunderte ordninger?

Leksografisk sammenlikning er ikke velfundert

>lpo er velfundert hvis spec har et endelig antall funksjonssymboler

Definisjon monoton: 4.6 s.78

▼ Vektfunksjoner

Teorem

En spesifikasjon er terminerende hvis det finnes en velfundert strikt partiell ordning $(S, \succ)$ *og en funksjon* $vekt : \mathcal{T}_\Sigma \to S$ *slik at* $t \rightsquigarrow t'$ *medfører* $vekt(t) \succ vekt(t')$ *for alle grunntermer* t, t'

- Generalisering av vektfunksjon der $(\mathbb{N}, >)$ generaliseres til en velfundert strikt partielt ordnet mengde $(S, \succ)$

Bevise terminering med “vektfunksjoner”

Metoder:

- Naturlige tall >
- Lister av nat med samme lengde >lex

Eksempler:

{f (x) = g(x), g(b) = f (a)}

vekt(t) = (2 ⇤ “antall b’er i t”) + “antall f ’er i t”

{f (x) = g(x), g(b) = f (a)}

- vekt(a) = 1
- vekt(b) = 88
- vekt(f (t)) = 4 + vekt(t)
- vekt(g(t)) = vekt(t)

{f(g(x)) = g(f(x))}

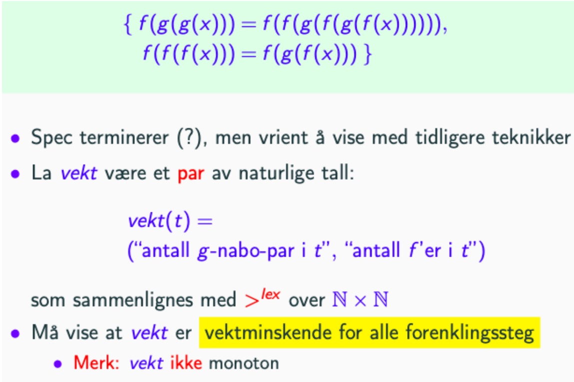
- vekt(a) = 2 for hver konstant a
- vekt(f(t)) = $(vekt(t))^3$
- vekt(g(t)) = 2 · vekt(t)

{g(f(x)) = f(h(x)), h(x) = g(x)}

- vekt(g(t)) = $(vekt(t))^3$
- vekt(f(t)) = vekt(t)
- vekt(h(t)) = $2 * (vekt(t))^3$
- vekt(a) = 1 for hver konstant a

{*f*(*g*(*x*))=*h*(*x*), *h*(*x*)=*g*(*f*(*x*)), *a*=*b*}

- w(a) = 2



$$\{\, f(g(g(x))) = f(f(g(f(g(f(x)))))),$$
$$f(f(f(x))) = f(g(f(x)))\,\}$$

- Spec terminerer (?), men vrient å vise med tidligere teknikker
- La *vekt* være et par av naturlige tall:

$$vekt(t) = (\text{"antall } g\text{-nabo-par i } t\text{", "antall } f\text{'er i } t\text{"})$$

  som sammenlignes med $>^{lex}$ over $\mathbb{N} \times \mathbb{N}$
- Må vise at *vekt* er vektminskende for alle forenklingssteg
  - Merk: *vekt* ikke monoton

▼ Tuppel vektfunksjon

Eks:

g(g(x)) = f(g(h(x))), f(h(g(x))) = g(h(x))

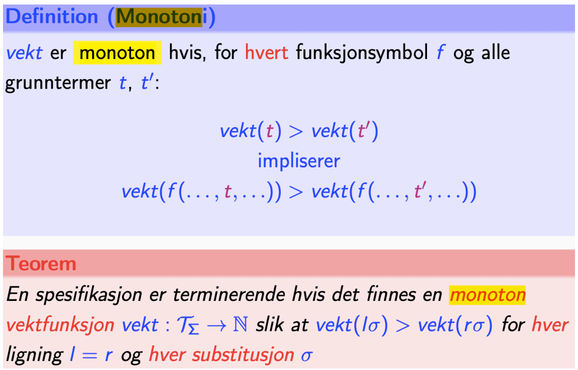
Vekt er par: (#g(t), size(t))

- #g(t) antall g-er
- size(t) termens strl

Monoton, antakelse w(t) > w(u)

- Vis at: w(f(t)) > w(f(u)), w(g(t)) > w(g(u)) og w(h(t)) > w(h(u))

▼ Monoton



Definition (Monotoni)

vekt er monoton hvis, for hvert funksjonsymbol f og alle grunntermer t, t':

$$vekt(t) > vekt(t')$$
$$\text{impliserer}$$
$$vekt(f(\ldots,t,\ldots)) > vekt(f(\ldots,t',\ldots))$$

Teorem

En spesifikasjon er terminerende hvis det finnes en monoton vektfunksjon $vekt : \mathcal{T}_\Sigma \to \mathbb{N}$ slik at $vekt(l\sigma) > vekt(r\sigma)$ for hver ligning $l = r$ og hver substitusjon σ

Krav: Alle variabler må regnes med i vektfunksjonene.

Unntak: Noen ganger er systemer hvor det ikke finnes monoton vektfunksjon fortsatt terminerende.

Hvis S er teminerende bevist med LPO/MPO så finnes det også en monoton vektfunksjon.

Anta `vekt(t) > vekt(u)`

`vekt(h(t)) > vekt(h(u))`

og `vekt(g(t)) > vekt(g(u))`

BEVISE en vektfunksjon som monoton:

w(a)=3

w(f(t,u)) = 2*w(t)+w(u)

Anta w(t1) > w(t2)

MÅ VISE:

w(f(t1,u)) > w(f(t2,u)) og

w(f(u,t1)) > w(f(u,t2))

GIR:

2*w(t1)+w(u) >? 2*w(t2)+w(u)

2*w(u)+w(t1) >? 2*w(u)+w(t2)

Siden w(t1) > w(t2) er vektfunksjonen monoton.

BRUKES TIL MOTBEVIS OGSÅ

- HELST MED MOTEKSEMEL

{ fhhx = ffhfhfx, fffx = fhfx }

< #1 h-par, #2 ant f-er >

fhhx > ffhfhfx OK med #1

fffx > fhfx OK med #2

Terminerer med denne ikke-monotone vektfunksjonen.

Ikke monoton fordi:

w(f(h(h(x)))) > w(h(f(x)))

w(h(f(h(h(x)))) /> w(h(h(f(x))))

▼ Stiordninger: lpo mpo

lpo 1: $f(...,t_i,...) \succ_{lpo} u$ if ($t_i \succ_{lpo} u$ or $t_i = u$)

lpo 2: $f(t_1,...,t_n) \succ_{lpo} g(u_1,...,u_m)$

if ($f \succ g$ and $f(t_1,...,t_n) \succ_{lpo} u_i$ for each $i \leq m$).

lpo 3: $f(t_1,...,t_n) \succ_{lpo} f(u_1,...,u_m)$

if ($(t_1,...,t_n) \succ_{lpo}^{lex} (u_1,...,u_n)$ and $f(t_1,...,t_n) \succ_{lpo} u_i$ for each $2 \leq i \leq n$).

mpo 3: $f(t_1,...,t_n) \succ_{mpo} f(u_1,...,u_m)$

if ($[t_1,...,t_n] \succ_{mpo}^{mul} [u_1,...,u_n]$.

- Hvis S er teminerende bevist av LPO/MPO så finnes det også en monoton vektfunksjon

E1 = {f(a,b) = f(b,a)} Vises terminerende med LPO (ikke MPO)
E2 = {g(x, a) = g(b, x)}. Vises terminerende med MPO (ikke LPO)

Er {f(g(x), a) = f(f(x,x), b)} terminerende?

  - Hvis g > f og g > b så gir lpo-1 at denne er lpo-minskende hvis g(x) >lpo f(f(x,x),b). Hvis g>f så gir lpo-2 at vi må vise g(x) >lpo f(x,x) og g(x) >lpo b.

- Begge disse er greie, så denne er lpo-terminerende.

Motbevis

- Ikke-terminerende hvis det er en ny variabel i høyresiden av en ligning: {f(x)=g(x,y)} gir gjentagende f(x) -^→ g(x,f(x))
- Ikke-terminerende hvis venstreside er en variabel: {x = g(a)}
- Ikke-terminerende hvis hele venstresiden eksisterer i høyresiden
- Ikke-terminerende hvis høyresiden kan vokse uendelig f(x) → f(f(x)) → …

Toyamas moteksempel

- Kombinere to terminerende system som gir et ikke-terminerende system

```
{f(a,b,x) = f(x,x,x)} U
{g(x,y) = x
g(x,y) = y}

f(a,b,g(a,b)) -^-> f(g(a,b), g(a,b), g(a,b))
              -^-> f(g(a,b), b, g(a,b))
              -^-> f(a, b, g(a,b))
```

Classic Toyama counterexample: a looping derivation
f(a, b, g(a, b)) → f(g(a, b), g(a, b), g(a, b)) → f(a, g(a, b), g(a, b)) → f(a, b, g(a, b))

Konfluens

Konfluens: Hvis $t \overset{*}{\rightsquigarrow} t_1$ og $t \overset{*}{\rightsquigarrow} t_2$, så må det finnes en u slik at $t_1 \overset{*}{\rightsquigarrow} u$ og $t_2 \overset{*}{\rightsquigarrow} u$

Konfluens + terminering $\Longrightarrow$ kun ett "resultat/verdi" av et uttrykk, uansett hvordan ligningene brukes

Maude antar at din spec er konfluent

Uavgjørbart hvorvidt en spec er konfluent

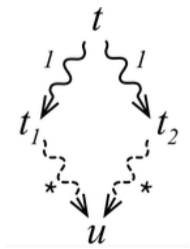
Avgjørbart hvis spec'en er terminerende

Newman's Lemma: tilstrekkelig å sjekke termer t_1 og t_2 som kan nås i ett steg fra startterm t

Critical Pairs Lemma: endelig mengde starttermer t

Terminering + lokal konfluens = global konfluens

Lokal konfluens



1. Undersøk venstresider
2. Omnavn felles variabler
3. Se etter kritiske par
    a. Ingen kritiske par impliser lokal konfluens
    b. Eventuelt sjekk konfluens for alle kritiske par
        i. Omnavn
        ii. Hvis du finner et overlappsterm som ikke er lokalt konfluent er specen ikke konfluent.

Terminerende systemer der man kan bevises lokal konfluens, er konfluente

Undersøk alle venstresider og sjekk om de kan representeres i en annen venstreside (alle subtermer)

Eksempel på spec som er lokalt men ikke globalt konfluent:

{a=b, a=c, b=c, b=d}

▼ 5 slutningsregler

E ⊢ t = u holder hvis det kan utledes fra følgende aksiomskjemaer og slutningsregler for likhetslogikk:

- Substitutivitet: $E \vdash l\sigma = r\sigma$ holder for hver ligning $l = r$ i E
- Refleksivitet: $E \vdash t = t$ holder for hver term t
- Symmetri: Hvis $E \vdash t = u$ holder, så holder $E \vdash u = t$
- Transitivitet: hvis både $E \vdash t_1 = t_2$ og $E \vdash t_2 = t_3$ holder, så holder $E \vdash t_1 = t_3$
- Kongruens: Hvis $E \vdash t_1 = u_1, ..., E \vdash t_n = u_n$ alle holder, så holder $E \vdash f(t_1, ..., t_n) = f(u_1, ..., u_n)$, for hvert funksjonssymbol f.

- Start med aksiomer (Substitutivitet og Refleksivitet)
- Bruk slutningsreglene til ønsket likhet oppnådd
- Se bort fra alt annet: kun disse 5 aksiomskjemaene/slutningsreglene kan brukes!
- Maude dersom spesifikasjonen er terminerende og konfluent

▼ Likhetslogikk

Definisjon (Følge fra en mengde ligninger)
$t_1 = t_2$ følger fra en mengde ligninger E hvis og bare hvis $t_1 = t_2$ holder i alle matematiske modeller hvor ligningene E holder

Definere om et utsagn følger logisk fra en mengde likninger — er to termer like?

Umulig å sjekke enhver E-modell

Likhetslogikk er uavgjørbart

- Vi vet at addisjon er kommutativ
- Men $\mathtt{NAT-ADD} \vdash \mathtt{M} + \mathtt{N} = \mathtt{N} + \mathtt{M}$ holder ikke
- Likheten over følger ikke fra NAT-ADD!
  - feil spesifikasjon eller teoremer?
  - $\mathtt{NAT-ADD} \vdash \mathtt{M} + \mathtt{N} = \mathtt{N} + \mathtt{M}$ betyr at $\mathtt{M} + \mathtt{N} = \mathtt{N} + \mathtt{M}$ holder i alle systemer hvor ligningene i NAT-ADD holder
  - likheten holder ikke i NAT-ADD utvidet med konstant a
- $m + n = n + m$ holder for alle tall/termer 0, s(0), s(s(0)), ...
- Induktive teoremer: holder for alle konstruktør-grunntermer

Et godt (optimalt?) bevissystem:
• sunt: kan bare bevise ting som er sanne
• komplett: kan bevise alle utsagn som holder
• sjekkes algoritmisk: kan sjekke hvorvidt et gitt "bevis" er et gyldig bevis
Likhetslogikk er et "optimalt" bevissystem for likheter som holder i alle E-modeller

Likhetslogikk:
- sunt/komplett bevissystem for likheter som holder i alle modeller hvor ligningene holder
- avgjørbart i terminerende og konfluente spec'er

Induktive teoremer:
- likheter som holder i intendert/ønsket modell
- mer relevant for oss
- ikke sunt/komplett bevissystem
- generelt induksjonsprinsipp for datatyper
- kan bevise noe for alle elementer
  - umulig med testing, etc
- ikke-trivielt:
  - trenger ofte hjelpe-lemmaer
  - hva hvis bevis ikke fører frem?
- nødvendig i sikkerhetskritiske systemer (fly, ...: PVS, Isabelle/HOL, Coq, ...)

▼ Teorem 6.2 og 6.5

S |- X = Y

THEOREM 6.2

E ⊢ t=u hvis og bare hvis $t \leftrightsquigarrow_E^* u$

For hver likning l = r er også r = l (symm)

THEOREM 6.4

I terminerende og konfluente spesifikasjoner kan vi sjekke hvorvidt E ⊢ t = u ved å beregne deres normalformer:
$E \vdash t = u$ hvis og bare hvis $t!_E = u!_E$

Fremgangsmåte

- Bevis/begrunn terminering
- Bevis/begrunn (global) konfluens
- Sammenlikn normalform:

  Sjekk om X! =? Y!

  Må kunne reduseres sort-uavhengig med likningene i E.

▼ Likhetslogikk eksempler

E = {f(x) = g(x), a=b}

Bevis: $E \vdash b = a$

Substitutivitet gir $E \vdash a = b$

Bevis: $E \vdash f(a) = g(b)$

Substitutivitet $E \vdash a = b$

Substitutivitet $E \vdash f(a) = g(a)$

Kongruens $E \vdash g(a) = g(b)$

Transitivitet $E \vdash f(a) = g(b)$

E' = {f(a,x) = f(b,x), c=d}

Bevis: $E' \vdash f(b,c) = f(a,c)$

Subst. $E' \vdash f(a,c) = f(b,c)$

Symm. $E' \vdash f(b,c) = f(a,c)$

Bevis: $E' \vdash f(a,a) = f(b,b)$

Subst. $E' \vdash f(a,a) = f(b,a)$

Subst. $E' \vdash f(a,b) = f(b,b)$

/stagnerer, umulig å bevise

Teorem 6.5

/Normalformen til $f(b,a) = / = f(b,b)$

Induktive teoremer

Forskjell på likhetslogikk og induktiv likhetslogikk er at induktiv kun gjelder for den gjeldende definisjonen.

LIKNINGEN E HOLDER FOR ALLE VÅRE ELEMENTER X

Finnes ikke "optimalt" bevissystem for likheter som skal holde i den intenderte modellen til en spesifikasjon

Prinsipp: Noe skal holde for alle elementer

Induksjon: bevise at P(n) holder for alle naturlige tall n:

1. Bevise at P(0) holder
2. Ta et "tilfeldig" tall k, og bevis at P(k) holder, når du kan anta induksjonshypotesen at P(j) holder for enhver j < k
    a. k er ofte"n+1"

Prinsipp for å bevise $P(t)$ for enhver konstruktør-grunnterm t:

1. Bevis $P(c)$ for enhver konstruktør-konstant c
2. Bevis, for hver konstruktør f, at $P(f(t_1, \ldots, t_n))$ holder for alle $t_1, \ldots, t_n$, når du kan anta induksjonshypotesen $P(t_i)$ for hver t_i (av samme sort)

For å bevise at $P(t)$ holder for hver konstruktør-grunnterm t av sort `Nat` i `NAT-ADD`, må man bevise:

1. $P(0)$ holder
2. $P(s(t))$ holder for "vilkårlig" t, når du kan anta at induksjonshypotesen $P(t)$ holder

1e Induktive teoremer Eksamen 2012

Generell versjon:

EKSEMPEL: Primitiv list-int

```
sort List .
op nil : -> List [ctor] .
op __ : List Nat -> List [ctor] .
```

Bevise $P(l)$ for hver konstruktør-grunnterm l av sort `List`:

1. Bevis $P(\texttt{nil})$
2. Bevis $P(l\ n)$ for "vilkårlige" kgt. l og n, når du kan anta induksjonshypotesen $P(l)$

```
op length : List -> Nat .
op concat : List List -> List .
var N : Nat .    vars L L' : List .

eq length(nil) = 0 .
eq length(L N) = s(length(L)) .
eq concat(L, nil) = L .
eq concat(L, L' N) = concat(L, L') N .
```

Ønsker å bevise

$$\mathtt{LIST-NAT1} \vdash \mathtt{length}(\mathtt{concat}(l, l')) = \mathtt{length}(l) + \mathtt{length}(l')$$

for alle konstruktør-grunntermer l og l' av sort List

ved induksjon på l':

1. Må bevise

   $$\mathtt{LIST-NAT1} \vdash \mathtt{length}(\mathtt{concat}(l, \mathtt{nil})) = \mathtt{length}(l) + \mathtt{length}(\mathtt{nil})$$

   for hver l
2. Må bevise

   $$\mathtt{LIST-NAT1} \vdash \mathtt{length}(\mathtt{concat}(l, l' \; n)) = \mathtt{length}(l) + \mathtt{length}(l' \; n)$$

   når man antar induksjonshypotesen

   $$\mathtt{LIST-NAT1} \vdash \mathtt{length}(\mathtt{concat}(l, l')) = \mathtt{length}(l) + \mathtt{length}(l')$$

EKSEMPEL: Bintree

1. $\mathtt{BTRE} \vdash \mathtt{size}(\mathtt{reverse}(\mathtt{empty})) = \mathtt{size}(\mathtt{empty})$
2. $\mathtt{BTRE} \vdash \mathtt{size}(\mathtt{reverse}(\mathtt{bintree}(t_1, n, t_2))) = \mathtt{size}(\mathtt{bintree}(t_1, n, t_2))$ når man antar induksjonshypotesene
   - $\mathtt{BTRE} \vdash \mathtt{size}(\mathtt{reverse}(t_1)) = \mathtt{size}(t_1)$ og
   - $\mathtt{BTRE} \vdash \mathtt{size}(\mathtt{reverse}(t_2)) = \mathtt{size}(t_2)$

(Kommutativitet av addisjon i NAT-ADD bevist i læreboken)

EKSEMPEL: Nat-add

- Husk: kan bruke $t \overset{*}{\leftrightsquigarrow} u$ i stedet for $E \vdash t = u$
- La Maude (prøve å) gjøre beviset:

```
fmod NAT-ADD-IND-PROOF is extending NAT-ADD .
  op t : -> Nat .        --- generic constant for induction
  eq t + 0 = t .         --- induction hypothesis
endfm

--- Inductive proof of NAT-ADD |- T + 0 = T.

--- Base case: T is 0. Prove |- 0 + 0 = 0 .
red 0 + 0 == 0 .    *** returns 'true'

--- Induction step. T is s(t):
red s(t) + 0 == s(t) .    *** returns 'true'
```

▼ Search

Search [antall tilfeller man søker etter] init ⇒* (antall steg) < tilstand > REST:Configuration such that (kriterier) .)

(search [1] init(3) =>! < O:Oid : Spill | rekke : L:RuteList > C:Configuration
such that L:RuteList == (sK ; sK ; sK ; tR ; hK ; hK ; hK) .)

(search [1] init(3) =>! (s;s;s;t;h;h;h) .)

Tre case:

Finner tilfelle

No solution

Uendelig

▼ Oppnåelighet, uendelighet, sluttiltstander

Endelig eller uendelig antall oppnåelige tilstander

FEIL: "Sluttilstander eksisterer bare dersom programmet er terminerende."

Nei, se på filosofene: det KAN finnes sluttilstander (deadlocks) men det finnes

også masse uendelige kjøringer, så denne er IKKE terminrende.

(Husk: terminerende betyr at det ikke FINNES uendelige kjøringer)

rl a => b

rl b => a

rl b => c

er IKKE terminerende (har evig løkke a->b->a->b->a …

men HAR en sluttilstand c.

Uendelig antall tilstander dersom:

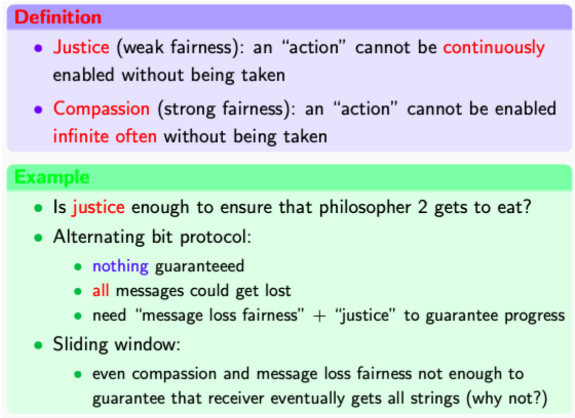
- attributt som øker i verdi
- ubegrenset antall uleste meldinger i systemet
- ubegrenset antall objekter

▼ Deadlock, livelock

Exercise 155, kap 10, oppgsett.9

`Maude> (search [1] init ⇒! C:Configuration .)` systemet bør ikke ha sluttilstand, derfor søker vi etter et brudd på dette

▼ Fairness: Justice (weak), compassion (strong)



Definition

- Justice (weak fairness): an "action" cannot be continuously enabled without being taken
- Compassion (strong fairness): an "action" cannot be enabled infinite often without being taken

Example

- Is justice enough to ensure that philosopher 2 gets to eat?
- Alternating bit protocol:
  - nothing guaranteeed
  - all messages could get lost
  - need "message loss fairness" + "justice" to guarantee progress
- Sliding window:
  - even compassion and message loss fairness not enough to guarantee that receiver eventually gets all strings (why not?)

Dining philosopher

The *fairness* assumptions about this problem are: an eating philosopher *eventually* stops eating; a thinking philosopher eventually becomes hungry; and a philosopher will eventually pick up a needed chopstick if it is available infinitely often.

op justice : Oid -> Formula .

eq justice(0) = (<> [] (O is outCS)) -> ([] <> (O is waitCS)) .

op globalJustice : -> Formula .

eq globalJustice = justice(n1) /\ justice(n2) /\ justice(n3) /\ justice(n4) /\ justice(n5) .

Tilstandsutsagn og hendelsesutsagn

State proposition - tilstandsutsagn og hendelsesutsagn

- Kun nåsituasjon
- Ikke kunnskap om tidligere states
- Kan søkes etter i maude

Action proposition - noe som skjer

- Undersøke om et event kan forkomme
- Noen kan søkes etter i Maude, men ikke alle

Kombinasjon action og state proposition

- Fairness conditions

Temporallogikk

□ P : P is invariant ("always P")

<> P : P is guaranteed ("eventually P") Kan defineres true U P

<>[] P : På et eller annet tidspunkt vil det gjelde resten.

[]<> P : Uendelig mange ganger.

□ (P → <> Q) : Response

P er invariant

- P holder i hver tilstand som kan nås fra initialtilstanden

P er garantert

- P vil holde i én eller annen tilstand i hver mulige oppførsel/kjøring (fra initialtilstanden)

P er oppnåelig

- en tilstand som tilfredsstiller P kan nås fra initialtilstanden

Respons

- hver P-tilstand vil lede til en Q-tilstand i hver oppførsel

P er stabil

- alle tilstander etter en P-tilstand vil være P-tilstander

P until Q

- i hver oppførsel, P holder til vi kommer til en Q-tilstand
  - strong until: Q må holde før eller siden i hver oppførsel
  - weak until: Q trenger ikke nødvendigvis å nås

Det er uendelig mange P-tilstander (i hver oppførsel)

Kombinerte egenskaper: P er garantert og stabil

true er en formel

et tilstands-utsagn p er en formel

hvis P og Q er formler, så er følgende også formler:

- $\neg P$ (*ikke* P)
- $P \wedge Q$ (P *og* Q)
- $P \vee Q$ (P *eller* Q)
- $P \rightarrow Q$ (P *impliserer* Q)
- $P \leftrightarrow Q$ (P *hvis og bare hvis* Q)
- $\Box P$ (P *holder i hver tilstand*)
- $\Diamond P$ (P *holder i én eller annen tilstand*)
- $P \, \mathcal{U} \, Q$ (P *(strong-) until* Q)
- $P \, \mathcal{W} \, Q$ (P *(weak-) until* Q)
- $\bigcirc P$ (P *holder i neste tilstand*)

Vi definerer ($M, \pi \models P$) induktivt over P:

- $M, \pi \models \mathit{true}$ *holder alltid*
- $M, \pi \models p$ *(for tilstands-utsagn p) holder hvis og bare hvis p holder i $\pi(0)$*
- $M, \pi \models \neg P$ *holder hvis og bare hvis $M, \pi \models P$ ikke holder*
- $M, \pi \models P \wedge Q$ *holder hvis og bare hvis både $M, \pi \models P$ og $M, \pi \models Q$ holder*
- $M, \pi \models P \vee Q$ *holder hvis og bare hvis enten $M, \pi \models P$ eller $M, \pi \models Q$ (eller begge) holder*
- $M, \pi \models P \rightarrow Q$ *holder hvis og bare hvis (enten $M, \pi \models P$ ikke holder eller $M, \pi \models Q$ holder)*

- $M, \pi \models P \leftrightarrow Q$ *holder hvis og bare hvis ($M, \pi \models P$ holder hvis og bare hvis $M, \pi \models Q$ holder)*
- $M, \pi \models \Box P$ *holder hvis og bare hvis $M, \pi^i \models P$ holder for hver $i \in \mathbb{N}$*
- $M, \pi \models \Diamond P$ *holder hvis og bare hvis det finnes en $k \in \mathbb{N}$ slik at $M, \pi^k \models P$ holder*
- $M, \pi \models P \, \mathcal{U} \, Q$ *holder hvis og bare hvis det finnes en $k \in \mathbb{N}$ slik at $M, \pi^k \models Q$ holder, og $M, \pi^i \models P$ holder for hver $i < k$*
- $M, \pi \models P \, \mathcal{W} \, Q$ *holder hvis og bare hvis $M, \pi \models (P \, \mathcal{U} \, Q) \vee (\Box P)$ holder*
- $M, \pi \models \bigcirc P$ *holder hvis og bare hvis $M, \pi^1 \models P$ holder*

P er invariant (i modell M og initial-tilstand t_0): $M, t_0 \models \Box P$

P er oppnåelig: oppgave

P er garantert (å nås): $M, t_0 \models \Diamond P$

Respons: hver P-tilstand **må** (før eller siden) lede til en Q-tilstand: $M, t_0 \models \Box (P \rightarrow \Diamond Q)$

P er stabil: $M, t_0 \models \Box (P \rightarrow (\Box P))$

Uendelig mange P-tilstander i hver oppførsel: $M, t_0 \models \Box \Diamond P$

Compassion fairness: $M, t_0 \models (\Box \Diamond P) \rightarrow (\Box \Diamond Q)$

Justice fairness: $M, t_0 \models (\Diamond \Box P) \rightarrow (\Box \Diamond Q)$

▼ Temporal modellsjekker

```
op xxx : xxx → Prop [ctor] .
op TensingHarOxFlash : → Prop [ctor] . ---term av sort prop

eq REST:Configuration <tilstand> |= prop = bool .
eq <Tensing:Person | sekk : item(ox…);item(flash…);IS2>

red modelCheck(init, FORMEL)
red modelCheck(init, [] (TensingHarOxFlash)
```

```
(omod MODEL-CHECK-PHIL1 is
  protecting DINING-PHILOSOPHERS .
  including MODEL-CHECKER .

  subsort Configuration < State .

  vars REST : Configuration .
  vars I J K : Nat .

  op phil_hasOneStickAndOtherAvailable : Nat -> Prop [ctor] .
  ceq (REST  < I : Philosopher | #sticks : 1 >  chopstick(J))
    |= phil I hasOneStickAndOtherAvailable = true
    if I can use stick J .

  op phil_eating : Nat -> Prop [ctor] .
  eq (REST < I : Philosopher | #sticks : K >)
    |= phil I eating = (K == 2) .

--- additional fairness assumptions here
endom)

Filosof 3 spiser uendelig mange ganger (i alle oppførsler)
Maude> (red modelCheck(initState, [] <> phil 3 eating) .)

Filosofene 3 og 4 spiser aldri samtidig:
Maude> (red modelCheck(initState, [] ~ ((phil 3 eating) /\ (phil 4 eating))) .)
```

```
--- Compassion og justice fairness

op fair : Nat -> Formula .
eq fair(I) =
      ([] <> phil I hasOneStickAndOtherAvailable) -> ([] <> phil I eating) .
op globalFairness : -> Formula .
eq globalFairness = fair(1) /\ fair(2) /\ fair(3) /\ fair(4) /\ fair(5) .
```

```
Maude> (red modelCheck(initState,
                       globalFairness -> ([] <> phil 3 eating)) .)
```

Rekkefølge på eksekvering

`(red modelCheck(init3,  ~ (<> (exInCS(n8) /\ <> (exInCS(n6) /\ <>  (exInCS(root) /\ <> exInCS(n4)))))) .)`

`(red modelCheck(init3, ~ (<> (exInCS(n2) /\ <> (~ exInCS(n2) /\ <> exInCS(n2))))) .)`

▼ MUTEX

1. To prosesser er aldri inCS samtidig
   a. Hvordan sjekke?
2. Hver prosess eksekverer uendelig ofte inCS
   a. Hvordan sjekke?

      Modelchecker

```
subsort Configuration > State .
ops outCS waitCS inCS : Oid -> Prop [ctor] .

vars O O1 O2 : Oid . var REST : Configuration .
vars P Q : Formula . var N : Nat .

eq REST < O : Node | state : outCS > |= outCS(O) = true .
eq REST < O : Node | state : waitCS > |= waitCS(O) = true .
eq REST < O : Node | state : inCS > |= inCS(O) = true .

*** Lack of fairness
--- Maude> (red modelCheck(init(3), [] inCS(n1:Oid))) .)
--- result ModelCheckResult : counterexample(...)

op justiceRule1 : Oid -> Formula .
eq justiceRule1(O) = (<> [] outCS(O)) -> <> [] outCS(O) .

*** FRA MUTEX-RING.maude kode
op justice : Oid -> Formula .
eq justice(0) = (<> [] (O is outCS)) -> ([] <> (O is waitCS)) .
op globalJustice : -> Formula .
eq globalJustice = justice(n1) /\ justice(n2) /\ justice(n3) /\ justice(n4) /\ justice(n5) .
```

3. Prosessene får tilgang til CS i den rekkefølgen som de ønsket seg inn
   a. Hvordan sjekke?

▼ Om emnet: Modell-sjekking og teorem-bevis