Statisk semantisk analyse - Kap. 6

- Generelt om statisk semantisk analyse
- Attributt-grammatikker
- Symboltabell
- Datatyper og typesjekking

INF 5110 - 2011

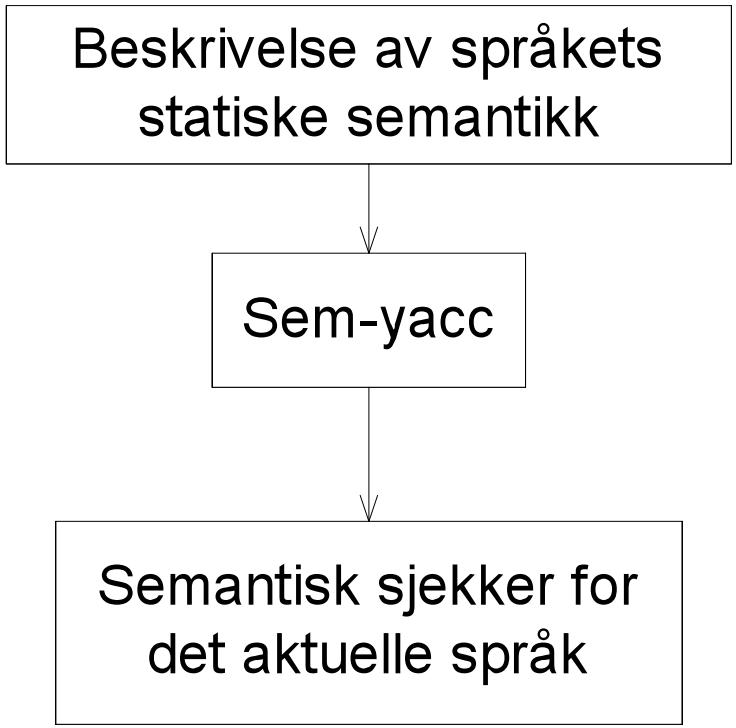
Generelt om semantisk analyse

- Oppgave: Sjekke alle krav i språkdefinisjonen som
  - kan sjekkes før utførelsen
  - ikke naturlig sjekkes under syntaktisk analyse
- Merk
  - Ikke altid klart hvor grensen mellom syntaktisk/semantisk sjekk går (eller 'bør velges')
  - if a then ... Må godkjennes syntaktisk
  - if a+b then ... ??

Poenger og begreper

- Typiske ting som sjekkes
  - at bruk av navn er konsistent med deres deklarasjon
  - at typen av (sub)-uttrykk stemmer med operasjonene
- Det vil altid være ting som ikke kan sjekkes før utførelsen
  - ”index out of range” i arrayer
  - ”none-test” for ”remote access” r.a
- Stor forskjell på språk med og uten typer på variable og parametre
  - data er egentlig altid typet
  - om variable ikke er typet: testing hele tiden på at data-operasjoner er OK
  - om variable er typede: Nesten all sjekk kan gjøres på kompileringstidspunktet, og det kan genereres bedre kode

En drøm



men

- Intet standard beskrivelsesspråk
- Input til semantisk sjekker er rimelig komplisert
- Det er altid en masse ad-hoc regler

Derfor

- Man ser på generelle metoder
- Men, de må programmeres i hvert tilfelle

Why MDA PIM to PSM via a model of the platform will stay a dream

Attributt og attributt-grammatikker

- En attributt er en egenskap ved et språkbegrep
- Eksempler:

  - The data type of a variable — statisk? også: typen til uttrykk
  - The value of an expression — dynamisk? (av og til statisk)
  - The location of a variable in memory — {Fortran: statisk, Java: dynamisk}
  - The object code of a procedure — statisk?
  - The number of significant digits in a number — statisk

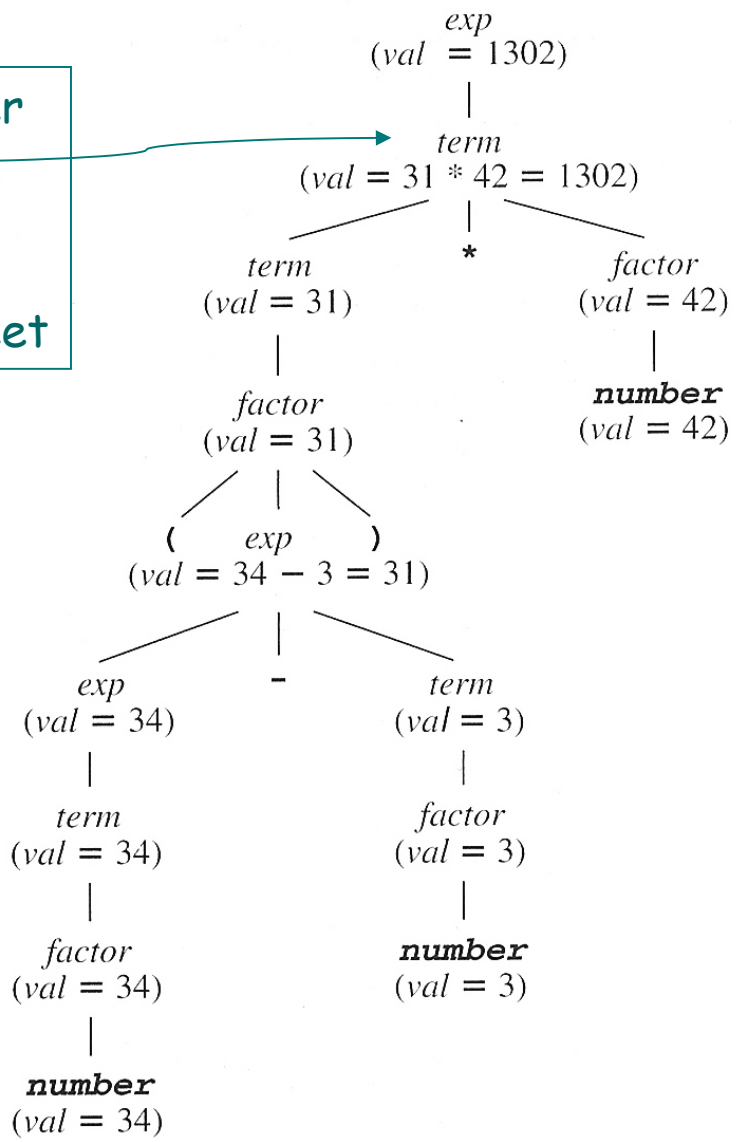
- Statiske attributter: Kan beregnes før utførelsen
- Dynamiske attributter: Må beregnes under utførelsen

- For attributt-grammatikker er alle attributter statiske, og de er definert i tilknytning til grammatikken for språket

Attributt-grammatikker

$exp \rightarrow exp$ **+** $term \mid exp$ **-** $term \mid term$
$term \rightarrow term$ ***** $factor \mid factor$
$factor \rightarrow$ **(** exp **)** $\mid$ **number**

Attributter er variable knyttet til nodene i parseringstreet



Hver semantisk regel er knyttet til en produksjon

Deres verdier definert ved semantiske regler

Grammar Rule	Semantic Rules
$exp_1 \rightarrow exp_2$ **+** $term$	$exp_1.val = exp_2.val + term.val$
$exp_1 \rightarrow exp_2$ **-** $term$	$exp_1.val = exp_2.val - term.val$
$exp \rightarrow term$	$exp.val = term.val$
$term_1 \rightarrow term_2$ ***** $factor$	$term_1.val = term_2.val * factor.val$
$term \rightarrow factor$	$term.val = factor.val$
$factor \rightarrow$ **(** exp **)**	$factor.val = exp.val$
$factor \rightarrow$ **number**	$factor.val =$ **number**$.val$

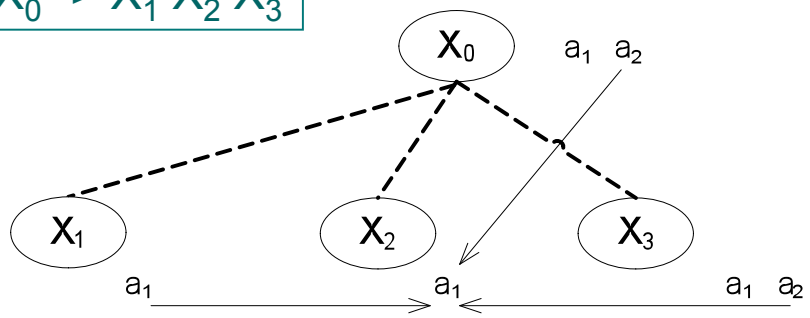
Bare ett attribut til hvert symbol, alle har samme navn, bare een regel pr produksjon

Attributt-grammatikker

- Gitt en grammatikk på ren BNF-form
  - For hvert grammatikk-symbol X (terminal eller ikke-terminal) skal det være gitt en mengde (navnede) attributter
  - Attributt-mengdene for de forskjelige symboler kan generelt være helt forskjellige, men har ofte mye felles
  - Attributtene er ment å materialisere seg som variable knyttet til nodene i et parseringstre
  - Attributten a til noden X skrives X.a
  - Attributtene er definert ved at det til hver regel er knyttet en likning av formen:

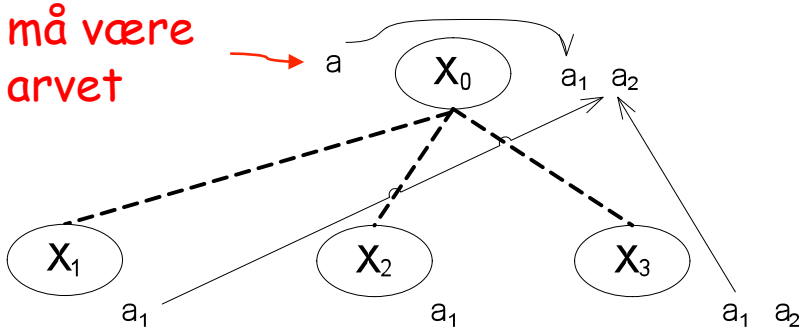
$$X_i.a_j = f_{ij}(X_0.a_1, \ldots, X_0.a_k, X_1.a_1, \ldots, X_1.a_k, \ldots, X_n.a_1, \ldots, X_n.a_k)$$

X_0 -> X_1 X_2 X_3



arvet attributt

må være arvet



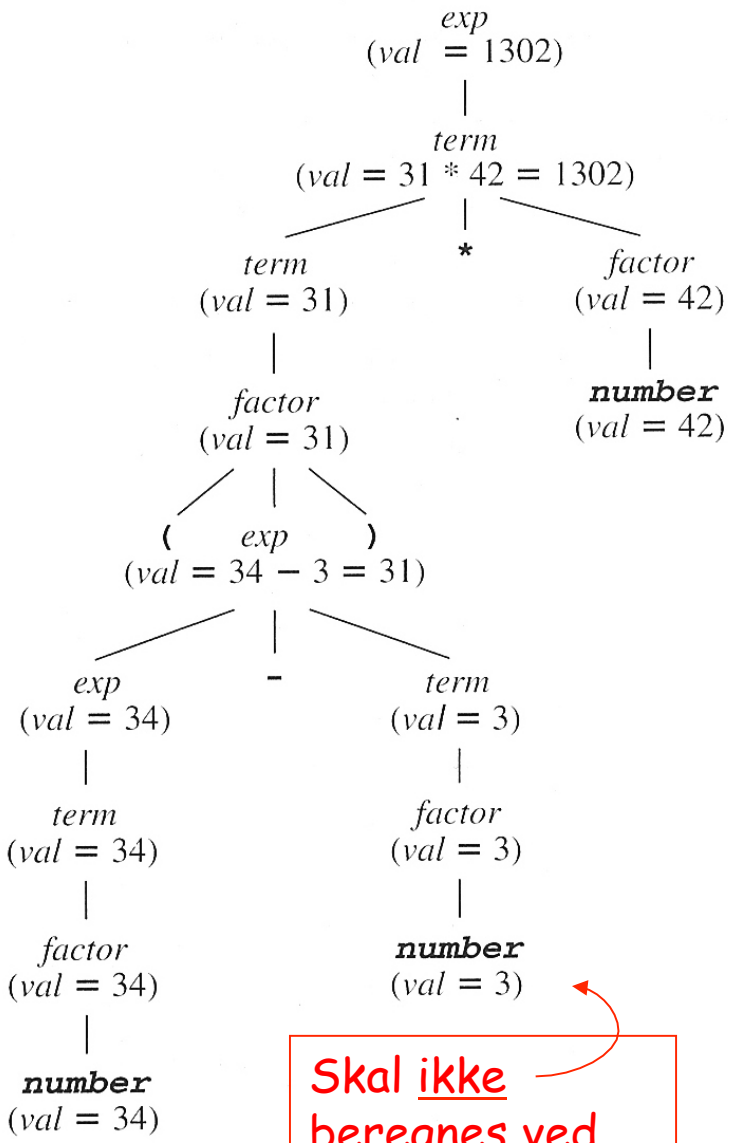
syntetisert attributt

Eksempel

$exp \rightarrow exp$ **+** $term \mid exp$ **-** $term \mid term$
$term \rightarrow term$ ***** $factor \mid factor$
$factor \rightarrow$ **(** exp **)** $\mid$ **number**

Alle symboler (bortsett fra '+', '-' og '*') har det ene attributtet 'val'.
exp_i brukes hvis samme symbol fler ganger

Grammar Rule	Semantic Rules
$exp_1 \rightarrow exp_2$ **+** $term$	$exp_1.val = exp_2.val + term.val$
$exp_1 \rightarrow exp_2$ **-** $term$	$exp_1.val = exp_2.val - term.val$
$exp \rightarrow term$	$exp.val = term.val$
$term_1 \rightarrow term_2$ ***** $factor$	$term_1.val = term_2.val * factor.val$
$term \rightarrow factor$	$term.val = factor.val$
$factor \rightarrow$ **(** exp **)**	$factor.val = exp.val$
$factor \rightarrow$ **number**	$factor.val =$ **number**$.val$



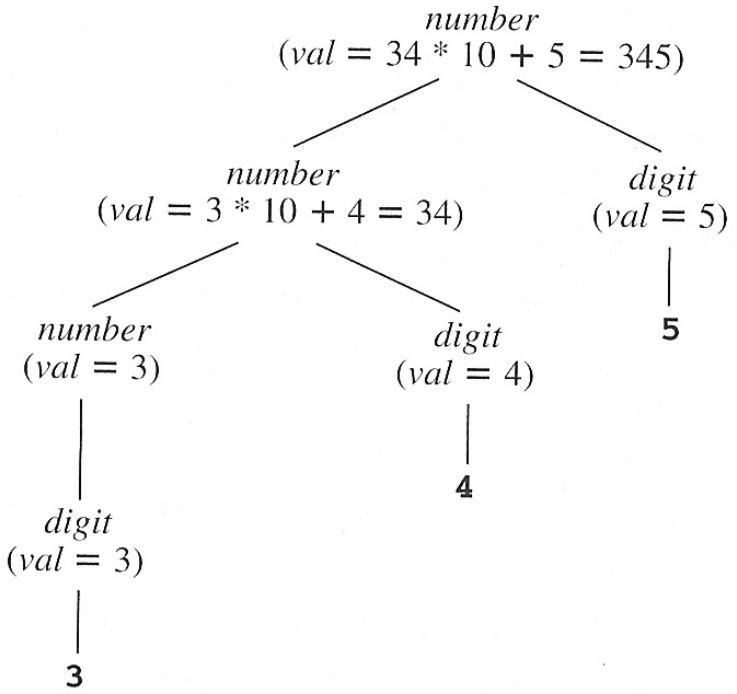
Skal ikke beregnes ved semantiske regler

Eksempel

number → *number digit* | *digit*
digit → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Attributter
number: val
digit: val
Terminaler: - (ingen)

Grammar Rule	Semantic Rules
$number_1 \rightarrow number_2\ digit$	$number_1.val = number_2.val * 10 + digit.val$
$number \rightarrow digit$	$number.val = digit.val$
$digit \rightarrow$ **0**	$digit.val = 0$
$digit \rightarrow$ **1**	$digit.val = 1$
$digit \rightarrow$ **2**	$digit.val = 2$
$digit \rightarrow$ **3**	$digit.val = 3$
$digit \rightarrow$ **4**	$digit.val = 4$
$digit \rightarrow$ **5**	$digit.val = 5$
$digit \rightarrow$ **6**	$digit.val = 6$
$digit \rightarrow$ **7**	$digit.val = 7$
$digit \rightarrow$ **8**	$digit.val = 8$
$digit \rightarrow$ **9**	$digit.val = 9$



Litt merkelig eksempel, da dette vanligvis gjøres i scanneren

INF 5110 - 2011

Syntetiserte attributter

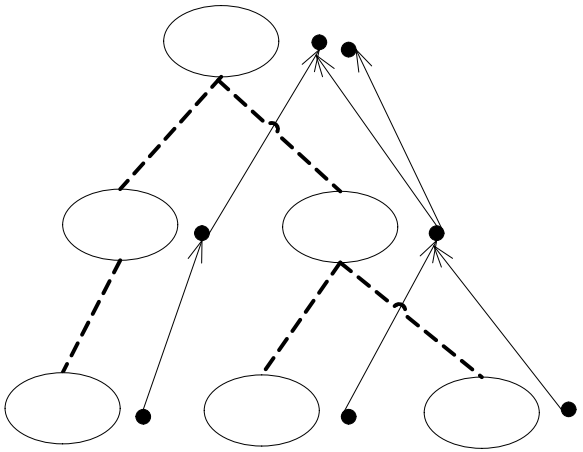
An attribute is **synthesized** if all its dependencies point from child to parent in the parse tree. Equivalently, an attribute a is synthesized if, given a grammar rule $A \rightarrow X_1 X_2 \ldots X_n$, the only associated attribute equation with an a on the left-hand side is of the form

$$A.a = f(X_1.a_1, \ldots, X_1.a_k, \ldots, X_n.a_1, \ldots, X_n.a_k)$$

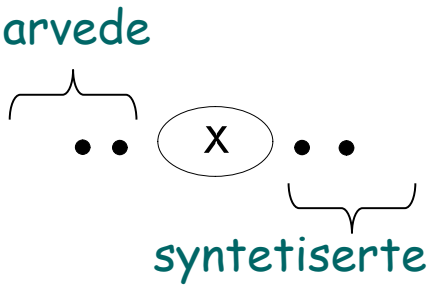
An attribute grammar in which all the attributes are synthesized is called an **S-attributed grammar**.

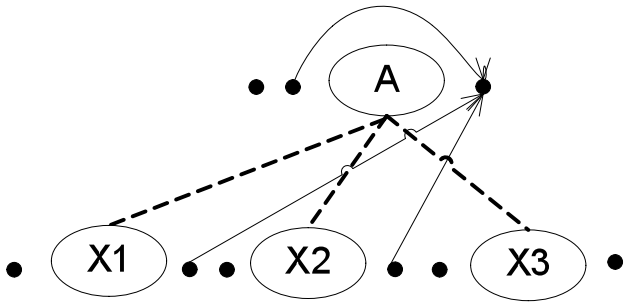
Trykkfeil: A.a får også være avhengig av A.b dersom b er et arvet attributt

- NB: Hvert attributt må velges til enten å være syntetisert eller arvet



Avhengighet i en S-attribut-grammatikk





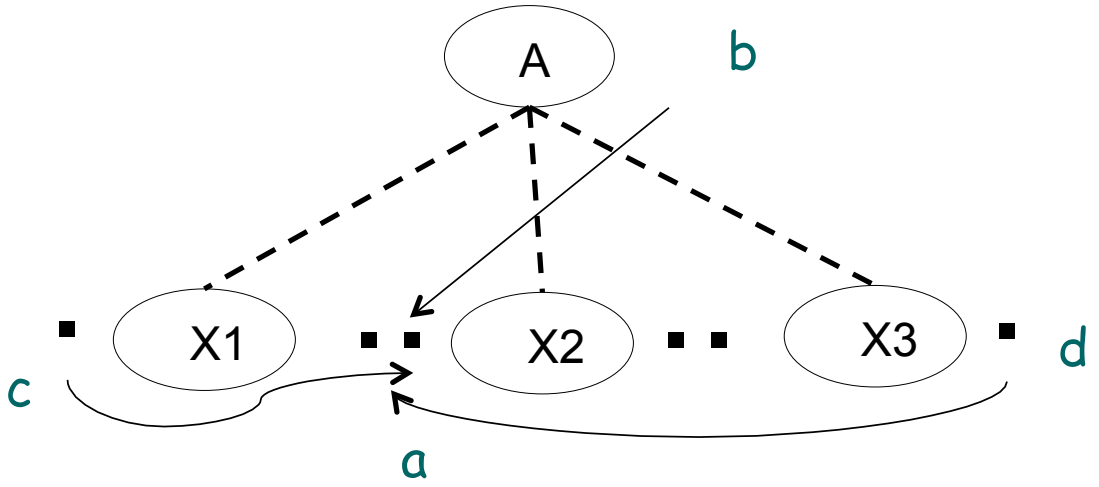
Mulig avhengighet for syntetisert attributt

Arvede attributter

- ”An attribute that is not synthesized is called an inherited attribute.”

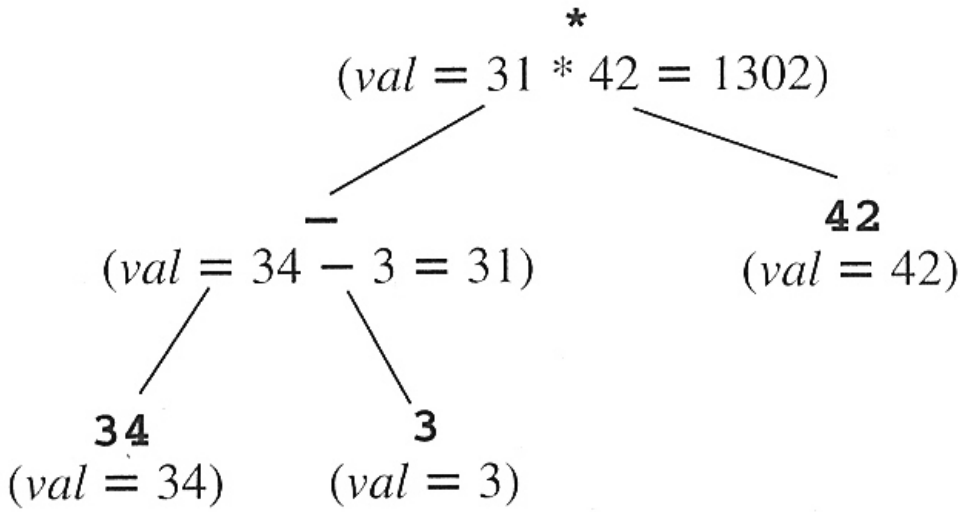
Heller:
et attributt sies å være arvet, når det defineres for et symbol på høyresiden av de produksjoner det opptrer i

$A \rightarrow X_1\ X_2\ X_3$ $\qquad$ $X_2.a = A.b + X_1.c + X_3.d$



Man kan gjerne tenke i

- flertydige grammatikker
- abstrakte syntakstrær



Attributter
exp: val
number: val (fra scanneren)
terminaler: -

$$exp \rightarrow exp \texttt{ + } exp \mid exp \texttt{ - } exp \mid exp \texttt{ * } exp \mid \texttt{(} exp \texttt{)} \mid \textbf{number}$$

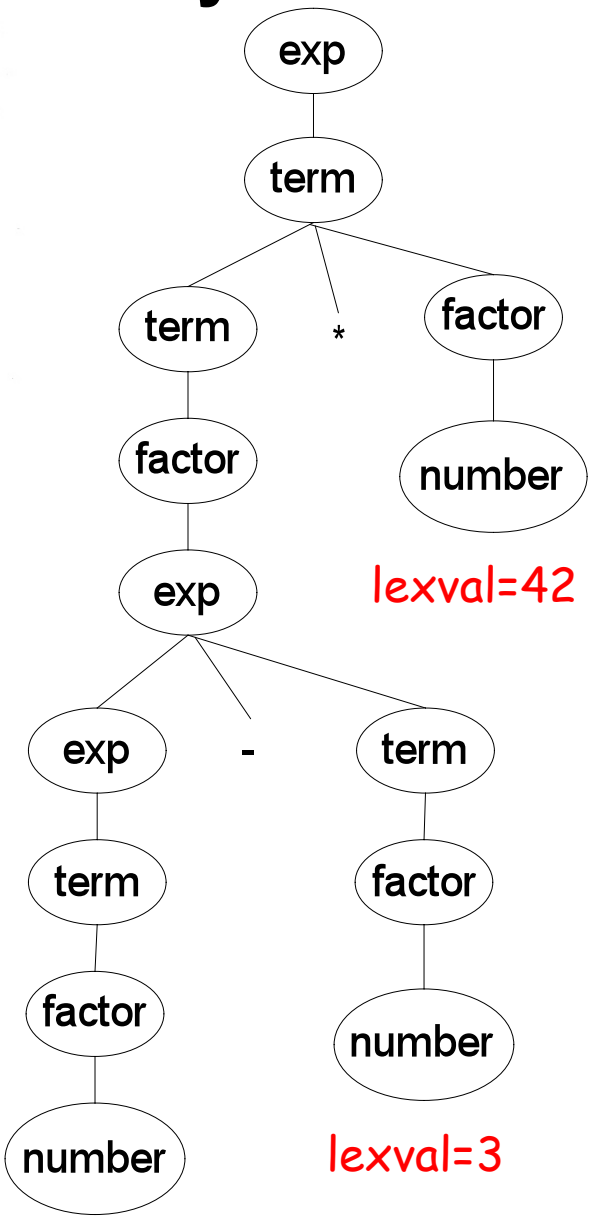
Grammar Rule	Semantic Rules
$exp_1 \rightarrow exp_2 \texttt{ + } exp_3$	$exp_1.val = exp_2.val + exp_3.val$
$exp_1 \rightarrow exp_2 \texttt{ - } exp_3$	$exp_1.val = exp_2.val - exp_3.val$
$exp_1 \rightarrow exp_2 \texttt{ * } exp_3$	$exp_1.val = exp_2.val * exp_3.val$
$exp_1 \rightarrow \texttt{(} exp_2 \texttt{)}$	$exp_1.val = exp_2.val$
$exp \rightarrow \textbf{number}$	$exp.val = \textbf{number}.val$

Attr.-grammatikk som 'bygger' abstrakt syntakstre

Grammar Rule	Semantic Rules
$exp_1 \rightarrow exp_2$ **+** $term$	$exp_1.tree =$ $mkOpNode(+, exp_2.tree, term.tree)$
$exp_1 \rightarrow exp_2$ **-** $term$	$exp_1.tree =$ $mkOpNode(-, exp_2.tree, term.tree)$
$exp \rightarrow term$	$exp.tree = term.tree$
$term_1 \rightarrow term_2$ ***** $factor$	$term_1.tree =$ $mkOpNode(*, term_2.tree, factor.tree)$
$term \rightarrow factor$	$term.tree = factor.tree$
$factor \rightarrow$ **(** exp **)**	$factor.tree = exp.tree$
$factor \rightarrow$ **number**	$factor.tree =$ $mkNumNode($**number**$.lexval)$

- Attributter:
  - exp: tree
  - term: tree
  - factor: tree
  - number: lexval

Merk: det abstrakte syntakstreet bygges ut fra at vi har det konkrete parseringstreet



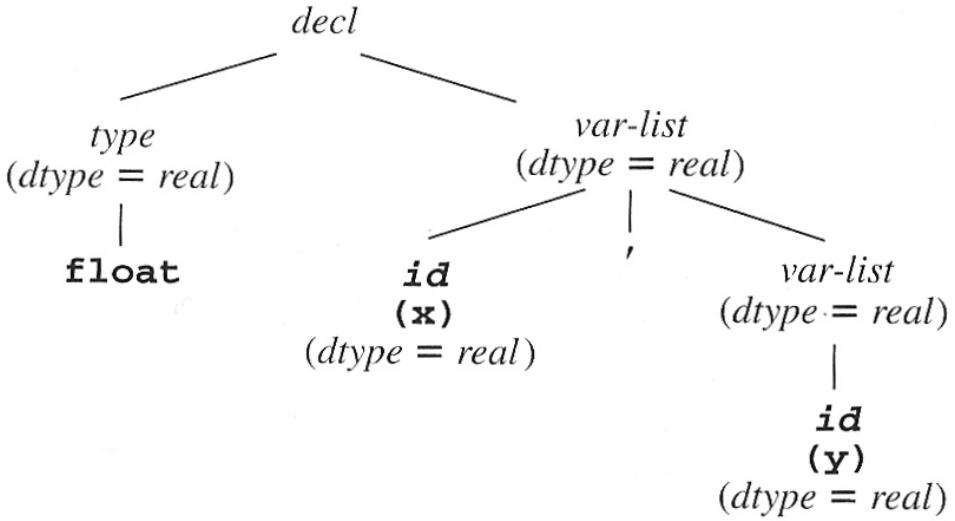
INF 5110 - 2011

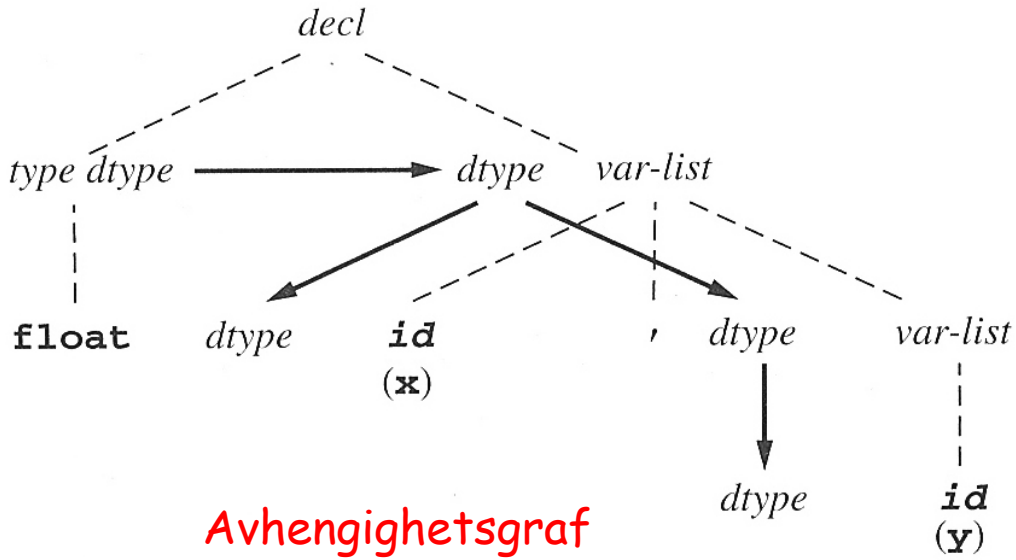
Grammatikk med arvede attributter

Grammar Rule	Semantic Rules
$decl \rightarrow type\ var\text{-}list$	$var\text{-}list.dtype = type.dtype$
$type \rightarrow$ **int**	$type.dtype = integer$
$type \rightarrow$ **float**	$type.dtype = real$
$var\text{-}list_1 \rightarrow$ **id,** $var\text{-}list_2$	**id** $.dtype = var\text{-}list_1.dtype$
	$var\text{-}list_2.dtype = var\text{-}list_1.dtype$
$var\text{-}list \rightarrow$ **id**	**id** $.dtype = var\text{-}list.dtype$

- Attributtet 'dtype' er her et arvet attributt
- Vi kan skrive om grammatikken slik at den blir 'nesten' syntetisert (eksempel 6.18)

Attributter
decl: -
type, varlist,id: dtype





Avhengighetsgraf

Eksempel 6.18

Denne ga arvede attr.:

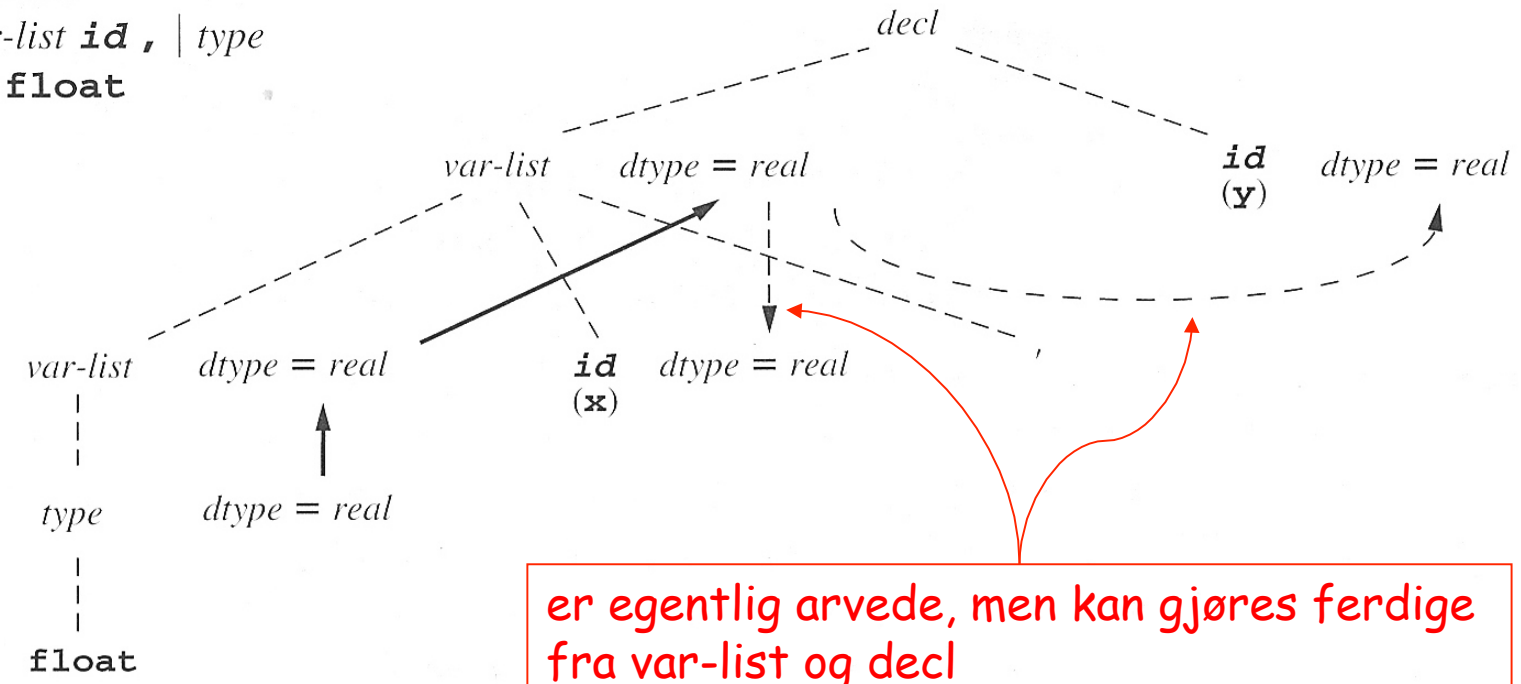
decl → *type var-list*
type → **int** | **float**
var-list → **id** , *var-list* | **id**

Grammar Rule	Semantic Rules
decl → *var-list* **id**	**id** .*dtype* = *var-list.dtype*
$var\text{-}list_1 \rightarrow var\text{-}list_2$ **id** ,	$var\text{-}list_1$.*dtype* = $var\text{-}list_2$.*dtype*
	id .*dtype* = $var\text{-}list_1$.*dtype*
var-list → *type*	*var-list.dtype* = *type.dtype*
type → **int**	*type.dtype* = *integer*
type → **float**	*type.dtype* = *real*

Nytt forslag:

decl → *var-list* **id**
var-list → *var-list* **id** , | *type*
type → **int** | **float**

dtype er nå nesten bare syntetisert



er egentlig arvede, men kan gjøres ferdige fra var-list og decl

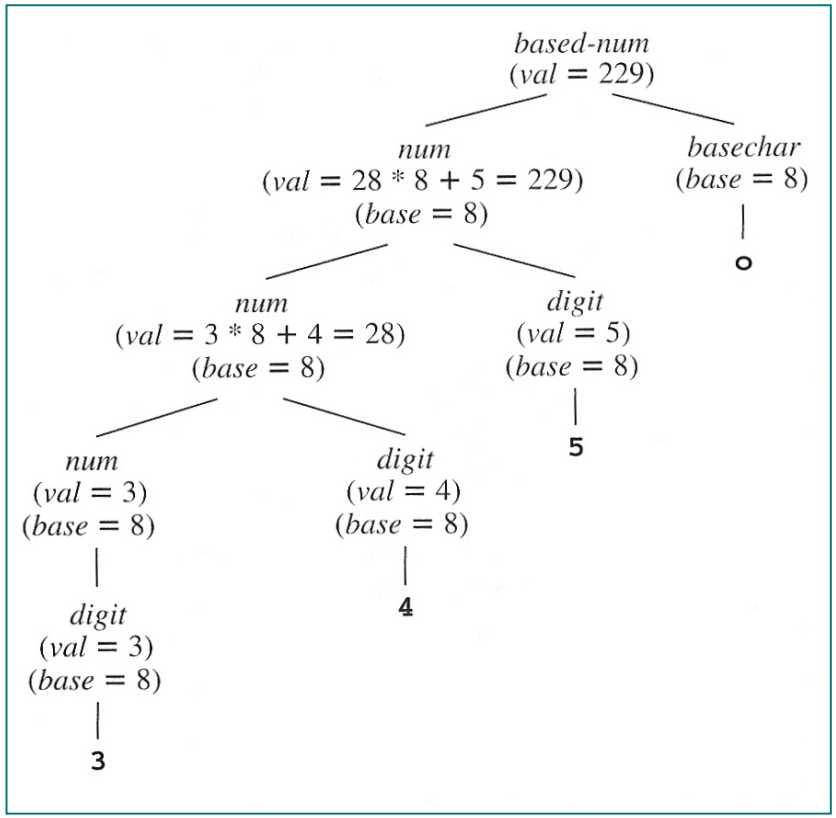
Eksempel på arvede attributter

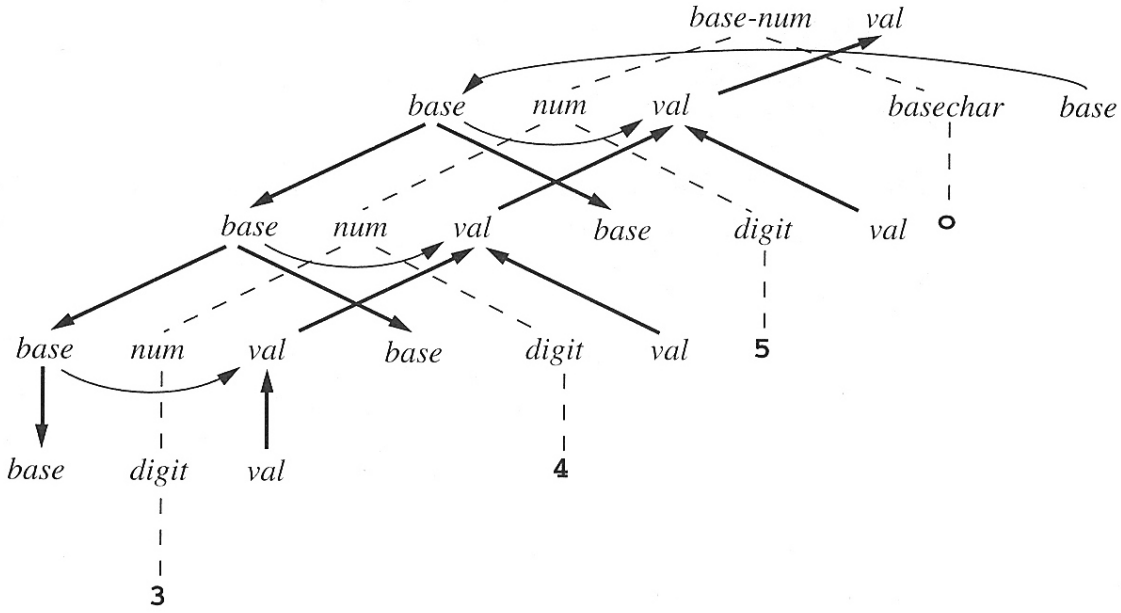
based-num → *num basechar*
basechar → `o` | `d`
num → *num digit* | *digit*
digit → `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

Grammar Rule	Semantic Rules
based-num → *num basechar*	*based-num.val* = *num.val* *num.base* = *basechar.base*
basechar → `o`	*basechar.base* = 8
basechar → `d`	*basechar.base* = 10
$num_1 \rightarrow num_2\ digit$	$num_1.val$ = **if** *digit.val* = *error* **or** $num_2.val$ = *error* **then** *error* **else** $num_2.val * num_1.base + digit.val$ $num_2.base = num_1.base$ $digit.base = num_1.base$
num → *digit*	*num.val* = *digit.val* *digit.base* = *num.base*
digit → `0`	*digit.val* = 0
digit → `1`	*digit.val* = 1
. . .	. . .
digit → `7`	*digit.val* = 7
digit → `8`	*digit.val* = **if** *digit.base* = 8 **then** *error* **else** 8
digit → `9`	*digit.val* = **if** *digit.base* = 8 **then** *error* **else** 9

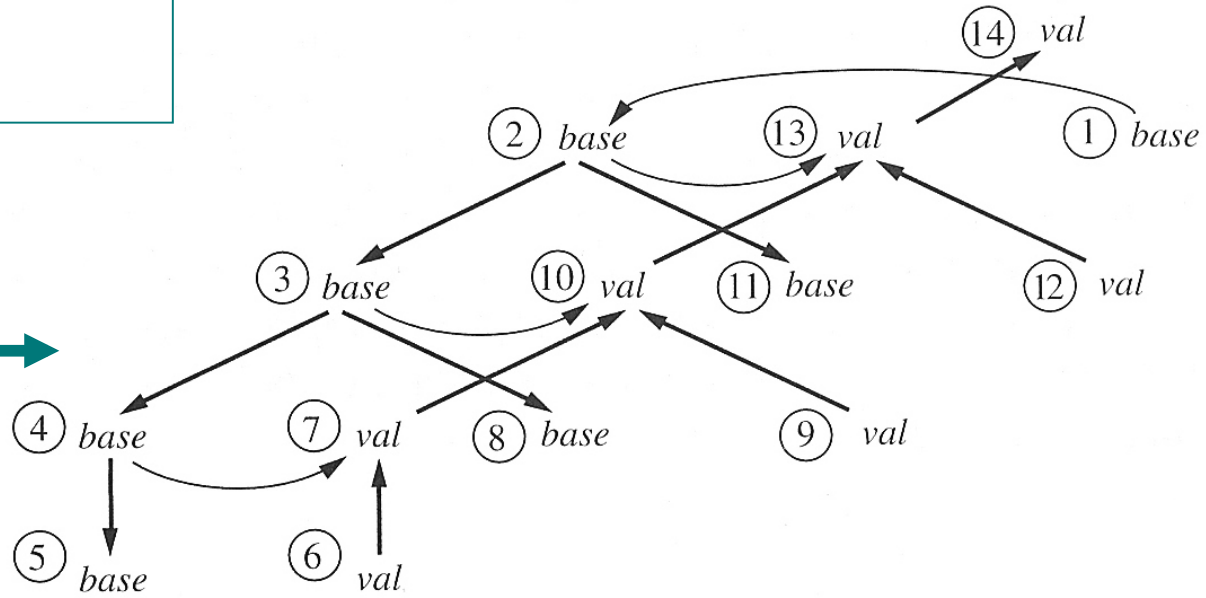
Attributter
based-num: val
basechar: base
(syntetisert)
num: val, base
(base er arvet)
digit: val

(val er overalt syntetisert)





Mulig beregningsrekkefølge



INF 5110 - 2011

Konsistens og beregnbarhet

- Alle attributter er enten arvede eller syntetiserte
  - Men merk: OK at basechar.base syntetisert, mens num.base arvet
- Alle attributter må være definerte. Gjøres ved at innen hver produksjon:
  - alle syntetiserte attributter i venstresiden er definert
  - alle arvede attributter i høyresiden er definert
  - det må ikke være noen lokale avhengighets-løkker

- Siden alle attributter er enten arvede eller syntetiserte, er hvert attributt i et parseringstre definert een og bare een gang

Løkker i avhengighets-grafen

- Ikke i noe syntakstre må det være løkker i avhengighetsgrafen!

- Generelt er dette vanskelig å garantere
- Man kan sjekke det **etter** at treet er bygget. Om ikke løkker: Finn 'topologisk sortering' som gir brukbar rekkefølge
- Det som brukes
  - Man overbeviser seg på forhånd om at det ikke kan bli løkker i noe syntakstre
  - Forhåndsbestemmer en rekkefølge for beregning

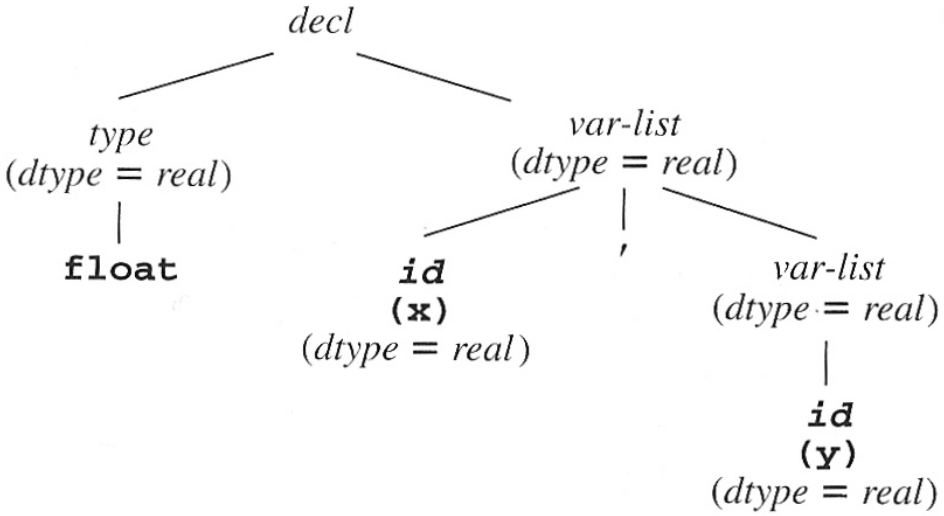
Beregningsalgoritme for type-grammatikk

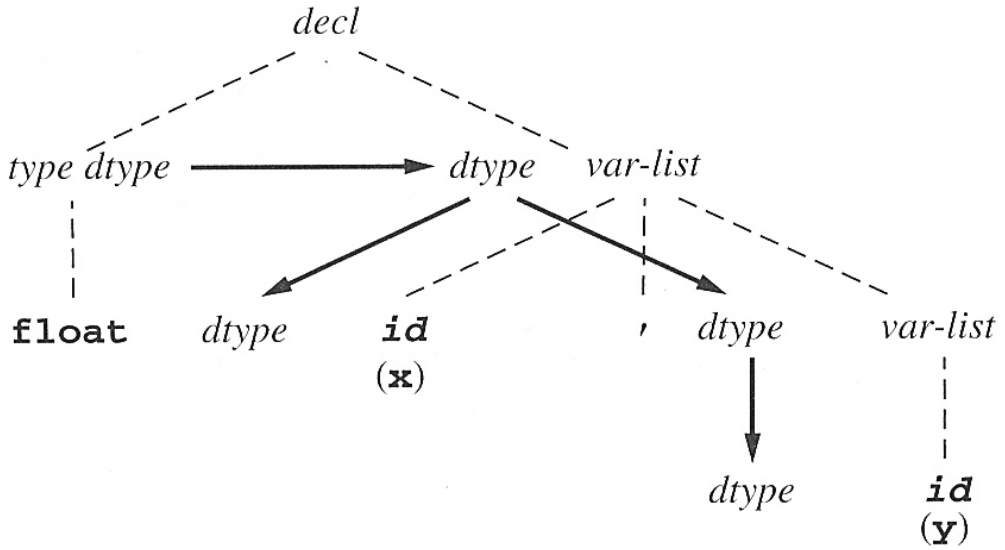
decl → *type var-list*
type → **`int`** | **`float`**
var-list → **`id`**, *var-list* | **`id`**

```
procedure EvalType ( T: treenode );
begin
   case nodekind of T of
   decl:
         EvalType ( type child of T );
         Assign dtype of type child of T to var-list child of T;
         EvalType ( var-list child of T );
   type:
         if child of T = int then T.dtype := integer
         else T.dtype := real;
   var-list:
         assign T.dtype to first child of T;
         if third child of T is not nil then
            assign T.dtype to third child;
            EvalType ( third child of T );
   end case;
end EvalType;
```

- NB: Treet antas å være bygget ferdig på forhånd
- Filosofi
  - Syntetiserte attributter settes i T ut fra barnas verdier
  - Arvede attributter setts i barna til T ut fra verdiene i T og i andre barn

Grammar Rule	Semantic Rules
decl → *type var-list*	*var-list.dtype* = *type.dtype*
type → **int**	*type.dtype* = *integer*
type → **float**	*type.dtype* = *real*
$var\text{-}list_1$ → **id,** $var\text{-}list_2$	**id** .*dtype* = $var\text{-}list_1$.*dtype*
	$var\text{-}list_2$.*dtype* = $var\text{-}list_1$.*dtype*
var-list → **id**	**id** .*dtype* = *var-list.dtype*





Beregning

- for 'base'-grammatikken

- Antar ferdiglaget tre
- Samme filosofi som for foil 20.

```
procedure EvalWithBase ( T: treenode );
begin
  case nodekind of T of
  based-num:
      EvalWithBase ( right child of T );
      assign base of right child of T to base of left child;
      EvalWithBase ( left child of T );
      assign val of left child of T to T.val;
  num:
      assign T.base to base of left child of T;
      EvalWithBase ( left child of T );
      if right child of T is not nil then
        assign T.base to base of right child of T;
        EvalWithBase ( right child of T );
        if vals of left and right children ≠ error then
            T.val := T.base*(val of left child) + val of right child
        else T.val := error;
      else T.val := val of left child;
  basechar:
      if child of T = o then T.base := 8
      else T.base := 10;
  digit:
      if T.base = 8 and child of T = 8 or 9 then T.val := error
      else T.val := numval ( child of T );
  end case;
end EvalWithBase;
```

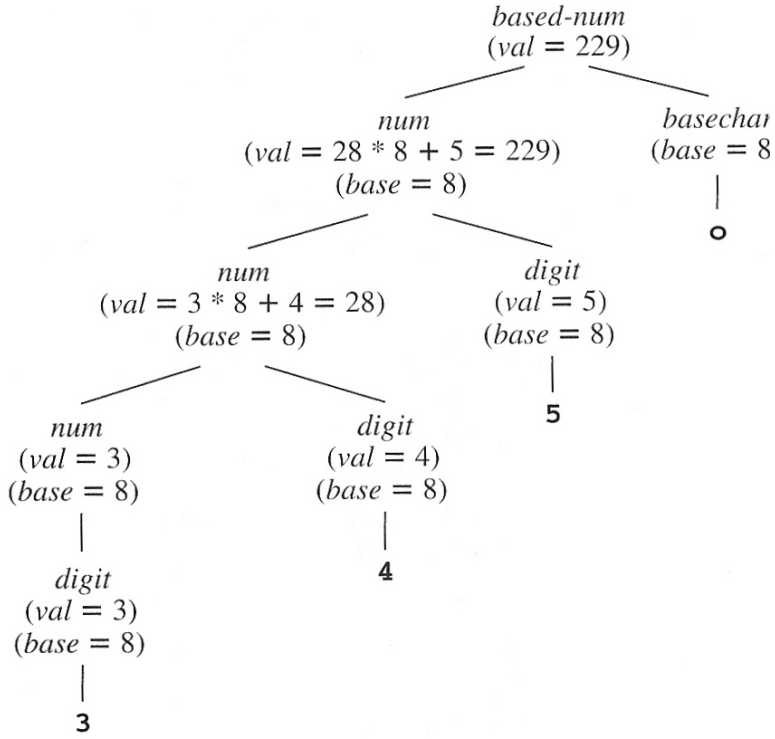
Alternativ til EvalWithBase

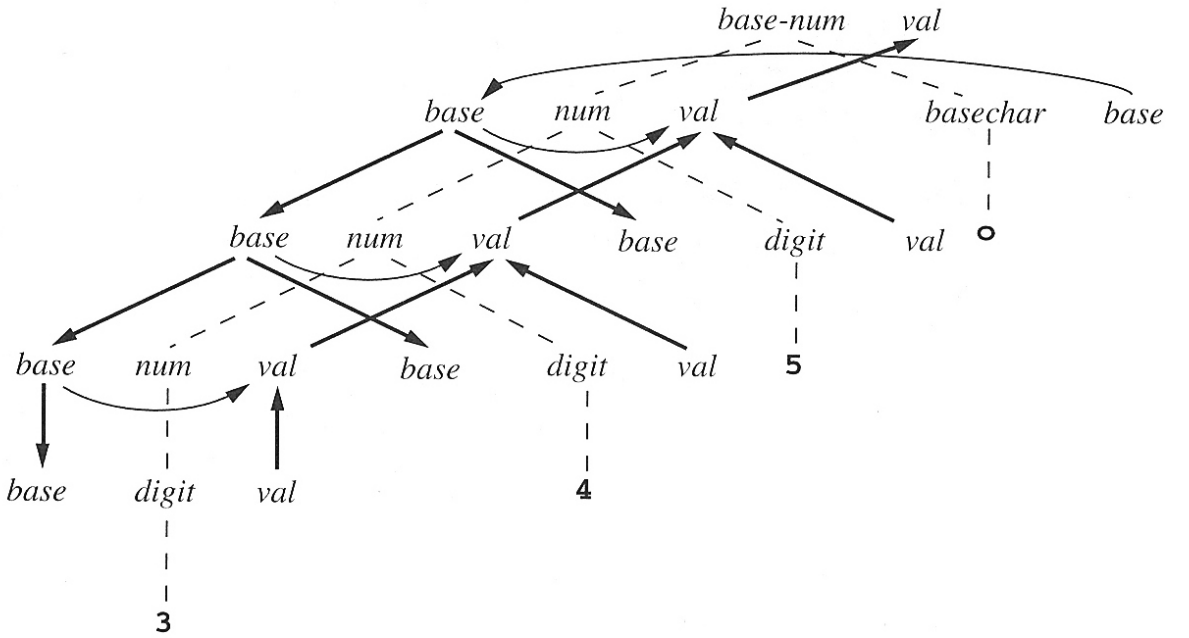
For 'basechar' er den 'base', ellers er den 'val'

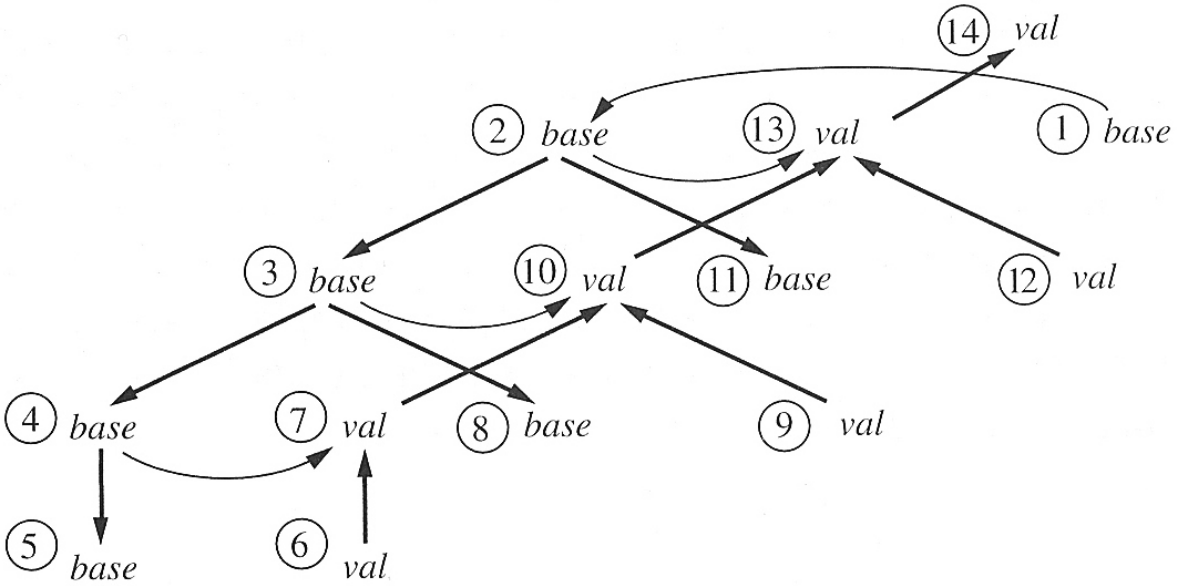
Mangler teknisk en parameter, men uten betydning

- Ny filosofi
  - Returverdier er syntetiserte attributter
  - Parametere er arvede attributter

```
function EvalWithBase ( T: treenode; base: integer ): integer;
var temp, temp2: integer;
begin
   case nodekind of T of
   based-num:
         temp := EvalWithBase ( right child of T );
         return EvalWithBase ( left child of T, temp );
   num:
         temp := EvalWithBase ( left child of T, base );
         if right child of T is not nil then
            temp2 := EvalWithBase ( right child of T, base );
            if temp ≠ error and temp2 ≠ error then
                  return base*temp + temp2
            else return error;
         else return temp;
   basechar:
         if child of T = 0 then return 8
         else return 10;
   digit:
         if base = 8 and child of T = 8 or 9 then return error
         else return numval ( child of T );
   end case;
end EvalWithBase;
```


based-num
(val = 229)
num
(val = 28 * 8 + 5 = 229)
(base = 8)
basechar
(base = 8
o
num
(val = 3 * 8 + 4 = 28)
(base = 8)
digit
(val = 5)
(base = 8)
5
num
(val = 3)
(base = 8)
digit
(val = 4)
(base = 8)
4
digit
(val = 3)
(base = 8)
3


base-num
val
base
num
val
basechar
base
base
num
val
base
digit
val
o
base
num
val
base
digit
val
5
base
digit
val
4
3


14
val
2
base
13
val
1
base
3
base
10
val
11
base
12
val
4
base
7
val
8
base
9
val
5
base
6
val

For beregning under parsering venstre -> høyre

An attribute grammar for attributes $a_1, \ldots, a_k$ is **L-attributed** if, for each inherited attribute a_j and each grammar rule

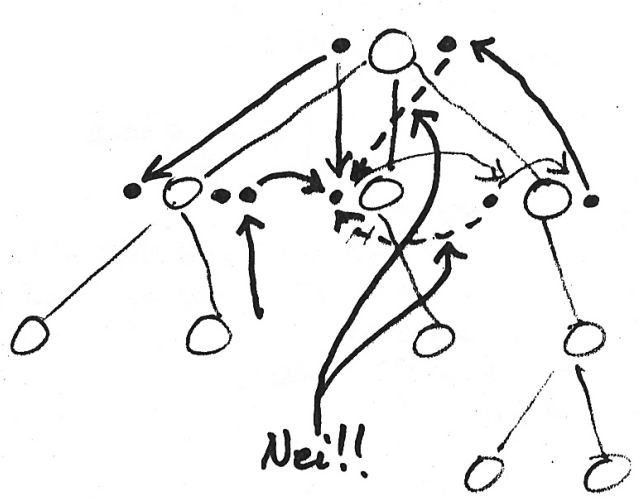
$$X_0 \rightarrow X_1 X_2 \ldots X_n$$

the associated equations for a_j are all of the form

$$X_i.a_j = f_{ij}(X_0.a_1, \ldots, X_0.a_k, X_1.a_1, \ldots, X_1.a_k, \ldots, X_{i-1}.a_1, \ldots, X_{i-1}.a_k)$$

That is, the value of a_j at X_i can only depend on attributes of the symbols $X_0, \ldots, X_{i-1}$ that occur to the left of X_i in the grammar rule.

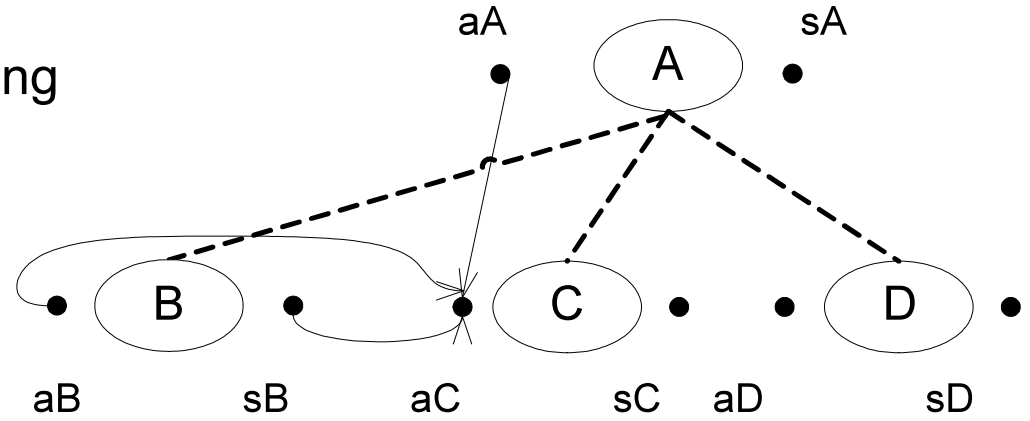
Trykkfeil: Krever også $X_0.a_1$, ... $X_o.a_k$ må alle være arvede attributter (ellers kan det oppstå løkker, og man kan få høyre->venstre avhengigheter



Beregning av L-attribut-gramm.

- under recursive descent parsering

A -> ... | B C D | ...




```
int A(aA){
  ...
  if <A-> BCD skal velges> then {
    int aB,sB,aC,sC,aD,sD
    aB = fB(aA);
    sB = B(aB);
    aC = fC(aA,aB,sB);
    sC = C(aC);
    aD = fD(aA,aB,sB,aC,sC);
    sD = D(aD);
    return gA(aA,aB,sB,aC,sC,aD,sD);
  } else
  ...
}
```

Beregning av sA

Generalisering av eksempel 6.16, men for L-attributt grammatikk