3
Algebraisk Specifikation af Abstrakte Datatyper.

Specifikation kontra program.

Bestanddele af en algebraisk specifikation.

Klassificering af funktioner i en ADT.

Systematisk definition af ligninger.

Fejlhåndtering.

PS1 -- Algebraisk specifikation

Noter

Specifikation i forhold til program.

- En specifikation af en abstrakt datatype udtrykker, *hvad* operationerne gør, snarere end *hvordan* .
- En specifikation af en abstrakt datatype tjener som et forlæg og en rettesnor for en mængde mulige implementationer af typen.
- Specifikationsteknikker:
  - Algebraiske specifikationer.
  - Specifikation af logiske udtryk.

Noter

I den foregående forelæsning har vi introduceret *ideen* om abstrakte datatyper.

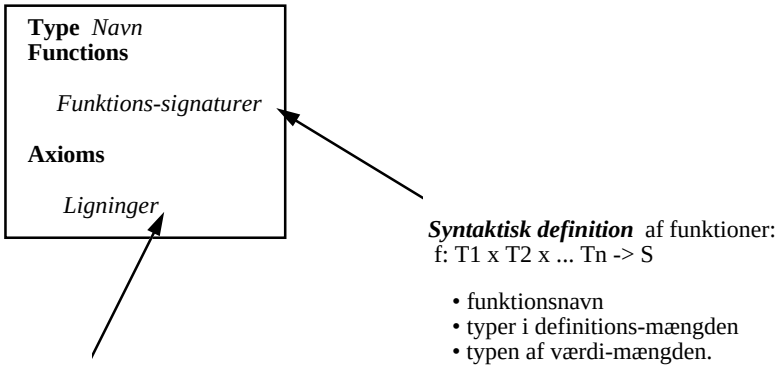
På basis af disse ideer kan der gives konkrete bud på, hvordan vi beskriver abstrakte datatyper i forskellige programmeringssprog. Det er her, det programmeringssproglige begreb *klasser* kommer ind i billedet. Det vil vi bruge meget tid på lidt senere i kurset.

Man kan også gå "i den modsatte retning". Givet ideen om abstrakte datatyper, kan vi bevæge os i den matematiske retning, og på et mere formelt grundlag beskrive egenskaber ved abstrakte datatyper.

I denne forelæsning vil vi se på én sådan formalisme til specifikation af abstrakte datatyper; den såkaldte algebraiske specifikationsformalisme.

Senere i kurset vil vi studere en anden specifikationsmetode, hvor logiske udsagn i begyndelsen og i slutningen af operationer angiver, hvad disse skal gøre.

Bestanddele af en algebraisk specifikation.



Syntaktisk definition af funktioner:
f: T1 x T2 x ... Tn -> S

- funktionsnavn
- typer i definitions-mængden
- typen af værdi-mængden.

Semantisk definition af funktioner:

Ligninger, som udtrykker gensidige afhængigheder mellem funktionerne, typisk gennem brug af rekursion.

PS1 -- Algebraisk specifikation

Noter

I en algebraisk specifikation er operationerne alle funktioner. Dette udelukker altså operationer, som påvirker et objekt ved at ændre objektets tilstand. I den idealiserede verden, hvori specifikationer fungerer, er dette imidlertid ikke nogen begrænsning. Enhver ændring i tilstanden af et objekt (af en eller anden type) kan beskrives ved en funktion, der returnerer en modificeret kopi af objektet som resultat.

Eksempel: Hvis vi specificerer en mængde (af heltal) vil man typisk i en implementation have en operation, der inkluderer et tal i mængden. Dette ændrer mængde-objektets tilstand. I en algebraisk specifikation tænker vi på denne operation som en funktion, der givet en mængde og et element returnerer en ny mængde som resultat. Vi vil i denne forelæsning vende tilbage til dette eksempel.

Eksempel: specifikation af naturlige tal.

type Nat0
functions
null: -> Nat0
succ: Nat0 -> Nat0
plus: Nat0 x Nat0 -> Nat0
leq: Nat0 x Nat0 -> Boolean

axioms
for all i, j in Nat0
plus (null(), j) = j
plus(succ(i),j) = succ(plus(i, j))
leq(null(), j) = true
leq(succ(i), null()) = false
leq(succ(i), succ(j)) = leq(i, j)

Noter

Værdier af den abstrakte datatype, i dette eksempel, naturlige tal, udtrykkes i termer af konstruktorerne. Et vilkårligt naturligt tal kan udtrykkes som succ(succ (... null () ...)); altså som det antal gange vi skal addere tallet 1 til 0 for at opnå det pågældende tal.

Man kan sige, at et objekt repræsenteres ved dets *operationshistorie*.

Reduktion af udtryk. Givet et vilkårligt udtryk U kan man omskrive U ved brug af en eller flere af ligningerne. En særlig interessant omskrivning består i at reducere udtrykket til et udtryk, som kun består af konstruktorer (se næste slide).

Eksempel: plus(succ(succ(null())), succ(null())). Dette udtryk kan ved to anvendelser af ligning 2 og én anvendelse af ligning 1 reduceres til succ(succ(succ(null()))). Vi har i alt beskedenhed udledt, at 2+1 er lig med 3.

Det ses, at en værdi i en abstrakt datatype repræsenteres som et udtryk i konstruktorerne. Dette udtryk afspejler direkte eller indirekte historien af operationer udført på instansen af typen.

Princippet i opskrivningen af ligningerne kaldes *generator induktion* (konstruktor induktion), idet der ligesom ved induktion er basistilfælde samt induktionstrin, der gradvis nærmer sig basis-situationen.

Grundlaget for specifikationen af Nat0 er Peano's klassiske definition af de naturlige tal.

Opgave: Udvid specifikationen med en subtraktionsfunktion *inden for de naturlige tal*.

Klassificering af funktioner i en ADT:

Specifikation af en abstrakt datatype T:

- **Konstruktorer** f: ... -> T

  De funktioner, der er nødvendige og tilstrækkelige til at generere værdier af den abstrakte datatype.

- **Transformatorer** f: ... -> T

  Andre funktioner, der har værdier af T som resultat-type.

- **Accessorer** f: ... T ... -> S (S en anden type end T).

  Funktioner, der afbilder T over i andre typer (giver oplysninger om T).

Noter

I litteraturen benyttes ofte en anderledes terminologi:

- Konstruktorer kaldes for generatorer.
- Transformatorer kaldes for extensorer.
- Accessorer kaldes for observatorer.

Konstruktorer og transformatorer er har det tilfælles, at de er *producenter* af værdier af den abstrakte datatype T, som vi er igang med a specificere.

Det er et spørgsmål om smag og behag, og måske også om perspektiv, der afgør om man bruger det ene eller det andet sæt af terminologi. Konstruktorer, transformatorer og accessorer passer godt til det objekt-orienterede perspektiv. Med et mere funktionelt perspektiv er det ofte mere naturligt at bruge de alternative ord.

Eksempel: specifikation af mængder.

```
type Set[X]
functions
  empty:  -> Set[X]
  insert: Set[X] x X -> Set[X]
  delete: Set[X] x X -> Set[X]
  member: Set[X] x X -> boolean
  intersection: Set[X] x Set[X] -> Set[X]
axioms
 for all s, t in set[X],  e, f in X:
  delete (empty (), e) = empty()
  delete (insert (s, e), f) =  if e=f
                               then delete(s,f)
                               else insert (delete(s, f), e)
  member(empty(), e) = false
  member(insert(s, e), f) = if e = f
                             then true
                             else member(s,f)
  intersection(empty(), s) = empty()
  intersection(insert(s, e), t ) = if member(t, e)
                                   then insert(intersection (s, t), e)
                                   else intersection (s, t)
```

PS1 -- Algebraisk specifikation

Noter

Kontruktorerne er empty og insert. Empty() betegner den tomme mængde, og insert indsætter et element af typen X i mængden. Den resulterende mængde returneres som resultat af funktionen insert. Delete og intersection er transformatorer. Delete(s, e) betegner mængden S \ {e}; intersection(s,t) betegner fællesmængden mellem s og t. Member er en accessor, som oplyser hvorvidt et element tilhører en mængde.

En karakteristisk egenskab ved mængder er jo, at et element kun forekommer én gang i en mængde. Det er vigtigt at konstatere, hvordan vi håndterer dette i ovenstående specifikation. Vi kan ikke i en algebraisk specifikation forhindre, at et bestemt element indsættes to eller flere gange via insert. Men vi skal udaf til efterlade det indtryk, at elementet kun forekommer een gang. Hvis vi f.eks. indsætter et element e to gange i mængden s, og derefter sletter det (én gang) med delete, så skal member(s,e) returnere falsk.

Det er værd at bemærke, at vi i denne specifikation har et eksempel på, at én af de specificerede accessorer kan udnyttes internt i specifikationen af en anden funktion. Det drejer sig om funktionen member, som benyttes i transformatoren intersection.

Opgave: Udvid specifikationen med operationerne union (foreningsmængde) og cardinalitet (antallet af elementer i mængden).

Tommelfingerregler for definition af ligninger.

Systematisk generator-induktion:

Vi er igang med at specificere T:

- f: A1 x A2 x ... An -> A er en ligning i specifikationen.
- f er enten en transformator eller en accessor.

> - Alle kombinationer af konstruktorerne på de pladser, hvor Ai = T giver anledning til en ligning i specifikationen.
> - Det er ofte muligt at slå to eller flere ligninger sammen til én.

Fuldstændighed af specifikationen: *Er der ligninger nok?*

Et vilkårligt udtryk kan reduceres til et udtryk i konstruktorerne eller til en værdi i en anden type.

Konsistens af specifikationen: *Er der ligninger, der giver modstrid?*

Et udtryk kan ikke reduceres til mere end ét resultat.

PS1 -- Algebraisk specifikation

Noter

Princippet om systematisk generator-induktion (konstruktor induktion) giver en meget praktisk og håndfast måde at opskrive ligner på. Hvis man følger princippet er det helt trivielt at afgøre hvormange ligninger der er nødvendige og tilstrækkelige, således at man opnår fuldstændighed og konsistens.

Det ønskelige i at "slå nogle ligninger sammen" er af sekundær betydning. Dog kan det bidrage til et bedre overblik over, og en form for "indre skønhed" i specifikationen, når antallet af ligninger er slået passende sammen.

Fejlhåndtering.

Det er nødvendigt at kunne håndtere fejl i specifikationen.

Introducere fejlværdier.

En funktion anvendt på en fejlværdi giver en ny fejlværdi.

Introducere pre-betingelser.

En funktion kan kun anvendes, hvis dets pre-betingelse er opfyldt.

Noter

Af de to alternative måder at håndtere fejl på vælger vi som regel fejlværdier.

Eksempel: Hvis pred: Nat0 -> Nat0 betegner forgænger operationen på naturlige tal (incl. 0) vil vi have en ligning:

pred(null()) = **error**

idet forgængeren af 0 plejer at være -1, og -1 er ikke et element i Nat0.

I OOSC er det vist, hvordan man kan bruge prebetingelser for at fange forskellige fejlsituationer. Prebetingelsen for pred skal udtrykke, at parameteren til pred skal være mindst tallet en.