cardelli’s motivation

separate compilation
› vital for writing, delivering, maintaining libraries
› but often things go wrong
› module designers lost sight of original aims

linking has become complicated
› so develop a formal model to help reason
› treat linking outside the programming language

Luca Cardelli. Program Fragments, Linking & Modularization. POPL 1997
separate compilation goofs

- Class loading problems
- May link with old library by mistake
- Covariance in Eiffel
- Runtime type errors despite static types

- Overloading in Java (David Griswold)
- Overloading in Ada, Modules-3
- Templates in C++, Ada, Modules-3
- Can't compile library without client

- Global analysis required
- Multimethods, overloading, MT modules

- No interface/impl distinction, esp. in untyped languages
- Missing language features

Separate compilation goofs
our motivation

4

explore variants
check theorems automatically
might be simpler: no decl ordering
recast Cardelli's theory in Alloy
challenger: a nice final project
use as reference model for more complex systems
shows how to capture essence of a problem
nice example of simple theory
have much in common with Cardelli's model
will be presented on Wednesday by Finder
Units: A new modularity mechanism
Empty for a program, non-empty for a library
its imports and exports
A linkset has an external interface
An export
A list of imports
A name
Each fragment has
A linkset is a collection of fragments to be linked
Linksets: a simple configuration language
Linking is substitution (i.e., initializing)
Compilation is fragmenting bindings
Work at source code level
Focus on type checking: the hardest part

Key Ideas
A well-formed judgment \( \forall x \neq i \{ f(x) \} \) in the environment \( \forall x_1 : A_1, \ldots, x_n : A_n \) means that the expression \( f \) has type \( A \) in the environment \( E \). Examples include:

\[
\begin{align*}
g & : \text{int} \to \text{int} \\
\lambda x : \text{int} \to \text{int} \ {\{\text{return}\} \ x} & : \text{int} \to \text{int} \\
\end{align*}
\]

The judgments are:

- \( E \vdash A \)
Lambda calculus

- A toy language for theoretical investigation
- If you understand Scheme, you’ll understand it

Cardelli uses the variant ‘F1’: explicit, first-order types

Syntax

<table>
<thead>
<tr>
<th>Terms</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda x : A ) b</td>
<td>A \rightarrow B</td>
</tr>
<tr>
<td>( x )</td>
<td>A, B :: K</td>
</tr>
<tr>
<td>( a )</td>
<td>a :: x</td>
</tr>
<tr>
<td>( b (a) )</td>
<td>(a)</td>
</tr>
</tbody>
</table>
Type checking

\[
\begin{align*}
E \vdash x : E(x) & \quad \vdash E \\
E \vdash \lambda x : A \, p : A \rightarrow B & \quad \vdash E, x : A \vdash p : B, E' \\
E \vdash f(a) : B & \quad \vdash E, f : A \rightarrow B, E' \vdash a : A
\end{align*}
\]

... then \( f(a) \) has the type \( B \)
... and \( a \) has the type \( A \)
If \( f \) has the type \( A \rightarrow B \) (in environment \( E \))
Expressed as inference rules
begin lambda (x: int) x : int end
fragment main : int
begin f(3) : int end

f : int -> int
import f : int -> int
import nothing
fragment f : int -> int

represents these two fragments:
General form of a linkset $\mathcal{L}$

$\emptyset = \text{imp}(\mathcal{L}) \cup \text{exp}(\mathcal{L})$

$\text{dom } E \subseteq \text{exp}(\mathcal{L})$

$(E^0, E)$ is an environment

$\text{imports}(\mathcal{L})$ and $\text{exports}(\mathcal{L})$ are environments

Well-formed iff

$\emptyset \text{formed iff}$

$\{x^1, \ldots, x^n\} = \text{exports}(\text{exports}(\mathcal{L}))$

$\text{imp}(\mathcal{L}) = (E^0)$

$\text{exports names} \text{exports } \text{exports}(\mathcal{L})$

$\{x^1, \ldots, x^n\} = \text{exp}(\mathcal{L})$

$\text{exports names} \text{exp}(\mathcal{L})$

$(E^0) \text{dom}(E) = \text{imp}(\mathcal{L})$

Defined notions

$u^i : u^n \rightarrow u^n$

$u^i : u^n \rightarrow u^n \# x^i, A^i \rightarrow E^1 \# x^i, A^i \rightarrow E^1 | E^0 \equiv E^1$ General form of a linkset

Definitions
In this case, if each component is well-typed in isolation, the entire linkset is well-formed. Each component is well-formed if:

\[ E_0, E_1 \vdash a_1 : A_1 \]

General form of a linkset:

\[ E_0 \vdash E_1 \vdash \ldots \vdash E_n \]

Type checking (1)
type checking (2)

general form of a linkset \( \langle L \rangle \)

\[
E_0 | x_1 \quad \ldots \quad E_n | x_n : A_1, \ldots, x_n : A_n \quad \text{linkset is inter-checked}
\]

\[
\text{iff it's intra-checked } \quad \text{and } x \text{ is } x_j \quad \text{then } a \text{ is } A_j
\]

\[
\text{it's intra-checked } \quad \text{if } E_1 \text{ has the form } E_1, x : A, E
\]

\[
\text{could use subtyping instead } \quad \text{here requires exact match}
\]

\[
\text{type matching}
\]

\[
L \equiv E_0 | x_1 \quad E_1 | a_1 : A_1, \ldots, x_n : A_n \quad E_n | a_n : A_n
\]

General form of a linkset

(2) type checking
algorithm

so build bottom-up, and no cycles

Linking requires an empty environment

Note

Write $\mathcal{L}' \leftarrow \mathcal{L}$ for reflexive transitive closure

is a linking step

 escrit step

"\text{Let } \mathcal{L} \equiv \mathcal{L}' \text{ even when there are cycles.}"

(\mathcal{L}' \leftarrow \mathcal{L}')
Theorem 2.1: Given linksets L and L', if \( L > L' \) and \( L \) is linking soundness, then \( \text{inter-check}(L') \) and \( L' \) are compatible linksets.

Essentially, the algorithm implements maximally linking soundness and reduction soundness & completeness.

If \( \text{inter-check}(L) \) and \( L \) are compatible linksets, then \( L' \) and \( L' \) are compatible linksets.

Properties of Linking
a binding is like a linkset

compilation transforms a binding into a linkset

compilation

binding is like a linkset

client doesn't name the module itself

a module exports a binding

a simple fragment exports a value

Key Idea

modules

x : int

a signature is a list of declarations

x : int = 3

... x : int

a binding is a list of definitions

bindings & signatures
separate compilation

Key theorem

Given two modules whose interfaces are compatible, their compiled linksets are inter-checked.
Key Ideas

- Two phases
  - *Intra-checking:* Module is well typed
  - *Inter-checking:* Module interfaces match

Theory assumes:
- Modules are outermost (no hiding)
- Import/export types match exactly
- No mutual references

Linking criteria:
- Linking preserves well-formedness properties
- Easy to understand and predict behaviour
- Inter-checking: Module interfaces match
- Inter-checking: Module is well typed
- Allow partial linking order independent

- Intra-checking: Module is well typed
- Intra-checking: Module interfaces match
- Import/export types match exactly