Softwaretechnik

Lecutre 05: Linking

Peter Thiemann

Universität Freiburg, Germany

SS 2008

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 programm runable after linking?

Question

- What happens while linking a program?
- Is a programm runable after linking?
- Request: Linking of prgram fracmants with suiteable interfaces yields to a type correct, runable program
- Aim: modell for that

Question

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

Goal of Cardellis Work

- Problem: in some Languages it is not possible to type check and compile components seperate from it's the usage
- Examples: C++ Templates, Ada, Modula-3, Eiffel
- Goal:
  - Count all Requirenments, that allows to type check and compile seperate from useage
  - interfaces are known
- Approchs : compact modell, that only focus on the problem

Szenario

One Modul and it's Usage

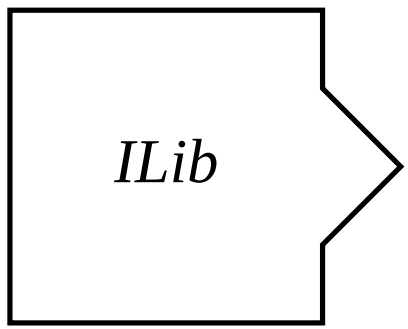
- In Toki a libary modul *Lib* will be developed
- In Stuttgart a program *Usr* will be developed
- The program *Usr* uses some functions from *Lib*.

One Modul and it's Usage

- In Toki a libary modul *Lib* will be developed
- In Stuttgart a program *Usr* will be developed
- The program *Usr* uses some functions from *Lib*.
- Additional complexity: The programmer can only comunicate throw code and interfaces ...

Day 1: Description of Libary Modules

- The interface I_{Lib} wil be published
- There is no implementation
- Reason: The program designer can start with their work
- Assumation: There exists interface descriptions, that does not contain code



Day 1: Description of Libary Modules

- The interface I_{Lib} wil be published
- There is no implementation
- Reason: The program designer can start with their work
- Assumation: There exists interface descriptions, that does not contain code

Problems

- Some languages do not split interface and implementation
- "Small" and untyped languages do not know interfaces.
- Some language criteria needs to analyse the hole program (multimethodes, overloading)

Day 2: Description of the Program

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

Day 2: Description of the Program

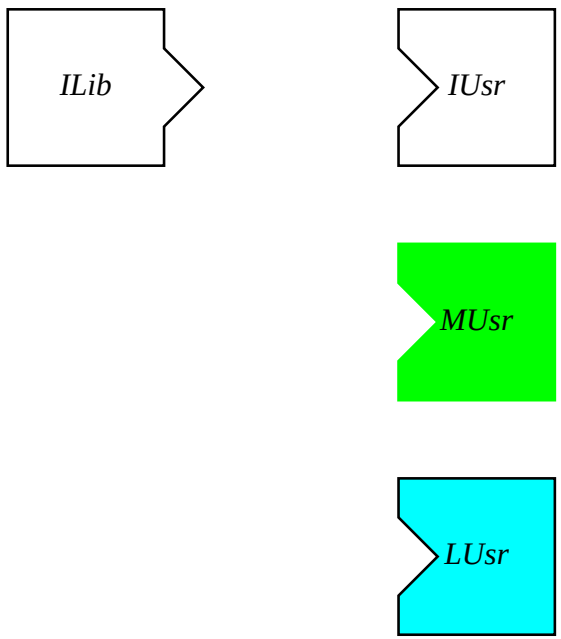
- The interface of the program I_{Usr} wil be described.
- At first without implementation
- Reason: first focus on the design of the program and the interaction beween it and *Lib*
- The interface I_{Usr} may use I_{Lib}

Problems

- I_{Lib} can define types, that are used in the program
- Some languaes does not allow this in interfaces

Day 3: Compiling the Program

- The program modul 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 runable program is created, because no implementation of I_{Lib} exists!



Day 3: Compiling the Program

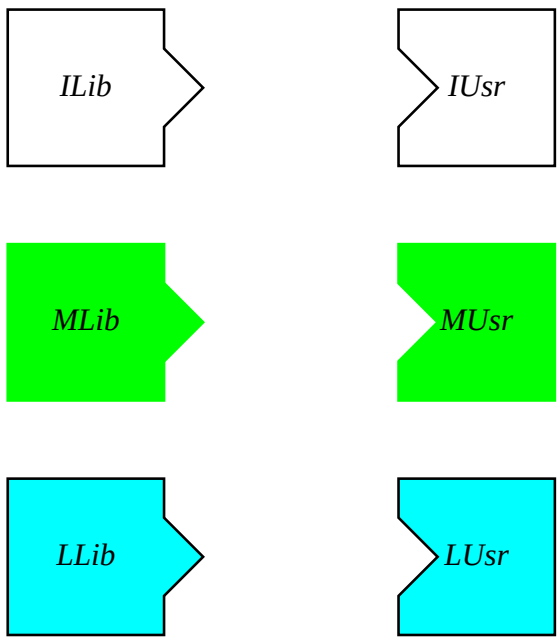
- The program modul M_{Usr} will be finished and compiled
- It is compatible to I_{Usr} and I_{Lib}.
- The compilation creates a linkable binary L_{Usr}
- no runable program is created, because no implementation of I_{Lib} exists!

Problems

- In some languages: Compiling of M_{Usr} without implementation of I_{Usr} is not possible
- Instatiation of generic moduls can show errors in I_{Lib}
- Code of super classes have to be checked another time.
- M_{Usr} can depends on the cocrete implementation of I_{Lib}
- The information is not enough to create to linkable binary.

Day 4: Compiling of the Libary Modul

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



Day 4: Compiling of the Libary Modul

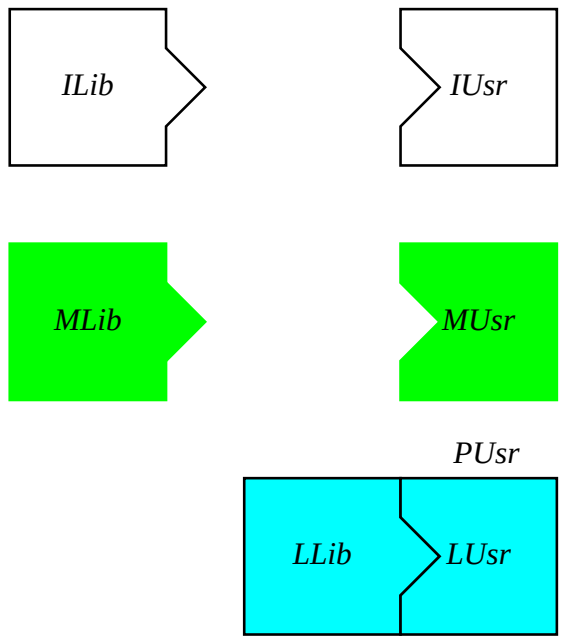
- Creation of M_{Lib}, compatible to I_{Lib}
- Compiling creates a linkable binary L_{Lib}
- The combination (I_{Lib}, L_{Lib}) will be publish in the repository.
- The source code of M_{Lib} stays secret.

Problems

- Generic moduls can not be compiled without the client

Day 5: Linking of the Program

- The user procures 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

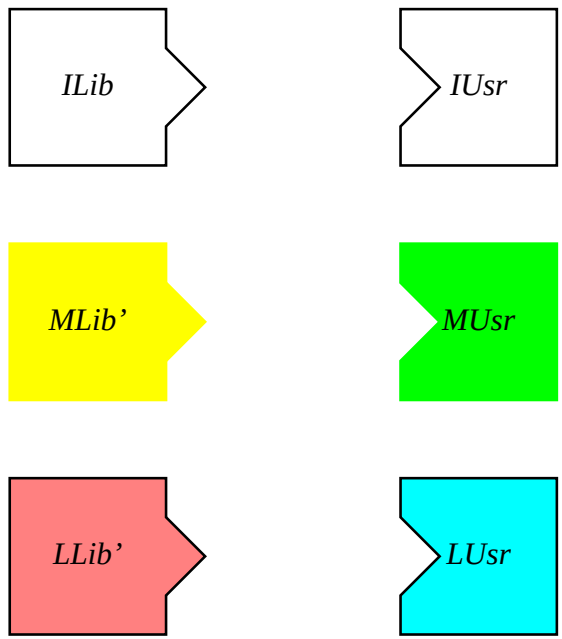
- The user procures the linkable binary L_{Lib} that is suitable for I_{Lib}
- The user creates a program $P_{Usr}$by linking L_{Lib} with L_{Usr}.

Problems

- It may happen in some languages (Eifel), 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 runtime erros.
- Some system do their consistency checks while linking, because the error may happen in doing so.
- The function of the linked program should conform to the progam that is the result of compiling the textual subsumption of all source code files.

Day 6: Futher Development Implementing the Libary

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



Day 6: Futher Development Implementing the Libary

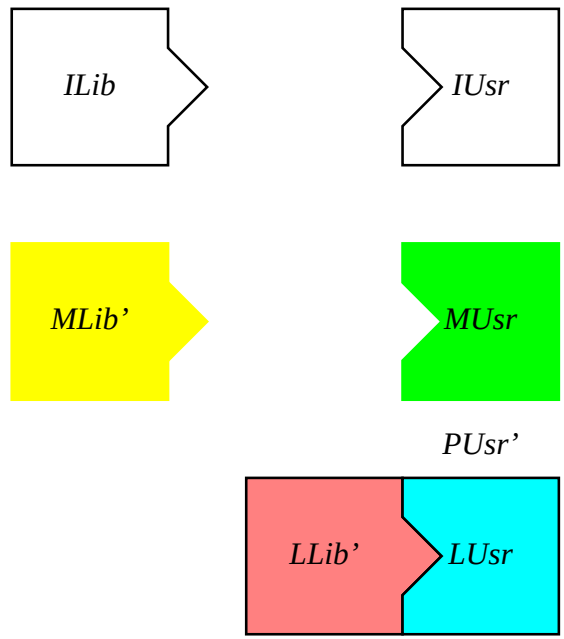
- A new implementation M'_{Lib} of I_{Lib} will be created
- In the repository M_{Lib} will be replaced by M'_{Lib}

Problems

- Changes in super classes may cause in recompiling the program, even if no interfaces changed
- If the libaries in the repository depends on each other, the user may obtain inconsitence libaries.

Day 7: Relinking of the Program

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



Day 7: Relinking of the Program

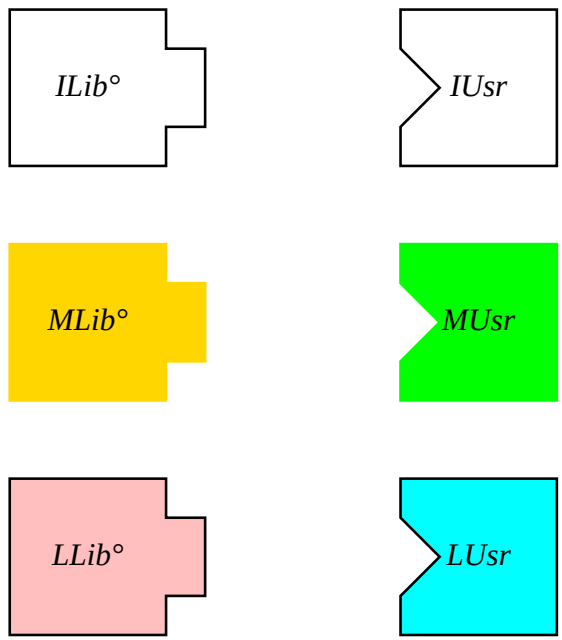
- P_{Usr} is not up to date, but I_{Lib} is unchanged
- No recompiling of M_{Usr} necessary
- aktuell program P'_{Usr} results from linking L_{Usr} with L'_{Lib}.

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 Libary

- The libaries interface revises 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 Libary

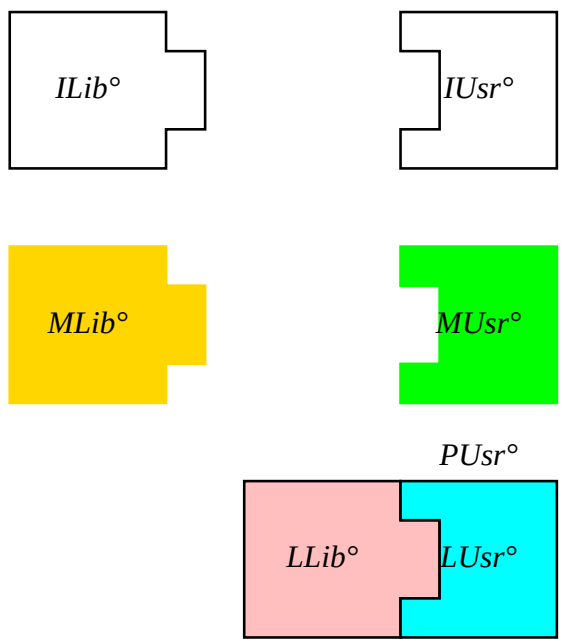
- The libaries interface revises to $I_{Lib}{}^{\circ}$
- For that we create an implementation of $M_{Lib}{}^{\circ}$ and compile it to $L_{Lib}{}^{\circ}$.
- The record $(I_{Lib}{}^{\circ}, L_{Lib}{}^{\circ})$ replaces (I_{Lib}, L'_{Lib}) in the repository

Problems

- inconsistent states on 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 to $I_{Lib}{}^{\circ}$.
- I_{Usr} will be changed into $I_{Usr}{}^{\circ}$
- The implemenation 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 to $I_{Lib}{}^{\circ}$.
- I_{Usr} will be changed into $I_{Usr}{}^{\circ}$
- The implemenation 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 correct, L_{Usr} and $L_{Lib}{}^{\circ}$ or $L_{Usr}{}^{\circ}$ and L'_{Lib} may linked together. This may yield to a erroneous program.

Simple Linking

Program Fragments

- A *program fragment* is a program phrase (e.g. an exprssion) *with free variables*.
- *Seperate compilation* means:
  - each program fragment can be type checked seperatly
  - code generation for each program fragment can done seperatly
- (for simplicity we ignore code generation)
- We need enough information in form of a type assumption A about absence program fragments
- A type judment $A \vdash e : t$ 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 *libary* is a result of an uncomplete linking.

A Configuration Language

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

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

  consists of
  - a type assamation 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

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

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

$\Rightarrow$ No linking needed

Examples for Linksets/2

- Two fragments

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

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

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 defintion of y
- check type consistency : intramodular and intermodular
- Goal: Linking using substitution

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

Examples for Linkssets/3

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

Examples for Linkssets/3

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

- linking not possible, self reference

Examples for Linkssets/3

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

- linking not possible, self reference
- simmilar case

$$\emptyset \mid \begin{array}{l} x \approx y : \texttt{int} \vdash y - 1 : \texttt{int} \\ y \approx x : \texttt{int} \vdash x + 1 : \texttt{int} \end{array}$$

- linking not possible, zyclic dependencies not allowed

Examples for Linkssets/3

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

- linking not possible, self reference
- simmilar case

$$\emptyset \mid \begin{array}{l} x \approx y : \texttt{int} \vdash y - 1 : \texttt{int} \\ y \approx x : \texttt{int} \vdash x + 1 : \texttt{int} \end{array}$$

- linking not possible, zyclic dependencies not allowed
- rekursive dependencies/modules in realisitic 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

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

Well Formed Linksets

Names are used consitent

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 assumantion)
- $imp(L) \cap exp(L) = \emptyset$

Examples : Well Formed Linkssets

Not well formed

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

$$y : \texttt{bool} \mid \begin{array}{l} (x_1 \approx y : \texttt{bool} \vdash y : \texttt{bool}) \\ (y \approx x_1 : \texttt{bool} \vdash !x_1 : \texttt{bool}) \end{array}$$

Well formed

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

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

Intramodular Consitent Linksets

A Structur

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

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

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

Means, that each defintion passes her type check

Example: Intramodular consistent Linksets

Not intramodular consistent

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

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

Intramodular consistent

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

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

Intermodular consistent Linksets

A structur

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

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

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

Example: Intermodular consitent Linksets

Not intermodular consistent

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

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

Intermodular consistent

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

Merging of Linksets

Preperation

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

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

Merging of two Linksets

Given two linkssets

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

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

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

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

Example: Merging of Linkssets

Linkset L_{Lib} (library)

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

Linkset L_{Usr} (program)

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

Example: Merging of Linkssets

Linkset L_{Lib} (library)

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

Linkset L_{Usr} (program)

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

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

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

Properties of Merging Linkssets

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 Linkssets

Definition: Compatibility of Linksets

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

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

Keeps intramodulare consistency

If *intra-checked*(L), *intra-checked*(L') and $L \div L'$, then *intra-checked*$(L + L')$.

Properties of Merging Linkssets

Definition: Compatibility of Linksets

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

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

Keeps intramodulare consistency

Wenn *inter-checked*(L), *inter-checked*(L') und $L \div L'$, dann *inter-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

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

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

Example: Linking Step

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

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

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

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

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

Alternative:

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

Example: Linking Step

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

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

Alternative, subsitute 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 $L \rightsquigarrow L'$, then $\mathit{linkset}(L')$.

Linking steps preserve *inter-checked* property

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

Comment

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

Link Algorithm

The Linking step relation terminates and yields a unique result.

Every interation removes one type assumation. Successfull termination yields to a *full linked program*.

Example: Linking up to Full Linked Program

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

Example: Linking up to Full Linked Program

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

Example: Linking up to Full Linked Program

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

Example: Linking up to Full Linked Program

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

Example: Linking up to Full Linked Program

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

Example: Linking up to Full Linked Program

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

Example: Linking up to Full Linked Program

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

No further step possible!

Easy Modules

Notation for easy Module

```
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;
}
```

Signeaturs and Bindings

Type check for odules

Signaturs S (typed export lists)

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

Bindings d (modules)

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

Example: Two Modules

- module `M1`

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

- module `M2`

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

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

$$|\emptyset \vdash (x : \texttt{int} = 3) \therefore (x : \texttt{int})|$$
$$=$$
$$\emptyset \mid (x \approx \emptyset \vdash 3 : \texttt{int})$$

- module `M2` is a linkset

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

Properties of Compilation

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

seperate Compilation:

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

seperate 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 over seperate Compilation

Konventions

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

- $\mathit{valid}(M)$: M is deducable, type check of module M is successful
- $M \div M'$: module M and M' are type compatible
- $\mathit{link}(L) = L'$, if $L \rightsquigarrow^* L'$ and $L' \not\rightsquigarrow$

Assertions over zspsdfs Compilation

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

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

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

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

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

Subsumption

- We formilize link process using substition
- Under certain Assumtions about modules and signaturs the compiler can ensure, that the linking does not fail, if each module is checked against the sigantures of the imported modules.
- recompiling is not necessary
- sequence of linking steps and merging negligible
- many more papers/work builds this basics (amongst others for Java)