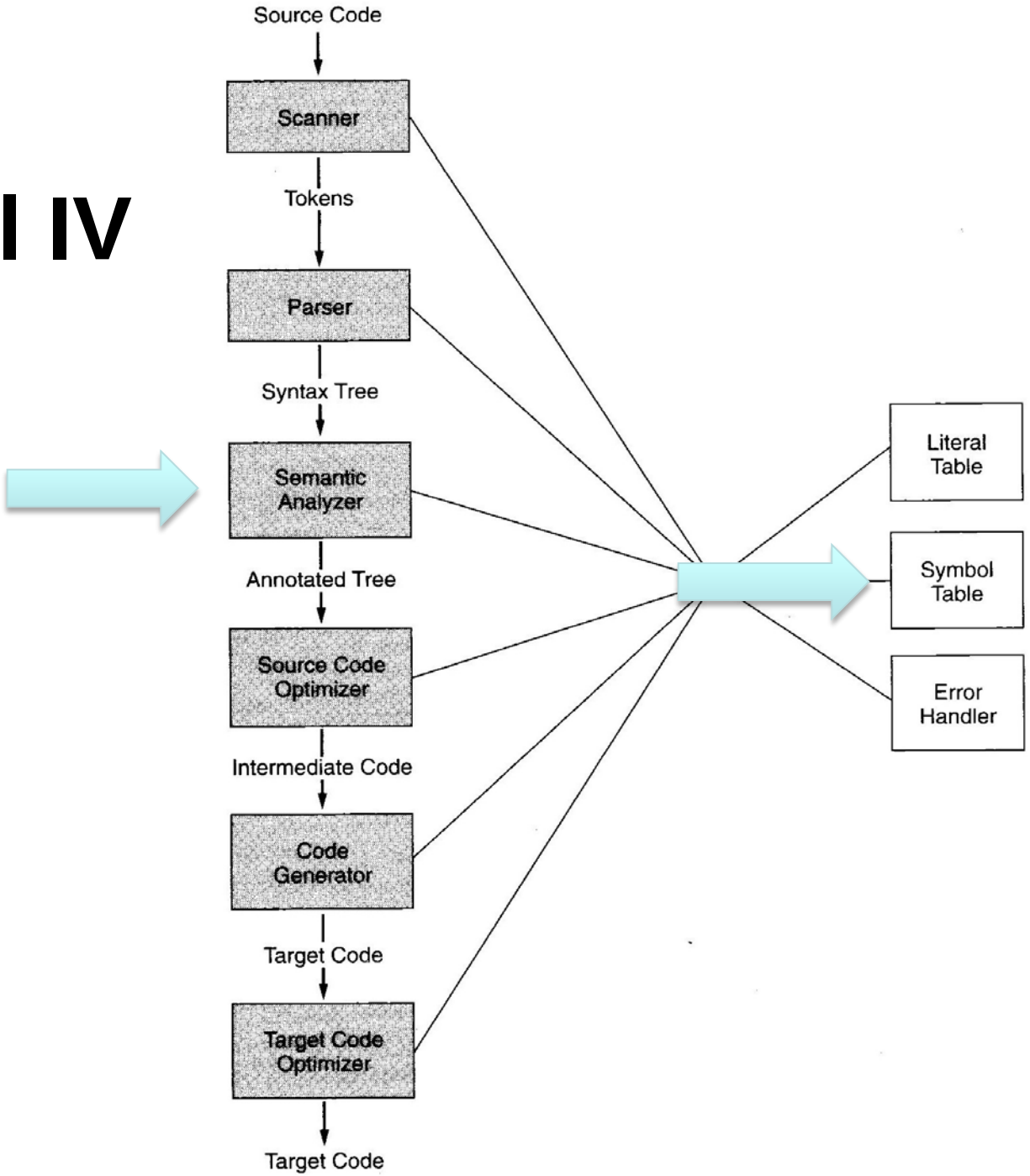
Semantisk Analyse del IV

Typesjekking

Kapittel 6.4 forts



INF 5110 - 2013

TYPE-LIKHET

Når er to typer like?

```
var a,b:
       record
         int i;
         double d;
       end
```

```
var c: record
         int i;
         double d;
       end
```

```
typedef idRecord:record
                   int i;
                   double d;
                 end
```

```
var d: idRecord;
var e: idRecord;
```

```
a:= c;        a:= b;
a:= d;        d:= e;
```

Strukturlikhet?

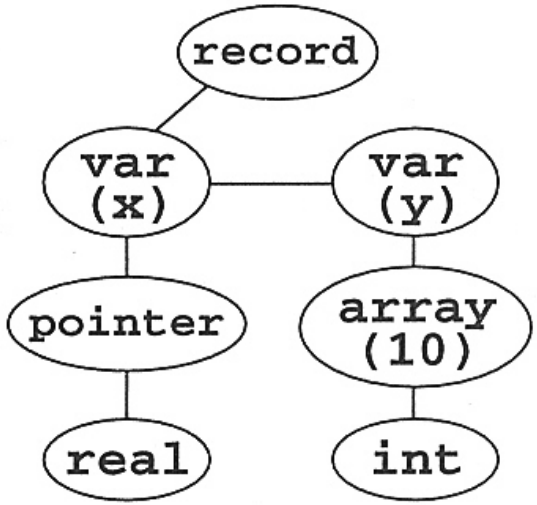
Navnelikhet?

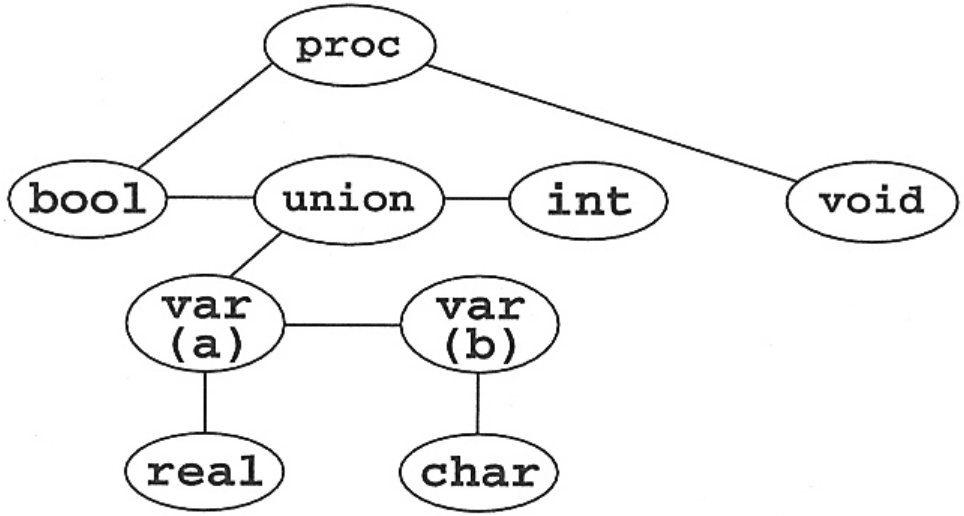
Type-uttrykk representert som (abstrakt) syntakstre

Uten typedef:
strukturlikhet

```
proc(bool,union a:real; b:char end,int):void
```

```
record
   x: pointer to real;
   y: array [10] of int
end
```





INF 5110 - 2013

Oppgave 21a

Gitt følgende (del-)grammatikk:

fun-decls → **fun id (** *var-decls* **) :** *type-exp* **;** *body*

var-decls → *var-decls; var-decl* | *var-decl*

var-decl → **id :** *type-exp*

Innholdet i type-exp og body er ikke relevant i denne sammenhengen.

Bestem en passende tre-struktur for slike funksjonstyper, og skriv pseudokode for en metode *typeEqual* som sjekker om to funksjonstyper er like.

Navne-likhet

Alle typer må gis navn, og type-likhet krever samme navn.

```
record
   x: pointer to real;
   y: array [10] of int
end
```

```
t1 = pointer to real;
t2 = array [10] of int;
t3 = record
        x: t1;
        y: t2
end
```

```
function typeEqual ( t1, t2 : TypeExp ) : Boolean;
var temp : Boolean ;
    p1, p2 : TypeExp ;
begin
   if t1 and t2 are of simple type then
      return t1 = t2
   else if t1 and t2 are type names then
      return t1 = t2
   else return false ;
end;
```

Bare navnelikhet (eller samme basaltype) gir likhet

Type-deklarasjon ekvivalens

```
t2 = t1;
```

```
t1 = int;
t2 = int
```

```
t1 = array [10] of int;
t2 = array [10] of int;
t3 = t1;
```

TYPE-SJEKKING

Eksempel-grammatikk

program → *var-decls* **;** *stmts*
var-decls → *var-decls* **;** *var-decl* | *var-decl*
var-decl → **id** **:** *type-exp*
type-exp → **int** | **bool** | **array** **[num]** **of** *type-exp*
stmts → *stmts* **;** *stmt* | *stmt*
stmt → **if** *exp* **then** *stmt* | **id** **:=** *exp*

exp → *exp* **+** *exp* | *exp* **or** *exp* | *exp* **[** *exp* **]**

Sjekking av type

Grammar Rule	Semantic Rules
var-decl → **id** : *type-exp*	*insert*(**id** .*name*, *type-exp.type*)
type-exp → **int**	*type-exp.type* := *integer*
type-exp → **bool**	*type-exp.type* := *boolean*
$type\text{-}exp_1$ → **array** **[num]** **of** $type\text{-}exp_2$	$type\text{-}exp_1$.*type* := *makeTypeNode*(*array*, **num** .*size*, $type\text{-}exp_2$.*type*)
stmt → **if** *exp* **then** *stmt*	**if not** *typeEqual*(*exp.type*, *boolean*) **then** *type-error*(*stmt*)
stmt → **id** := *exp*	**if not** *typeEqual*(*lookup*(**id** .*name*), *exp.type*) **then** *type-error*(*stmt*)
exp_1 → exp_2 + exp_3	**if not** (*typeEqual*(exp_2 .*type*, *integer*) **and** *typeEqual*(exp_3 .*type*, *integer*)) **then** *type-error*(exp_1) ; exp_1 .*type* := *integer*
exp_1 → exp_2 **or** exp_3	**if not** (*typeEqual*(exp_2 .*type*, *boolean*) **and** *typeEqual*(exp_3 .*type*, *boolean*)) **then** *type-error*(exp_1) ; exp_1 .*type* := *boolean*
exp_1 → exp_2 **[** exp_3 **]**	**if** *isArrayType*(exp_2 .*type*) **and** *typeEqual*(exp_3 .*type*, *integer*) **then** exp_1 .*type* := exp_2 .*type.child1* **else** *type-error*(exp_1)
exp → **num**	*exp.type* := *integer*
exp → **true**	*exp.type* := *boolean*
exp → **false**	*exp.type* := *boolean*
exp → **id**	*exp.type* := *lookup*(**id** .*name*)

Oppgave 21b

Gitt følgende (del-)grammatikk:

fun-decls → **fun id (** *var-decls* **) :** *type-exp* **;** *body*

var-decls → *var-decls; var-decl* | *var-decl*

var-decl → **id :** *type-exp*

exp → **id (** *exps* **)**

exps → *exps* **,** *exp* | *exp*

Skriv semantiske regler for å type-sjekke funksjons-deklarasjoner og funksjons-kall. Det er ikke nødvendig å lage semantiske regler for type-exp, body og andre alternativer for exp.

Oppgave 20

Se på følgende (tvetydige) grammatikk:

exp → *exp* **+** *exp* | *exp* **-** *exp* | *exp* ***** *exp* | *exp* **/** *exp* | **(** *exp* **)** | **num | num.num**

num er av typen integer, **num.num** av typen flyttall.

Regel: Hvis to del-uttrykk har forskjellig type, skal integer-uttrykket konverteres til flyttall og operasjonen for flyttall brukes.

Skriv en attributt-grammatikk som gjør om uttrykket slik at denne konverteringen gjøres eksplisitt ved hjelp av funksjonen **FLOAT**. Divisjonsoperatoren / skal være **div** dersom begge operandene er integer.

Det er ikke nødvendig å lage regler for – og * .