Formal Methods
Seminar 3

TypeChecker Implementation

- Please implement in Ocaml a typechecker for CoreJava language
- The type checking rules of the CoreJava type system are described in the next slides

Your typechecker takes as input a CoreJava program (that you have to manually represent as an AST), does pattern matching over the AST root and calls recursivelly itself for the AST children. For expressions, it returns the computed type of that expression or throws specific exceptions for the encountered errors. For classes/methods definitions it returns true/false (or throws specific exceptions for the encountered errors).

CoreJava Type System

CoreJava Type System

- In the following we present the type checking rules for CoreJava.
- The presentation is not so formal as in the literature
- The judgements have the following form

conditions to be met		**(IF conds to be met)**
---------------------------	**<==>**	**THEN**
context |- type rule		**for the given context**
		The type rule is true

CoreJava Type System

- It consists of the following judgements for:
  - A well-typed program
  - A well-typed class declaration
  - Well-typed field declarations
  - A well-typed method declaration
  - A well-typed expression
  - Subtyping

- Please start the implementation with the followings:
  - Subtyping relation
  - Auxilliary functions
  - Simple Expression Type checking

Subtyping Judgement

- In order to denote that a type t1 is a subtype of type t2 we used the following notation t1 <: t2
- The rules of the subtyping relation are enumerated in the following
- If none of the following rule is applicable that means that t1 is not subtype of t2

Subtyping Judgement

```
                    (inheritance rule)

    Class cn1 extends cn2 {...} is a declared class in P
---------------------------------------------------------
                 P |-  cn1 <: cn2



   (reflexivity)                    (transitivity)

                                P |-  t1<:t2   and   P |- t2<:t3

   -------------------          ------------------------------------

     P |- t<:t                          P |- t1 <:t3
```

Subtyping Judgement

$$\frac{\text{Cn is a declared class in P}}{\text{P |- bot <: cn}}$$

$$\frac{\text{cn is a declared class in P}}{\text{P |- cn <: Object}}$$

- **Note that the above 5 rules directly imply the followings:**
  - **int <:int ,**
  - **float<:float ,**
  - **void <:void**
  - **bool <: bool**

Subtyping Judgement Implementation

Subtyping judgement can be implemented as a function

Subtype (prog: progr) (t1:typ) (t2:typ) = ...

- It returns a bool

- it has a different body according to the patterns of t1 and t2

fieldlist

```
------------------------------------------------
            fieldlist(P,Object) = []


     class cn1 extends cn2 {t1 f1;...;tn fn # ....}
     ----------------------------------------------------------
     fieldlist(P,cn1)= fieldlist(P,cn2) ++ [(f1,t1);...(fn,tn))]
```

- It computes all fields of a class

Fieldlist Implementation

it can be implemented as a function

Fieldlist (prog: progr) (classname: string) =

It returns a list of pairs field name and type: (string*typ) list

Well-typed expressions

for the following rules you have to implement a function:

TypeCheckExp (prog: progr) (environment: (string*typ) list) (expCrt: exp)=

-It returns typ or throws exception when a condition does not hold

- its body depends on the pattern of the current expression expCrt

Well-typed expressions

```
----------------------------------------
                 P,TE   |- null:bot

-------------------------------------
          P,TE |- kint: int

----------------------------------------
          P,TE   |- kfloat:float

-------------------------------------
          P,TE |- (): void

----------------------------------------
          P,TE   |- false:bool

-------------------------------------
          P,TE |- true: bool
```

Well typed expressions

```
          TE={...(v:t)...}
----------------------------------
          P,TE |-  v : t
```

- The type of the variable v is the declared type of the variable v
- The declared type of a variable is stored in the type environment

Well typed expressions

```
            P,TE |- v: cn  and
      (cn is a declared class in P) and
      ( (f,t) is defined in fieldlist(P,cn))
------------------------------------------
            P,TE |-  v.f : t
```

- First we get the type of v, that type must be a class
- Second we get the type of the field f

Well typed expressions

```
          P,TE |- v: t1 and
         P,TE |- e : t2 and
          P |- t2 <: t1
---------------------------------
          P,TE |-  v=e : void
```

- The type t2 of the expression e must be a subtype of the variable v type t1

Well typed expressions

P,TE |- v.f :t1

P,TE |- e : t2 and

P |- t2 <: t1

P,TE |- v.f=e : void

Well typed expressions

```
        P, {v:t} +TE  |- e : t1
------------------------------------------------
           P,TE |-  {(t v)  e} : t1


        P, TE  |- e : t1
------------------------------------------------
           P,TE |-  {e} : t1
```

Well typed expressions

```
          P,TE |- e1 :t1    and
         P,TE |-  e2 : t2
------------------------------------
            P,TE |-  e1;e2 : t2
```

Well typed expressions

P,TE |- v :tv   and P |- tv<:bool and

P,TE |- {e1} : t1 and P,TE |-{ e2}:t2 and

Find t such that

P |- t1 <: t   and P |- t2 <: t and

(t is the least maximum type of t1 and t2)

--

P,TE |- if v then {e1} else {e2} : t

Well typed expressions

```
        (opint is either + or – or * or /)
       P,TE |- e1 :t1    and P |- t1<:int and
        P,TE |-  e2 : t2  and P |- t2<:int
--------------------------------------------------------
              P,TE |-  e1 opint e2 : int
```

Well typed expressions

```
        (opbool is either && or ||) and
   P,TE |- e1 :t1     and     P |- t1<:bool and
   P,TE |-  e2 : t2    and     P |- t2<:bool
------------------------------------------------------------
          P,TE |-  e1 opbool e2 : bool



    P,TE |- e : t       and      P |- t<:bool
------------------------------------------------------------
          P,TE |- !e : bool
```

Well typed expressions

```
(opcmp is either < or <= or == or != or > or >=) and
            P,TE |- e1 :t1    and P,TE |-  e2 : t2  and
                        t1<:t2  and t2<:t1 and
            ( t1 is not a declared class in P)  and
           (t2 is not a declared class in P)
    ------------------------------------------
                P,TE |-  e1 opcmp e2 : bool
```

Well typed expressions

```
        (cn is a declared class in P) and
         P,TE |- v :t    and
         ( P |- cn <: t    or    P |- t<: cn)
------------------------------------------------------
              P,TE |-  (cn) v : cn
```

Well typed expressions

```
      (cn is a declared class in P) and
       P,TE |- v :t  and (t <: cn or cn<:t)
------------------------------------------------------
            P,TE |-  v instanceof  cn : bool
```

Well typed expressions

```
        (cn is a declared class in P) and
      [(f1,t1),...,(fn,tn)]=fieldlist(P,cn) and
  P,TE |- v1 :t1' and ... and P,TE |- vn:tn' and
   P |- t1'<:t1 and ...  and P |- tn'<:tn
--------------------------------------------
          P,TE |-  new cn(v1,...,vn) : cn
```

Well typed expressions

```
P,TE |- v:t     and  P |- t <: bool and
             P,TE |- e : te
-----------------------------------------------------------
        P,TE |- while v {e} : void
```