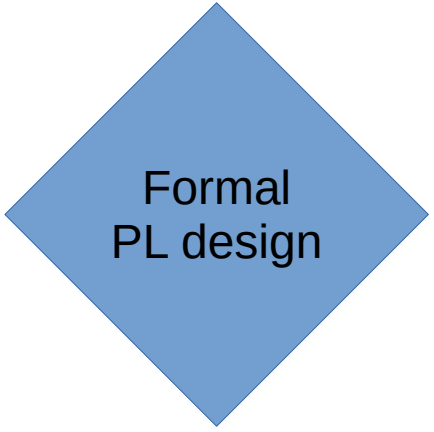
SWEN423 - Featherweight Java



Lect6: State / Field Update

Adding Field update to FJ

- What about field update?

$$e_0.f = e_1$$

Note: this is not local variable update.

Remember, in java “this.” can be omitted.
In Java foo=bar; is a field update only if it is a “shortcut” for this.foo=bar.

Field update requires object identity

If one reference to an object has a field update, then any other reference to the same object will observe the updated value.

With field update we use locations (l) as values. We add them to the grammar, together with the memory/store S

e ::= x | e.m(es) | new C(es) | e.f | e_0.f=e_1 | l

S ::= l_1 : C_1(ls_1) .. l_n : C_n(ls_n)

e ::= x | e.m(es) | new C(es) | e.f | e_0.f=e_1 | l

cd ::= class C implements Cs{ Fs K Ms }
| interface C extends Cs{ MH_1; .. MH_k; }

K ::= C(C_1 x_1 .. C_n x_n){this.x_1=x_1;..this.x_n=x_n;}

F ::= C f; M ::= MH{return e;}

MH ::= C m(C_1 x_1 .. C_n x_n)

S::= l_1 : C_1(ls_1) .. l_n : C_n(ls_n)

Ɛv ::= □ | Ɛv.m(es) | l.m(ls,Ɛv,es)| new C(ls,Ɛv,es) | Ɛv.f
| Ɛv.f=e | l.f=Ɛv

Γ ::= x_1 : C_1 .. x_n : C_n

#Define e_0[x=e_1] = e_2
x[x=e] = e
x[x'=e] = x *where:* x != x'
e_0.m(e_1..e_n)[x=e] = e_0[x=e].m(e_1[x=e]..e_n[x=e])
new C(e_1..e_n)[x=e] = new C(e_1[x=e]..e_n[x=e])
e_0.f[x=e] = e_0[x=e].f
l[x=e]=l

With field update we use locations (l) as values. We add them to the grammar, together with the memory/store S

e ::= x | e.m(es) | new C(es) | e.f | e_0.f=e_1 | l
S ::= l_1 : C_1(ls_1) .. l_n : C_n(ls_n)

Reduction will work on pairs of stores and expressions:

$S_0|e_0$ --> $S_1|e_1$

e ::= x | e.m(es) | new C(es) | e.f | e_0.f=e_1 | l

cd ::= class C implements Cs{ Fs K Ms }
| interface C extends Cs{ MH_1; .. MH_k; }

K ::= C(C_1 x_1 .. C_n x_n){this.x_1=x_1;..this.x_n=x_n;}

F ::= C f; M ::= MH{return e;}

MH ::= C m(C_1 x_1 .. C_n x_n)

S::= l_1 : C_1(ls_1) .. l_n : C_n(ls_n)

Ɛv ::= □ | Ɛv.m(es) | l.m(ls,Ɛv,es)| new C(ls,Ɛv,es) | Ɛv.f
| Ɛv.f=e | l.f=Ɛv

Γ ::= x_1 : C_1 .. x_n : C_n

#Define e_0[x=e_1] = e_2
x[x=e] = e
x[x'=e] = x *where:* x != x'
e_0.m(e_1..e_n)[x=e] = e_0[x=e].m(e_1[x=e]..e_n[x=e])
new C(e_1..e_n)[x=e] = new C(e_1[x=e]..e_n[x=e])
e_0.f[x=e] = e_0[x=e].f
l[x=e]=l

S **well formedness**: no l is repeated, the order of entries in S is irrelevant

All reduction rules

$$\text{(f-update)}\ \frac{\texttt{class C \_\{ \_ } C_0\ x_0;..C_n\ x_n\texttt{; K Ms \} in cds}}{l':C(ls\ l_0..l_n)\ S \mid l'.x_0{=}l \Rightarrow l':C(ls\ l\ l_1..l_n)\ S \mid l}$$

$$\text{(ctx)}\ \frac{S_0 \mid e_0 \Rightarrow S_1 \mid e_1}{S_0 \mid \mathcal{E}v[e_0] \Rightarrow S_1 \mid \mathcal{E}v[e_1]} \qquad \text{(f-access)}\ \frac{\texttt{class C \_\{ } C_1\ x_1;..C_n\ x_n\texttt{; K Ms \} in cds}}{l:C(l_1..l_n)\ S \mid l.x_i \Rightarrow l:C(l_1..l_n)\ S \mid l_i}$$

$$\text{(new)}\ \frac{}{S \mid \texttt{new C(ls)} \Rightarrow l:C(ls)\ S \mid l} \qquad \text{(m-call)}\ \frac{\texttt{class C \_\{ \_ } C_0\ \texttt{m(C1 x1..Cn xn)\{return e;\} \_ \} in cds}}{l:C(ls)\ S \mid l.m(l_1..l_n) \Rightarrow l:C(ls)\ S \mid e[this{=}l\ x_1{=}l_1..x_n{=}l_n]}$$

```
e  ::= x | e.m(es) | new C(es) | e.f | e0.f=e1 | l

cd ::= class C implements Cs{ Fs K Ms }
     | interface C extends Cs{ MH1; .. MHk; }

K  ::= C(C1 x1 .. Cn xn){this.x1=x1;..this.xn=xn;}

F  ::= C f;        M  ::= MH{return e;}

MH ::= C m(C1 x1 .. Cn xn)

S::= l1 : C1(ls1) .. ln : Cn(lsn)

Ev ::= □ | Ev.m(es) | l.m(ls,Ev,es)| new C(ls,Ev,es) | Ev.f
  | Ev.f=e | l.f=Ev

Γ  ::= x1 : C1 .. xn : Cn
```

```
_______
#Define  e0[x=e1] = e2
  x[x=e] = e
  x[x'=e] = x where: x != x'
  e0.m(e1..en)[x=e] = e0[x=e].m(e1[x=e]..en[x=e])
  new C(e1..en)[x=e] = new C(e1[x=e]..en[x=e])
  e0.f[x=e] = e0[x=e].f
  l[x=e]=l
```

Typing:

Two ways of doing typing:

- ignoring the memory: just add a typing rule for e.f=e':

$$\text{(f-update-t)}\ \frac{\Gamma \vdash e_0 : C \quad \Gamma \vdash e : C_i \quad \text{class } C \ \_\{\ C_1\ x_1; .. C_n\ x_n;\ K\ Ms\ \} \text{ in cds}}{\Gamma \vdash e_0.x_i = e : C_i}$$

  - This way is useful if we just want to implement a type checker, since the term “l” is never contained in the source code.

- Redefine all typing rules to keep memory and locations in account.
  - This way is useful if we want to do prove correctness properties on our reduction.

Now we have an imperative FJ

- In literature, this version of FJ is used to study type systems and other techniques to control aliasing and mutations.
- This kind of control can be applied to provide safe parallelism or to other aspects of code verification and correctness.

Local variable update?

- Field update can be used to emulate local variable updates by using “box objects”.
- For example

```
class NumBox{ N inner; NumBox(N inner){this.inner=inner;}} // a box containing a number

class User{
  N meth(NumBox b){
    return this.op(b.inner=b.inner.sum(b.inner), b.inner);
    }

  N op(N unused,N res){return res;}//op emulates multiple statements
  }

//This can emulate both argument pass by reference and by value:

new User().meth(myNumBox); //argument passed by reference (as in C macro expansion)

new User().meth(new NumBox(myNumBox)); //argument passed by value (as in Java)
```

- pass by value:
  assigning the method parameter “x=foo” has no effect on the caller

- pass by reference:
  assigning the method parameter “x=foo” updates the value on the caller side

pass by value/reference **is not about field updates but only parameters update!**

Local variable update as a primitive

- We can also extend FJ to also support field update. Not very interesting from a research perspective, but it is a good exercise
- We need to add 'x=e' as an expression
- Now 'x' evaluates to a value

S **well formedness**: no l is repeated,
the order of entries in S is irrelevant

V **well formedness**: no x is repeated,
the order of entries in V is irrelevant

$V ::= x_1 : l_1 \;..\; x_n : l_n$

Reduction grammar: $S_0;V_0|e_0 \dashrightarrow S_1;V_1|e_1$

All reduction rules

$$\frac{}{S;V\ x{:}l \mid x \Rightarrow S;V\ x{:}l \mid l}\ (\text{v-access})$$

$$\frac{}{S;V\ x{:}l_0 \mid x{=}l \Rightarrow S;V\ x{:}l \mid l}\ (\text{v-update})$$

$$\frac{\text{class } C \_\{ \_\ C_0\ x_0;..C_n\ x_n;\ K\ Ms\ \} \text{ in } cds}{l'{:}C(ls\ l_0..l_n)\ S;V \mid l'.x_0{=}l \Rightarrow l'{:}C(ls\ l\ l_1..l_n)\ S;V \mid l}\ (\text{f-update})$$

$$\frac{S_0;V_0 \mid e_0 \Rightarrow S_1;V_1 \mid e_1}{S_0;V_0 \mid \mathcal{E}v[e_0] \Rightarrow S_1;V_1 \mid \mathcal{E}v[e_1]}\ (\text{ctx})$$

$$\frac{\text{class } C \_\{\ C_1\ x_1;..C_n\ x_n;\ K\ Ms\ \} \text{ in } cds}{l{:}C(l_1..l_n)\ S;V \mid l.x_i \Rightarrow l{:}C(l_1..l_n)\ S;V \mid l_i}\ (\text{f-access})$$

$$\frac{}{S;V \mid \text{new } C(ls) \Rightarrow l{:}C(ls)\ S;V \mid l}\ (\text{new})$$

$$\frac{\text{class } C \_\{ \_\ C_0\ m(C1\ x1..Cn\ xn)\{\text{return } e;\}\ \_\ \} \text{ in } cds}{l{:}C(ls)\ S;V \mid l.m(l_1..l_n) \Rightarrow l{:}C(ls)\ S;\ V\ this{:}l, x_1{:}l_1..x_n{:}l_n \mid e}\ (\text{m-call})$$

Error above! What is going wrong?

```
e  ::= x | e.m(es) | new C(es) | e.f | e0.f=e1 | x=e | l

cd ::= class C implements Cs{ Fs K Ms }
     | interface C extends Cs{ MH1; .. MHk; }

K  ::= C(C1 x1 .. Cn xn){this.x1=x1;..this.xn=xn;}

F  ::= C f;        M  ::= MH{return e;}

MH ::= C m(C1 x1 .. Cn xn)

S::= l1 : C1(ls1) .. ln : Cn(lsn)

V::= x1 : l1 .. xn : ln

Ev ::= □ | Ev.m(es) | l.m(ls,Ev,es)| new C(ls,Ev,es) | Ev.f
  | Ev.f=e | l.f=Ev | x=Ev

Γ  ::= x1 : C1 .. xn : Cn
```

S **well formedness**: no l is repeated,
the order of entries in S is irrelevant

V **well formedness**: no x is repeated,
the order of entries in V is irrelevant

$$\frac{\text{class C \_\{ \_ } C_0 \text{ m(C1 x1..Cn xn)\{return e;\} \_ \} in cds}}{l{:}C(ls)\ S;V|l.m(l_1..l_n) \rightarrow l{:}C(ls)\ S;\ V\ this{:}l,x_1{:}l_1..x_n{:}l_n|e}\ \text{(m-call)}$$

Error above! What is going wrong?

```
e  ::= x | e.m(es) | new C(es) | e.f | e0.f=e1 | x=e | l

cd ::= class C implements Cs{ Fs K Ms }
     | interface C extends Cs{ MH1; .. MHk; }

K  ::= C(C1 x1 .. Cn xn){this.x1=x1;..this.xn=xn;}

F  ::= C f;        M  ::= MH{return e;}

MH ::= C m(C1 x1 .. Cn xn)

S::= l1 : C1(ls1) .. ln : Cn(lsn)

V::= x1 : l1 .. xn : ln

Ev ::= □ | Ev.m(es) | l.m(ls,Ev,es)| new C(ls,Ev,es) | Ev.f
  | Ev.f=e | l.f=Ev | x=Ev

Γ  ::= x1 : C1 .. xn : Cn
```

We are adding the method formal parameter names to V.
What if V contains some those variable names already? It would be a non well formed V, thus the metarule could not be instantiated.

We could use a technique called alpha renaming: where variables are consistently renamed with variables with different names. In our specific case we could define a notation

$$V|e \cong V'|e'$$

where V' and e' are like V and e but variables names are consistently renamed.

We can then write the rule (m-call) using such notation

S **well formedness**: no l is repeated,
the order of entries in S is irrelevant

V **well formedness**: no x is repeated,
the order of entries in V is irrelevant

$V ::= x_1 : l_1 \;..\; x_n : l_n$

Reduction grammar: $S_0,V_0|e_0 \dashrightarrow S_1,V_1|e_1$

```
e  ::= x | e.m(es) | new C(es) | e.f | e0.f=e1 | x=e | l

cd ::= class C implements Cs{ Fs K Ms }
     | interface C extends Cs{ MH1; .. MHk; }

K  ::= C(C1 x1 .. Cn xn){this.x1=x1;..this.xn=xn;}

F  ::= C f;        M  ::= MH{return e;}

MH ::= C m(C1 x1 .. Cn xn)

S::= l1 : C1(ls1) .. ln : Cn(lsn)

V::= x1 : l1 .. xn : ln

Ev ::= □ | Ev.m(es) | l.m(ls,Ev,es)| new C(ls,Ev,es) | Ev.f
  | Ev.f=e | l.f=Ev | x=Ev

Γ  ::= x1 : C1 .. xn : Cn
```

All reduction rules

$$\text{(v-access)}\ \frac{}{S;V\ x:l \mid x \rightarrow S;V\ x:l \mid l}$$

$$\text{(v-update)}\ \frac{}{S;V\ x:l_0 \mid x=l \rightarrow S;V\ x:l \mid l}$$

$$\text{(f-update)}\ \frac{\text{class C \_\{ \_ } C_0\ x_0;..C_n\ x_n;\ K\ Ms\ \}\ \text{in cds}}{l':C(ls\ l_0..l_n)\ S;V \mid l'.x_0=l \rightarrow l':C(ls\ l\ l_1..l_n)\ S;V \mid l}$$

$$\text{(ctx)}\ \frac{S_0;V_0 \mid e_0 \rightarrow S_1;V_1 \mid e_1}{S_0;V_0 \mid Ev[e_0] \rightarrow S_1;V_1 \mid Ev[e_1]}$$

$$\text{(f-access)}\ \frac{\text{class C \_\{ } C_1\ x_1;..C_n\ x_n;\ K\ Ms\ \}\ \text{in cds}}{l:C(l_1..l_n)\ S;V \mid l.x_i \rightarrow l:C(l_1..l_n)\ S;V \mid l_i}$$

$$\text{(new)}\ \frac{}{S;V \mid new\ C(ls) \rightarrow l:C(ls)\ S;V \mid l}$$

$$\text{(m-call)}\ \frac{\text{class C \_\{ \_ } C_0\ m(C1\ x1..Cn\ xn)\{return\ e;\}\ \_\ \}\ \text{in cds}\quad this:l,x_1:l_1..x_n:l_n \mid e \cong V' \mid e'}{l:C(ls)\ S;V \mid l.m(l_1..l_n) \rightarrow l:C(ls)\ S;\ V\ V' \mid e'}$$

Tomorrow, Mock Term Test!

- It is a fundamental occasion to do exercise and to review this first part of the course.
- It will contains some other extensions to FJ, and you will have to spot and fix the mistakes, complete the formalism and answer questions.
- It is NOT MARKED, but it is mandatory to submit it. -What about this week surprise?