Softwaretechnik

Lecutre 05: Linking

Peter Thiemann

Universität Freiburg, Germany

SS 2009

Table of Contents

Live circle of a program

- approximate design of the architecture
- split in components / define the interfaces
- develop the components
- test the components
- *link* the components

Question

- What happens while linking a program?
- Is a program runnable after linking?

Question

- What happens while linking a program?
- Is a program runnable after linking?
- Request: Linking of program fragments with suitable interfaces yields to a type correct, runnable program
- Aim: model for that

Question

- What happens while linking a program?
- Is a program runnable after linking?
- Request: Linking of program fragments with suitable interfaces yields to a type correct, runnable program
- Aim: model for that
- Basics: Luca Cardelli. Program Fragments, Linking, and Modularization. In: Principles of Programming Languages POPL1997. S.266-277. ACM Press, 1997.

Goal of Cardelli's Work

- Problem: in some languages it is not possible to type check and compile a component separate from its use
- Examples: C++ Templates, Ada, Modula-3, Eiffel
- Goal:
  - Collect all requirements that enable type checking and compilation separate from usage
  - Interfaces are known
- Approach: compact model that only focuses on the problem

Scenario

One Module and its Use

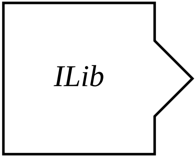
- In Tokyo a library module *Lib* will be developed
- In Stuttgart a program *Usr* will be developed
- The program *Usr* uses some functions from *Lib*.

One Module and its Use

- In Tokyo a library module *Lib* will be developed
- In Stuttgart a program *Usr* will be developed
- The program *Usr* uses some functions from *Lib*.
- Additional complexity: The programmers can only communicate through code and interfaces . . .

Day 1: Description of Library Modules

- The interface I_{Lib} will be published
- There is no implementation
- Reason: The program designer can start with their work
- Assumption: Interface descriptions exists that do not contain code



Day 1: Description of Library Modules

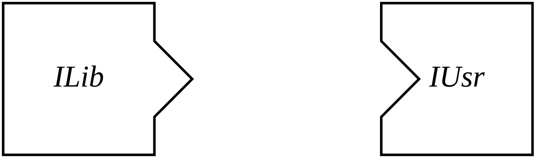
- The interface I_{Lib} will be published
- There is no implementation
- Reason: The program designer can start with their work
- Assumption: Interface descriptions exists that do not contain code

Problems

- Some languages do not distinguish interface and implementation
- "Small" and untyped languages do not support interfaces.
- Some language features require a whole program analysis (multimethods, overloading)

Day 2: Description of the Program

- The interface of the program I_{Usr} will be described.
- At first without implementation
- Reason: first focus on the design of the program and the interaction between it and *Lib*
- The interface I_{Usr} may use I_{Lib}



Day 2: Description of the Program

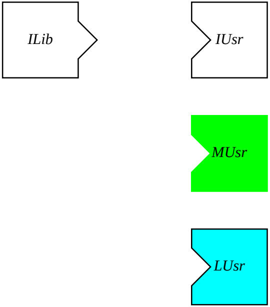
- The interface of the program I_{Usr} will be described.
- At first without implementation
- Reason: first focus on the design of the program and the interaction between it and *Lib*
- The interface I_{Usr} may use I_{Lib}

Problems

- I_{Lib} can define types that are used in the program
- Some languages do not allow such uses in interfaces

Day 3: Compiling the Program

- The program module M_{Usr} will be finished and compiled
- It is compatible to I_{Usr} and I_{Lib}.
- The compilation creates a linkable binary L_{Usr}
- no runnable program is created, because no implementation of I_{Lib} exists!



Day 3: Compiling the Program

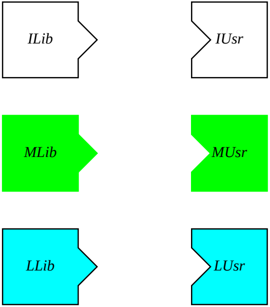
- The program module M_{Usr} will be finished and compiled
- It is compatible to I_{Usr} and I_{Lib}.
- The compilation creates a linkable binary L_{Usr}
- no runnable program is created, because no implementation of I_{Lib} exists!

Problems

- In some languages: Compilation of M_{Usr} without implementation of I_{Usr} is not possible
- Instantiation of generic modules can show errors in I_{Lib}
- Code of superclasses has to be checked another time.
- M_{Usr} may depend on the concrete implementation of I_{Lib}
- The information is not sufficient to create to linkable binary.

Day 4: Compiling the Library Module

- Creation of M_{Lib}, compatible to I_{Lib}
- Compiling creates a linkable binary L_{Lib}
- The combination (I_{Lib}, L_{Lib}) will be published in the repository.
- The source code of M_{Lib} remains secret.



Day 4: Compiling the Library Module

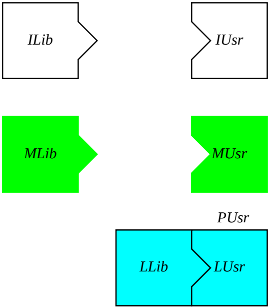
- Creation of M_{Lib}, compatible to I_{Lib}
- Compiling creates a linkable binary L_{Lib}
- The combination (I_{Lib}, L_{Lib}) will be published in the repository.
- The source code of M_{Lib} remains secret.

Problems

- Generic modules cannot be compiled without the client

Day 5: Linking of the Program

- The user acquires the linkable binary L_{Lib} that is suitable for I_{Lib}
- The user creates a program P_{Usr} by linking L_{Lib} with L_{Usr}.



Day 5: Linking of the Program

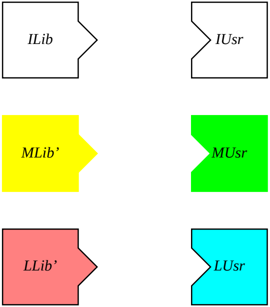
- The user acquires the linkable binary L_{Lib} that is suitable for I_{Lib}
- The user creates a program P_{Usr} by linking L_{Lib} with L_{Usr}.

Problems

- It may happen in some languages (Eiffel), that M_{Lib} is suitable for I_{Lib}, M_{Usr} is suitable for I_{Usr} and I_{Usr} is suitable for I_{Lib}, but P_{Usr} contains run-time errors.
- Some system perform their consistency checks while linking, because the error may happen in doing so.
- The function of the linked program should conform to the program that is the result of compiling the textual concatenation of all source code files.

Day 6: Further Development Implementing the Library

- A new implementation M'_{Lib} of I_{Lib} will be created
- In the repository L_{Lib} will be replaced by L'_{Lib}



Day 6: Further Development Implementing the Library

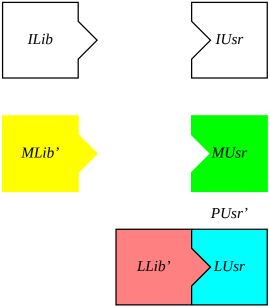
- A new implementation M'_{Lib} of I_{Lib} will be created
- In the repository L_{Lib} will be replaced by L'_{Lib}

Problems

- Changes in superclasses may cause recompilation of the program, even if no interfaces changed
- If the libraries in the repository depend on each other, the user may obtain inconsistent libraries.

Day 7: Relinking of the Program

- P_{Usr} is not up to date, but I_{Lib} is unchanged
- No recompiling of M_{Usr} necessary
- Current program P'_{Usr} results from linking L_{Usr} with L'_{Lib}.



Day 7: Relinking of the Program

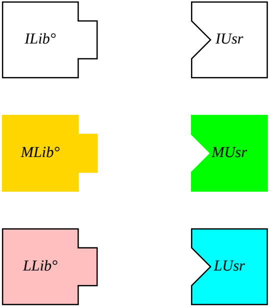
- P_{Usr} is not up to date, but I_{Lib} is unchanged
- No recompiling of M_{Usr} necessary
- Current program P'_{Usr} results from linking L_{Usr} with L'_{Lib}.

Problems

- Is the result the same as if we recompile M_{Usr}?
- In early Java versions, this was not the case.

Day 8: Evaluation of the Library

- The library's interface is revised to $I_{Lib}{}^{\circ}$
- For that we create an implementation of $M_{Lib}{}^{\circ}$ and compile it to $L_{Lib}{}^{\circ}$.
- The record $(I_{Lib}{}^{\circ}, L_{Lib}{}^{\circ})$ replaces (I_{Lib}, L'_{Lib}) in the repository



Day 8: Evaluation of the Library

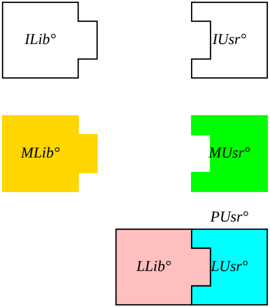
- The library's interface is revised to ${I_{Lib}}^\circ$
- For that we create an implementation of ${M_{Lib}}^\circ$ and compile it to ${L_{Lib}}^\circ$.
- The record $({I_{Lib}}^\circ, {L_{Lib}}^\circ)$ replaces (I_{Lib}, L'_{Lib}) in the repository

Problems

- inconsistent states of recursive dependencies

Day 9: Adaption of the Program

- the program is out of date, because $M_{Lib}{}^\circ$ is now available
- M_{Usr} and I_{Usr} does not fit with $I_{Lib}{}^\circ$.
- I_{Usr} will be changed into $I_{Usr}{}^\circ$
- The implementation of M_{Usr} will be changed int $M_{Usr}{}^\circ$, so that is is suitable to $I_{Lib}{}^\circ$. The it will be compiled to $L_{Usr}{}^\circ$.
- a new program $P_{Usr}{}^\circ$ is created by linking $L_{Usr}{}^\circ$ with $L_{Lib}{}^\circ$.



Day 9: Adaption of the Program

- the program is out of date, because ${M_{Lib}}^\circ$ is now available
- M_{Usr} and I_{Usr} does not fit with ${I_{Lib}}^\circ$.
- I_{Usr} will be changed into ${I_{Usr}}^\circ$
- The implementation of M_{Usr} will be changed int ${M_{Usr}}^\circ$, so that is is suitable to ${I_{Lib}}^\circ$. The it will be compiled to ${L_{Usr}}^\circ$.
- a new program ${P_{Usr}}^\circ$ is created by linking ${L_{Usr}}^\circ$ with ${L_{Lib}}^\circ$.

Problems

- If dependencies are not tracked correctly, L_{Usr} and ${L_{Lib}}^\circ$ or ${L_{Usr}}^\circ$ and L'_{Lib} may be linked together. This may yield an erroneous program.

Simple Linking

Program Fragments

- A *program fragment* is a program phrase (e.g. an expression) *with free variables*.
- *Separate compilation* means:
  - each program fragment can be type checked separately
  - code generation for each program fragment can done separately
- (for simplicity we ignore code generation)
- We need enough information in form of a type assumption A about absent program fragments
- A type judgment $A \vdash e : t$ yields the type of a program fragment e.

Programs

- An *complete program* is a program fragment without free variables.
- Program fragments can be linked together. The result can be a program fragment (with free variables).
- A *library* is a result of an incomplete linking.

A Configuration Language

- Input of the link process
  - Set of program fragments
  - rules to combine fragments
- cf. project files, Makefiles, etc.
- A *Linkset*

$$A_0 \mid x_1 \approx A_1 \vdash \mathcal{I}_1, \dots, x_n \approx A_n \vdash \mathcal{I}_n$$

  consists of
  - a type assumption A_0, the *external interface* (empty for a complete program)
  - judgments $A_i \vdash \mathcal{I}_i$, labeled with variable x_i, so that $A_0, A_i \vdash \mathcal{I}_i$ holds
- It's allowed to use the variables x_i in the other judgments.

Examples of Linksets

- A complete program

$$\emptyset \mid \mathit{main} \approx \emptyset \vdash 3 + 1 : \mathtt{int}$$

- *main* is closed (it contains no variables)

$\Rightarrow$ No linking needed

Examples for Linksets/2

- Two fragments

$$\begin{array}{ll} \emptyset \mid & y \approx \emptyset \vdash 17 : \texttt{int} \\ & \mathit{main} \approx y : \texttt{int} \vdash y + 4 : \texttt{int} \end{array}$$

- y is closed
- *main* needs the definition of y
- check type consistency : intramodular and intermodular

Examples for Linksets/2

- Two fragments

$$\begin{array}{ll} \emptyset \mid & y \approx \emptyset \vdash 17 : \mathtt{int} \\ & main \approx y : \mathtt{int} \vdash y + 4 : \mathtt{int} \end{array}$$

- y is closed
- *main* needs the definition of y
- check type consistency : intramodular and intermodular
- Goal: Linking using substitution

$$\begin{array}{ll} \emptyset \mid & y \approx \emptyset \vdash 17 : \mathtt{int} \\ & main \approx \emptyset \vdash (y + 4)[17/y] : \mathtt{int} \end{array}$$

Examples for Linksets/3

$$\emptyset \mid y \approx y : \mathtt{int} \vdash y + 1 : \mathtt{int}$$

Examples for Linksets/3

$$\emptyset \mid y \approx y : \texttt{int} \vdash y + 1 : \texttt{int}$$

- linking not possible, self reference

Examples for Linksets/3

$$\emptyset \mid y \approx y : \texttt{int} \vdash y + 1 : \texttt{int}$$

- linking not possible, self reference
- similar case

$$\begin{aligned} \emptyset \mid \quad & x \approx y : \texttt{int} \vdash y - 1 : \texttt{int} \\ & y \approx x : \texttt{int} \vdash x + 1 : \texttt{int} \end{aligned}$$

- linking not possible, cyclic dependencies not allowed

Examples for Linksets/3

$$\emptyset \mid y \approx y : \mathtt{int} \vdash y + 1 : \mathtt{int}$$

- linking not possible, self reference
- similar case

$$\begin{array}{ll} \emptyset \mid & x \approx y : \mathtt{int} \vdash y - 1 : \mathtt{int} \\ & y \approx x : \mathtt{int} \vdash x + 1 : \mathtt{int} \end{array}$$

- linking not possible, cyclic dependencies not allowed
- recursive dependencies/modules in realistic languages (Java) possible

Linking Lemma

If $A_1, x : t, A_3 \vdash \mathcal{I}$
and $A_1, A_2 \vdash e : t$
and $dom(x : t, A_3) \cap dom(A_2) = \emptyset$
then $A_1, A_2, A_3 \vdash \mathcal{I}[e/x]$.

Properties of Linksets

Access to Parts

For the structure

$$L \equiv A_0 \mid (x_i \approx A_i \vdash e_i : t_i)^{1 \leq i \leq n}$$

we define

$$\begin{array}{lll}
imp(L) = & dom(A_0) & \text{imported names} \\
exp(L) = & \{x_1, \ldots, x_n\} & \text{exported names} \\
names(L) = & imp(L) \cup exp(L) & \text{all names} \\
imports(L) = & A_0 & \text{import environment} \\
exports(L) = & x_1 : t_1, \ldots, x_n : t_n & \text{export environment}
\end{array}$$

Well Formed Linksets

Names are used consistently

A structure

$$L \equiv A_0 \mid (x_i \approx A_i \vdash e_i : t_i)^{1 \leq i \leq n}$$

is a linkset, *linkset*(L), if

- *imports*(L) and *exports*(L) defines each variable only once
- for all i holds : A_0, A_i defines each variable only once and $dom(A_i) \subseteq exp(L)$
  (a definition exists for each type assumption)
- $imp(L) \cap exp(L) = \emptyset$

Examples : Well Formed Linksets

Not well formed

$$\emptyset \mid (x \approx \emptyset \vdash 5 : \texttt{int})$$
$$(x \approx \emptyset \vdash 9 : \texttt{int})$$

$$y : \texttt{bool} \mid (x_1 \approx y : \texttt{bool} \vdash y : \texttt{bool})$$
$$(y \approx x_1 : \texttt{bool} \vdash !x_1 : \texttt{bool})$$

Well formed

$$z : \texttt{int} \mid (x_1 \approx x_2 : \texttt{int} \vdash z + x_2 : \texttt{int})$$
$$(x_2 \approx \emptyset \vdash z + z : \texttt{int})$$

$$z : \texttt{bool} \mid (x_1 \approx x_2 : \texttt{bool} \vdash z + x_2 : \texttt{int})$$
$$(x_2 \approx \emptyset \vdash z + z : \texttt{int})$$

Intramodular Consistent Linksets

A Structure

$$L \equiv A_0 \mid (x_i \approx A_i \vdash e_i : t_i)^{1 \leq i \leq n}$$

is a *intramodular consistent Linkset*, *intra-checked*(L), if

- *linkset*(L)
- for all i holds: $A_0, A_i \vdash e_i : t_i$ holds

Means, that each definition passes her type check

Example: Intramodular consistent Linksets

Not intramodular consistent

$$\begin{array}{r|l} \emptyset & (x \approx \emptyset \vdash 5 : \texttt{int}) \\ & (x \approx \emptyset \vdash 9 : \texttt{int}) \end{array}$$

$$\begin{array}{r|l} z : \texttt{bool} & (x_1 \approx x_2 : \texttt{int} \vdash z + x_2 : \texttt{int}) \\ & (x_2 \approx \emptyset \vdash z + z : \texttt{int}) \end{array}$$

Intramodular consistent

$$\begin{array}{r|l} z : \texttt{int} & (x_1 \approx x_2 : \texttt{int} \vdash z + x_2 : \texttt{int}) \\ & (x_2 \approx \emptyset \vdash z + z : \texttt{int}) \end{array}$$

$$\begin{array}{r|l} z : \texttt{int} & (x_1 \approx x_2 : \texttt{int} \vdash z + x_2 : \texttt{int}) \\ & (x_2 \approx \emptyset \vdash \texttt{false} : \texttt{bool}) \end{array}$$

Intermodular consistent Linksets

A structure

$$L \equiv A_0 \mid (x_i \approx A_i \vdash e_i : t_i)^{1 \le i \le n}$$

is a *intermodular consistent Linkset*, *inter-checked*(L), if

- *intra-checked*(L)
- for all $1 \le j, k \le n$: if $x_j : t \in A_k$, then $t = t_j$ holds

Example: Intermodular consistent Linksets

Not intermodular consistent

$$\begin{array}{rl} z : \texttt{bool} \mid & (x_1 \approx x_2 : \texttt{bool} \vdash z + x_2 : \texttt{int}) \\ & (x_2 \approx \emptyset \vdash z + z : \texttt{int}) \end{array}$$

$$\begin{array}{rl} z : \texttt{int} \mid & (x_1 \approx x_2 : \texttt{int} \vdash z + x_2 : \texttt{int}) \\ & (x_2 \approx \emptyset \vdash \texttt{false} : \texttt{bool}) \end{array}$$

Intermodular consistent

$$\begin{array}{rl} z : \texttt{int} \mid & (x_1 \approx x_2 : \texttt{int} \vdash z + x_2 : \texttt{int}) \\ & (x_2 \approx \emptyset \vdash z + z : \texttt{int}) \end{array}$$

Merging of Linksets

Preparation

$$A_0 \mid (x_i \approx A_i \vdash e_i : t_i)^{1 \le i \le n}$$

- The environment A_0 describes the absence definitions.
- Goal: a *full joined Linkset*, where A_0 and all type assumptions are empty
- We provide absence definitions by *merging linksets*
- auxiliary definitions
  - $A \setminus X$ removes from a type assumption A the binding for all variables from X
  - In $A \mid X$ only bindings for variables in X remains from A
  - compatible type assumptions: $A_1 \div A_2$, if for all $x \in dom(A_1) \cap dom(A_2)$ hold $A_1(x) = A_2(x)$.
  - Merging of two type assumptions: $A_1 + A_2 = A_1, (A_2 \setminus dom(A_1))$

Merging of two Linksets

Given two linksets

$$L \equiv A_0 \mid (x_i \approx A_i \vdash \mathcal{I}_i)^{1 \leq i \leq n}$$

$$L' \equiv A'_0 \mid (x'_i \approx A'_i \vdash \mathcal{I}'_i)^{1 \leq i \leq n'}$$

with *linkset*(L), *linkset*(L'), $exp(L) \cap exp(L') = \emptyset$. Then their *merging* $L + L'$ is defined by

$$\begin{aligned}(A_0 \setminus exp(L')) + (A'_0 \setminus exp(L)) \mid \quad & (x_i \approx A_0 \mid exp(L'), A_i \vdash \mathcal{I}_i)^{1 \leq i \leq n}, \\ & (x'_i \approx A'_0 \mid exp(L), A'_i \vdash \mathcal{I}'_i)^{1 \leq i \leq n'}\end{aligned}$$

Example: Merging of Linksets

Linkset L_{Lib} (library)

$$
\begin{array}{rl}
m : \mathtt{int} \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx s : \mathtt{int} \vdash m - s : \mathtt{int}) \\
& (h \approx s : \mathtt{int} \vdash m + s : \mathtt{int})
\end{array}
$$

Linkset L_{Usr} (program)

$$
\begin{array}{rl}
l : \mathtt{int}, h : \mathtt{int} \mid & (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Example: Merging of Linksets

Linkset L_{Lib} (library)

$$m : \mathtt{int} \mid \begin{array}{l} (s \approx \emptyset \vdash 30 : \mathtt{int}) \\ (l \approx s : \mathtt{int} \vdash m - s : \mathtt{int}) \\ (h \approx s : \mathtt{int} \vdash m + s : \mathtt{int}) \end{array}$$

Linkset L_{Usr} (program)

$$l : \mathtt{int}, h : \mathtt{int} \mid \begin{array}{l} (m \approx \emptyset \vdash 42 : \mathtt{int}) \\ (ok \approx m : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool}) \end{array}$$

Linkset $L_{Usr} + L_{Lib}$

$$\emptyset \mid \begin{array}{l} (s \approx \emptyset \vdash 30 : \mathtt{int}) \\ (l \approx m : \mathtt{int}, s : \mathtt{int} \vdash m - s : \mathtt{int}) \\ (h \approx m : \mathtt{int}, s : \mathtt{int} \vdash m + s : \mathtt{int}) \\ (m \approx \emptyset \vdash 42 : \mathtt{int}) \\ (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool}) \end{array}$$

Properties of Merging Linksets

Definition: Compatibility of Linksets

Two Linksets are compatible, $L \div L'$, if

- $exp(L) \cap exp(L') = \emptyset$ (linkset)

Linkset+Linkset = Linkset

If $linkset(L)$, $linkset(L')$ and $L \div L'$, then $linkset(L + L')$.

Properties of Merging Linksets

Definition: Compatibility of Linksets

Two Linksets are compatible, $L \div L'$, if

- $\mathit{exp}(L) \cap \mathit{exp}(L') = \emptyset$ (linkset)
- $\mathit{imports}(L) \div \mathit{imports}(L')$ (intra)

Keeps intramodular consistency

If $\mathit{intra\text{-}checked}(L)$, $\mathit{intra\text{-}checked}(L')$ and $L \div L'$, then $\mathit{intra\text{-}checked}(L + L')$.

Properties of Merging Linksets

Definition: Compatibility of Linksets

Two Linksets are compatible, $L \div L'$, if

- $\mathit{exp}(L) \cap \mathit{exp}(L') = \emptyset$ (linkset)
- $\mathit{imports}(L) \div \mathit{imports}(L')$ (intra)
- $\mathit{imports}(L) \div \mathit{exports}(L')$ (inter)
- $\mathit{imports}(L') \div \mathit{exports}(L)$ (inter)

Keeps intramodular consistency

If $\mathit{inter\text{-}checked}(L)$, $\mathit{inter\text{-}checked}(L')$ und $L \div L'$, then $\mathit{inter\text{-}checked}(L + L')$.

Linking

Definition: Linking Step

Given

$$L \equiv A_0 \mid \ldots, (x \approx \emptyset \vdash e : t), \ldots, (y \approx x : t', A' \vdash \mathcal{I}), \ldots$$

then L do one step to L', $L \rightsquigarrow L'$, with

$$L' \equiv A_0 \mid \ldots, (x \approx \emptyset \vdash e : t), \ldots, (y \approx A' \vdash \mathcal{I}[e/x]), \ldots$$

Example: Linking Step

Before: $L_{Usr} + L_{Lib}$

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx m : \mathtt{int}, s : \mathtt{int} \vdash m - s : \mathtt{int}) \\
& (h \approx m : \mathtt{int}, s : \mathtt{int} \vdash m + s : \mathtt{int}) \\
& (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Example: Linking Step

Before: $L_{Usr} + L_{Lib}$

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \texttt{int}) \\
& (l \approx m : \texttt{int}, s : \texttt{int} \vdash m - s : \texttt{int}) \\
& (h \approx m : \texttt{int}, s : \texttt{int} \vdash m + s : \texttt{int}) \\
& (m \approx \emptyset \vdash 42 : \texttt{int}) \\
& (ok \approx m : \texttt{int}, l : \texttt{int}, h : \texttt{int} \vdash (l < m) \&\& (m < h) : \texttt{bool})
\end{array}
$$

After:

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \texttt{int}) \\
& (l \approx s : \texttt{int} \vdash 42 - s : \texttt{int}) \\
& (h \approx m : \texttt{int}, s : \texttt{int} \vdash m + s : \texttt{int}) \\
& (m \approx \emptyset \vdash 42 : \texttt{int}) \\
& (ok \approx m : \texttt{int}, l : \texttt{int}, h : \texttt{int} \vdash (l < m) \&\& (m < h) : \texttt{bool})
\end{array}
$$

Example: Linking Step

Before: $L_{Usr} + L_{Lib}$

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx m : \mathtt{int}, s : \mathtt{int} \vdash m - s : \mathtt{int}) \\
& (h \approx m : \mathtt{int}, s : \mathtt{int} \vdash m + s : \mathtt{int}) \\
& (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Alternative:

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx m : \mathtt{int}, s : \mathtt{int} \vdash m - s : \mathtt{int}) \\
& (h \approx s : \mathtt{int} \vdash 42 + s : \mathtt{int}) \\
& (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Example: Linking Step

Before: $L_{Usr} + L_{Lib}$

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \texttt{int}) \\
& (l \approx m : \texttt{int}, s : \texttt{int} \vdash m - s : \texttt{int}) \\
& (h \approx m : \texttt{int}, s : \texttt{int} \vdash m + s : \texttt{int}) \\
& (m \approx \emptyset \vdash 42 : \texttt{int}) \\
& (ok \approx m : \texttt{int}, l : \texttt{int}, h : \texttt{int} \vdash (l < m) \&\& (m < h) : \texttt{bool})
\end{array}
$$

Alternative, substitute s

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \texttt{int}) \\
& (l \approx m : \texttt{int} \vdash m - 30 : \texttt{int}) \\
& (h \approx m : \texttt{int}, s : \texttt{int} \vdash m + s : \texttt{int}) \\
& (m \approx \emptyset \vdash 42 : \texttt{int}) \\
& (ok \approx m : \texttt{int}, l : \texttt{int}, h : \texttt{int} \vdash (l < m) \&\& (m < h) : \texttt{bool})
\end{array}
$$

Properties Preserved by Linking Steps

Linking steps preserve *linkset* property

If $\mathit{linkset}(L)$ and $L \leadsto L'$, then $\mathit{linkset}(L')$.

Linking steps preserve *inter-checked* property

If $\mathit{inter\text{-}checked}(L)$ and $L \leadsto L'$, then $\mathit{inter\text{-}checked}(L')$.

Comment

$\mathit{intra\text{-}checked}(L)$ is **not** preserved by linking steps.

Link Algorithm

The Linking step relation terminates and yields a unique result.

Every iteration removes one type assumption. Successful termination yields to a *full linked program*.

Example: Linking up to Full Linked Program

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx m : \mathtt{int} \vdash m - 30 : \mathtt{int}) \\
& (h \approx m : \mathtt{int}, s : \mathtt{int} \vdash m + s : \mathtt{int}) \\
& (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Example: Linking up to Full Linked Program

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx m : \mathtt{int} \vdash m - 30 : \mathtt{int}) \\
& (h \approx m : \mathtt{int} \vdash m + 30 : \mathtt{int}) \\
& (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Example: Linking up to Full Linked Program

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx \emptyset \vdash 42 - 30 : \mathtt{int}) \\
& (h \approx m : \mathtt{int} \vdash m + 30 : \mathtt{int}) \\
& (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Example: Linking up to Full Linked Program

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \mathtt{int}) \\
& (l \approx \emptyset \vdash 42 - 30 : \mathtt{int}) \\
& (h \approx \emptyset \vdash 42 + 30 : \mathtt{int}) \\
& (m \approx \emptyset \vdash 42 : \mathtt{int}) \\
& (ok \approx m : \mathtt{int}, l : \mathtt{int}, h : \mathtt{int} \vdash (l < m) \&\& (m < h) : \mathtt{bool})
\end{array}
$$

Example: Linking up to Full Linked Program

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \texttt{int}) \\
& (l \approx \emptyset \vdash 42 - 30 : \texttt{int}) \\
& (h \approx \emptyset \vdash 42 + 30 : \texttt{int}) \\
& (m \approx \emptyset \vdash 42 : \texttt{int}) \\
& (ok \approx l : \texttt{int}, h : \texttt{int} \vdash (l < 42) \&\& (42 < h) : \texttt{bool})
\end{array}
$$

Example: Linking up to Full Linked Program

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \texttt{int}) \\
& (l \approx \emptyset \vdash 42 - 30 : \texttt{int}) \\
& (h \approx \emptyset \vdash 42 + 30 : \texttt{int}) \\
& (m \approx \emptyset \vdash 42 : \texttt{int}) \\
& (ok \approx h : \texttt{int} \vdash ((42 - 30) < 42) \&\& (42 < h) : \texttt{bool})
\end{array}
$$

Example: Linking up to Full Linked Program

$$
\begin{array}{ll}
\emptyset \mid & (s \approx \emptyset \vdash 30 : \texttt{int}) \\
& (l \approx \emptyset \vdash 42 - 30 : \texttt{int}) \\
& (h \approx \emptyset \vdash 42 + 30 : \texttt{int}) \\
& (m \approx \emptyset \vdash 42 : \texttt{int}) \\
& (ok \approx \emptyset \vdash ((42 - 30) < 42) \&\& (42 < (42 + 30)) : \texttt{bool})
\end{array}
$$

No further step possible!

Easy Modules

Notation for Easy Modules

```
module M1 {
  import {}                                // import list
  export { x: int }                        // export list

  x: int = 3;                              // definitions
}

module M2 {
  import { x: int }
  export { y: int, z: boolean }

  y: int = 42 - x;
  z: boolean = y < 0;
}
```

Signatures and Bindings

Type Checking for Modules

Signatures S (typed export lists)

$$\textsc{Sig-Empty}\ \frac{}{A \vdash \emptyset} \qquad \textsc{Sig-X}\ \frac{A, x : t \vdash S}{A \vdash x : t, S}$$

Bindings d (modules)

$$\textsc{Bind-Empty}\ \frac{}{A \vdash \emptyset \mathrel{.\!\!\cdot\!\!.} \emptyset} \qquad \textsc{Bind-X}\ \frac{A, x : t \vdash d \mathrel{.\!\!\cdot\!\!.} S \quad A \vdash e : t}{A \vdash (x : t = e, d) \mathrel{.\!\!\cdot\!\!.} (x : t, S)}$$

Example: Two Modules

- module `M1`

$$\emptyset \vdash (x : \mathtt{int} = 3)$$
$$\therefore (x : \mathtt{int})$$

- module `M2`

$$x : \mathtt{int} \vdash (y : \mathtt{int} = 42 - x, z : \mathtt{bool} = y < 0)$$
$$\therefore (y : \mathtt{int}, z : \mathtt{bool})$$

Compiling a Binding Judgment from a Linkset

$|A \vdash d \therefore S|$ is the compilation of a valid judgment from one linkset.
Examples:

- module `M1` is a linkset

$$
\begin{array}{l}
|\emptyset \vdash (x : \mathtt{int} = 3) \therefore (x : \mathtt{int})| \\
= \\
\emptyset \mid (x \approx \emptyset \vdash 3 : \mathtt{int})
\end{array}
$$

- module `M2` is a linkset

$$
\begin{array}{l}
|x : \mathtt{int} \vdash (y : \mathtt{int} = 42 - x, z : \mathtt{bool} = y < 0) \therefore (y : \mathtt{int}, z : \mathtt{bool})| \\
= \\
x : \mathtt{int} \mid (y \approx \emptyset \vdash 42 - x : \mathtt{int})(z \approx y : \mathtt{int} \vdash y < 0 : \mathtt{bool})
\end{array}
$$

Properties of Compilation

Given $|A \vdash d \therefore S|$, a compilation of a valid judgment to a linkset.

Separate Compilation:

If $A \vdash d \therefore S$, then *inter-checked*$(|A \vdash d \therefore S|)$ holds.

Separate Compilation and Merging:

If $A \vdash d \therefore S$, $A' \vdash d' \therefore S'$ and $(A \vdash S) \div (A' \vdash S')$, then *inter-checked*$(|A \vdash d \therefore S| + |A' \vdash d' \therefore S'|)$.

Assertions about Separate Compilation

Conventions

For a module $M = A \vdash d \therefore S$, a linkset L and a linkset $|M|$ resulting from compilation of M, it holds:

- $valid(M)$: M is derivable, type checking of module M is successful
- $M \div M'$: module M and M' are type compatible
- $link(L) = L'$, if $L \leadsto^* L'$ and $L' \not\leadsto$

Statements about Separate Compilation

$$\textsc{Comp}\ \frac{\mathit{valid}(M)}{\mathit{inter\text{-}checked}(M)}$$

$$\textsc{Comp-Comp}\ \frac{\mathit{valid}(M) \quad \mathit{valid}(M') \quad M \div M'}{|M| \div |M'|}$$

$$\textsc{Link}\ \frac{\mathit{inter\text{-}checked}(L) \quad \mathit{link}(L) = L'}{\mathit{inter\text{-}checked}(L')}$$

$$\textsc{Link-Comp}\ \frac{\mathit{inter\text{-}checked}(L) \quad \mathit{inter\text{-}checked}(L') \quad L \div L' \quad \mathit{link}(L) = L''}{L'' \div L'}$$

$$\textsc{Merge}\ \frac{\mathit{inter\text{-}checked}(L) \quad \mathit{inter\text{-}checked}(L') \quad L \div L'}{\mathit{inter\text{-}checked}(L + L')}$$

Summary

- Linking formalized with substitution
- Under certain assumptions about modules and signatures the compiler can ensure that the linking does not fail, if each module is checked against the signatures of the imported modules.
- Recompilation is not necessary
- Order of linking steps and merging irrelevant
- $\exists$ further work extending these ideas (*e.g.*, for Java)