March 11, 2002

6898: Advanced Topics in Software Design

MIT Lab for Computer Science

Daniel Jackson

Linker

Cardellis
Luca Cardelli. Program Fragments, Linking & Modularization. POPL 1997

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
separate compilation goofs

- may link with old library by mistake
- covariant in Eiffel
- runtime type errors despite static types
- can’t compile library without client
- multimethods, overloading, ML modules
- global analysis required
- no interface/implement distinction, esp. in untyped languages

overloading in Java (David Griswold)
- templates in C++, Ada, Modula-3
our motivation

- explore variants
- check theorems automatically
- might be simpler: no decl ordering
- recast Cardelli's theory in Alloy
  
  challenge: 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

Units: a new modularity mechanism

- will be presented on Wednesday by Findler
Focus on type checking; the hardest part

Key ideas
well formed if $x_1 \neq x_j$ for $i \neq j$

Examples

expression a has type A
in the environment $E$

the judgment $E \vdash a: A$ means

judgments
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

Terms

Variables

Abstractions

Function types

Base types

Types

\[ \text{application} \]

\[ (a) \]

\[ \text{abstraction} \]

\[ \text{variable} \]

\[ \lambda a : A \ b \]

\[ a \leftarrow b \]

\[ A \rightarrow B \]

\[ A, B : K \]

\[ x = : b \]

\[ \text{types} \]

\[ \text{syntax} \]
type checking

\[
\begin{align*}
\Gamma, \Delta \vdash x : E(x) \\
\hline
\vdash \Gamma
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash \lambda a : A . b : B, \quad \Delta \vdash a : A \\
\hline
\Gamma, \Delta \vdash \lambda a : A . b : B
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash f(a) : B, \quad \Delta \vdash a : A \\
\hline
\Gamma, \Delta \vdash f(a) : B
\end{align*}
\]

... then \( f(a) \) has the type \( B \)

... and \( a \) has the type \( A \)

... if \( f \) has the type \( A \rightarrow B \) (in environment \( \Gamma \))

expressed as inference rules
\begin{verbatim}
begin lambda (x: int) x end

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

f(3) : int
\end{verbatim}

represents these two fragments:

(main# (\empt : int \multimap \empt : int) (\empt# (\empt \multimap \empt \multimap \empt)) f# (\empt \multimap \empt) (\empt \multimap \empt) lambda (x: int) (\empt \multimap \empt)) \end{verbatim}

\textbf{Linksets}
\[ \emptyset = (\text{Imp}(L) \cup (\text{Exp}(L)) \setminus \text{dom } E \subseteq \text{Exp}(L) \setminus \text{dom } E \]

is an environment \( E \) if \( \text{imp}(L) = \text{dom } E \) and \( \text{exp}(L) = \{x_1, \ldots, x_n\} \) are environments

well-formed if

\[ \{x^1, \ldots, x^m\} = (\text{Exp}(L) \setminus \text{Exp}(L)) \]

\( E = (\text{Imp}(L) \setminus \text{Imp}(L)) \) and \( \{x^1, \ldots, x^m\} = (\text{Exp}(L) \setminus \text{Exp}(L)) \)

\( \text{dom } (E) = (\text{Exp}(L) \setminus \text{Exp}(L)) \)

defined notions

\[ \text{general form of a linkset} \]

\( L \equiv (E^1 \setminus \text{Exp}(L)) \setminus \text{Exp}(L) \)
Type checking (1)

General form of a linkset

\[ E_0, E_1, \ldots, E_n \vdash a_1 : A_1, \ldots, a_n : A_n \]

Linkset is intra-checked iff \( E_0 \) is well-formed

\[ \text{each fragment is well-typed in isolation} \]

\[ L \equiv E_0 \mid E_1 \vdash a_1 : A_1, \ldots, E_n \vdash a_n : A_n \]

Type checking (1)
type checking

general form of a linkset

\[
E_0 \mid x_1 \ldots \mid x_n \ldots \mid E_n
\]

linkset is inter-checked iff it's intra-checked

if \( E_i \) has the form \( E', x : A, E \)

then \( A \) is \( A' \)

and \( x \) is \( x' \)

if it's intra-checked

\[
E \equiv x^n_{E_1 \ldots E_n} \quad A' \quad A_1 \ldots A_n
\]

general form of a linkset

type checking (2)
or stuck (no further steps possible and some $E^1$ not empty).

Either linked (all environments except $E^0$ empty).

Keep doing linking steps until

Algorithm

so build bottom-up, and no cycles

Linking requires an empty environment

Note

Write $L \leftarrow L'$ for reflexive transitive closure

is a linking step

Let $L \equiv 0 \quad E^0 \quad (\forall x : \mathcal{A}, \forall X : \mathcal{A}, \forall \alpha : \langle \emptyset \rangle \# x \# \dot{E} \# \dot{E} \}, \forall \dot{E} \# \dot{E} \}$

Then $L \leftarrow L'$

Let $L \equiv 0 \quad E^0 \quad (\forall x : \mathcal{A}, \forall X : \mathcal{A}, \forall \alpha : \langle \emptyset \rangle \# x \# \dot{E} \# \dot{E} \}, \forall \dot{E} \# \dot{E} \}$

Linking steps
Properties of Linking

...
modules

a binding is like a linkset

compilation transforms a binding into a linkset

a binding is like a linkset

compilation

t : int

a signature is a list of declarations

... x : int = 3, ... 

a binding is a list of definitions

bindings & signatures

client doesn't name the module itself

a module exports a binding

a module exports a binding

a simple fragment exports a value

key idea

modules
key theorem
> given two modules whose interfaces are compatible
> their compiled linksets are inter-checked

separate compilation
Key ideas

- Intra-checking: module is well typed
- Inter-checking: module interfaces match

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

Linking preserves well-formedness properties
- Linking preserves well-formedness properties
- Allow partial linking, order independent
- Easy to understand and predict behaviour

Linking criteria

- Inter-checking: module interfaces match
- Intra-checking: module is well typed

Two phases