CoreC++

Multiple Inheritance in C++

Tobias Nipkow

Technische Universität München
What is CoreC++?

Roughly speaking:

$$\text{Jinja} \subset \text{CoreC++} \subset \text{C++}$$

Intention:

CoreC++ models multiple inheritance exactly as in C++.

Warning:

CoreC++ lacks many C++ features, incl. overloading.
Overview

• Multiple inheritance in C++
• 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

- window
- window with border
- window with menu
- window with border & menu
Motivation:

Modelling?
Efficiency?
Multiple inheritance in C++

C++ keyword: virtual

shared: A

repeated: A

class diagram:

object layout:

A

B

A

B

A

B
The shared diamond

Class diagram

Object layout
The repeated diamond

Class diagram

Object layout
Multiple inheritance only for *interfaces*
Multiple inheritance in C++
A formal model of *subobjects*
Examples
Semantics and type system
The Rossie-Friedman model of subobjects
©1995
Identify (nested) subobjects by access path
Unfolding repeated inheritance

Paths are ordered
Outline

- Multiple inheritance in C++
- A formal model of *subobjects*
- Examples
- Semantics and type system
t: T;
t := (L) new B;
t.x

Cast adjusts pointer

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

Assignment performs implicit cast
What is called?

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

[B,R] “dominates” [T]
What is called?

Static type of object may disambiguate method call

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

l:L; l := b;

t:T; t := l;

(R)t

Statically allowed. But at run time ...
class A: \{f() : \text{Top} = \ldots\}\]
class B: A \{f() : \text{Bot} = \ldots\}\]

a : A;
a := \text{new} B;
a.f().x

\text{f() : Bot is illegal.}
Semantic domains
Paths

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

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

A $B$ object in the shared diamond:

$$(B, \{ ([T], [x \mapsto \ldots]), ([B,L], \ldots), ([B,R], \ldots), ([B], \ldots) \})$$

Better:

$$obj = \text{cname} \times \text{subo set}$$

$$\text{subo} = \text{path} \times (\text{vname} \rightarrow \text{val})$$

Better: $$obj = \text{cname} \times (\text{path} \times \text{vname} \rightarrow \text{val})$$
References

References must point to subobjects
Subobjects are identified by paths

\[ \text{Addr } a \quad \text{(Jinja)} \quad \sim \quad \text{Ref} \ (a, \ Cs) \quad \text{(CoreC++)} \]

\[ ref(a, Cs) \equiv Val(Ref(a, Cs)) \]
If $e :: \text{Class } C$ and $e \Rightarrow \text{ref}(a, [C_1, \ldots, C_n])$
then $C = C_n$
Path functions

\[ P \vdash \text{path } C \text{ to } D \text{ via } Cs \equiv P \vdash Cs \text{ path from } C \land \text{last } Cs = D \]

\[ P \vdash \text{path } C \text{ to } D \text{ unique} \equiv \]
\[ \exists ! Cs. P \vdash Cs \text{ path from } C \land \text{last } Cs = D \]

\[ Cs @_p Cs' \equiv \text{if last } Cs = \text{hd } Cs' \text{ then } Cs @ \text{tl } Cs' \text{ else } Cs' \]

Example:

In repeated diamond: \[ [B, L] @_p [L, T] = [B, L, T] \]
In shared diamond: \[ [B, L] @_p [T] = [T] \]
Semantics and type system

Assignment

Cast

Field access

Method call
Assignment: typing

\[ E \ V = \lfloor T \rfloor \quad P, E \vdash e :: T' \quad P \vdash T' \leq T \]

\[ P, E \vdash V := e :: T \]

where

\[ P \vdash \text{path } C \text{ to } D \text{ unique} \]

\[ P \vdash \text{Class } C \leq \text{Class } D \]

\[ P \vdash T \leq T \quad P \vdash NT \leq \text{Class } C \]
Assignment: semantics

\[ P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{Val } v, (h, l) \rangle \quad E \; V = \lfloor T \rfloor \]

\[ P \vdash T \text{ casts } v \text{ to } v' \quad l' = l(V \mapsto v') \]

\[ P, E \vdash \langle V := e, s_0 \rangle \Rightarrow \langle \text{Val } v', (h, l') \rangle \]

Examples:

\[ P \vdash \text{Integer casts } v \text{ to } v \]

\[ P \vdash \text{Class } T \text{ casts } \text{Ref} \ (b, [B, R]) \text{ to } \text{Ref} \ (b, [B, R, T]) \]

(repeated diamond)
Assignment: casting

\[
P \vdash \text{path last } Cs \text{ to } C \text{ via } Cs'
\]

\[
P \vdash \text{Class } C \text{ casts } \text{Ref (a, Cs)} \text{ to } \text{Ref (a, Cs } @_{p} Cs')
\]

\[
\forall C. \ T \neq \text{Class } C
\]

\[
P \vdash \ T \text{ casts } v \text{ to } v
\]

\[
P \vdash \text{Class } C \text{ casts } \text{Null to Null}
\]
Semantics and type system

Assignment

Cast

Field access

Method call
Three casts!

- C-style cast - unsafe
- Static cast - unsafe
- Dynamic cast - safe but complicated
Dynamic cast: typing

\[
P, E \vdash e :: \text{Class } D \quad \text{is-class } P \ C \\
\quad \quad P, E \vdash \text{dyn_cast } C \ e :: \text{Class } C
\]

Why potentially ok even if neither \( C \preceq^* D \) nor \( D \preceq^* C \)?
Dynamic up cast

Up cast extends path

shared diamond  repeated diamond

dyn_cast \( T (\text{new } B) \)  \( \Rightarrow \text{ref } (b, [T]) \)  \( \Rightarrow \text{null} \)

\[
P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref } (a, Cs), s_1 \rangle
\]

\[
P \vdash \text{path last Cs to C unique}
\]

\[
P \vdash \text{path last Cs to C via Cs'}
\]

\[
P, E \vdash \langle \text{dyn_cast } C e, s_0 \rangle \Rightarrow \langle \text{ref } (a, Cs @_p Cs'), s_1 \rangle
\]
Dynamic down cast (repeated)

Down cast shortens path

dyn_cast \( R \) \( (\text{ref} \ (b, \ [B, \ R, \ T])) \) \( \Rightarrow \) \( \text{ref} \ (b, \ [B, \ R]) \)

\[
P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, \ Cs \@\ [C] \@\ Cs'), s_1 \rangle
\]

\[
P, E \vdash \langle \text{dyn_cast} \ C \ e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, \ Cs \@\ [C]), s_1 \rangle
\]
Dynamic down cast (shared)

Wanted:
\[
dyn\_cast\ R\ (ref\ (b, [T]))) \Rightarrow ref\ (b, [B, R])
\]

Need to consult dynamic class of object!
Dynamic cross cast

In an all-shared diamond:

\[
dyn\_cast \ R (\text{ref} \ (b, \ [L])) \Rightarrow \text{ref} \ (b, \ [R])
\]

Need to consult dynamic class of object!
Dynamic cast (general)

\[ P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref} (a, Cs), (h, l) \rangle \]

\[ h a = \lfloor (D, S) \rfloor \]

\[ P \vdash \text{path } D \text{ to } C \text{ via } Cs' \]

\[ P, E \vdash \langle \text{dyncast } C e, s_0 \rangle \Rightarrow \langle \text{ref} (a, Cs'), (h, l) \rangle \]
All 3 rules are required

\[
P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, Cs), (h, l) \rangle \quad \text{h} \ a = \lfloor (D, S) \rfloor
\]

\[
P \vdash \text{path } D \text{ to } C \text{ via } Cs'
\]

\[
P, E \vdash \langle \text{dyn_cast } C \ e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, Cs'), (h, l) \rangle
\]

\[
P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, Cs), s_1 \rangle
\]

\[
P \vdash \text{path last } Cs \text{ to } C \text{ unique}
\]

\[
P \vdash \text{path last } Cs \text{ to } C \text{ via } Cs'
\]

\[
P, E \vdash \langle \text{dyn_cast } C \ e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, Cs @ p \ Cs'), s_1 \rangle
\]

\[
P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, Cs @ [C] @ Cs'), s_1 \rangle
\]

\[
P, E \vdash \langle \text{dyn_cast } C \ e, s_0 \rangle \Rightarrow \langle \text{ref} \ (a, Cs @ [C]), s_1 \rangle
\]
If all else fails

\[ P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref} (a, Cs), (h, l) \rangle \]

\[ h a = \lfloor (D, S) \rfloor \quad \neg P \vdash \text{path D to C unique} \]

\[ \neg P \vdash \text{path last Cs to C unique} \quad C \notin \text{set Cs} \]

\[ P, E \vdash \langle \text{dyn\_cast} C e, s_0 \rangle \Rightarrow \langle \text{null}, (h, l) \rangle \]
Type system and semantics

Assignment

Cast

Field access

Method call
Field access: syntax

\[ e.F\{Cs\} \]

\(Cs\) is path from static class of \(e\) to class declaring \(F\)
Typing examples

\[
\text{new } B.x\{[T]\} \text{ legal} \quad \text{new } B.x\{[\_]\} \text{ illegal}
\]
Field access: typing

\[
P, E \vdash e :: \text{Class } C \quad P \vdash C \text{ has least } F : T \text{ via } Cs
\]

\[
P, E \vdash e.F\{Cs\} :: T
\]

• Path Cs leads from C to unique lowest declaration of F.
Field access: semantics

\[ P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref } (a, Cs'), (h, l) \rangle \]

\[ h \ a = \lfloor (D, S) \rfloor \]

\[(Cs' \ @_p Cs, fs) \in S\]

\[ fs \ F = \lfloor v \rfloor \]

\[ P, E \vdash \langle e. F\{Cs\}, s_0 \rangle \Rightarrow \langle \text{Val } v, (h, l) \rangle \]
Semantics and type system

Assignment

Cast

Field access

Method call
Method call: typing

\[ P, E \vdash e :: \text{Class } C \]

\[ P \vdash C \text{ has least } M = (Ts, T, _, _) \text{ via } _\]

\[ P, E \vdash es [::] Ts' \quad P \vdash Ts' [\leq] Ts \]

\[ P, E \vdash e.M(es) :: T \]

- Must have unique lowest definition of \( M \)
Method call: semantics

\[ P, E \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{ref} (a, Cs), s_1 \rangle \]

\[ P, E \vdash \langle ps, s_1 \rangle \Rightarrow \langle \text{map val vs}, (h_2, l_2) \rangle \]

\[ h_2 \ a = \lfloor (C, \_ \_ \) \rfloor \]

P \vdash \text{last Cs has least } M = (Ts', T', pns', body') \text{ via } Ds

P \vdash (C, Cs @_p Ds) \text{ selects } M = (Ts, T, pns, body) \text{ via } Cs'

\vdots

\[ P, E \vdash \langle e. M(ps), s_0 \rangle \Rightarrow \langle e', (h_3, l_2) \rangle \]
Method selection

\[ P \vdash C \text{ has least } M = \text{mthd via } Cs' \]

\[ P \vdash (C, Cs) \text{ selects } M = \text{mthd via } Cs' \]

\[ \forall \text{mthd } Cs'. \neg P \vdash C \text{ has least } M = \text{mthd via } Cs' \]

\[ P \vdash (C, Cs) \text{ has overrider } M = \text{mthd via } Cs' \]

\[ P \vdash (C, Cs) \text{ selects } M = \text{mthd via } Cs' \]
Method overriding: unique covariance

Wellformedness:

if $M : Ts \rightarrow \text{Class A in C}$
and $M : Us \rightarrow \text{Class B in D}$
and $P \vdash D \preceq^\ast C$
then $P \vdash \text{path B to A unique}$
Type Safety

CoreC++ is type safe

Proof similar to Jinja