$\mathcal{L}_2$-a formal, minimal, imperative, class based, object oriented language with inheritance, without overloading

$\mathcal{L}_2$= $\mathcal{L}_1$+ inheritance.

We shall use $\mathcal{L}_2$ for a basic study of inheritance, and will then continue with the discussion of implementation issues.

As for any language, the formal description of $\mathcal{L}_2$ consists of

- the syntax
- the operational semantics
- the type system
- agreement between heap, frame and program, environment.
- type soundness, demonstrated through a subject reduction theorem.

An Example in $\mathcal{L}_2$

Consider the following program

$P_{ei} \equiv$

```
class Stdt {
        bool eat( Food x) {  x.tasty( true )  }
}
class Food {
        bool  fat
        bool tasty( bool x) {  false  }
        Food mix( Food x) {  x  }
}
class Pizza extends Food{
        Food ingrds
        bool tasty( bool  x) {  true  }
}
```

An Example in $\mathcal{L}_2$ - 2

How does the following programme behave?

```
1.  Food f = new Food;
2.  f.fat            \\ ???
3.  f.ingrds         \\ ???
4.  f.tasty(...)     \\ ???

5.  f = new Pizza;
6.  f.fat            \\ ???
7.  f.ingrds         \\ ???
8.  f.tasty(...)     \\ ???
9.  f.mix(f)         \\ ???

10. Pizza p = new Pizza;
11. p.fat            \\ ???
12. p.ingrds         \\ ???
13. p.tasty(...)     \\ ???
14. p.mix(p)         \\ ???
```

An Example in $\mathcal{L}_2$ - 3

The program from previous slide behaves as follows:

1. Food f = new Food;
2. f.fat \\ returns false
3. f.ingrds \\ TYPE ERROR
4. f.tasty(...) \\ returns false

5. f = new Pizza;
6. f.fat \\ returns false
7. f.ingrds \\ TYPE ERROR
8. f.tasty(...) \\ returns true
9. f.mix(f) \\ ...

10. Pizza p = new Pizza;
11. p.fat \\ returns false
12. p.ingrds \\ returns null
13. p.tasty(...) \\ returns true
14. p.mix(p) \\ ...

An Example in $\mathcal{L}_2$ - 4

Our example demonstrates

- A subclass inherits all fields of superclass (here line 11)
- A subclass inherits all methods from superclass (here line 14)
- A subclass object may appear where a superclass object expected (here line 14)
- A subclass may override methods from a superclass; methods are bound dynamically (here line 8 and 13)
- Difference static type and dynamic class (lines 7 vs 12)

The syntax of $\mathcal{L}_2$ expressions, the structure of $\mathcal{L}_2$ programs

$$
\begin{array}{lcll}
\mathit{Progr} & = & \mathit{ClassId} \longrightarrow & (\ \mathit{ClassId} \\
 & & & \quad \times \\
 & & & (\mathit{FieldId} \longrightarrow \mathit{type}\,) \\
 & & & \quad \times \\
 & & & (\mathit{MethId} \longrightarrow \mathit{meth}\,) \ )
\end{array}
$$

where

$$
\begin{array}{lcl}
\mathit{meth} & ::= & \mathit{type}\ m \ \ \mathbf{(}\ \mathit{type}\,\mathrm{x}\ \mathbf{)} \ \ \mathbf{\{} \ e \ \mathbf{\}} \\
\mathit{type} & ::= & \mathrm{bool} \ \mid \ c \\
e & ::= & \mathrm{if} \ \ e \ \ \mathrm{then} \ \ e \ \ \mathrm{else} \ \ e \ \ \mid \\
 & & e\,.f \ \mid \ e\,.f := e \ \mid \ e\,.m\,\mathbf{(}\ e\ \mathbf{)} \ \mid \\
 & & \mathrm{new}\ c \ \mid \ x \ \mid \ \mathrm{this} \ \mid \ \mathrm{true} \ \mid \ \mathrm{false} \ \mid \ \mathrm{null}\ \mathbf{.}
\end{array}
$$

Question: What are the differences between $\mathcal{L}_1$ and $\mathcal{L}_2$?

The example P_{SF} is represented in our system as

$$
\begin{array}{llll}
P_{SF} \equiv & \mathsf{Stdt} \mapsto & (& \mathsf{Object}, \\
& & & \emptyset, \\
& & & \mathsf{eat} \mapsto \mathsf{bool\ eat}\mathbf{(}\ \mathsf{Food\ \ x}\mathbf{)}\ \mathbf{\{}\ \ \mathsf{x.tasty}\mathbf{(}\ \mathsf{true}\ \mathbf{)}\ \ \mathbf{\}} \qquad), \\
& \mathsf{Food} \mapsto & (& \mathsf{Object}, \\
& & & \mathsf{fat} \mapsto \mathsf{bool}, \\
& & & (\ \ \mathsf{tasty} \mapsto \mathsf{bool\ tasty}\mathbf{(}\ \mathsf{bool\ x}\mathbf{)}\ \mathbf{\{}\ \ \mathsf{false}\ \ \mathbf{\}}\ , \\
& & & \quad \mathsf{mix} \mapsto \mathsf{Food\ mix}\mathbf{(}\ \mathsf{Food\ x}\mathbf{)}\ \mathbf{\{}\ \ \mathsf{x}\ \ \mathbf{\}}\ \ ) \qquad), \\
& \mathsf{Pizza} \mapsto & (& \mathsf{Food}, \\
& & & \mathsf{ingrds} \mapsto \mathsf{Food}, \\
& & & \mathsf{tasty} \mapsto \mathsf{bool\ tasty}\mathbf{(}\ \mathsf{bool\ x}\mathbf{)}\ \mathbf{\{}\ \ \mathsf{true}\ \ \mathbf{\}} \qquad).
\end{array}
$$

Subclasses, and Acyclic class hierarchies

The judgement $\mathsf{P} \vdash \mathsf{c} \sqsubseteq \mathsf{c}'$ means that c is a subclass of c'; the judgement $Acyclic(\mathsf{P})$ means that the class hierarchy in P is acyclic.

$$\frac{}{\mathsf{P} \vdash \mathsf{Object} \sqsubseteq \mathsf{Object}} \qquad \frac{\mathsf{P}(\mathsf{c}) \downarrow_1 = \mathsf{c}'}{\begin{array}{c}\mathsf{P} \vdash \mathsf{c} \sqsubseteq \mathsf{c} \\ \mathsf{P} \vdash \mathsf{c} \sqsubseteq \mathsf{c}'\end{array}} \qquad \frac{\begin{array}{c}\mathsf{P} \vdash \mathsf{c} \sqsubseteq \mathsf{c}' \\ \mathsf{P} \vdash \mathsf{c}' \sqsubseteq \mathsf{c}''\end{array}}{\mathsf{P} \vdash \mathsf{c} \sqsubseteq \mathsf{c}''}$$

$$Acyclic(\mathsf{P}) \quad \equiv \quad \forall \mathsf{c}, \mathsf{c}'. \begin{cases} (\mathsf{P} \vdash \mathsf{c} \sqsubseteq \mathsf{c}' \text{ and } \mathsf{P} \vdash \mathsf{c}' \sqsubseteq \mathsf{c} \quad \Longrightarrow \quad \mathsf{c} = \mathsf{c}') \\ \text{and} \\ (\mathsf{P}(\mathsf{c}) \downarrow_1 = \mathsf{c}' \quad \Longrightarrow \quad \mathsf{c} \neq \mathsf{c}') \end{cases}$$

For example, in $\mathsf{P_{SF}}$:

$\mathsf{P_{SF}} \vdash \mathsf{Object} \sqsubseteq \mathsf{Object}$

$\mathsf{P_{SF}} \vdash \mathsf{Stdt} \sqsubseteq \mathsf{Stdt}$ $\qquad$ $\mathsf{P_{SF}} \vdash \mathsf{Stdt} \sqsubseteq \mathsf{Object}$

$\mathsf{P_{SF}} \vdash \mathsf{Food} \sqsubseteq \mathsf{Food}$ $\qquad$ $\mathsf{P_{SF}} \vdash \mathsf{Food} \sqsubseteq \mathsf{Object}$

$\mathsf{P_{SF}} \vdash \mathsf{Pizza} \sqsubseteq \mathsf{Pizza}$ $\qquad$ $\mathsf{P_{SF}} \vdash \mathsf{Pizza} \sqsubseteq \mathsf{Food}$ $\qquad$ $\mathsf{P_{SF}} \vdash \mathsf{Pizza} \sqsubseteq \mathsf{Object}$

The above are *all* the subclass relationships in $\mathsf{P_{SF}}$, therefore,

$Acyclic(\mathsf{P_{SF}})$.

On the other hand, for the program $\mathsf{P_{cyc}}$ corresponding to

`class A extends B{ ... }` $\qquad$ `class B extends A{ ... }`

we have that $\quad$ NOT($Acyclic(\mathsf{P_{cyc}})$).

Field and method lookup functions

We need to define field and method lookup, so that it takes inheritance into account. E.g. $\mathcal{F}(\mathsf{P_{SF}}, \mathsf{Pizza}, \mathsf{fat}) = \mathsf{bool}$,
and $\mathcal{M}(\mathsf{P_{SF}}, \mathsf{Pizza}, \mathsf{mix}) = $ Food mix**(** Food x**) {** x **}**

space for students' deliberations

Field and method lookup functions - 2

For program P with $Acyclic(\mathsf{P})$, identifiers c, f, and m, we define:

$$\mathcal{FD}(\mathsf{P}, \mathsf{c}, \mathsf{f}) \quad = \quad \mathsf{P}(\mathsf{c}) \downarrow_2 (\mathsf{f}).$$

$$\mathcal{F}(\mathsf{P}, \mathsf{c}, \mathsf{f}) \quad = \quad \begin{cases} \mathcal{FD}(\mathsf{P}, \mathsf{c}, \mathsf{f}) & \textit{if } \mathcal{FD}(\mathsf{P}, \mathsf{c}, \mathsf{f}) \text{ is defined ,} \\ \mathcal{F}(\mathsf{P}, \mathsf{P}(\mathsf{c}) \downarrow_1, \mathsf{f}) & \textit{otherwise.} \end{cases}$$

$$\mathcal{F}(\mathsf{P}, \mathsf{Object}, \mathsf{f}) \quad \text{is undefined .}$$

$$\mathcal{F}s(\mathsf{P}, \mathsf{c}) \quad = \quad \{\mathsf{f} \mid \mathcal{F}(\mathsf{P}, \mathsf{c}, \mathsf{f}) \text{ is defined } \}.$$

$$\mathcal{MD}(\mathsf{P}, \mathsf{c}, \mathsf{m}) \quad = \quad \mathsf{P}(\mathsf{c}) \downarrow_3 (\mathsf{m}).$$

$$\mathcal{M}(\mathsf{P}, \mathsf{c}, \mathsf{m}) \quad = \quad \begin{cases} \mathcal{MD}(\mathsf{P}, \mathsf{c}, \mathsf{m}) & if\ \mathcal{MD}(\mathsf{P}, \mathsf{c}, \mathsf{m}) \text{ is defined }, \\ \mathcal{M}(\mathsf{P}, \mathsf{P}(\mathsf{c}) \downarrow_1, \mathsf{m}) & otherwise. \end{cases}$$

$$\mathcal{M}(\mathsf{P}, \mathsf{Object}, \mathsf{m}) \quad \text{is undefined .}$$

Questions: 1. Why did we require $Acyclic(\mathsf{P})$? 2. Could we have dropped the requirement that $Acyclic(\mathsf{P})$?

For example,

$$
\begin{array}{lcl}
\mathcal{FD}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Food}, \mathsf{fat}) & = & \mathsf{bool}, \\
\mathcal{FD}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Pizza}, \mathsf{fat}) & & \text{is undefined}, \\
\mathcal{FD}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Food}, \mathsf{ingrds}) & & \text{is undefined}, \\
\mathcal{FD}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Pizza}, \mathsf{ingrds}) & = & \mathsf{Food}, \\
\mathcal{F}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Food}, \mathsf{fat}) & = & \mathsf{bool}, \\
\mathcal{F}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Food}, \mathsf{ingrds}) & & \text{is undefined}, \\
\mathcal{F}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Pizza}, \mathsf{fat}) & = & \mathsf{bool}, \\
\mathcal{F}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Pizza}, \mathsf{ingrds}) & = & \mathsf{Food}, \\
\mathcal{F}s(\mathsf{P}_{\mathsf{SF}}, \mathsf{Pizza}) & = & \{\ \mathsf{fat}, \mathsf{ingrds}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{M}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Food}, \mathsf{mix}) & = & \mathsf{Food\ mix}\textbf{(}\ \mathsf{Food\ x}\textbf{)}\ \textbf{\{}\ \ \mathsf{x}\ \textbf{\}} \\
\mathcal{M}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Food}, \mathsf{tasty}) & = & \mathsf{bool\ tasty}\textbf{(}\ \mathsf{bool\ x}\textbf{)}\ \textbf{\{}\ \ \mathsf{false}\ \ \textbf{\}} \\
\mathcal{M}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Pizza}, \mathsf{mix}) & = & \mathsf{Food\ mix}\textbf{(}\ \mathsf{Food\ x}\textbf{)}\ \textbf{\{}\ \ \mathsf{x}\ \textbf{\}} \\
\mathcal{M}(\mathsf{P}_{\mathsf{SF}}, \mathsf{Pizza}, \mathsf{tasty}) & = & \mathsf{bool\ tasty}\textbf{(}\ \mathsf{bool\ x}\textbf{)}\ \textbf{\{}\ \ \mathsf{true}\ \ \textbf{\}}
\end{array}
$$

The Operational Semantics of $\mathcal{L}_2$

space for students' deliberations

The Operational Semantics of $\mathcal{L}_2$

... is identical to that of $\mathcal{L}_1$.

For example, the following stack frame ϕ_0 and heap χ_0 correspond to some execution of $\mathsf{P_{SF}}$:

$$
\begin{array}{lcl}
\phi_0 & = & (\iota_3, \iota_4) \\
\chi_0(\iota_3) & = & (\ \mathsf{Stdt},\ \emptyset) \\
\chi_0(\iota_4) & = & (\ \mathsf{Pizza},\ (\mathsf{fat} \mapsto \mathsf{true}\ , \mathsf{ingrds} \mapsto \iota_6)\ ) \\
\chi_0(\iota_5) & = & (\ \mathsf{Food},\ (\mathsf{fat} \mapsto \mathsf{false}\ )\ ) \\
\chi_0(\iota_6) & = & (\ \mathsf{Pizza},\ (\mathsf{fat} \mapsto \mathsf{true}\ , \mathsf{ingrds} \mapsto \iota_5)\ )
\end{array}
$$

The operational semantics gives:

$$\mathsf{this.eat(\ x)}\ , \phi_0, \chi_0 \leadsto_{\mathsf{P_{BP}}} \mathsf{true}\ , \chi_0$$

which shows that tasty was bound dynamically to that from class Pizza.

Determinism of the operational semantics

We can prove determinism for $\mathcal{L}_2$ executions for acyclic programs.

Lemma For program P with $Acyclic(\mathsf{P})$, and any expression e, if

$$\mathsf{e},\phi,\chi \rightsquigarrow_{\mathsf{P}} \mathsf{r}',\chi' \quad \text{and} \quad \mathsf{e},\phi,\chi \rightsquigarrow_{\mathsf{P}} \mathsf{r}'',\chi''$$

then

$$\mathsf{r}' = \mathsf{r}'', \text{ and } \chi' = \chi''$$

up to renaming of addresses.

Proof: similar to that for $\mathcal{L}_1$.

Note: Compare with corresponding Lemma for $\mathcal{L}_1$.

Further properties of the operational semantics

Execution has the following properties

- preserves the classes of all objects
- preserves the existence of any fields in an object

Lemma For program P with $Acyclic(\mathsf{P})$, and any expression e, if

$$\mathsf{e},\phi,\chi \rightsquigarrow_{\mathsf{P}} \mathsf{r}',\chi'$$

then

- $\chi(\iota)$ is defined $\implies$ $\chi(\iota)\downarrow_1 = \chi'(\iota)\downarrow_1$
- $\chi(\iota)(\mathsf{f})$ is defined $\implies$ $\chi'(\iota)(\mathsf{f})$ is defined

Proof by structural induction over the derivation $\mathsf{e},\phi,\chi \rightsquigarrow_{\mathsf{P}} \mathsf{r}',\chi'$.

The Type System of $\mathcal{L}_2$

As for $\mathcal{L}_1$, typing is a judgement of the form:

$$\mathsf{P}, \Gamma \vdash \mathsf{e} : \ \mathsf{t}$$

i.e. , in context of program P and environment Γ, expression e has type t.

We also consider subtypes. A value of a type t, which is a subtype of t' may appear wherever a value of type t' is expected.

The subtype relationship is the projection of the subclass relationship onto types – that is, we have *name* type equivalence as opposed to *structural* type equivalence:

$$\frac{\mathsf{P} \vdash \mathsf{c} \sqsubseteq \mathsf{c}'}{\mathsf{P} \vdash \mathsf{c} \leq \mathsf{c}'} \qquad\qquad \frac{}{\mathsf{P} \vdash \mathsf{bool} \leq \mathsf{bool}}$$

We define $IsCls(\mathsf{P}, \mathsf{c})$ and $IsCls(\mathsf{P}, \mathsf{t})$ as for $\mathcal{L}_1$.

Types of Expressions

litVarThis

$$\frac{}{\begin{array}{l}P, \Gamma \vdash \mathsf{true} : \mathsf{bool} \\ P, \Gamma \vdash \mathsf{false} : \mathsf{bool} \\ P, \Gamma \vdash x : \Gamma(x) \\ P, \Gamma \vdash \mathsf{this} : \Gamma(\mathsf{this})\end{array}}$$

newNull

$$\frac{IsCls(P, c)}{\begin{array}{l}P, \Gamma \vdash \mathsf{null} : c \\ P, \Gamma \vdash \mathsf{new}\ c : c\end{array}}$$

fld

$$\frac{\begin{array}{l}P, \Gamma \vdash e : c \\ \mathcal{F}(P, c, f) = t\end{array}}{P, \Gamma \vdash e.f : t}$$

fldAss

$$\frac{\begin{array}{l}P, \Gamma \vdash e.f : t \\ P, \Gamma \vdash e' : t' \\ P \vdash t' \leq t\end{array}}{P, \Gamma \vdash e.f := e' : t'}$$

cond

$$\frac{\begin{array}{l} P, \Gamma \vdash e : \ \text{bool} \\ P, \Gamma \vdash e_1 : \ t_1 \\ P, \Gamma \vdash e_2 : \ t_2 \\ P \vdash t_i \leq t \text{ for } i \in 1, 2 \end{array}}{P, \Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \ t}$$

methCall

$$\frac{\begin{array}{l} P, \Gamma \vdash e_0 : \ c \\ P, \Gamma \vdash e_1 : \ t_1' \\ \mathcal{M}(P, c, m) = t \ m\mathbf{(} \ t_1 \ x\mathbf{)} \ \mathbf{\{} \ e \ \mathbf{\}} \\ P \vdash t_1' \leq t_1 \end{array}}{P, \Gamma \vdash e_0.m\mathbf{(} \ e_1\mathbf{)} \ : \ t}$$

Properties of types of expressions

Do the following properties hold?

- $P, \Gamma \vdash e : t$ and $P, \Gamma \vdash e : t'$ $\Longrightarrow$ $t = t'$
- $P, \Gamma \vdash e : t$ and $P, \Gamma \vdash e' : t$ $\Longrightarrow$ $e = e'$

Well-formed class

$$
\mathit{ClssWF}(\mathsf{P},\mathsf{c}) \equiv \begin{cases}
\mathsf{P}(\mathsf{c}) \downarrow_1 = \mathsf{c}' \text{ and } (\mathsf{c}' \neq \mathsf{Object} \implies \mathsf{c}' \in \mathit{dom}(\mathsf{P})\ ) \\
\text{and} \\
\forall \mathsf{f} :\ \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) = \mathsf{t} \implies \mathit{IsTyp}(\mathsf{P},\mathsf{t}) \text{ and } \mathcal{F}(\mathsf{P},\mathsf{c}',\mathsf{f}) \text{ undef.} \\
\text{and} \\
\forall \mathsf{m} :\ \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) = \mathsf{t}\ \mathsf{m}\mathbf{(}\ \mathsf{t}_1\ \mathsf{x}\mathbf{)}\ \ \mathbf{\{}\ \mathsf{e}\ \mathbf{\}} \implies \\
\qquad \mathit{IsTyp}(\mathsf{P},\mathsf{t}), \\
\qquad \text{and} \\
\qquad \mathit{IsTyp}(\mathsf{P},\mathsf{t}_1), \\
\qquad \text{and} \\
\qquad \mathsf{P}, \mathsf{t}_1\ \mathsf{x}, \mathsf{c}\ \mathsf{this} \vdash \mathsf{e} :\ \mathsf{t}', \text{ and } \mathsf{P} \vdash \mathsf{t}' \leq \mathsf{t} \quad \text{for some type } \mathsf{t}', \\
\qquad \text{and} \\
\qquad \mathcal{M}(\mathsf{P},\mathsf{c}',\mathsf{m}) \text{ undef. or } \mathcal{M}(\mathsf{P},\mathsf{c}',\mathsf{m}) = \mathsf{t}\ \mathsf{m}\mathbf{(}\ \mathsf{t}_1\ \mathsf{x}\mathbf{)}\ \ \mathbf{\{}\ \mathsf{e}'\ \mathbf{\}}\ .
\end{cases}
$$

$$
\mathit{ProgWF}(\mathsf{P}) \equiv \mathit{Acyclic}(\mathsf{P}) \text{ and } \forall \mathsf{c} \in \mathit{dom}(\mathsf{P}).\mathit{ClssWF}(\mathsf{P},\mathsf{c}).
$$

Soundness of the $\mathcal{L}_2$ Type System

The type system is sound in the sense that a converging well-typed expression returns either a value of the same type as the expression, or the nullPntrExc, but does not get stuck. Furthermore, in both cases, the resulting heap "agrees" with the program and the environment, *i.e.* its consistency is preserved.

Agreement

We introduce agreement notions between programs, heaps, and values:

As for L_1, we first define an auxiliary, "basic" agreement notion:

$$\frac{}{\mathsf{P},\chi\vdash\mathsf{true} <:\mathsf{bool}} \qquad \frac{}{\mathsf{P},\chi\vdash\mathsf{false} <:\mathsf{bool}}$$

$$\frac{IsCls(\mathsf{P},\mathsf{t})}{\mathsf{P},\chi\vdash\mathsf{null}<:\mathsf{t}} \qquad \frac{\chi(\iota)\downarrow_1 = \mathsf{c}}{\mathsf{P},\chi\vdash\iota<:\mathsf{c}} \qquad \frac{\mathsf{P},\chi\vdash\mathsf{v}<:\mathsf{t}' \qquad \mathsf{P}\vdash\mathsf{t}'\leq\mathsf{t}}{\mathsf{P},\chi\vdash\mathsf{v}<:\mathsf{t}}$$

Based on the "basic" agreement notion, we define agreement:

$$\mathsf{P},\chi\vdash\mathsf{v}\lhd\mathsf{t} \;\equiv\; \begin{cases} \mathsf{P},\chi\vdash\mathsf{v}<:\mathsf{t}, & \text{if } \mathsf{v}\in\{\mathsf{true},\mathsf{false},\mathsf{null}\}, \\ \mathsf{P},\chi\vdash\iota<:\mathsf{c}, \text{ and} & \\ \forall \mathsf{f} : \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f})=\mathsf{t}' \implies \mathsf{P},\chi\vdash\chi(\iota)(\mathsf{f})<:\mathsf{t}' & \text{if } \mathsf{v}=\iota, \text{ and } \mathsf{t}=\mathsf{c}, \\ \text{false} & \text{otherwise.} \end{cases}$$

What is difference between the definition of agreement for L_1 and for L_2?

Well-formed heap and stack frame

$$\mathsf{P}, \Gamma \vdash (\iota, \mathsf{v}), \chi \diamond \quad \equiv \quad \begin{cases} \mathsf{P}, \chi \vdash \iota \triangleleft \Gamma(\mathsf{this}), \quad \text{and} \\ \mathsf{P}, \chi \vdash \mathsf{v} \triangleleft \Gamma(\mathsf{x}), \quad \text{and} \\ \forall \iota' \in dom(\chi) : \mathsf{P}, \chi \vdash \iota' \triangleleft \chi(\iota') \downarrow_1 \end{cases}$$

Lemma

If

$$\mathsf{P}, \Gamma \vdash \phi, \chi \diamond \quad \text{and} \quad \chi(\iota) \downarrow_1 = \mathsf{c} \quad \text{and} \quad \mathsf{f} \in \mathcal{F}s(\mathsf{P}, \mathsf{c}),$$

then

$$\mathsf{P}, \chi \vdash \chi(\iota)(\mathsf{f}) \triangleleft \mathcal{F}(\mathsf{P}, \mathsf{c}, \mathsf{f})$$

Remember our example, where

$$
\begin{array}{lll}
P_{SF} \equiv & \mathrm{Stdt} \mapsto & (\ \mathrm{Object},\ \emptyset,\ \ldots\ ) \\
& \mathrm{Food} \mapsto & (\ \mathrm{Object},\ \mathrm{fat} \mapsto \mathrm{bool},\ \ldots\ ) \\
& \mathrm{Pizza} \mapsto & (\ \mathrm{Food},\ \mathrm{ingrds} \mapsto \mathrm{Food},\ \ldots)
\end{array}
$$

Take a heap χ_2, defined as follows

$$
\begin{array}{ll}
\chi_2(\iota_3) = (\ \mathrm{Stdt},\ (\mathrm{ingrds} \mapsto \iota_3)\ ) & \chi_2(\iota_4) = (\ \mathrm{Pizza},\ (\mathrm{fat} \mapsto \mathrm{true}\ )\ ) \\
\chi_2(\iota_5) = (\ \mathrm{Food},\ (\mathrm{fat} \mapsto \mathrm{false}\ )\ ) & \chi_2(\iota_6) = (\ \mathrm{Pizza},\ (\mathrm{fat} \mapsto \mathrm{true}\ ,\mathrm{ingrds} \mapsto \iota_4)\ )
\end{array}
$$

Then, which of the following judgments hold

$$
\begin{array}{ll}
P_{SF}, \chi_2 \vdash \iota_3 \lhd \mathrm{Stdt} & \\
P_{SF}, \chi_2 \vdash \iota_4 \lhd \mathrm{Food} & P_{SF}, \chi_2 \vdash \iota_4 \lhd \mathrm{Pizza} \\
P_{SF}, \chi_2 \vdash \iota_5 \lhd \mathrm{Food} & P_{SF}, \chi_2 \vdash \iota_6 \lhd \mathrm{Pizza}
\end{array}
$$

Continue with our example, where

$$\mathrm{P_{SF}} \equiv \mathsf{Stdt} \mapsto (\,\mathsf{Object},\, \emptyset,\ldots),\ \ \mathsf{Food} \mapsto (\,\mathsf{Object},\, (\mathsf{fat} \mapsto \mathsf{bool}),\ldots),$$
$$\mathsf{Pizza} \mapsto (\,\mathsf{Food},\, (\mathsf{ingrds} \mapsto \mathsf{Food}),\ldots)$$

Take a frame ϕ_0, and a heap χ_0, defined as follows:

$$\phi_0 \ = (\iota_3, \iota_4)$$
$$\chi_0(\iota_3) = (\ \mathsf{Stdt},\ \emptyset) \qquad \chi_0(\iota_4) = (\ \mathsf{Pizza},\ (\mathsf{fat} \mapsto \mathsf{true}\ , \mathsf{ingrds} \mapsto \iota_6)\ )$$
$$\chi_0(\iota_5) = (\ \mathsf{Food},\ (\mathsf{fat} \mapsto \mathsf{false}\ )\ ) \qquad \chi_0(\iota_6) = (\ \mathsf{Pizza},\ (\mathsf{fat} \mapsto \mathsf{true}\ , \mathsf{ingrds} \mapsto \iota_5)\ )$$

Then:

$$\mathrm{P_{SF}}, \chi_0 \vdash \iota_3 \lhd \mathsf{Stdt}$$
$$\mathrm{P_{SF}}, \chi_0 \vdash \iota_4 \lhd \mathsf{Pizza} \qquad \mathrm{P_{SF}}, \chi_0 \vdash \iota_4 \lhd \mathsf{Food}$$
$$\ldots \qquad \ldots$$
$$\mathrm{P_{SF}}, \mathsf{Food\ x}, \mathsf{Stdt\ this} \vdash \phi_0, \chi_0 \diamond \qquad \mathrm{P_{SF}}, \mathsf{Pizza\ x}, \mathsf{Stdt\ this} \vdash \phi_0, \chi_0 \diamond$$

On the other hand, χ_2 is so “badly formed”, that $\forall \Gamma, \phi: \quad \mathrm{P_{SF}}, \Gamma \not\vdash \phi, \chi_2 \diamond$.

Properties of Well-formed programs - preservation of members in subclasses

Lemma For program P, class identifiers c and c', method identifier m,

If $\mathit{ProgWF}(P)$, and $P \vdash c' \sqsubseteq c$, then

- $\mathcal{F}(P, c, f)$ is defined $\implies$ $\mathcal{F}(P, c', f) = \mathcal{F}(P, c, f)$.
- $\mathcal{M}(P, c, m) = t\ m(\ t'\ x)\ \{\ \_\ \}$ $\implies$ $\exists e: \mathcal{M}(P, c', m) = t\ m(\ t'\ x)\ \{\ e\ \}$.

Question Does the opposite direction hold, *e.g.* do $\mathit{ProgWF}(P)$, and $P \vdash c' \sqsubseteq c$, and $\mathcal{F}(P, c', f) = t$ imply that $\mathcal{F}(P, c, f) = t$?

Properties of Well-formed programs - preservation of types in more precise environments

Lemma For program P, environments Γ and Γ', expression e, and type t:

If $\mathit{ProgWF}(\mathsf{P})$, and $\mathsf{P}\vdash\Gamma'(\mathsf{this})\sqsubseteq\Gamma(\mathsf{this})$, and $\mathsf{P}\vdash\Gamma'(\mathsf{x})\leq\Gamma(\mathsf{x})$, then

- $\mathsf{P},\Gamma\vdash\mathsf{e}:\ \mathsf{t} \quad\Longrightarrow\quad \exists\mathsf{t}'$ with $\mathsf{P},\Gamma'\vdash\mathsf{e}:\ \mathsf{t}'$, $\mathsf{P}\vdash\mathsf{t}'\leq\mathsf{t}$.

Question Does the opposite direction hold, *i.e.* do $\mathit{ProgWF}(\mathsf{P})$, and $\mathsf{P}\vdash\Gamma'(\mathsf{this})\sqsubseteq\Gamma(\mathsf{this})$, and $\mathsf{P}\vdash\Gamma'(\mathsf{x})\leq\Gamma(\mathsf{x})$, and $\mathsf{P},\Gamma'\vdash\mathsf{e}:\ \mathsf{t}$ imply that $\mathsf{P},\Gamma\vdash\mathsf{e}:\ \mathsf{t}$?

Type Soundness

Theorem For program P, environment Γ, expression e, heap χ, stack frame ϕ, and type t, if

$$\mathit{ProgWF}(\mathsf{P}), \quad \text{and} \quad \mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{t}, \quad \text{and} \quad \mathsf{P}, \Gamma \vdash \phi, \chi \diamond, \quad \text{and} \quad \mathsf{e}, \phi, \chi \rightsquigarrow_{\mathsf{P}} \mathsf{r}, \chi',$$

then

- $\mathsf{r} \in val$, and $\mathsf{P}, \chi' \vdash \mathsf{r} \lhd \mathsf{t}$, and $\mathsf{P}, \Gamma \vdash \phi, \chi' \diamond$,

or

- $\mathsf{r} = \mathsf{nullPntrExc}$, and $\mathsf{P}, \Gamma \vdash \phi, \chi' \diamond$.

Question Do we not need to mention subtypes/subclasses in that Theorem?

Proof by structural induction over the derivation $\mathsf{e}, \phi, \chi \rightsquigarrow_{\mathsf{P}} \mathsf{r}, \chi'$.

Subsumption

As we said earlier, a value of a subtype may appear wherever a value of a supertype is expected.

This is usually formalized through a subsumption rule, which is:

$$\frac{\begin{array}{l} \mathsf{P}, \Gamma \vdash_s \mathsf{e} : \ \mathsf{t} \\ \mathsf{P} \vdash \mathsf{t} \leq \mathsf{t}' \end{array}}{\mathsf{P}, \Gamma \vdash_s \mathsf{e} : \ \mathsf{t}'} \ \text{Subsump}$$

Such a rule seems obvious. If we added such a rule to the type system of $\mathcal{L}_2$, we would not need to mention subtypes explicitly any more , *i.e.* , we would obtain:

Types of $\mathcal{L}_2$-Expressions with Subsumption

litVarThis

$$\frac{}{\ldots\ as\ before}$$

newNull

$$\frac{\ldots\ as\ before}{\ldots\ as\ before}$$

Subsump

$$\frac{\mathsf{P},\Gamma \vdash_s \mathsf{e} : \mathsf{t} \qquad \mathsf{P} \vdash \mathsf{t} \leq \mathsf{t}'}{\mathsf{P},\Gamma \vdash_s \mathsf{e} : \mathsf{t}'}$$

fld

$$\frac{\mathsf{P},\Gamma \vdash_s \mathsf{e} : \mathsf{c} \qquad \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) = \mathsf{t}}{\mathsf{P},\Gamma \vdash_s \mathsf{e.f} : \mathsf{t}}$$

fldAss

$$\frac{\mathsf{P},\Gamma \vdash_s \mathsf{e.f} : \mathsf{t} \qquad \mathsf{P},\Gamma \vdash_s \mathsf{e}' : \mathsf{t}}{\mathsf{P},\Gamma \vdash_s \mathsf{e.f}:=\mathsf{e}' : \mathsf{t}}$$

cond

$$\frac{\mathsf{P},\Gamma \vdash_s \mathsf{e} : \mathsf{bool} \qquad \mathsf{P},\Gamma \vdash_s \mathsf{e}_1 : \mathsf{t} \qquad \mathsf{P},\Gamma \vdash_s \mathsf{e}_2 : \mathsf{t}}{\mathsf{P},\Gamma \vdash_s \mathsf{if\ e\ then\ e}_1\ \mathsf{else\ e}_2 : \mathsf{t}}$$

methCall

$$\frac{\mathsf{P},\Gamma \vdash_s \mathsf{e}_0 : \mathsf{c} \qquad \mathsf{P},\Gamma \vdash_s \mathsf{e}_1 : \mathsf{t}_1 \qquad \mathcal{M}(\mathsf{P},\mathsf{c},\mathsf{m}) = \mathsf{t\ m}\textbf{(}\ \mathsf{t}_1\ \mathsf{x}\textbf{)}\ \ \textbf{\{}\ \mathsf{e}\ \textbf{\}}}{\mathsf{P},\Gamma \vdash_s \mathsf{e}_0.\mathsf{m}\textbf{(}\ \mathsf{e}_1\textbf{)} : \mathsf{t}}$$

Properties of the system $P, \Gamma \vdash_s e : t$

The type rules with subsumption are more elegant than those without.
... are they?

Do the following properties hold?

- $P, \Gamma \vdash_s e : t$ and $P, \Gamma \vdash_s e : t' \quad \Longrightarrow \quad t = t'$
- $P, \Gamma \vdash_s e : t$ and $P, \Gamma \vdash_s e' : t \quad \Longrightarrow \quad e = e'$
- $P, \Gamma \vdash_s e : t \quad \Longrightarrow \quad P, \Gamma \vdash e : t$
- $P, \Gamma \vdash e : t \quad \Longrightarrow \quad P, \Gamma \vdash_s e : t$

But ...

The type system with subsumption is NOT sound!

Here is a counterexample:

On the next slide we will "repair" the previous type system:

Types of $\mathcal{L}_2$-Expressions with Subsumption - revised

litVarThis

$$\frac{}{\dots\ as\ before}$$

newNull

$$\frac{\dots\ as\ before}{\dots\ as\ before}$$

fld

$$\frac{\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{c} \qquad \mathcal{F}(\mathsf{P}, \mathsf{c}, \mathsf{f}) = \mathsf{t}}{\mathsf{P}, \Gamma \vdash_r \mathsf{e.f} : \mathsf{t}}$$

cond

$$\frac{\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{bool} \qquad \mathsf{P}, \Gamma \vdash_r \mathsf{e}_1 : \mathsf{t} \qquad \mathsf{P}, \Gamma \vdash_r \mathsf{e}_2 : \mathsf{t}}{\mathsf{P}, \Gamma \vdash_r \mathsf{if\ e\ then\ e}_1\ \mathsf{else\ e}_2 : \mathsf{t}}$$

Subsump

$$\frac{\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t} \qquad \mathsf{P} \vdash \mathsf{t} \leq \mathsf{t}'}{\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t}'}$$

fldAss

$$\frac{\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{c} \qquad \mathsf{P}, \Gamma \vdash_r \mathsf{e}' : \mathsf{t} \quad \mathcal{F}(\mathsf{P}, \mathsf{c}, \mathsf{f}) = \mathsf{t}}{\mathsf{P}, \Gamma \vdash_r \mathsf{e.f} := \mathsf{e}' : \mathsf{t}}$$

methCall

$$\frac{\mathsf{P}, \Gamma \vdash_r \mathsf{e}_0 : \mathsf{c} \qquad \mathsf{P}, \Gamma \vdash_r \mathsf{e}_1 : \mathsf{t}_1 \qquad \mathcal{M}(\mathsf{P}, \mathsf{c}, \mathsf{m}) = \mathsf{t\ m}(\ \mathsf{t}_1\ \mathsf{x})\ \{\ \mathsf{e}\ \}}{\mathsf{P}, \Gamma \vdash_r \mathsf{e}_0.\mathsf{m}(\ \mathsf{e}_1)\ : \mathsf{t}}$$

Properties of the system $\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t}$

Do the following properties hold?

- $\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t}$ and $\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t}' \implies \mathsf{t} = \mathsf{t}'$
- $\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t}$ and $\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t} \implies \mathsf{e} = \mathsf{e}'$
- $\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t} \implies \mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{t}$
- $\mathsf{P}, \Gamma \vdash \mathsf{e} : \mathsf{t} \implies \mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t}$
- $\mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t} \implies \mathsf{P}, \Gamma \vdash_s \mathsf{e} : \mathsf{t}$
- $\mathsf{P}, \Gamma \vdash_s \mathsf{e} : \mathsf{t} \implies \mathsf{P}, \Gamma \vdash_r \mathsf{e} : \mathsf{t}$

Well-formed class – revisited

$$
\mathit{ClssWF}(\mathsf{P},\mathsf{c}) \equiv \begin{cases}
\mathsf{P}(\mathsf{c})\downarrow_1 = \mathsf{c}' \text{ and } (\mathsf{c}' \neq \mathsf{Object} \implies \mathsf{c}' \in dom(\mathsf{P})\ ) \\
\text{and} \\
\forall \mathsf{f}:\ \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) = \mathsf{t} \implies \mathit{IsTyp}(\mathsf{P},\mathsf{t}) \text{ and } \mathcal{F}(\mathsf{P},\mathsf{c}',\mathsf{f}) \text{ undef.} \\
\text{and} \\
\forall \mathsf{m}:\ \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) = \mathsf{t}\ \mathsf{m}\textbf{(}\ \mathsf{t}_1\ \mathsf{x}\textbf{)}\ \ \textbf{\{}\ \mathsf{e}\ \textbf{\}} \implies \\
\qquad \mathit{IsTyp}(\mathsf{P},\mathsf{t}), \\
\qquad \text{and} \\
\qquad \mathit{IsTyp}(\mathsf{P},\mathsf{t}_1), \\
\qquad \text{and} \\
\qquad \mathsf{P}, \mathsf{t}_1\ \mathsf{x}, \mathsf{c}\ \mathsf{this} \vdash_r \mathsf{e}:\ \mathsf{t}, \\
\qquad \text{and} \\
\qquad \mathcal{M}(\mathsf{P},\mathsf{c}',\mathsf{m}) \text{ undef. or } \mathcal{M}(\mathsf{P},\mathsf{c}',\mathsf{m}) = \mathsf{t}\ \mathsf{m}\textbf{(}\ \mathsf{t}_1\ \mathsf{x}\textbf{)}\ \ \textbf{\{}\ \mathsf{e}'\ \textbf{\}}\ .
\end{cases}
$$

Other issues

$\mathcal{L}_2$ is a minimal formalization of classes, objects, imperative issues and inheritance.

We have not covered

- overloaded methods,
- field hiding,
- objects on the stack frame,
- C++ references,
- ....

All these can be expressed as variations of $\mathcal{L}_2$.

The expressive power of $\mathcal{L}_2$

Encoding Booleans

We can encode booleans in basic object oriented languages (and actually in $\mathcal{L}_1$ as well). Consider, namely:

```
class Boolean extends Object {
        Boolean and ( Boolean x) { ... }
        Boolean or ( Boolean x) { ... }
        Boolean not ( ) { ... }
        Object ifThenElse ( Object thenPart, Object elsePart) { ... }
}
```

```
class True extends Boolean{
        Boolean and ( Boolean x) { x }
        Boolean or ( Boolean x) { this }
        Boolean not ( ) { new False }
        Object ifThenElse ( Object thenPart, Object elsePart) { thenPart }
}

class False extends Boolean{
        Boolean and ( Boolean x) { this }
        Boolean or ( Boolean x) { x }
        Boolean not ( ) { new True }
        Object ifThenElse ( Object thenPart, Object elsePart) { elsePart }
}
```

With the above, we could express:

```
if ( ( true or ( false and true ) ) and aBoolean )
then 20
else 200
```

as

```
((new True.or( new False.and( new True) ) ).and( aBoolean) ).ifThenElse( 20, 200)
```

Encoding Natural Numbers

We can also encode natural numbers.

space for students' deliberations

Encoding Natural Numbers - 2

space for students' deliberations

WOW!

This demonstrates the power of the object paradigm!

Out of the imperative, the functional, the object oriented, and the logic paradigm, which ones can/cannot encode booleans and numbers?

Note: the Smalltalk environment contains booleans as described here; as with all environment classes, one can modify these classes, with interesting effects...

However...

Note that the previous is not a *complete* encoding of booleans and numbers. We are still unable to express ...