Softwaretechnik

Vorlesung 05: Linking

Peter Thiemann

Universität Freiburg, Germany

SS 2008

Inhalt

Lebenslauf eines Programms

- Grober Entwurf der Architektur
- Aufteilung in Komponenten / Definition der Schnittstellen
- Entwicklung der Komponenten
- Komponententest
- Zusammensetzen der Komponenten (*Linking*)

Frage

- Was geschieht beim Linken?
- Ist das Programm nach dem Linken funktionsfähig?

Frage

- Was geschieht beim Linken?
- Ist das Programm nach dem Linken funktionsfähig?
- Wunsch: Zusammensetzen von Programmfragmenten mit passenden Schnittstellen liefert ein typkorrektes, funktionsfähiges Programm
- Ziel: Modell dafür

Frage

- Was geschieht beim Linken?
- Ist das Programm nach dem Linken funktionsfähig?
- Wunsch: Zusammensetzen von Programmfragmenten mit passenden Schnittstellen liefert ein typkorrektes, funktionsfähiges Programm
- Ziel: Modell dafür
- Grundlage: Luca Cardelli. Program Fragments, Linking, and Modularization. In: Principles of Programming Languages POPL1997. S.266-277. ACM Press, 1997.

Ziel von Cardellis Arbeit

- Problem: in manchen Sprachen ist es nicht möglich, die Typprüfung und die Übersetzung einer Software-Komponente getrennt von ihren Verwendungen durchzuführen
- Beispiele: C++ Templates, Ada, Modula-3, Eiffel
- Ziel:
  - Zusammenstellen von Voraussetzungen, sodass Typprüfung (und Compilierung) getrennt durchführbar ist.
  - Interfaces dürfen/müssen bekannt sein
- Ansatz: kompaktes Modell, das möglichst nur das Problem betrachtet

Szenario

Ein Modul und seine Verwendung

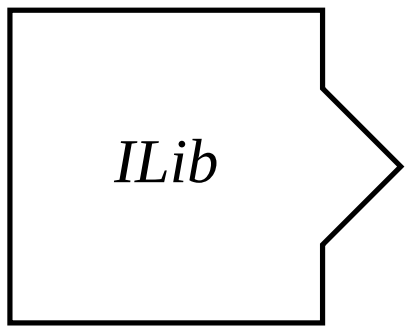
- In Tokyo wird ein Bibliotheksmodul *Lib* entwickelt.
- In Stuttgart wird ein Programm *Usr* entwickelt.
- Das Programm *Usr* verwendet die Funktionalität von *Lib*.

Ein Modul und seine Verwendung

- In Tokyo wird ein Bibliotheksmodul *Lib* entwickelt.
- In Stuttgart wird ein Programm *Usr* entwickelt.
- Das Programm *Usr* verwendet die Funktionalität von *Lib*.
- Zusätzliche Schwierigkeit: Die Entwickler können sich nur per Code / Schnittstellen verständigen...

Tag 1: Beschreibung des Bibliotheksmoduls

- Die Schnittstelle I_{Lib} wird veröffentlicht.
- Es gibt noch keine Implementierung.
- Grund: Anwendungsentwicklung kann beginnen.
- Annahme: Es gibt Schnittstellenbeschreibungen, die keinen Code enthalten



Tag 1: Beschreibung des Bibliotheksmoduls

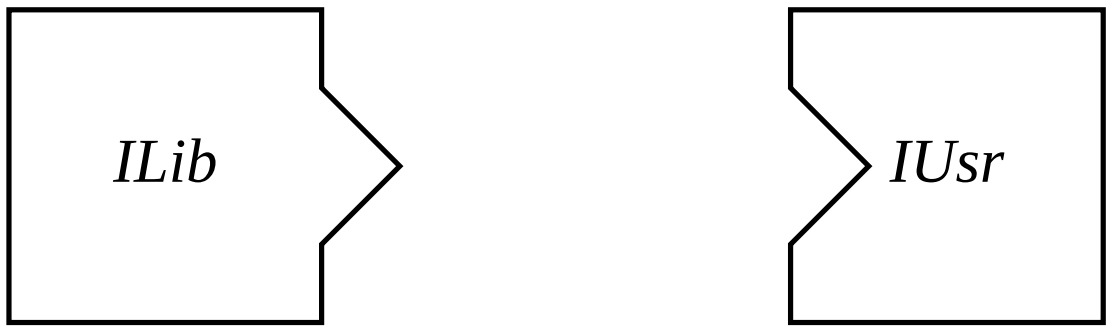
- Die Schnittstelle I_{Lib} wird veröffentlicht.
- Es gibt noch keine Implementierung.
- Grund: Anwendungsentwicklung kann beginnen.
- Annahme: Es gibt Schnittstellenbeschreibungen, die keinen Code enthalten

Schwierigkeiten

- Manche Programmiersprachen trennen nicht zwischen Interface und Implementierung.
- "Kleine" und ungetypte Sprachen kennen oft keine Interfaces.
- Manche Sprachmerkmale erfordern Analyse des ganzen Programms (Multimethoden, Überladung)

Tag 2: Beschreibung der Anwendung

- Das Anwendungsinterface I_{Usr} wird geschrieben.
- Zunächst ohne Implementierung.
- Grund: Entwurf der Anwendung und ihrer Interaktion mit *Lib*
- Interface I_{Usr} kann I_{Lib} verwenden.



Tag 2: Beschreibung der Anwendung

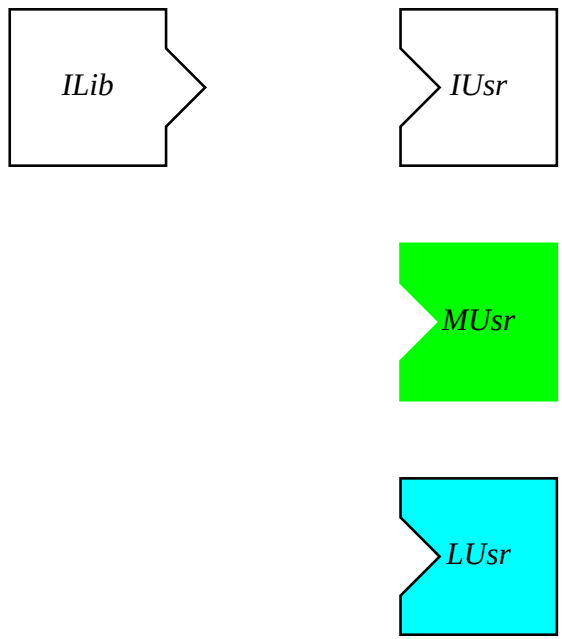
- Das Anwendungsinterface I_{Usr} wird geschrieben.
- Zunächst ohne Implementierung.
- Grund: Entwurf der Anwendung und ihrer Interaktion mit *Lib*
- Interface I_{Usr} kann I_{Lib} verwenden.

Schwierigkeiten

- I_{Lib} kann Typen definieren, die von der Anwendung benutzt werden.
- Manche Sprachen erlauben dies nicht in Schnittstellen.

Tag 3: Übersetzen der Anwendung

- Das Anwendungsmodul M_{Usr} wird fertiggestellt und übersetzt.
- Es ist kompatibel mit I_{Usr} und mit I_{Lib}.
- Die Übersetzung erzeugt ein linkbares Binary L_{Usr}
- Es gibt kein lauffähiges Programm, da keine Implementierung von I_{Lib} vorliegt!



Tag 3: Übersetzen der Anwendung

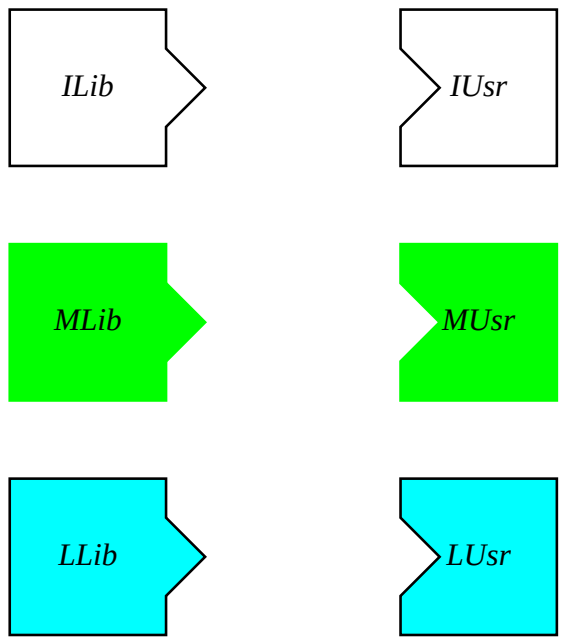
- Das Anwendungsmodul M_{Usr} wird fertiggestellt und übersetzt.
- Es ist kompatibel mit I_{Usr} und mit I_{Lib}.
- Die Übersetzung erzeugt ein linkbares Binary L_{Usr}
- Es gibt kein lauffähiges Programm, da keine Implementierung von I_{Lib} vorliegt!

Schwierigkeiten

- In manchen Sprachen: Compilierung von M_{Usr} ohne Implementierung von I_{Usr} unmöglich.
- Die Instantiierung von generischen Modulen kann in einer Implementierung von I_{Lib} Fehler aufdecken.
- Code von Superklassen muss ggf. erneut überprüft werden.
- M_{Usr} kann von konkreter Implementierung von I_{Lib} abhängen.
- Die Information reicht nicht aus um ein linkbares Binary zu erzeugen.

Tag 4: Übersetzen des Bibliotheksmoduls

- Erstellung von M_{Lib}, kompatibel zu I_{Lib}
- Übersetzung erzeugt ein linkbares Binary L_{Lib}
- Die Kombination (I_{Lib}, L_{Lib}) wird in einem Repository veröffentlicht.
- Quellcode M_{Lib} bleibt geheim.



Tag 4: Übersetzen des Bibliotheksmoduls

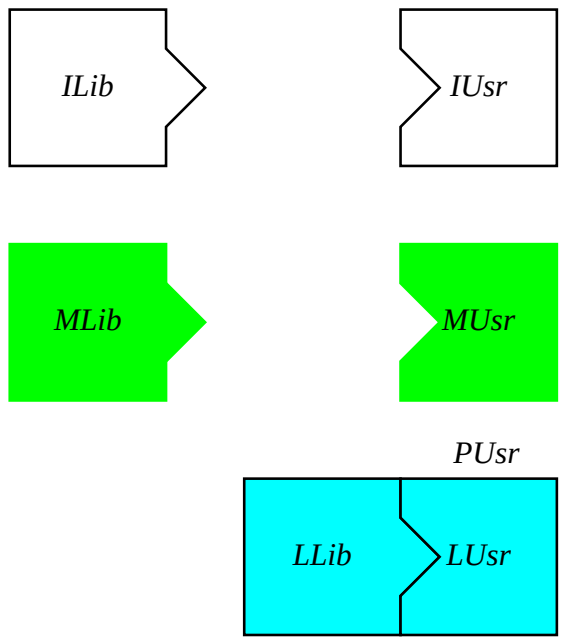
- Erstellung von M_{Lib}, kompatibel zu I_{Lib}
- Übersetzung erzeugt ein linkbares Binary L_{Lib}
- Die Kombination (I_{Lib}, L_{Lib}) wird in einem Repository veröffentlicht.
- Quellcode M_{Lib} bleibt geheim.

Schwierigkeiten

- Generische Module können teilweise nicht ohne Clienten übersetzt werden.

Tag 5: Linken der Anwendung

- Der Anwender verschafft sich das linkbare Binary L_{Lib}, das zu I_{Lib} passt.
- Der Anwender erzeugt ein Programm P_{Usr}, indem er L_{Lib} mit L_{Usr} linkt.



Tag 5: Linken der Anwendung

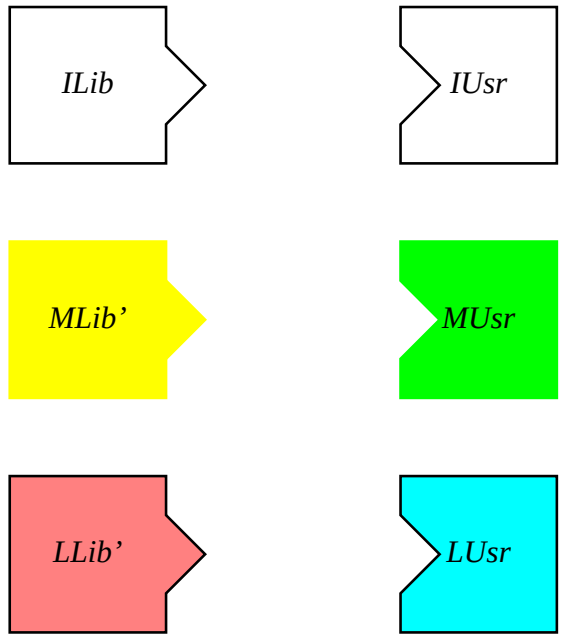
- Der Anwender verschafft sich das linkbare Binary L_{Lib}, das zu I_{Lib} passt.
- Der Anwender erzeugt ein Programm P_{Usr}, indem er L_{Lib} mit L_{Usr} linkt.

Schwierigkeiten

- In manchen Sprachen (Eiffel), kann es sein, dass M_{Lib} zu I_{Lib} passt, M_{Usr} zu I_{Usr} passt, I_{Usr} zu I_{Lib} passt und trotzdem P_{Usr} Laufzeitfehler hat.
- Einige Systeme verlegen Konsistenzprüfungen auf den Linkvorgang, bei dem dann Fehler auftreten können.
- Die Funktion des fertig verlinkten Programms sollte dem Programm entsprechen, dass bei der textlichen Zusammenfassung aller Quelldateien gesteht.

Tag 6: Weiterentwicklung der Implementierung der Bibliothek

- Eine neue Implementierung M'_{Lib} von I_{Lib} wird erstellt.
- Im Repository wird M_{Lib} durch M'_{Lib} ersetzt.



Tag 6: Weiterentwicklung der Implementierung der Bibliothek

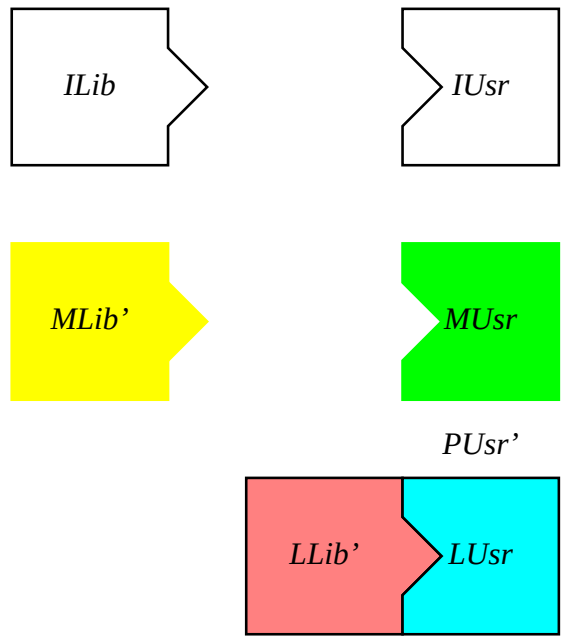
- Eine neue Implementierung M'_{Lib} von I_{Lib} wird erstellt.
- Im Repository wird M_{Lib} durch M'_{Lib} ersetzt.

Schwierigkeiten

- Änderungen an Superklassen können dazu führen, dass Anwendungscode neu übersetzt werden muss (auch wenn die öffentliche Schnittstelle sich nicht ändert)
- Wenn die Bibliotheken im Repository voneinander abhängen, können Anwender inkonsistente Binaries erhalten.

Tag 7: Erneutes Linken der Anwendung

- P_{Usr} ist nicht mehr aktuell, aber I_{Lib} ist unverändert.
- Keine erneute Übersetzung von M_{Usr} notwendig.
- Aktuelles Programm P'_{Usr} entsteht durch Linken von L_{Usr} mit L'_{Lib}.



Tag 7: Erneutes Linken der Anwendung

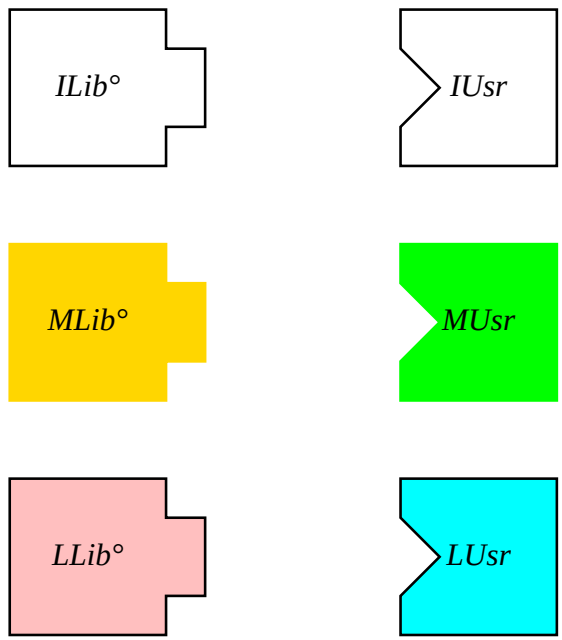
- P_{Usr} ist nicht mehr aktuell, aber I_{Lib} ist unverändert.
- Keine erneute Übersetzung von M_{Usr} notwendig.
- Aktuelles Programm P'_{Usr} entsteht durch Linken von L_{Usr} mit L'_{Lib}.

Schwierigkeiten

- Ist das Ergebnis das gleiche, als wenn M_{Usr} erneut übersetzt wird?
- In frühen Java-Versionen war das nicht der Fall.

Tag 8: Evolution der Bibliothek

- Die Schnittstelle der Bibliothek wird revidiert zu $I_{Lib}{}^{\circ}$
- Dazu wird eine Implementierung $M_{Lib}{}^{\circ}$ erstellt und nach $L_{Lib}{}^{\circ}$ übersetzt.
- Der Eintrag $(I_{Lib}{}^{\circ}, L_{Lib}{}^{\circ})$ ersetzt (I_{Lib}, L'_{Lib}) im Repository.



Tag 8: Evolution der Bibliothek

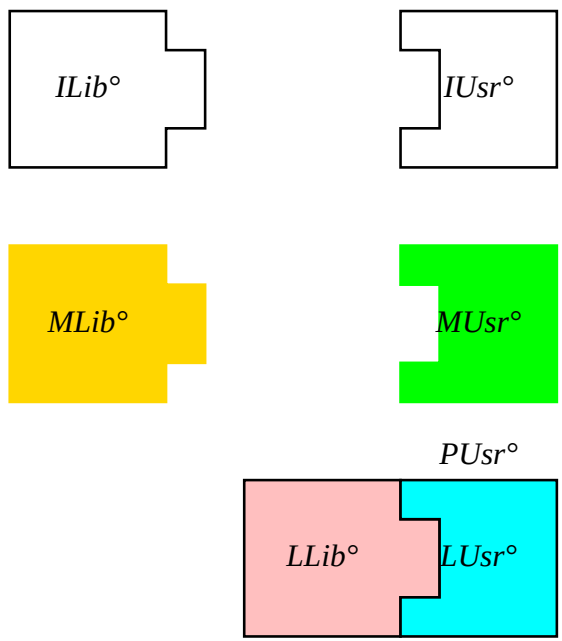
- Die Schnittstelle der Bibliothek wird revidiert zu $I_{Lib}{}^{\circ}$
- Dazu wird eine Implementierung $M_{Lib}{}^{\circ}$ erstellt und nach $L_{Lib}{}^{\circ}$ übersetzt.
- Der Eintrag $(I_{Lib}{}^{\circ}, L_{Lib}{}^{\circ})$ ersetzt (I_{Lib}, L'_{Lib}) im Repository.

Schwierigkeiten

- Inkonsistente Zustände bei gegenseitigen Abhängigkeiten

Tag 9: Adaptieren der Anwendung

- Das Anwendungsprogramm ist veraltet, da jetzt M_{Lib}° vorliegt.
- M_{Usr} und I_{Usr} passen nicht mehr zu I_{Lib}°.
- I_{Usr} wird zu I_{Usr}° abgeändert
- Die Implementierung M_{Usr} wird, passend zu I_{Lib}°, zu M_{Usr}° abgeändert und nach L_{Usr}° übersetzt.
- Neues Programm P_{Usr}° durch Linken von L_{Usr}° mit L_{Lib}° erzeugt.



Tag 9: Adaptieren der Anwendung

- Das Anwendungsprogramm ist veraltet, da jetzt M_{Lib}° vorliegt.
- M_{Usr} und I_{Usr} passen nicht mehr zu I_{Lib}°.
- I_{Usr} wird zu I_{Usr}° abgeändert
- Die Implementierung M_{Usr} wird, passend zu I_{Lib}°, zu M_{Usr}° abgeändert und nach L_{Usr}° übersetzt.
- Neues Programm P_{Usr}° durch Linken von L_{Usr}° mit L_{Lib}° erzeugt.

Schwierigkeiten

- Wenn nicht korrekt über Abhängigkeiten buchgeführt wird, kann versehentlich L_{Usr} mit L_{Lib}° oder L_{Usr}° mit L'_{Lib} gelinkt werden, was zu einem fehlerhaften Programm führt.

Einfaches Linken

Programmfragmente

- Ein *Programmfragment* ist eine Programmphrase (z.B. ein Ausdruck) *mit freien Variablen*.
- *Getrennte Übersetzung* meint:
  - Jedes Programmfragment kann einzeln der Typüberprüfung unterzogen werden
  - Für jedes Programmfragment kann einzeln Code erzeugt werden.
- (Zur Vereinfachung ignorieren wir Codeerzeugung.)
- Es muss ausreichend Information in Form eine Typannahme A über fehlende Programmfragmente vorliegen.
- Ein Urteil $A \vdash e : t$ liefert den Typ für ein Programmfragment e.

Programme

- Ein *vollständiges Programm* ist ein Programmfragment, das keine Variable enthält.
- Programmfragemente können verlinkt werden. Das Ergebnis kann wieder ein Programmfragment (mit freien Variablen) sein.
- Eine *Bibliothek* ist das Ergebnis einer unvollständigen Verlinkung.

Eine Konfigurationssprache

- Eingabe des Linkprozesses
  - Menge von Programmfragmenten
  - Vorschrift zur Kombination dieser Fragmente
- Vgl. Projektdateien, Makefiles, etc
- Ein *Linkset*

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

  besteht aus
  - Typannahme A_0, die *externe Schnittstelle* (leer bei einem vollständigen Programm)
  - durch Variable x_i benannte Urteile $A_i \vdash \mathcal{I}_i$, so dass $A_0, A_i \vdash \mathcal{I}_i$ gültig ist
- Die x_i dürfen in den anderen Urteilen verwendet werden.

Beispiele für Linksets

- Ein vollständiges Programm

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

- *main* ist abgeschlossen (enthält keine Variable)

$\Rightarrow$ Kein Linken erforderlich!

Beispiele für Linksets/2

- Zwei Fragmente

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

- y ist abgeschlossen
- *main* benötigt noch die Definition von y
- Überprüfung der Typkonsistenz: intramodular und intermodular

Beispiele für Linksets/2

- Zwei Fragmente

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

- y ist abgeschlossen
- *main* benötigt noch die Definition von y
- Überprüfung der Typkonsistenz: intramodular und intermodular
- Ziel: Verlinken durch Substitution

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

Beispiele für Linksets/3

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

Beispiele für Linksets/3

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

- Keine Verlinkung möglich, Selbstreferenz!

Beispiele für Linksets/3

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

- Keine Verlinkung möglich, Selbstreferenz!
- Ähnlicher Fall

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

- Keine Verlinkung möglich, zyklische Abhängigkeiten nicht erlaubt!

Beispiele für Linksets/3

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

- Keine Verlinkung möglich, Selbstreferenz!
- Ähnlicher Fall

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

- Keine Verlinkung möglich, zyklische Abhängigkeiten nicht erlaubt!
- Rekursive Abhängigkeiten/Module in realistischen Sprachen (Java) möglich.

Linking Lemma

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

Eigenschaften von Linksets

Zugriff auf Teile

Zur Struktur

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

definiere

$imp(L) =$	$dom(A_0)$	Importierte Namen
$exp(L) =$	$\{x_1, \ldots, x_n\}$	Exportierte Namen
$names(L) =$	$imp(L) \cup exp(L)$	Alle Namen
$imports(L) =$	A_0	Importumgebung
$exports(L) =$	$x_1 : t_1, \ldots, x_n : t_n$	Exportumgebung

Wohlgeformte Linksets

Namen werden konsistent verwendet

Eine Struktur

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

ist ein Linkset, *linkset*(L), falls

- *imports*(L) und *exports*(L) definieren jede Variable höchstens einmal
- für alle i gilt: A_0, A_i definiert jede Variable höchstens einmal und $dom(A_i) \subseteq exp(L)$
  (zu jeder internen Typannahme gibt es eine Definition)
- $imp(L) \cap exp(L) = \emptyset$

Beispiele: Wohlgeformte Linksets

Nicht wohlgeformt

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

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

Wohlgeformt

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

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

Intramodular konsistente Linksets

Eine Struktur

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

ist ein *intramodular konsistenter Linkset*, *intra-checked*(L), falls

- *linkset*(L)
- für alle i gilt: $A_0, A_i \vdash e_i : t_i$ ist gültig

Bedeutet nur, dass jede Definition für sich die Typüberprüfung besteht!

Beispiele: Intramodular konsistente Linksets

Nicht intramodular konsistent

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

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

Intramodular konsistent

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

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

Intermodular konsistente Linksets

Eine Struktur

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

ist ein *intermodular konsistenter Linkset*, *inter-checked*(L), falls

- *intra-checked*(L)
- für alle $1 \leq j, k \leq n$: falls $x_j : t \in A_k$, dann gilt $t = t_j$

Beispiele: Intermodular konsistente Linksets

Nicht intermodular konsistent

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

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

Intermodular konsistent

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

Verschmelzen von Linksets

Vorbereitung

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

- Die Umgebung A_0 beschreibt die noch fehlenden Definitionen.
- Ziel: ein *voll verlinketer Linkset*, in dem A_0 und auch alle anderen Typannahmen leer sind.
- Fehlende Definitionen werden durch *Verschmelzen von Linksets* bereitgestellt.
- Hilfsdefinitionen
  - $A \setminus X$ entfernt aus Typannahme A die Bindungen für Variable aus X
  - $A \mid X$ behält in A nur die Bindungen für Variable aus X bei
  - Kompatible Typannahmen: $A_1 \div A_2$, falls für alle $x \in dom(A_1) \cap dom(A_2)$ gilt $A_1(x) = A_2(x)$.
  - Verschmelzung zweier Typannahmen: $A_1 + A_2 = A_1, (A_2 \setminus dom(A_1))$

Verschmelzen von Linksets

Gegeben zwei Linksets

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

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

mit $\mathit{linkset}(L)$, $\mathit{linkset}(L')$, $\mathit{exp}(L) \cap \mathit{exp}(L') = \emptyset$. Dann ist ihre *Verschmelzung* $L + L'$ definiert durch

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

Beispiel: Verschmelzen von Linksets

Linkset L_{Lib} (Bibliothek)

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

Linkset L_{Usr} (Anwendung)

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

Beispiel: Verschmelzen von Linksets

Linkset L_{Lib} (Bibliothek)

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

Linkset L_{Usr} (Anwendung)

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

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

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

Eigenschaften der Verschmelzung von Linksets

Definition: Kompatibilität von Linksets

Zwei Linksets sind kompatibel, $L \div L'$, falls

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

Linkset+Linkset = Linkset

Wenn $linkset(L)$, $linkset(L')$ und $L \div L'$, dann $linkset(L + L')$.

Eigenschaften der Verschmelzung von Linksets

Definition: Kompatibilität von Linksets

Zwei Linksets sind kompatibel, $L \div L'$, falls

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

Intramodulare Konsistenz bleibt erhalten

Wenn *intra-checked*(L), *intra-checked*(L') und $L \div L'$, dann *intra-checked*$(L + L')$.

Eigenschaften der Verschmelzung von Linksets

Definition: Kompatibilität von Linksets

Zwei Linksets sind kompatibel, $L \div L'$, falls

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

Intermodulare Konsistenz bleibt erhalten

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

Linken

Definition: Linkschritt

Gegeben

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

dann führt L einen Linkschritt nach L' aus, $L \rightsquigarrow L'$, mit

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

Beispiel: Linkschritt

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

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

Beispiel: Linkschritt

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

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

Nachher:

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

Beispiel: Linkschritt

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

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

Oder nachher:

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

Beispiel: Linkschritt

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

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

Oder auch s einsetzen:

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

Eigenschaften von Linkschritten

Linkschritte bewahren die *linkset*-Eigenschaft

Wenn $\mathit{linkset}(L)$ und $L \rightsquigarrow L'$, dann $\mathit{linkset}(L')$.

Linkschritte bewahren die *inter-checked*-Eigenschaft

Wenn $\mathit{inter\text{-}checked}(L)$ und $L \rightsquigarrow L'$, dann auch $\mathit{inter\text{-}checked}(L')$.

Bemerkung

$\mathit{intra\text{-}checked}(L)$ wird durch Linkschritte **nicht** bewahrt.

Link-Algorithmus

Die Linkschrittrelation terminiert und liefert ein eindeutiges Ergebnis.

Jeder Linkschritt entfernt eine Typannahme. Erfolgreiche Termination führt zu einem *vollständig verlinkten Programm*.

Beispiel: Linken bis zum vollständig verlinkten Programm

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

Beispiel: Linken bis zum vollständig verlinkten Programm

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

Beispiel: Linken bis zum vollständig verlinkten Programm

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

Beispiel: Linken bis zum vollständig verlinkten Programm

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

Beispiel: Linken bis zum vollständig verlinkten Programm

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

Beispiel: Linken bis zum vollständig verlinkten Programm

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

Beispiel: Linken bis zum vollständig verlinkten Programm

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

Kein weiterer Linkschritt möglich!

Einfache Module

Notation für einfache Module

```
module M1 {
  import {}                                  // Importliste
  export { x: int }                          // Exportliste

  x: int = 3;                                // Definitionen
}

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

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

Signaturen und Bindungen

Typüberprüfung für Module

Signaturen S (getypte Exportlisten)

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

Bindungen d (Module)

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

Beispiel: Zwei Module

- Modul `M1`

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

- Modul `M2`

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

Übersetzung eines Bindungsurteils in einen Linkset

$|A \vdash d \therefore S|$ ist die Übersetzung eines gültigen Urteils in einen Linkset.
Beispiele:

- Modul `M1` als Linkset

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

- Modul `M2` als Linkset

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

Eigenschaften der Übersetzung

Sei $|A \vdash d \therefore S|$ die Übersetzung eines gültigen Urteils in einen Linkset.

Getrennte Übersetzung:

Falls $A \vdash d \therefore S$, dann gilt *inter-checked*$(|A \vdash d \therefore S|)$.

Getrennte Übersetzung und Verschmelzung:

Falls $A \vdash d \therefore S$, $A' \vdash d' \therefore S'$ und $(A \vdash S) \div (A' \vdash S')$, dann *inter-checked*$(|A \vdash d \therefore S| + |A' \vdash d' \therefore S'|)$.

Aussagen über getrennte Übersetzung

Konventionen

Sei $M = A \vdash d \therefore S$ ein Modul, L ein Linkset und $|M|$ der Linkset, der durch die Übersetzung von M entsteht.

- $\mathit{valid}(M)$: M ist herleitbar, Typüberprüfung von Modul M erforlgreich
- $M \div M'$: Module M und M' sind typkompatibel
- $\mathit{link}(L) = L'$, wenn $L \rightsquigarrow^* L'$ und $L' \not\rightsquigarrow$

Aussagen über getrennte Übersetzung

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

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

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

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

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

Zusammenfassung

- Linkprozess durch Substitution formalisiert.
- Unter gewissen Voraussetzungen an Module und Signaturen kann der Compiler sicherstellen, dass das Linken nicht fehlschlägt, wenn nur jedes einzelne Modul zusammen mit den Signaturen der importierten Module überprüft worden ist.
- Erneute Übersetzung ist nicht erforderlich.
- Reihenfolge der Linkschritte und des Verschmelzens von Modulen unerheblich.
- Viele weitere Arbeiten auf dieser Basis (u.a. für Java).