

Statisk semantisk analyse - Kap. 6

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

Generelt om semantisk analyse

- Formål: Sjekke alle krav i språkdefinisjonen som
 - kan sjekkes før utførelsen
 - ikke naturlig sjekkes under syntaktisk analyse
- Typisk
 - at bruk av navn er konsistent med deres deklarasjon
 - at typen av (sub)-uttrykk stemmer med operasjonene
- Merk
 - Ikke alltid klart hvor grensen mellom syntaktisk/semantisk sjekk går (eller 'bør gå')
 - if a then ... Må godkjennes syntaktisk
 - if a+b then ... ??
- Men
 - Ikke alt kan sjekkes før utførelsen
 - Stor forskjell på språk med og uten typer på variable og parametre

Run time type check

Dynamic languages

Does Java have runtime type check? Casting.

data er egentlig alltid typet

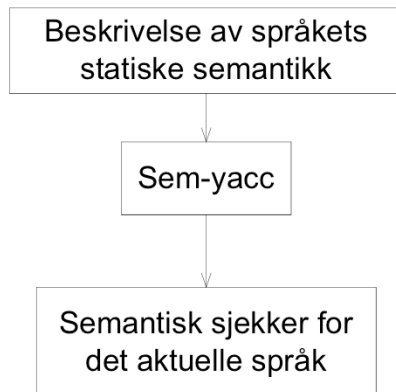
om variable ikke er typet: testing hele tiden på at data-operasjoner er OK

om variable er typede: Nesten all sjekk kan gjøres på kompileringstidspunktet, og det kan genereres bedre kode

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 alltid 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 alltid 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 beskrivesspråk
- Input til semantisk sjekker er rimelig komplisert
- Det er alltid 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

03/03/14

INF 5110 - 2014

5

In dynamic languages the type of a variable is not static,
some languages have untyped variables

Value of $3+4$, or $3+\text{const}$ is static

Object code may be produced during execution, or garbage collected.

Attributt-grammatikker

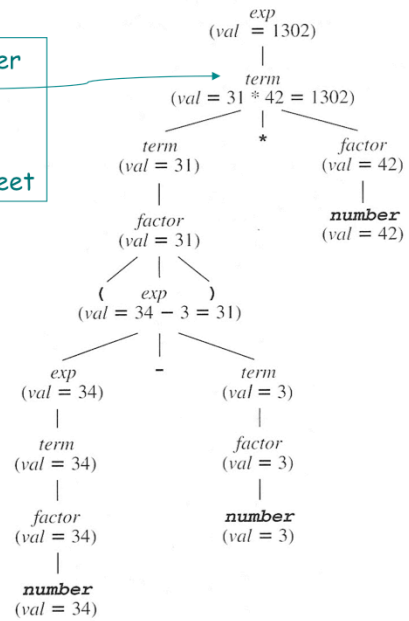
$exp \rightarrow exp + term \mid exp - term \mid term$
 $term \rightarrow term * factor \mid factor$
 $factor \rightarrow (exp) \mid \mathbf{number}$

Attributter er variable knyttet til nodene i parsringstreet

Hver semantisk regel er knyttet til en produksjon

Deres verdier definert ved semantiske regler

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



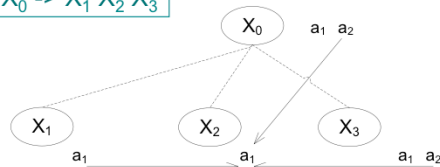
Bare ett attribut til hvert symbol, alle har samme navn, bare en regel pr produksjon

Attributt-grammatikker

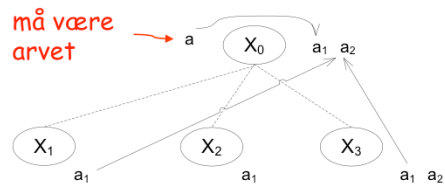
- Gitt en grammatikk på ren BNF-form
 - For hvert grammatikk-symbol X (terminal eller ikke-terminal) skal det være gitt en mengde (navnede) attributter
 - Attributt-mengdene for de forskjellige symboler kan generelt være helt forskjellige, men har ofte mye felles
 - Attributtene er ment å materialisere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

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

$$X_0 \rightarrow X_1 X_2 X_3$$



arvet attributt



syntetisert attributt

03/03/14

Syntetiserte attributter

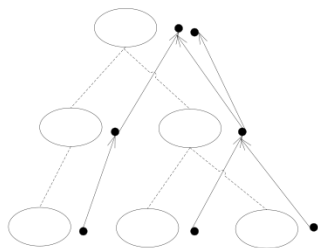
An attribute is **synthesized** if all its dependencies point from child to parent in the parse tree. Equivalently, an attribute a is synthesized if, given a grammar rule $A \rightarrow X_1 X_2 \dots X_n$, the only associated attribute equation with an a on the left-hand side is of the form

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

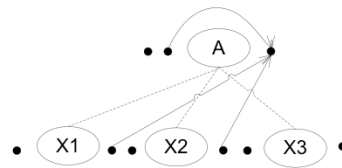
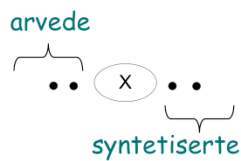
An attribute grammar in which all the attributes are synthesized is called an **S-attributed grammar**.

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

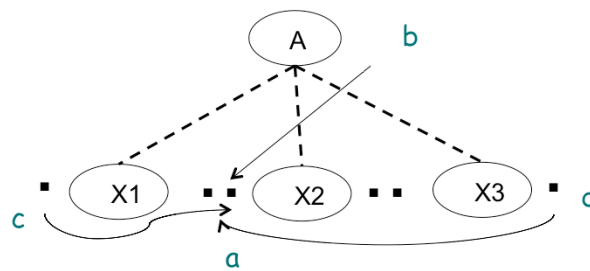
Arvede attributter

- "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

$A \rightarrow X_1 X_2 X_3$

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



Eksempel

$exp \rightarrow exp + term \mid exp - term \mid term$

$term \rightarrow term * factor \mid factor$

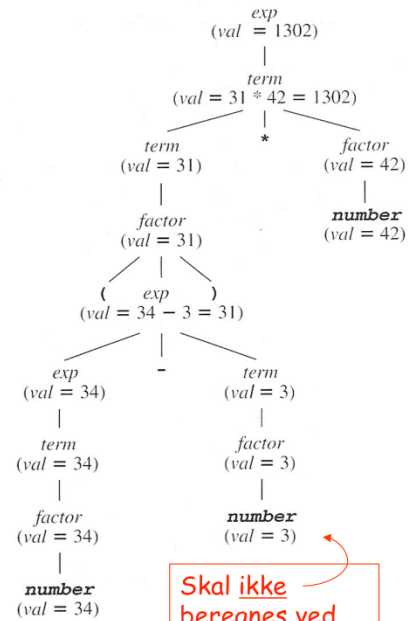
$factor \rightarrow (exp) \mid \mathbf{number}$

Alle symboler (bortsett fra '+', '-' og '*') har det ene attributtet 'val'.

exp_i brukes hvis samme symbol fler ganger



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



03/03/14

10

Eksempel

$number \rightarrow number\ digit \mid digit$
 $digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

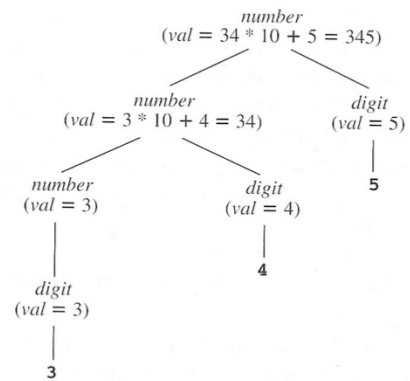
Attributter

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

Man kan gjerne tenke i

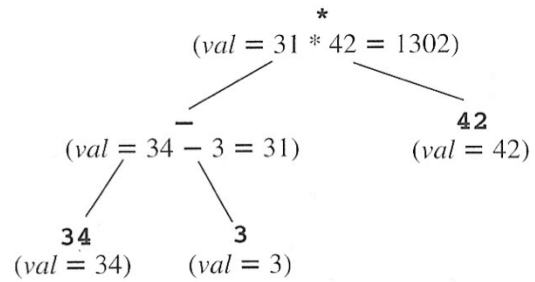
- flertydige grammatikker
- abstrakte syntakstrær

Attributter

exp: val

number: val (fra scanneren)

terminaler: -



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

Grammar Rule

Semantic Rules

$exp_1 \rightarrow exp_2 + exp_3$

$exp_1.val = exp_2.val + exp_3.val$

$exp_1 \rightarrow exp_2 - exp_3$

$exp_1.val = exp_2.val - exp_3.val$

$exp_1 \rightarrow exp_2 * exp_3$

$exp_1.val = exp_2.val * exp_3.val$

$exp_1 \rightarrow (exp_2)$

$exp_1.val = exp_2.val$

$exp \rightarrow \mathbf{number}$

$exp.val = \mathbf{number}.val$

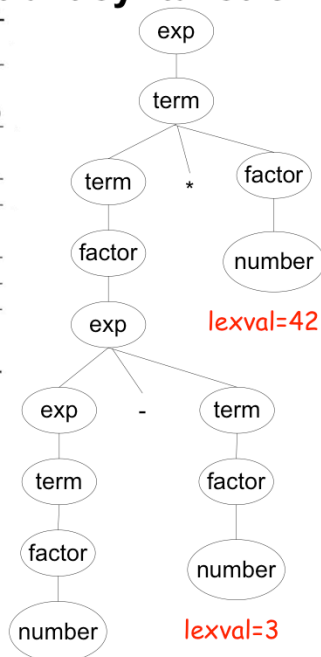
Attr.-grammatikk som 'bygger' abstrakt syntakstre

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

- **Attributter:**
 - tree, for
 - exp, term, factor
 - lexval (for number)

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

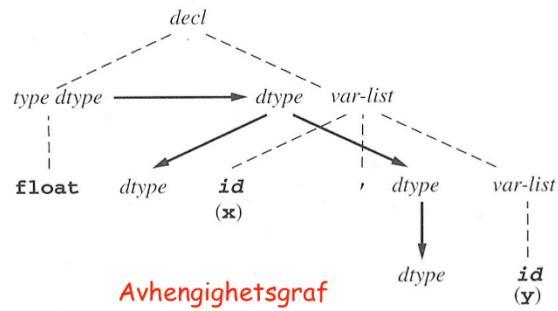
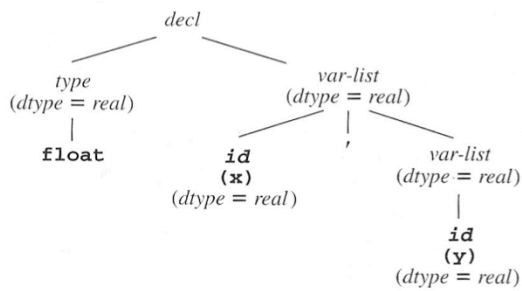
lexval=34



Grammatikk med arvede attributter

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

Attributter
 type, varlist, id: dtype



Eksempel på arvede og syntetiserte attributter

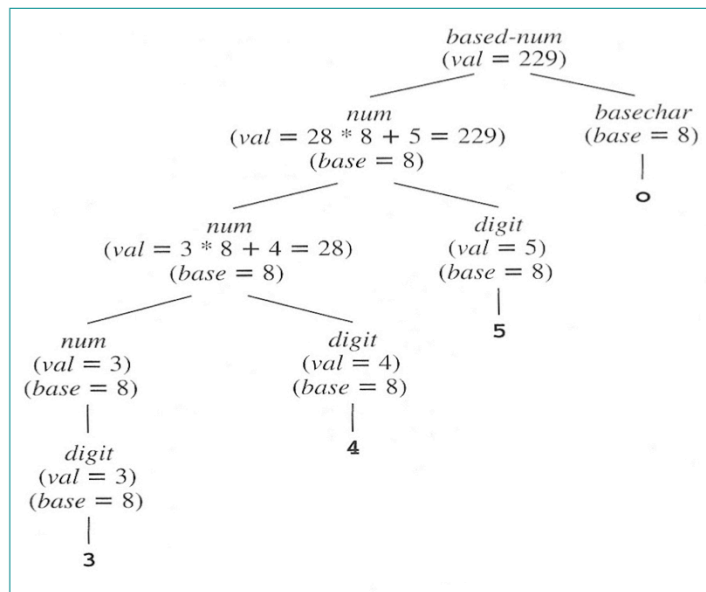
Grammar Rule	Semantic Rules
$based_num \rightarrow num\ basechar$	$based_num.val = num.val$ $num.base = basechar.base$
$basechar \rightarrow 0$	$basechar.base = 8$
$basechar \rightarrow \bar{a}$	$basechar.base = 10$
$num_1 \rightarrow num_2\ digit$	$num_1.val =$ if $digit.val = error$ or $num_2.val = error$ then $error$ else $num_2.val * num_1.base + digit.val$ $num_2.base = num_1.base$ $digit.base = num_1.base$
$num \rightarrow digit$	$num.val = digit.val$ $digit.base = num.base$
$digit \rightarrow 0$	$digit.val = 0$
$digit \rightarrow 1$	$digit.val = 1$
...	...
$digit \rightarrow 7$	$digit.val = 7$
$digit \rightarrow 8$	$digit.val =$ if $digit.base = 8$ then $error$ else 8
$digit \rightarrow 9$	$digit.val =$ 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)

Eksempel based-num forts.



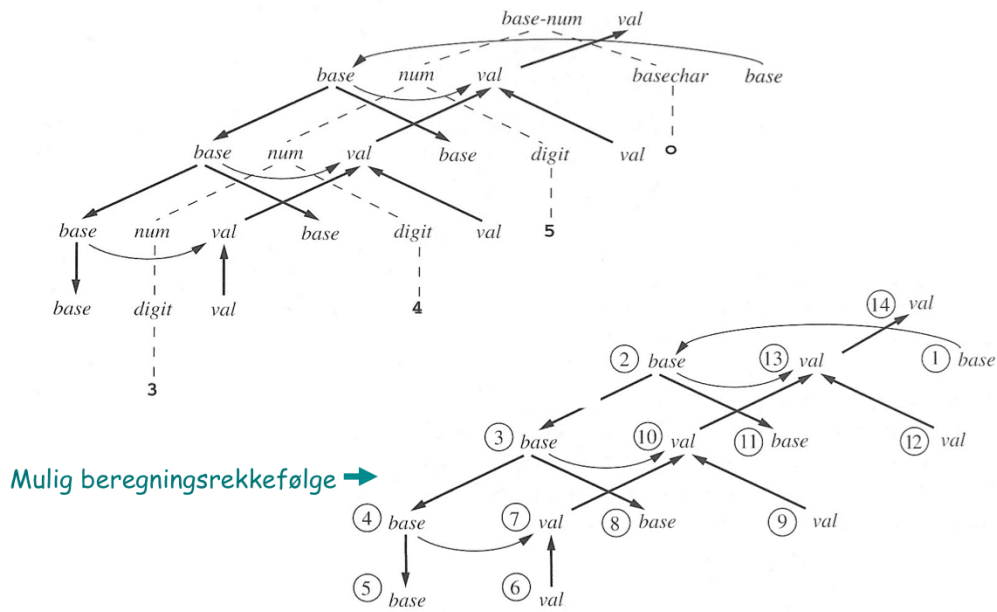
03/03/14

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 parsingstre definert een og bare een gang

Avhengighetsgraf for base-num - eksempelet



03/03/14

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

Omskriving fra arvede til syntetiserte attributter

- Denne ga arvede attributter:

$decl \rightarrow type\ var\text{-}list$
 $type \rightarrow \mathbf{int} \mid \mathbf{float}$
 $var\text{-}list \rightarrow \mathbf{id}, var\text{-}list \mid \mathbf{id}$

- Nytt forslag:

$decl \rightarrow var\text{-}list\ \mathbf{id}$
 $var\text{-}list \rightarrow var\text{-}list\ \mathbf{id}, \mid type$
 $type \rightarrow \mathbf{int} \mid \mathbf{float}$

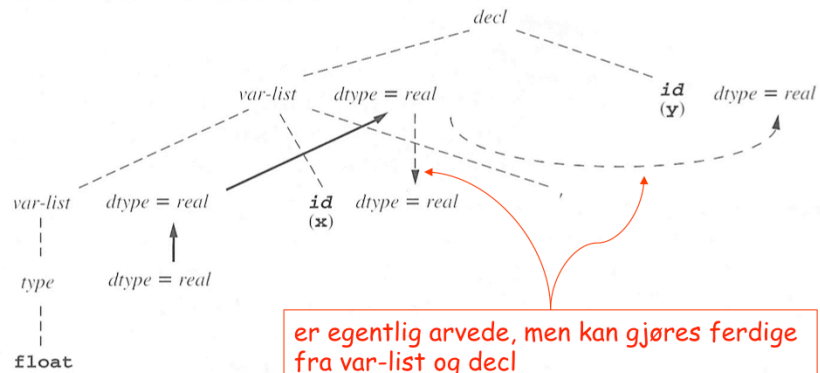
Med ny grammatikk

Grammar Rule

$decl \rightarrow var\text{-}list\ id$
 $var\text{-}list_1 \rightarrow var\text{-}list_2\ id\ ,$
 $var\text{-}list \rightarrow type$
 $type \rightarrow \mathbf{int}$
 $type \rightarrow \mathbf{float}$

Semantic Rules

$id.dtype = var\text{-}list.dtype$
 $var\text{-}list_1.dtype = var\text{-}list_2.dtype$
 $id.dtype = var\text{-}list_1.dtype$
 $var\text{-}list.dtype = type.dtype$
 $type.dtype = integer$
 $type.dtype = real$



03/03/14

Beregningsalgoritme for type-grammatikk

```
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;
```

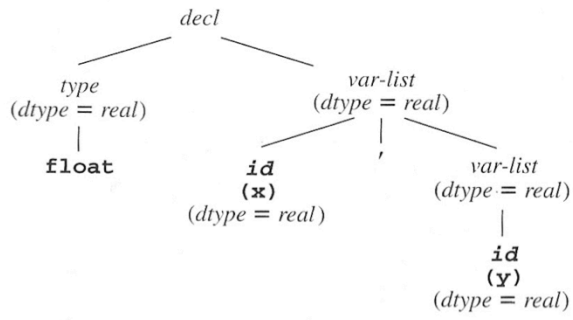
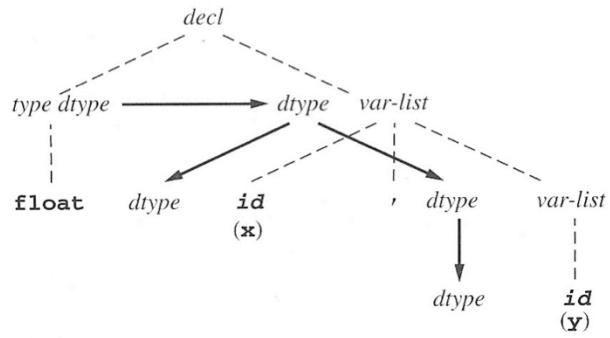
```
end EvalType;
```

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

- 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

INF 5110 - 2014

22



03/03/14

Beregning for 'base'- grammatikken

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

```
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 = 0 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;
```

03/03/14

INF 5110 - 2014

24

Alternativ til EvalWithBase

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

Mangler teknisk en parameter, men uten betydning

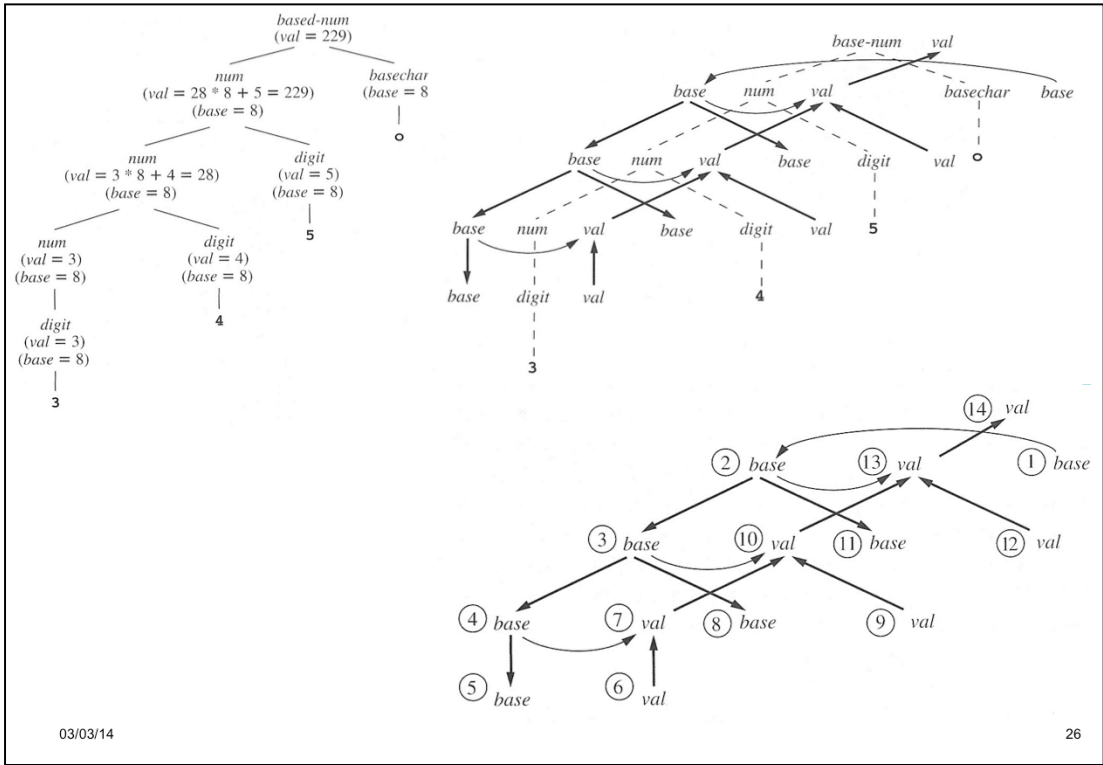
- Ny filosofi
 - Returverdier er syntetiserte attributter
 - Parametere er arvede attributter

```
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;
```

03/03/14

INF 5110 - 2014

25



03/03/14

26

For beregning under parsing venstre → høyre

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

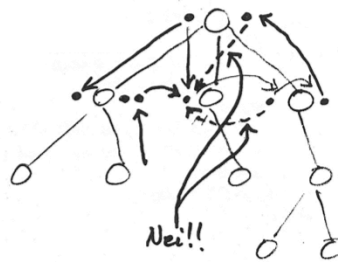
$$X_0 \rightarrow X_1 X_2 \dots X_n$$

the associated equations for a_j are all of the form

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

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

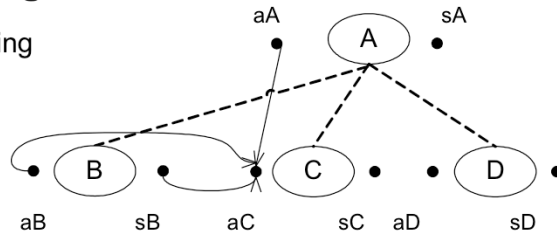
Trykkfeil: Krever også $X_0.a_1, \dots, X_0.a_k$ må alle være arvede attributter (ellers kan det oppstå løkker, og man kan få høyre-→venstre avhengigheter)



Beregning av L-attribut-gramm.

- under recursive descent parsing

$A \rightarrow \dots | B C D | \dots$



```

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

03/03/14