Part II

C+:

The Intricacies of C++-Like Multiple Inheritance

Joint work with Wasserrab, Snelting, Tip

What is C+?

C+ is not C++!

Jinja $\subset$ *C+* $\subset$ C++

Overview

- C++-like multiple inheritance
- A formal model of *subobjects*
- Examples
- Semantics and type system

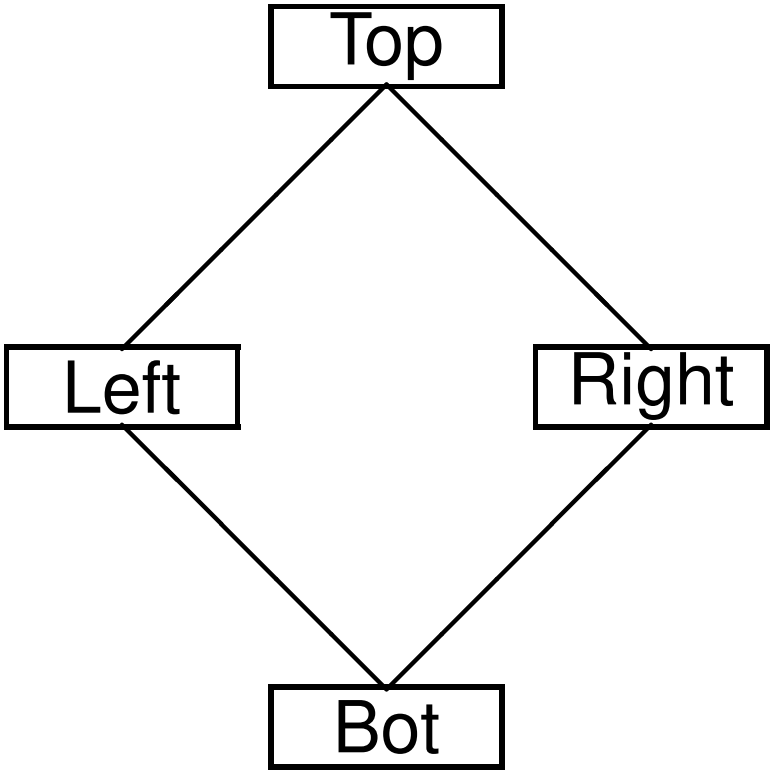
Why multiple inheritance and C++?

Not pretty but millions of lines of code out there:

- Want to understand them
- Want to port them automatically to safer languages

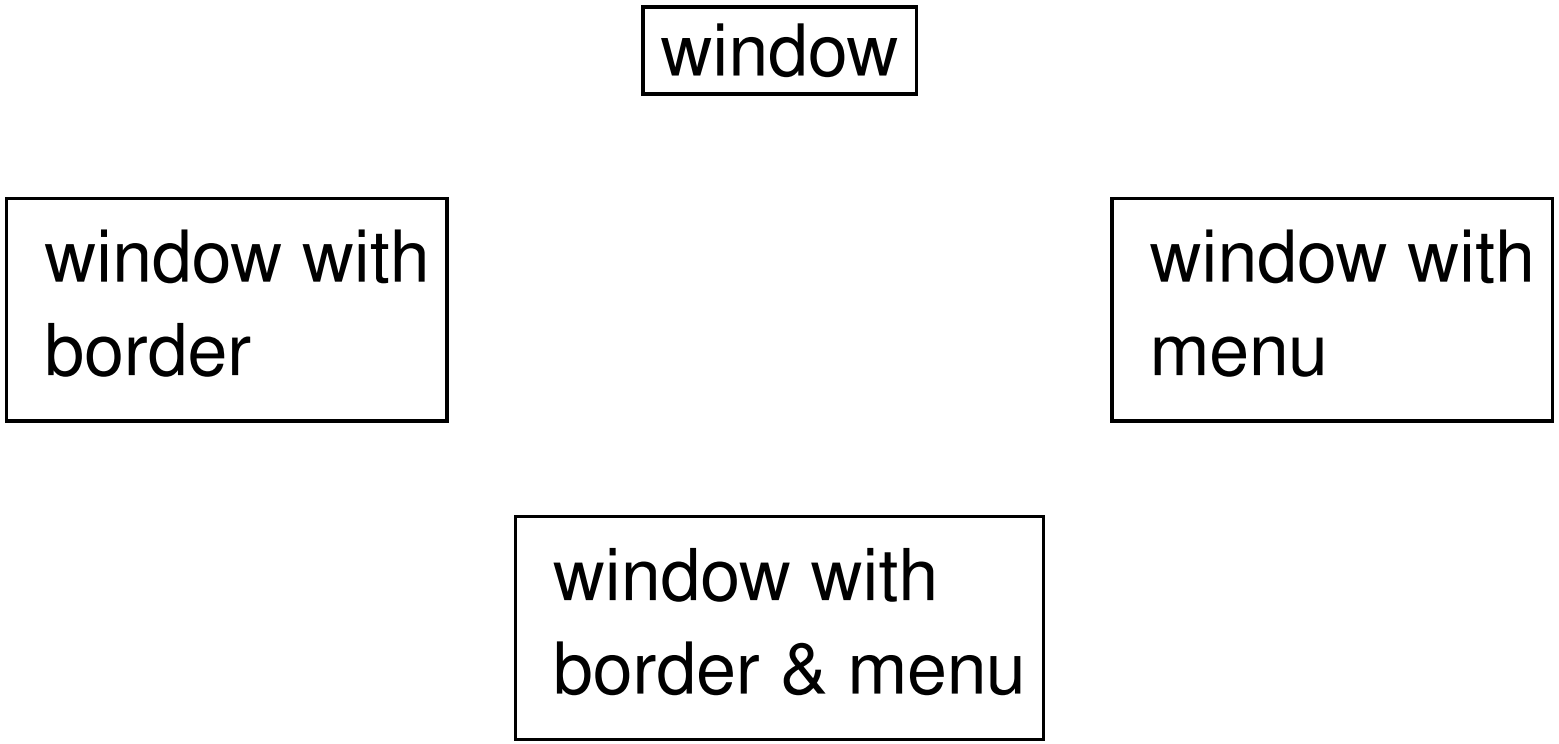
Informal definition of C++: pointers and tables (Stroustrup)
Challenge: abstract formal model

Multiple inheritance



1 or 2 instances of Top in Bot?

Shared inheritance

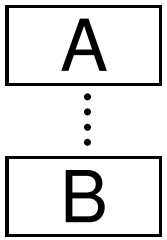
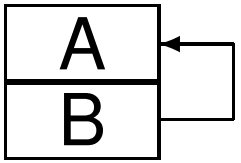
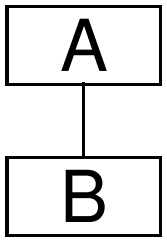
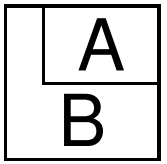


Repeated inheritance

Main motivation efficiency:

Avoid pointer indirection

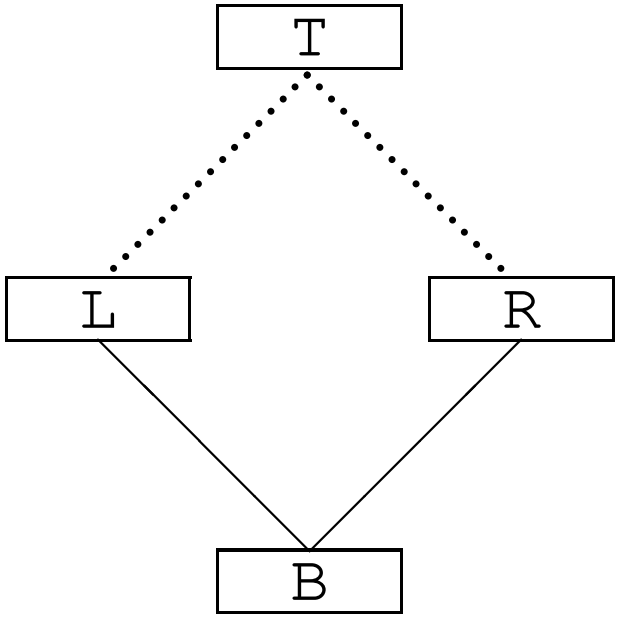
Multiple inheritance in C++

	C++ keyword	class diagram	object layout
shared:	`virtual`		
repeated:	`nonvirtual`		

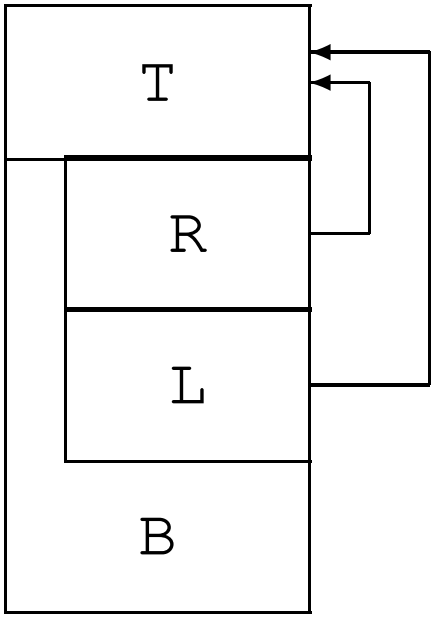
General form:

`class` C `repeats` A_1 `...` `shares` B_1 `...` `{...}`

The shared diamond

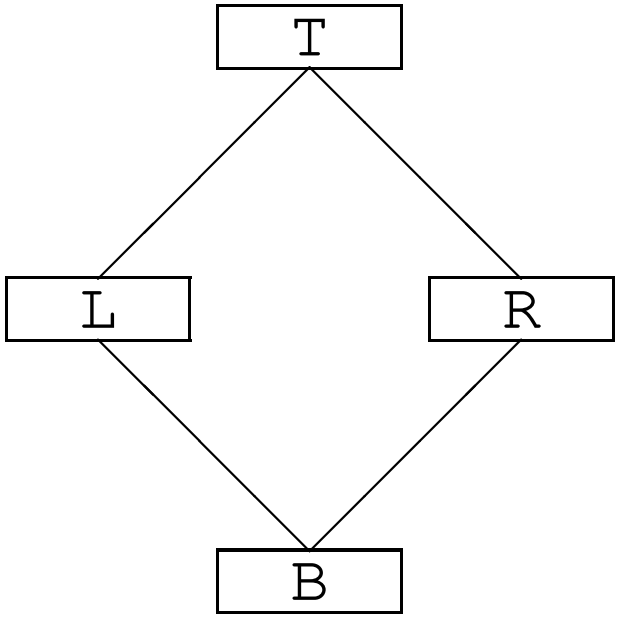


Class diagram

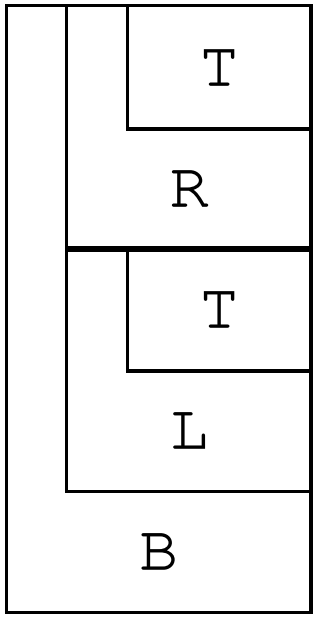


Object layout

The repeated diamond



Class diagram



Object layout

Alternative Java

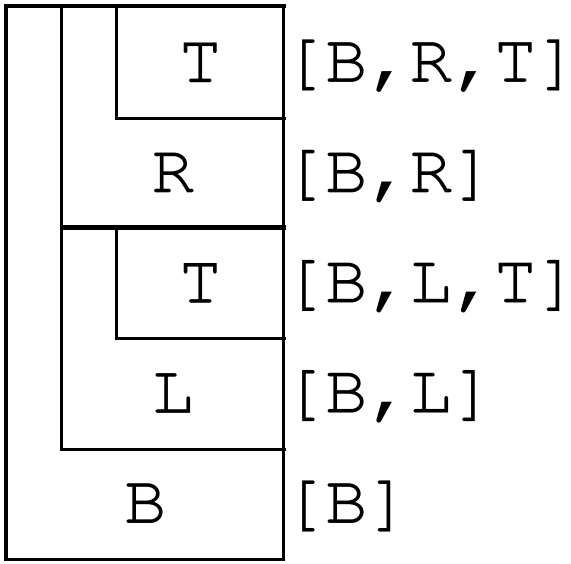
Multiple inheritance only for *interfaces*

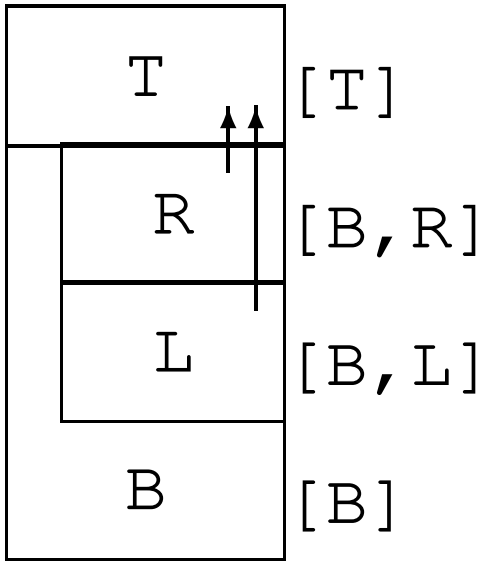
The Rossie-Friedman model of subobjects

©1995

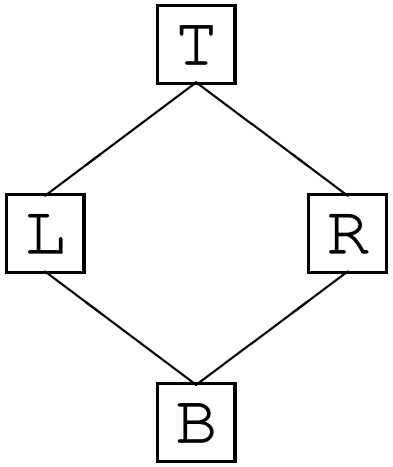
Paths

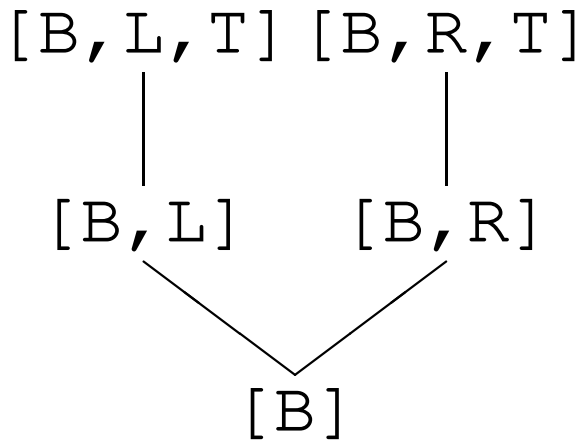
Identify (nested) subobjects by access path

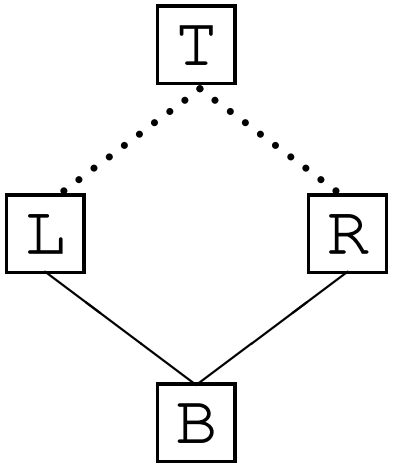


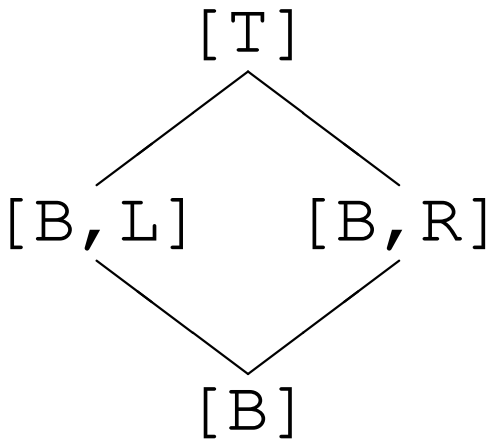


Unfolding repeated inheritance







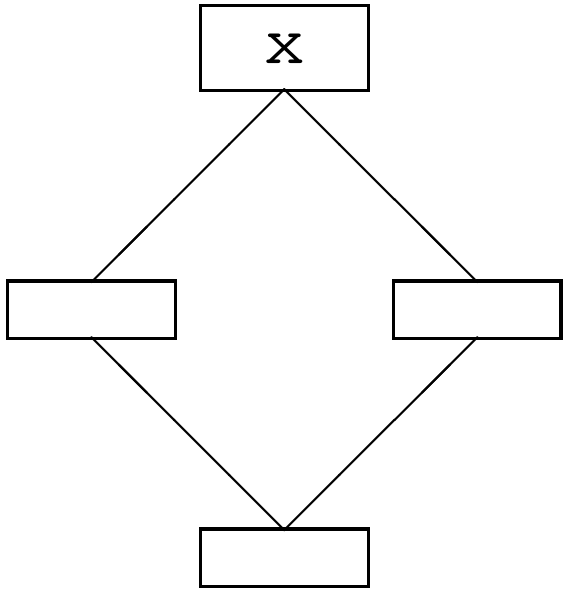


Paths are ordered

Examples

Is it ambiguous?

Cast



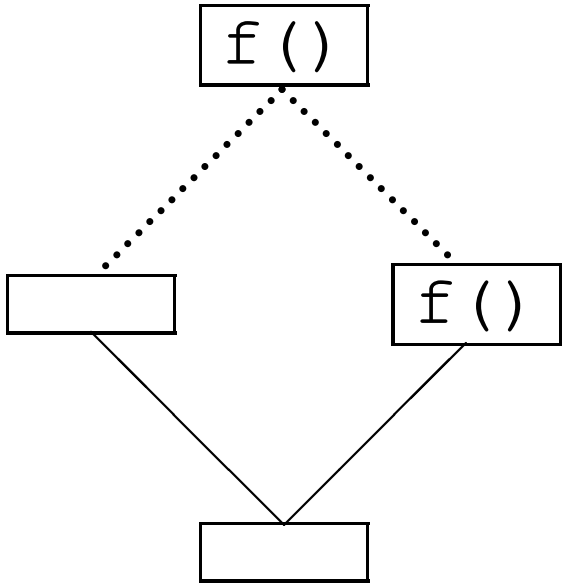

```
t:T;
t := (L) new B;
t.x


l:L;
l := new B;
l.x
```

Cast adjusts pointer

Assignment performs implicit cast

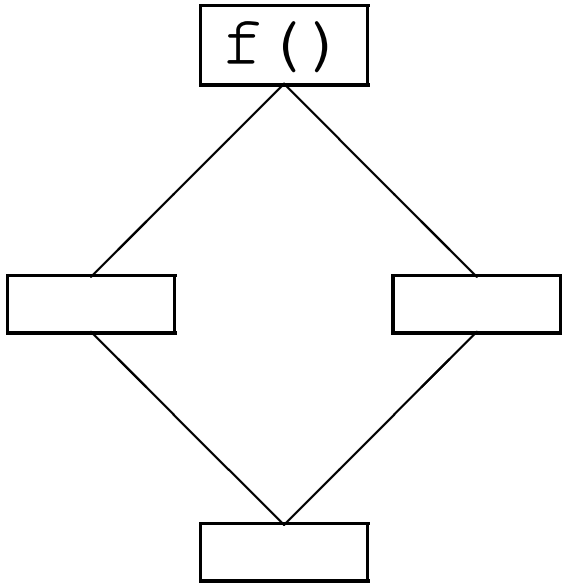
Dominance




```
l:L;
l := new B;
l.f()
```

`[B,R]` “dominates” `[T]`

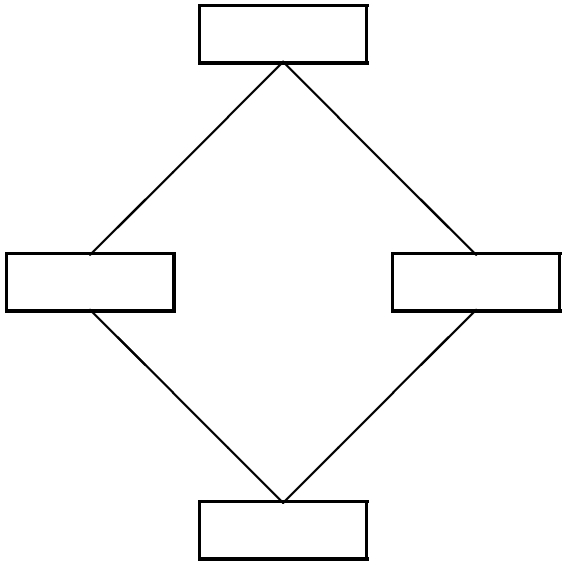
Method call




```
l:L;
l := new B;
l.f()
```

C++: static type of object may disambiguate method call

Legal?



```
b:B; b := new B;
l:L; l := b;
t:T; t := l;
(R)t
```

Statically allowed, but at run time ...

Semantic domains

Paths

$[C_1,\ldots,C_n]$ is a *path* in C if

- $C_1 \prec_R \ldots \prec_R C_n$ and
- $C = C_1$ or $\exists\, C'.\ C \preceq^* C' \prec_S C_1$.

Notation: $P \vdash Cs$ *path in* C

Objects

A B object in the shared diamond:

$$
\begin{array}{lll}
(B, & \{ & ([T], \quad [x \mapsto \ldots]), \\
 & & ([B,L], \quad \ldots), \\
 & & ([B,R], \quad \ldots), \\
 & & ([B], \quad \ldots) \\
 & \} & \\
) & &
\end{array}
$$

$$obj = cname \times subo\ set$$

$$subo = path \times (vname \rightharpoonup val)$$

$$obj \simeq cname \times (path \times vname \rightharpoonup val)$$

References

References must point to subobjects

Subobjects are identified by paths

$$\mathit{Addr}\ a \quad \text{(Jinja)} \qquad \rightsquigarrow \qquad \mathit{Ref}\ (a, Cs) \quad \text{(C+)}$$

$$ref(a,Cs) \equiv Val(Ref(a,Cs))$$

Intuition

If $e :: \mathit{Class}\ C$ and $e \Rightarrow \mathit{ref}\,(a,[C_1,\ldots,C_n])$
then $C = C_n$ (almost)

Semantics and type system

Cast

Field access

Method call

Semantics: Down cast

Shorten path:

$$\texttt{Cast}\ L\ (\mathit{ref}\ (a, [B, L, T])) \Rightarrow \mathit{ref}\ (a, [B, L])$$

$$\frac{P \vdash \langle e, s_0\rangle \Rightarrow \langle \mathit{ref}\ (a, Cs \mathbin{@} [C] \mathbin{@} _), s_1\rangle}{P \vdash \langle \texttt{Cast}\ C\ e, s_0\rangle \Rightarrow \langle \mathit{ref}\ (a, Cs \mathbin{@} [C]), s_1\rangle}$$

No down cast along shared inheritance
(could be ambiguous)

Semantics: Up cast

Extend path upwards:

$\texttt{Cast}\ L\colon [B] \mapsto [B, L]$

$\texttt{Cast}\ T\colon [B] \mapsto [T]$ shared $\diamond$

$\texttt{Cast}\ T\colon [B] \mapsto \texttt{THROW}\ \mathit{ClassCast}$ repeated $\diamond$

$$\frac{P \vdash \langle e, s_0\rangle \Rightarrow \langle \mathit{ref}\ (a, Cs), s_1\rangle \quad P \vdash \mathit{path}\ \mathit{last}\ Cs\ \mathit{to}\ C\ \mathit{via}\ Cs' \quad P \vdash \mathit{path}\ \mathit{last}\ Cs\ \mathit{to}\ C\ \mathit{unique}}{P \vdash \langle \texttt{Cast}\ C\ e, s_0\rangle \Rightarrow \langle \mathit{ref}\ (a, Cs\ @_p\ Cs'), s_1\rangle}$$

Path functions

$$P \vdash \textit{path } C \textit{ to } D \textit{ via } Cs \equiv P \vdash Cs \textit{ path in } C \wedge \textit{last } Cs = D$$

$$P \vdash \textit{path } C \textit{ to } D \textit{ unique} \equiv$$
$$\exists!Cs.\ P \vdash Cs \textit{ path in } C \wedge \textit{last } Cs = D$$

$$Cs \mathbin{@_p} Cs' \equiv \textit{if last } Cs = \textit{hd } Cs' \textit{ then } Cs \mathbin{@} \textit{tl } Cs' \textit{ else } Cs'$$

Cast exceptions

Invalid down or ambiguous up cast:

$$\frac{P \vdash \langle e, s_0 \rangle \Rightarrow \langle \mathit{ref}\ (a, Cs), s_1 \rangle \qquad C \notin \mathit{set}\ Cs \qquad \neg\ P \vdash \mathit{path}\ \mathit{last}\ Cs\ \mathit{to}\ C\ \mathit{unique}}{P \vdash \langle \texttt{Cast}\ C\ e, s_0 \rangle \Rightarrow \langle \texttt{THROW}\ \mathit{ClassCast}, s_1 \rangle}$$

$\texttt{Cast}\ B\ (\mathit{ref}\ (a, [T]))$ with $h\ a = \lfloor (B, \ldots) \rfloor$
is *not* a valid down cast

- `Cast` does not consult heap!
- models C++ (separate `dynamic_cast` operator)

C+ vs C++

C+ casts can throw exceptions

C++ casts simply "go wrong" ($\leadsto$ crash)
For safer casts use `dynamic_cast`

Type system: up cast

In repeated $\diamond$:

$$\texttt{Cast}\ T\ (\texttt{new}\ B)$$

$$\frac{P,E \vdash e :: \mathit{Class}\ D \qquad \mathit{is\text{-}class}\ P\ C \qquad P \vdash \mathit{path}\ D\ \mathit{to}\ C\ \mathit{unique}}{P,E \vdash \texttt{Cast}\ C\ e :: \mathit{Class}\ C}$$

Necessary but insufficient ambiguity test,
run time check still needed.

Type system: down cast

Down cast only along repeated inheritance chain

- Avoids some ambiguity:

  Repeated $\diamond$ + $T \prec_S C$:

  `Cast` T (`Cast` C (`new` B))

- Run time check still needed

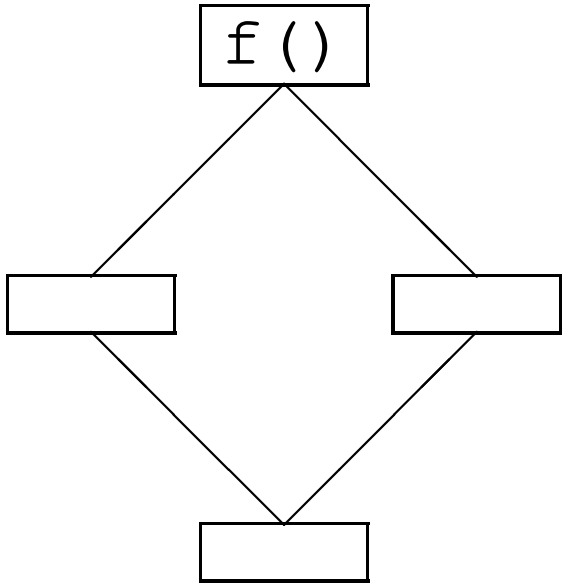
Semantics and type system

Cast

Method call

Field access

Remember?




```
l:L;
l := new B;
l.f()
```

C++: static type of object may disambiguate method call

C+: dynamic type *only* used for method call

Method call: semantics

$$\frac{\begin{array}{c} P \vdash \langle e, s_0 \rangle \Rightarrow \langle ref\ (a, Cs), s_1 \rangle \\ P \vdash \langle ps, s_1 \rangle\ [\Rightarrow]\ \langle map\ \texttt{Val}\ vs, (h_2, l_2) \rangle \\ h_2\ a = \lfloor (C, S) \rfloor \\ P \vdash C\ has\ least\ M = (Ts, T, pns, body)\ via\ Cs' \\ |vs| = |pns| \\ l_2' = [this \mapsto Ref\ (a, Cs'), pns\ [\mapsto]\ vs] \\ P \vdash \langle body, (h_2, l_2') \rangle \Rightarrow \langle e', (h_3, l_3) \rangle \end{array}}{P \vdash \langle e.M(ps), s_0 \rangle \Rightarrow \langle e', (h_3, l_2) \rangle}$$

The “dominance” ordering on paths

$Cs \sqsubseteq Ds$ if

- Cs is a prefix of Ds $\quad ([B] \sqsubseteq [B,R])$
- or $last\ Cs \preceq^* hd\ Ds$ $\quad ([B,R] \sqsubseteq [T])$

Ambiguous method call

If there is no least method definition . . .

`THROW` *MemberAmbigous*

Open problem:

Does disambiguation à la C++ obviate the need for *MemberAmbigous*?

Probably . . .

Method call: typing

$$\frac{P,E \vdash e :: Class\ C \qquad P \vdash C\ has\ least\ M = (Ts,\ T,\ _,\ _)\ via\ Cs \qquad P,E \vdash es\ [::]\ Ts}{P,E \vdash e.M(es) :: T}$$

- Leastness necessary to determine Ts and T
- Not sufficient to prevent dynamic ambiguity

Type system and semantics

Cast

Method call

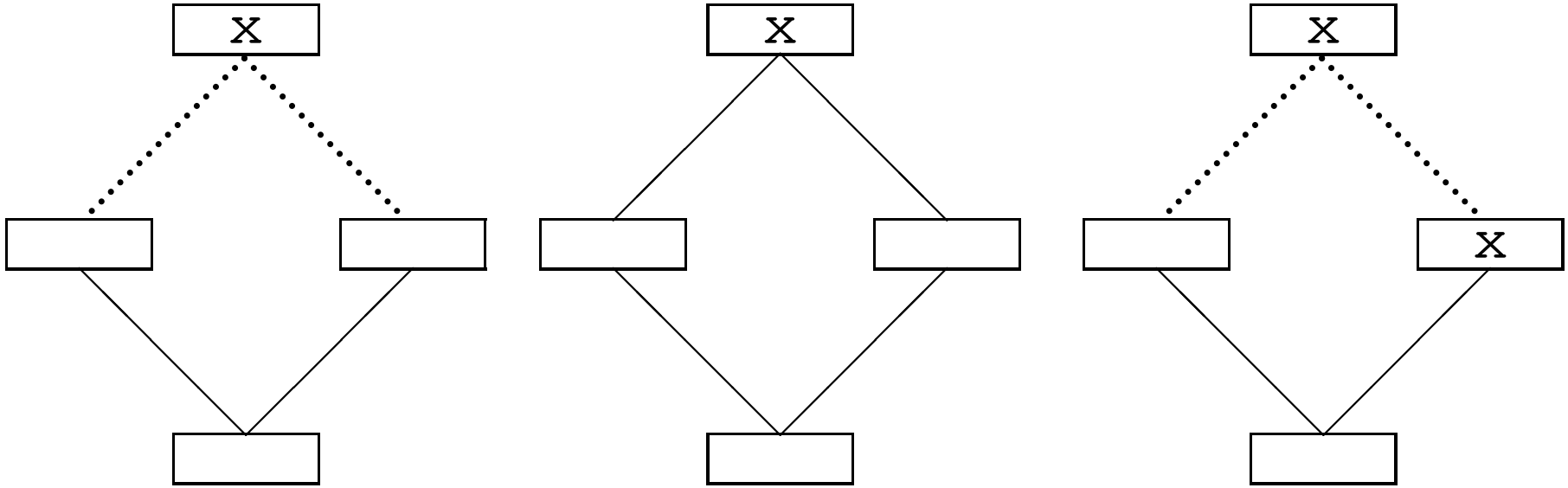
Field access

Field access: syntax

$$e.F\{Cs\}$$

Cs is path in static class of *e* to class declaring *F*

Field access can be statically ambiguous

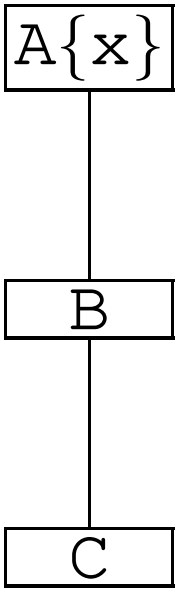


`new` *B*.*x*{[*T*]} `new` *B*.*x*{*?*} `new` *B*.*x*{[*B*, *R*]}

Field access: typing

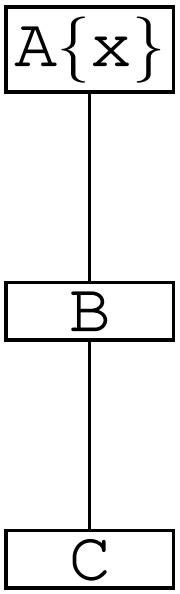
$$\frac{P,E \vdash e :: \mathit{Class}\ C \qquad P \vdash C\ \mathit{has least}\ F : T\ \mathit{via}\ Cs}{P,E \vdash e.F\{Cs\} :: T}$$

Field access semantics: the easy case



$$\texttt{Cast}\ B\ (\texttt{new}\ C).x\{[B, A]\}$$
$$\longrightarrow \texttt{Cast}\ B\ (\mathit{ref}\ (a, [C])).x\{[B, A]\}$$
$$\longrightarrow \mathit{ref}\ (a, [C, B]).x\{[B, A]\}$$
$$\rightsquigarrow [C, B, A]$$

A gap




```
class D                 { A f() {...} }
class E repeats D { C f() {...} }
```

$$
\begin{aligned}
&\texttt{Cast}\ D\ (\texttt{new}\ E).f([]).x\{[A]\} \\
\longrightarrow\ &\texttt{Cast}\ D\ (\mathit{ref}\ (e, [E])).f([]).x\{[A]\} \\
\longrightarrow\ &\mathit{ref}\ (e, [E, D]).f([]).x\{[A]\} \\
\longrightarrow\ &\mathit{ref}\ (c, [C]).x\{[A]\} \\
\rightsquigarrow\ &[C, B, A]
\end{aligned}
$$

The path from C to A must be unique

Method overriding: unique covariance

Wellformedness:

$$\begin{array}{l} \text{if} \quad M : Ts \to Class\ A \text{ in } C \\ \text{and } M : Us \to Class\ B \text{ in } D \\ \text{and } P \vdash D \preceq^* C \\ \text{then } P \vdash \textit{path } B \textit{ to } A \textit{ unique} \end{array}$$

Field access: semantics

$$\frac{P \vdash \langle e, s_0 \rangle \Rightarrow \langle \mathit{ref}\ (a, Cs'), (h, l) \rangle \qquad h\ a = \lfloor (D, S) \rfloor \qquad P \vdash \mathit{path}\ \mathit{last}\ Cs'\ \mathit{to}\ \mathit{hd}\ Cs\ \mathit{unique} \qquad P \vdash \mathit{path}\ \mathit{last}\ Cs'\ \mathit{to}\ \mathit{hd}\ Cs\ \mathit{via}\ Cs'' \qquad Ds = Cs' @_p Cs'' @_p Cs \qquad (Ds, fs) \in S \qquad fs\ F = \lfloor v \rfloor}{P \vdash \langle e.F\{Cs\}, s_0 \rangle \Rightarrow \langle \mathtt{Val}\ v, (h, l) \rangle}$$

Ambiguous field access

$$\frac{P \vdash \langle e, s_0 \rangle \Rightarrow \langle \mathit{ref}\ (a, Cs'), (h, l) \rangle \quad \neg\ P \vdash \mathit{path}\ \mathit{last}\ Cs'\ \mathit{to}\ \mathit{hd}\ Cs\ \mathit{unique}}{P \vdash \langle e.F\{Cs\}, s_0 \rangle \Rightarrow \langle \mathtt{THROW}\ \mathit{MemberAmbiguous}, (h, l) \rangle}$$

Open problem:

Can ambiguous gaps arise
in the context of unambiguous covariance?

We think not . . .

Type Safety

C+ is type safe

- Proof similar to Jinja
- New run time type system

C+ recognizes ambiguous and wrong casts

C+ may raise *MemberAmbiguous* exception

- Necessary?

From Jinja to C+: Reuse

- Could reuse much of Jinja ($\approx$ 60%)
- Many changes (from *C* to *Cs*)
- Only in retrospect easy

Conclusion

So far:

- First attempt at formalizing C++-like m.i.
- Proved type safety (of C+!)
- For details see `www.in.tum.de/~nipkow/pubs/`

To come:

- Remove *MemberAmbiguous*
- Relate C+ and C++ (as defined by Stroustrup)
- Remove m.i. by program transformation