Formal Methods
Seminar 4

TypeChecker Implementation

Please implement the following remaining rules of Core-Java type checker:

- Type checking rule for method invocation
- Type checking rule for method declaration
- Type checking rule for class declaration
- Type checking rule for the entire program
- Some auxilliary functions

Type rule for method invocation

```
P,TE |- v0:t0 and (t0 is a declared class in P) and
( P |-( tr mn(t1 v1,..., tn vn){e}) is a declared method in t0)   and
P,TE |- v1' :t1' and ... and P,TE |- vn':tn' and
P |- t1'<:t1 and ...  and P |- tn'<:tn
------------------------------------------------------
P,TE |-  v0.mn(v1',...,vn') : tr
```

Well-typed method declaration

```
P, {v1:t1,..., vn:tn}+TE |- e : t   and P |- t <: tr
-------------------------------------------------------------------
                         P,TE   |-mth-  tr mn(t1 v1, ..., tn vn) {e}
```

- A method is well typed if:
  - The method body is well typed
  - TE denotes the type environment
  - {v1:t1,..., vn:tn}+TE denotes the extension of a type environment TE with new mappings {v1:t1,...,vn:tn} corresponding to the formal parameters of the method
  - The judgement P,TE |- e:t says that the type of the expression e is t with respect to the program P and type environment TE
  - The type of the method body must be a subtype of the declared return type of the method

Well-typed class declaration

```
    ClsD= class cn extends cn' {fldD1...fldDn # mthD1...mthDn} and
    For each method declaration mthDi we have:
                 P, {this:cn} |-mth- mthDi
-------------------------------------------------------------
                     P   |-def- clsD
```

- A class is well typed if:
  - Each method from the class is well typed
  - {this:cn} denotes the initial type environment
  - A type environment is a dictionary containing mappings from the variable name to the type associated to that variable
  - Type environment is working as a stack where we continously push new mappings

Well-typed program

```
   |- WellFoundedClasses(P)  and P=clsD1;...;clsDn and
   For each class declaration clsDi we have:
        |- methsOnce(clsDi) and |-fieldsOnce(clsDi)  and
        P |- inheritanceOK(clsDi) and P |-def-   clsDi
----------------------------------------------------------------
                   |- P
```

- A program is well-typed if:
  - WellFoundedClasses: no duplicate definitions of the clases, no cycle in the class hierarchy and last class contains the main method
  - MethsOnce: no methods duplication in a class
  - FieldsOnce: no field duplication in a class
  - InheritanceOk: method overriding is sound
  - Each class is well typed

Auxilliary rules

```
   clsD = class cn extends cn' {...# mthD1...mthDn}
    For each i and j, 0<=i<=n and 0<=j<=n and i!=j
              name(mthDi) != name(mthDj)
-----------------------------------------------------------
              |- methsOnce(clsD)
```

- No method overloading/duplication in a class definition

Auxilliary rules

```
clsD = class cn extends cn' {fldD1...fldDn # ...}
For each i and j, 0<=i<=n and 0<=j<=n and i!=j
name(fldDi) != name(fldDj)
----------------------------------------------------------
|- fieldsOnce(clsD)
```

- No field duplication in a class definition

Auxilliary rules

```
  P=clsD1;...;clsDn  and clsDi= cni extends cni' {...} and
   IR={(cni,cni')| 1<=i<=n} and ID={(cni,cni)|1<=i<=n} and
TransitiveClosure(IR) intersect ID = {} and
For all i,j    cni != cnj and
ClsDn = class Main extends cn' { # void main() { e}}
      -----------------------------------------------------------------
                        |- WellFoundedClasses(P)
```

- no duplicate definitions of the clases, no cycle in the class hierarchy and last class contains the main method

Transitive Closure

IR={(cni,cni')| 1<=i<=n}

TransitiveClosure(IR) is computed as follows:

1. TransitiveClosure(IR)=IR

2. if (cn1,cn2) is in TransitiveClosure(IR) and (cn2,cn3) is in TransitiveClosure(IR) then the pair(cn1,cn3) is added to TransitiveClosure(IR)

3. Step 2 is performed until no modification can be done to TransitiveClosure(IR)

Auxilliary rules

```
  clsD= class cn extends cn' {...# meth1...methn} and
For all 1<=i<=n if exists a method meth' such that
    (meth' is a declared method in cn') and
    name(methi) == name(meth') then
             overridesOk(methi,meth')
    ----------------------------------------------------------------
                         P |- inheritanceOK(clsD)


meth1 = tr1 mn(t1 v1,...tn vn) {e1} and
Meth2 = tr2 mn(t1 v1,...tn vn) {e1} and tr1<:tr2
-----------------------------------------------------
            overridesOK(meth1,meth2)
```

Auxilliary rules

P= clsD1 ...clsDi...clsDn and

ClsDi = class cn extends cn' {...}

----

cn is a declared class in P

P |-(tr mn(t1 v1,..., tn vn){e}) is a directly declared method in cn

----

P |-(tr mn(t1 v1,..., tn vn){e}) is a declared method in cn

Auxilliary rules

class cn extends cn' {...} and

(P |-(tr mn(t1 v1,..., tn vn){e}) is a declared method in cn') and

NOT (P |-(tr mn(t1 v1,..., tn vn){e}) is a directly declared method in cn)

P |-(tr mn(t1 v1,..., tn vn){e}) is a declared method in cn

Class cn extends cn' { ...#meth1...methi...methn}

Methi = tr mn(t1 v1,..., tn vn){e}

P |-(tr mn(t1 v1,..., tn vn){e}) is a directly declared method in cn

Type Checking Example

Example

In the following we discuss the type checking for a simple program P written in CoreJava:

```
class A extends Object{
int f1;
#
int m1(int a, int b) { (int c)
                        c=a+b;this.f1=this.f1+c;c};
}
```

Example

```
class B extends A{
A f2;
#
A m2(A x, A y) {(A z) { (int n)
                          n=x.m1(1,2)-y.m1(2,1);
                          {(bool m)  m=(x.f1-y.f1)>n;
                             if m then {z=new A(m)} else {z=new A(n)}
                          }
                       };this.f2=z;z
}
```

Example

```
Class Main extends Object{  #
 Void main(){ (B o1) o1=new B(0,null);
                    { (A o2)  o2=new A(2);
                        { (A o3) o3=new A(3);
                             o2 =o1.m2(o2,o3)
                        }
                    }
              }
}
```

TypeChecking Example

```
|- WellFoundedClasses(P)  and P=clsA;clsB;clsMain and
   For each class declaration we have:
           |- methsOnce(clsDi) and |-fieldsOnce(clsDi)  and
           P |- inheritanceOK(clsDi) and P |-def-   clsDi
--------------------------------------------------------------------
                      |- P
```

TypeChecking Example

```
ClsA= class A extends Object {fldF1 # mthM1} and
              P, {this:A} |-mth- mthM1
-------------------------------------------------------------
                 P   |-def- clsA


ClsB= class B extends A {fldF2 # mthM2} and
              P, {this:B} |-mth- mthM2
-------------------------------------------------------------
                 P   |-def- clsB


ClsMain= class Main extends Object { # mthMain} and
              P, {this:Main} |-mth- mthMain
-------------------------------------------------------------
                 P   |-def- clsMain
```

TypeChecking Example

```
P, {c:int;a:int;b:int;this:A} |- c=a+b:?t1  and
 P, {c:int;a:int;b:int;this:A} |- this.f1=this.f1+c;c : ?t
------------------------------------ -----------------------------------------
P, {c:int;a:int;b:int;this:A} |- c=a+b;this.f1=this.f1+c;c :? t
--------------------------------------------------------
P, {a:int;b:int;this:A} |- { (int c) c=a+b;this.f1=this.f1+c;c} :? t
        and P |- ?t <: int
---------------------------------------------------------------------
                        P,{this:A}   |-mth-   int m1(int a, int a) {...}
```

TypeChecking Example

```
                                                  ?t11"=int    ?t12"=int

                                            P, {c:int;a:int;b:int;this:A} |- a:?t11":int

          ?t1'=int                          P, {c:int;a:int;b:int;this:A} |-b:?t12":int

 {c:int} in {c:int;a:int;b:int;this:A}          P |- ?t11<:int  and P|-?t12:int  TRUE

--------------------------------------   and --------------------------------------

P, {c:int;a:int;b:int;this:A} |- c:?t1'          P, {c:int;a:int;b:int;this:A} |- a+b:int

                                                        ?t1"=int

             P |- ?t1" <: ?t1'  TRUE

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

                 P, {c:int;a:int;b:int;this:A} |- c=a+b:void ?t1=void
```

TypeChecking Example

P, {c:int;a:int;b:int;this:A} |-this.f1=this.f1+c: ?t2

And

P, {c:int;a:int;b:int;this:A} |- c:?t   ?t=int

--

P, {c:int;a:int;b:int;this:A} |- this.f1=this.f1+c;c : ?t   ?t=int

TypeChecking Example

```
P, {c:int;a:int;b:int;this:A} |- c=a+b:?t1  and
 P, {c:int;a:int;b:int;this:A} |- this.f1=this.f1+c;c : int
----------------------------------- ----------------------------------------
P, {c:int;a:int;b:int;this:A} |- c=a+b;this.f1=this.f1+c;c :int
------------------------------------------------------
P, {a:int;b:int;this:A} |- { (int c) c=a+b;this.f1=this.f1+c;c} :int
        and P |- int <: int
-------------------------------------------------------------------
                       P,{this:A}   |-mth-   int m1(int a, int b) {...}
```

TypeChecking Example

P, {c:int;a:int;b:int;this:A} |-this.f1+c:?t22 and

P, {c:int;a:int;b:int;this:A} |-this.f1:?t21 and

P |- ?t22 <: ?t21

P, {c:int;a:int;b:int;this:A} |-this.f1=this.f1+c: void ?t2=void

TypeChecking Example

P, {c:int;a:int;b:int;this:A} |-this.f1:?t221   ?t221=int

and

P, {c:int;a:int;b:int;this:A} |-c:?t222   ?t222=int

and

P|- ?t221 <:int   and   P|- ?t222 <:int   BOTH TRUE

P, {c:int;a:int;b:int;this:A} |-this.f1+c:?t22   ?t22=int

TypeChecking Example

P, {c:int;a:int;b:int;this:A}|- this: ?t21' and ?t21'=A

(?t21' is a declared class in P) and TRUE

((f1,?t21) is defined in fieldlist(P,?t21')) fieldlist(P,A)={(f1,int)}

P, {c:int;a:int;b:int;this:A} |-this.f1:?t21 ?t21=int