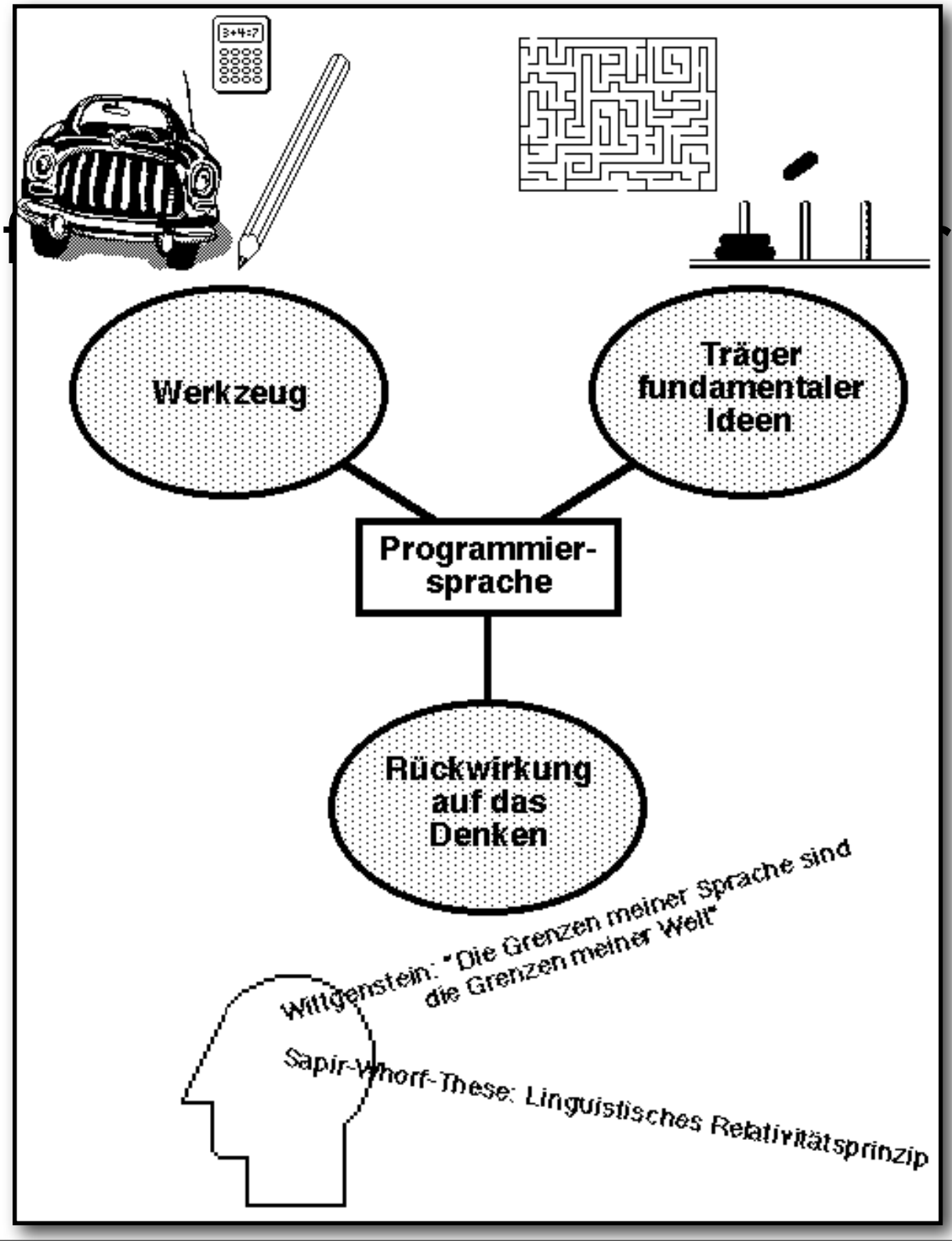
Kapitel 1 - Programmierstile

- Klassifikation von Programmiersprachen
- Prozedurale und deklarative Programmierung
- Beschreibungsmächtigkeit der Programmierstile
- Programmierstile im Detail
  - Imperative Programmierung
  - Funktionale Programmierung
  - Prädikative Programmierung

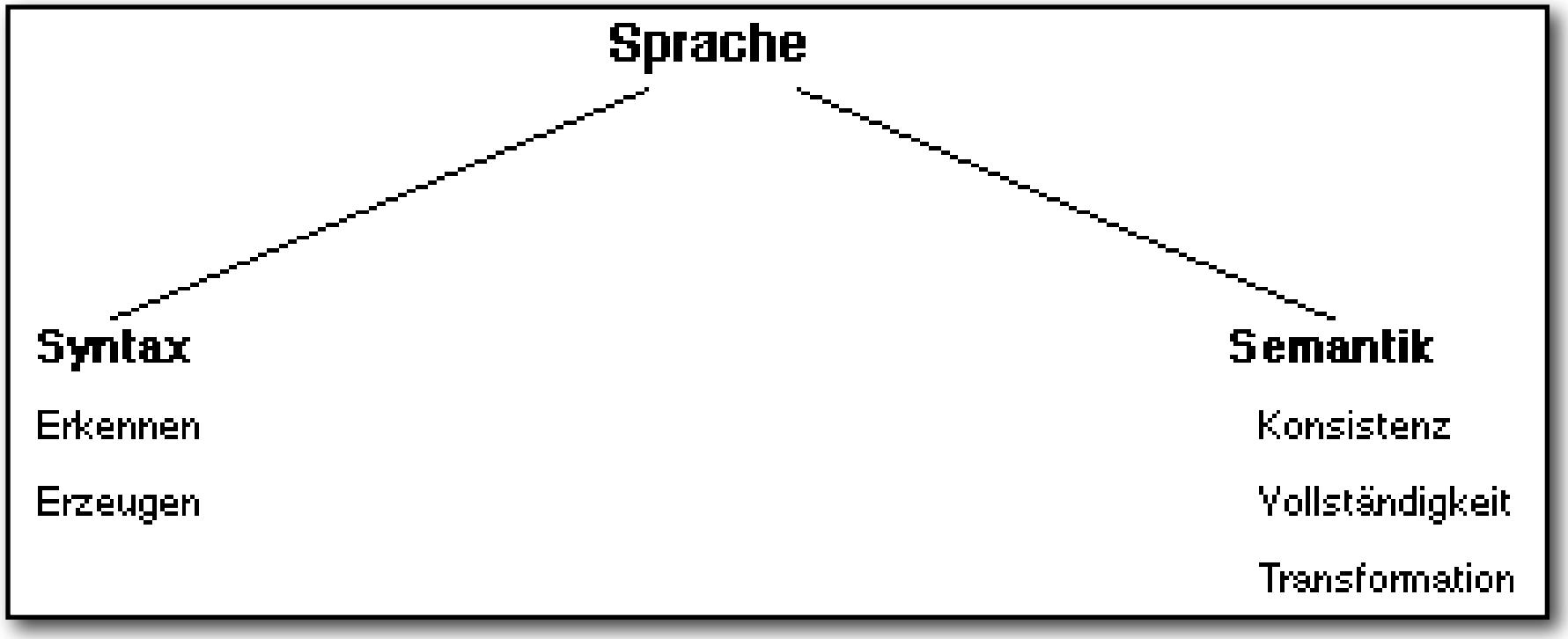
Programmierstile

Klassifikation von Programmiersprachen

Klassif rachen



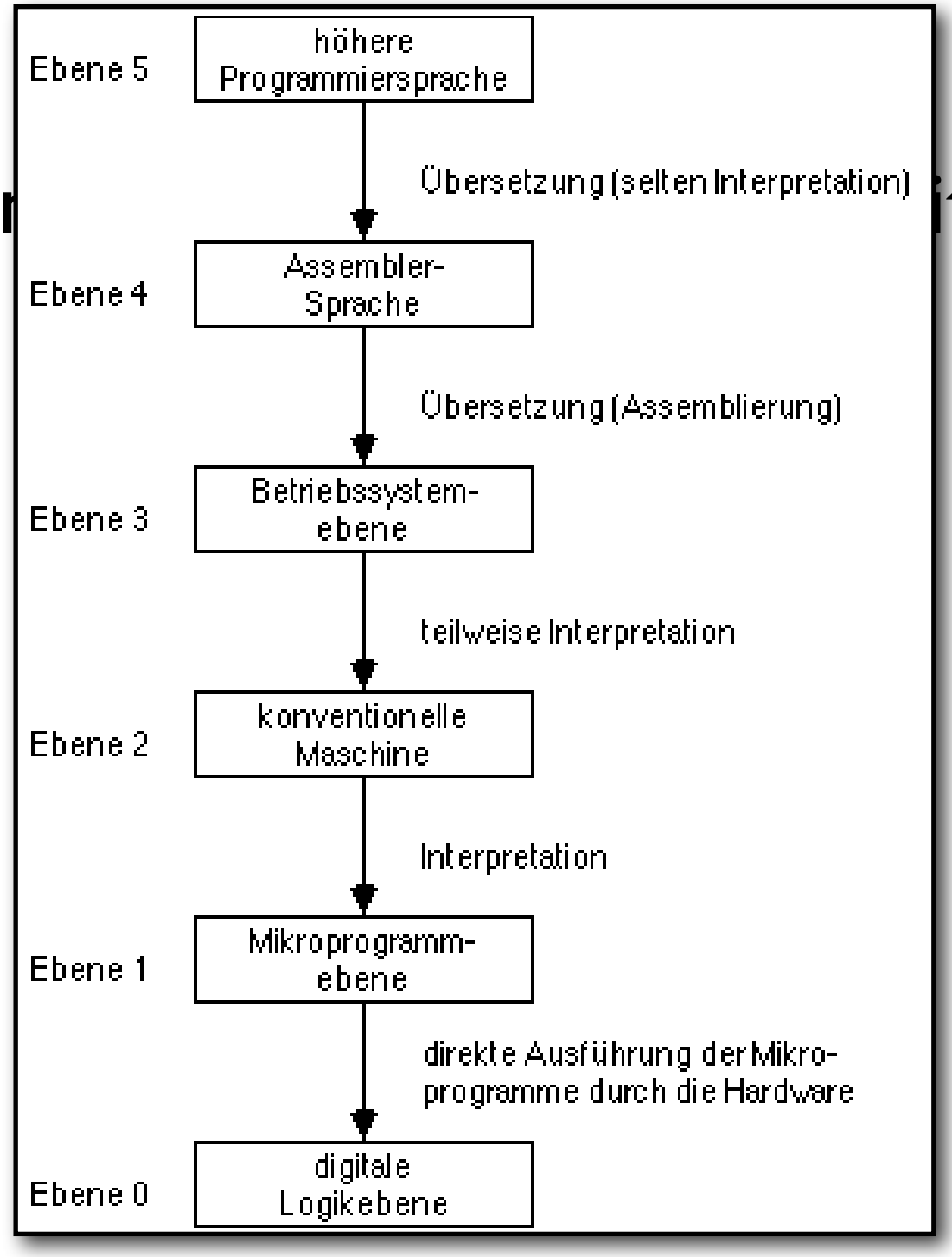
Programmierstile
Sprache als fundamentale Idee



Programmierstile

Ebenenmodell der Rechnerarchitektur

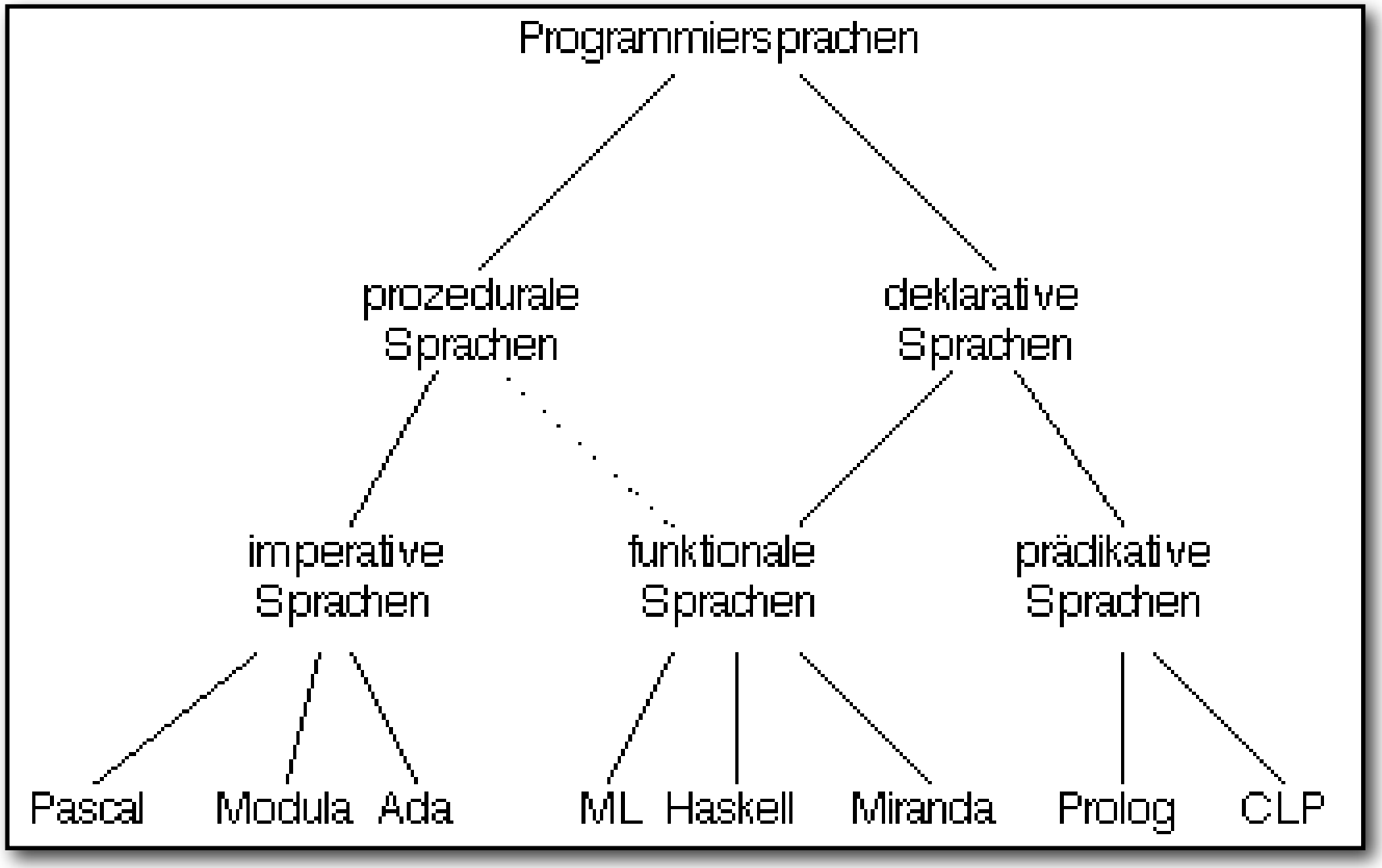
Ebener itektur



Programmierstile
Klassifikation von Programmiersprachen

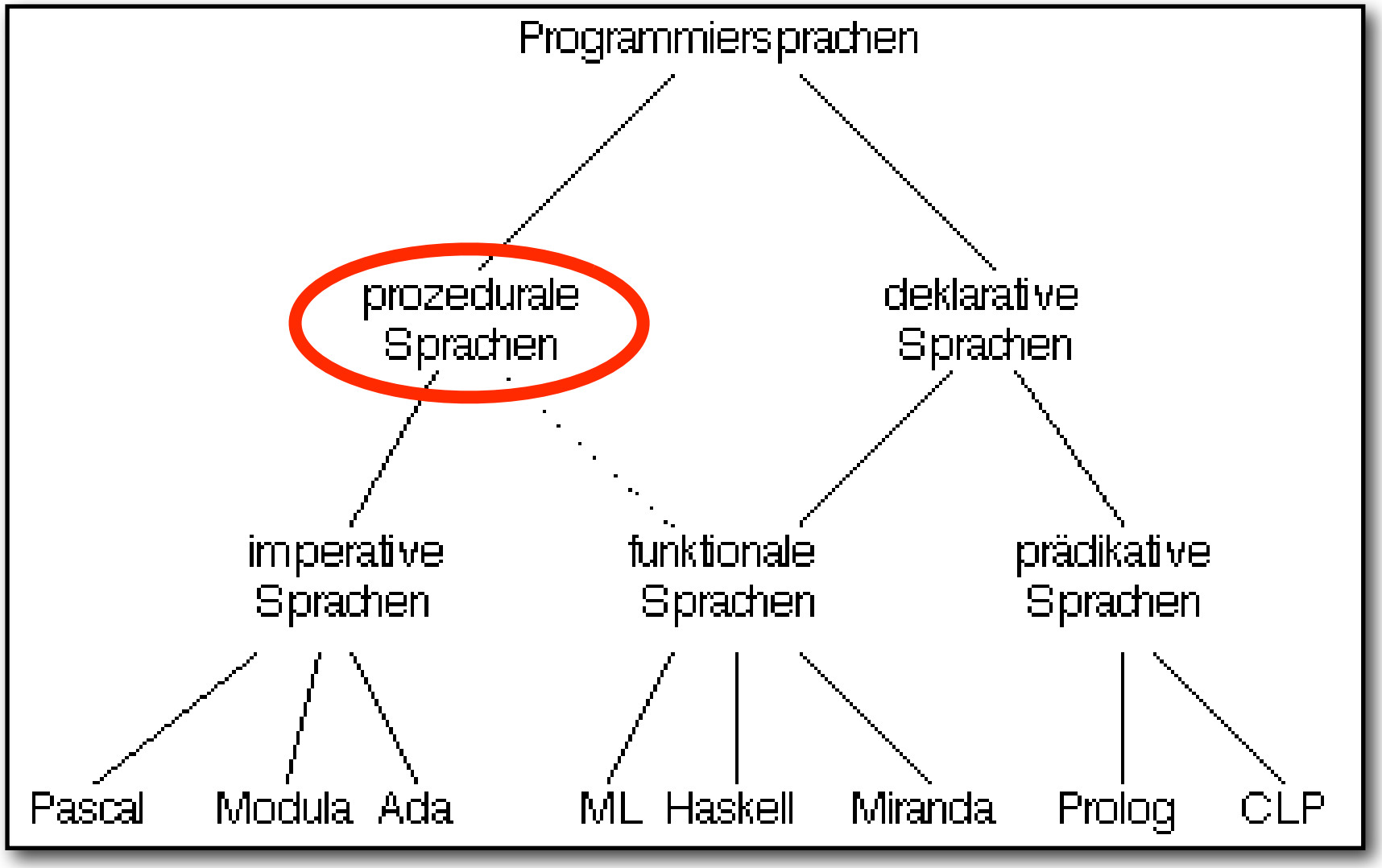
- Zahlreiche verschiedene Ansätze - Fülle von Kategorien
- drei relativ stabile Klassen
  - imperative Programmierung
  - prädikative Programmierung
  - funktionale Programmierung
- Unterschiede: Eleganz, Klarheit, Kürze, Mitteilbarkeit

Programmierstile
Übergeordnete Kategorien

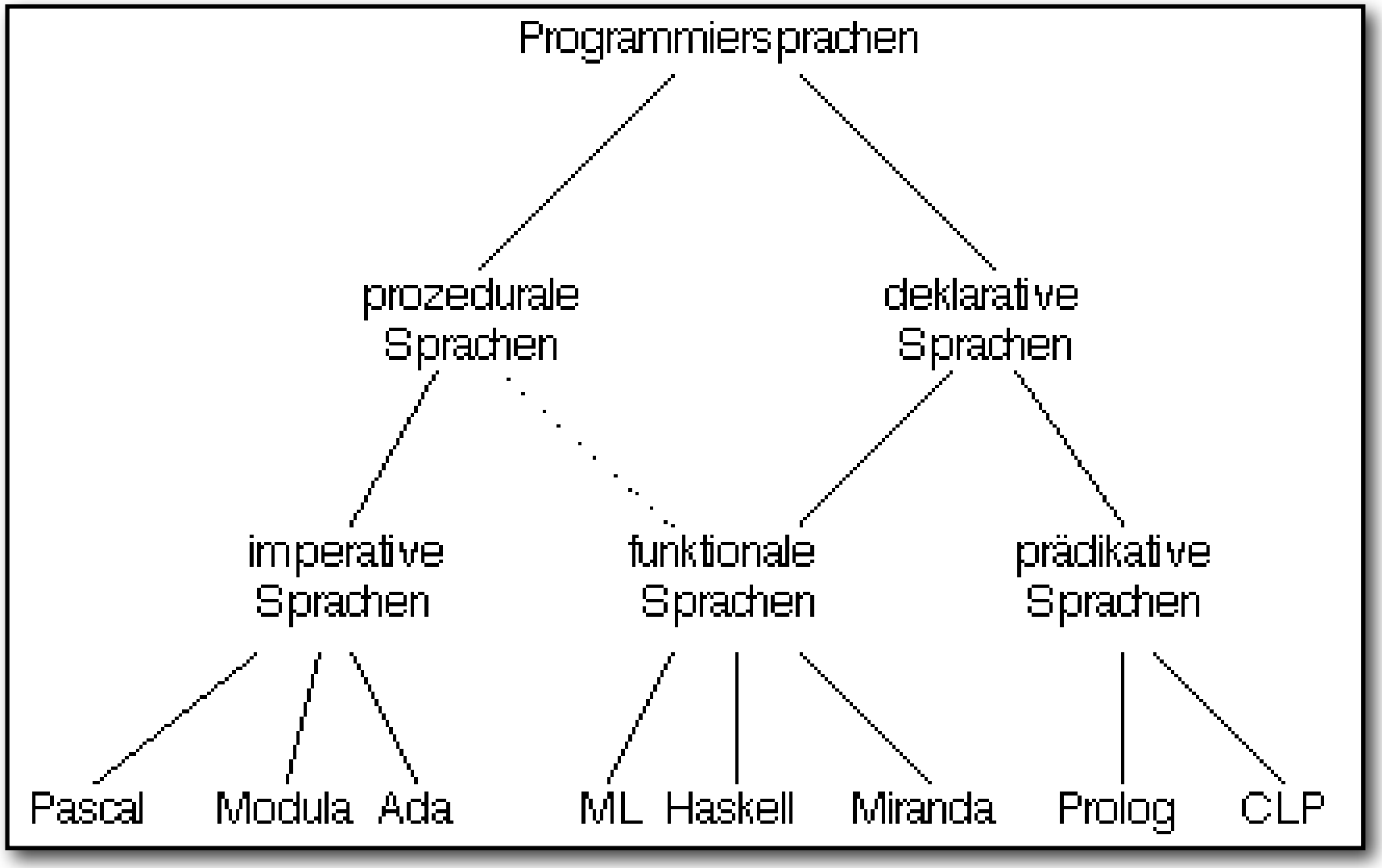


Programmierstile

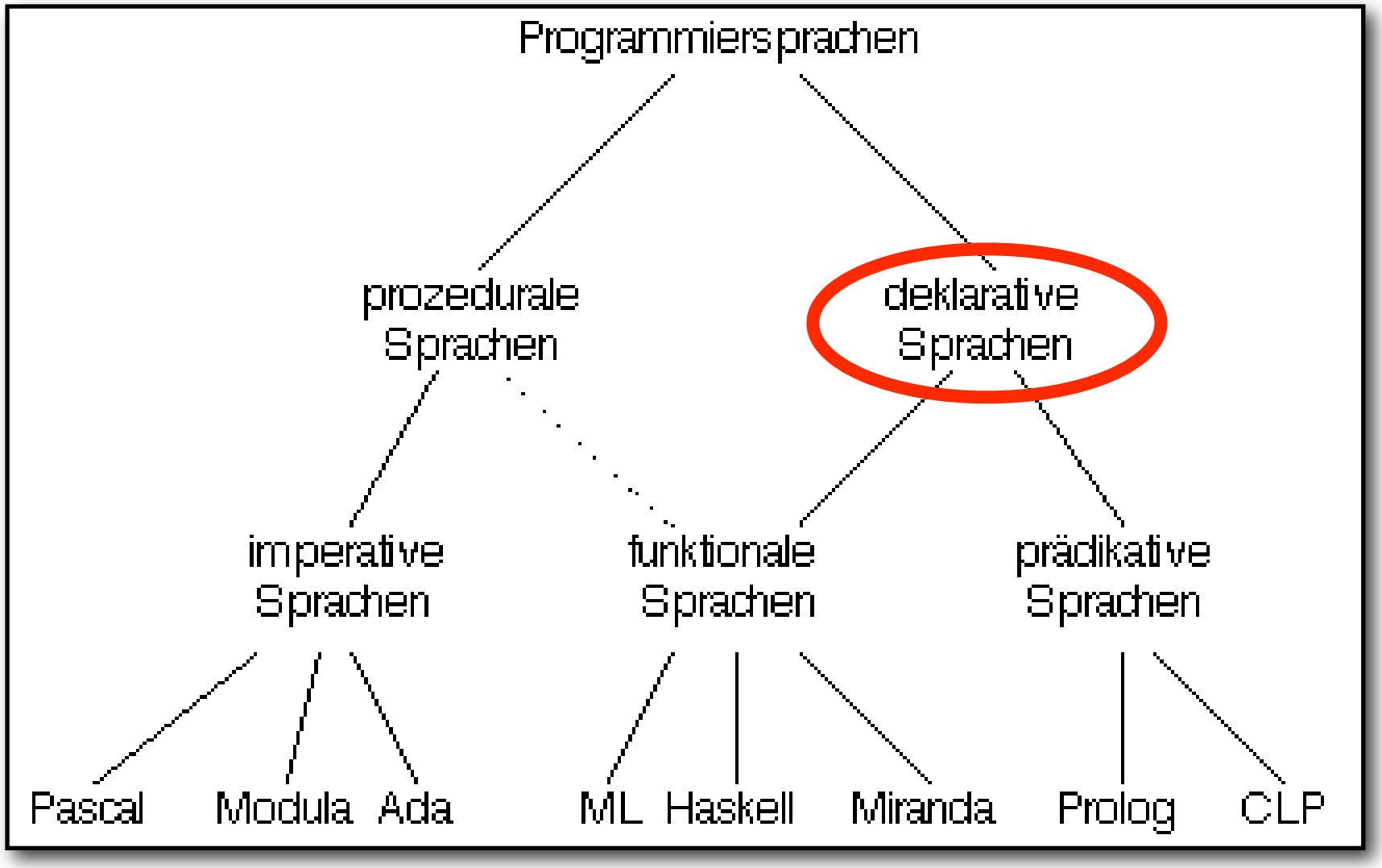
Übergeordnete Kategorien



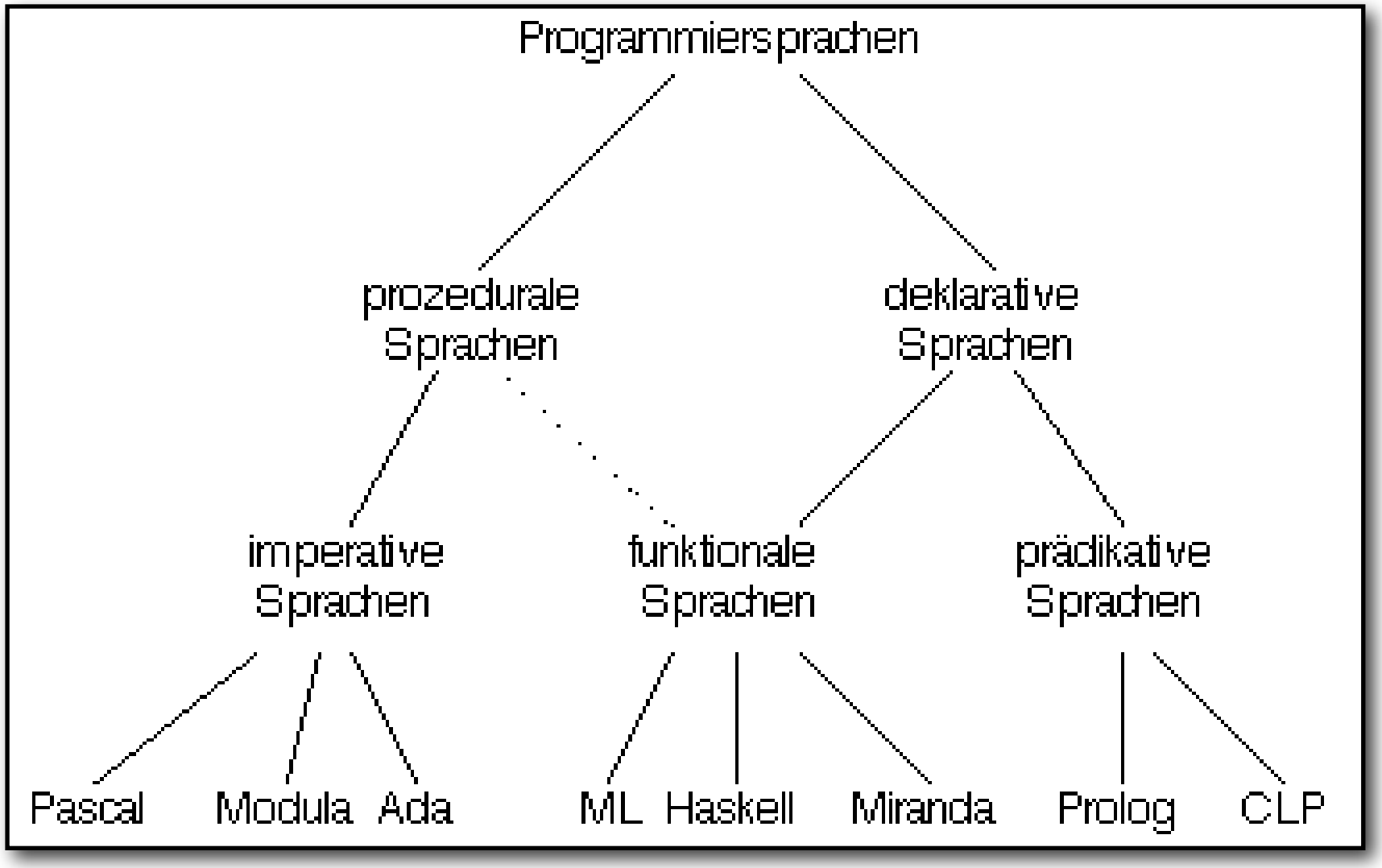
Programmierstile
Übergeordnete Kategorien



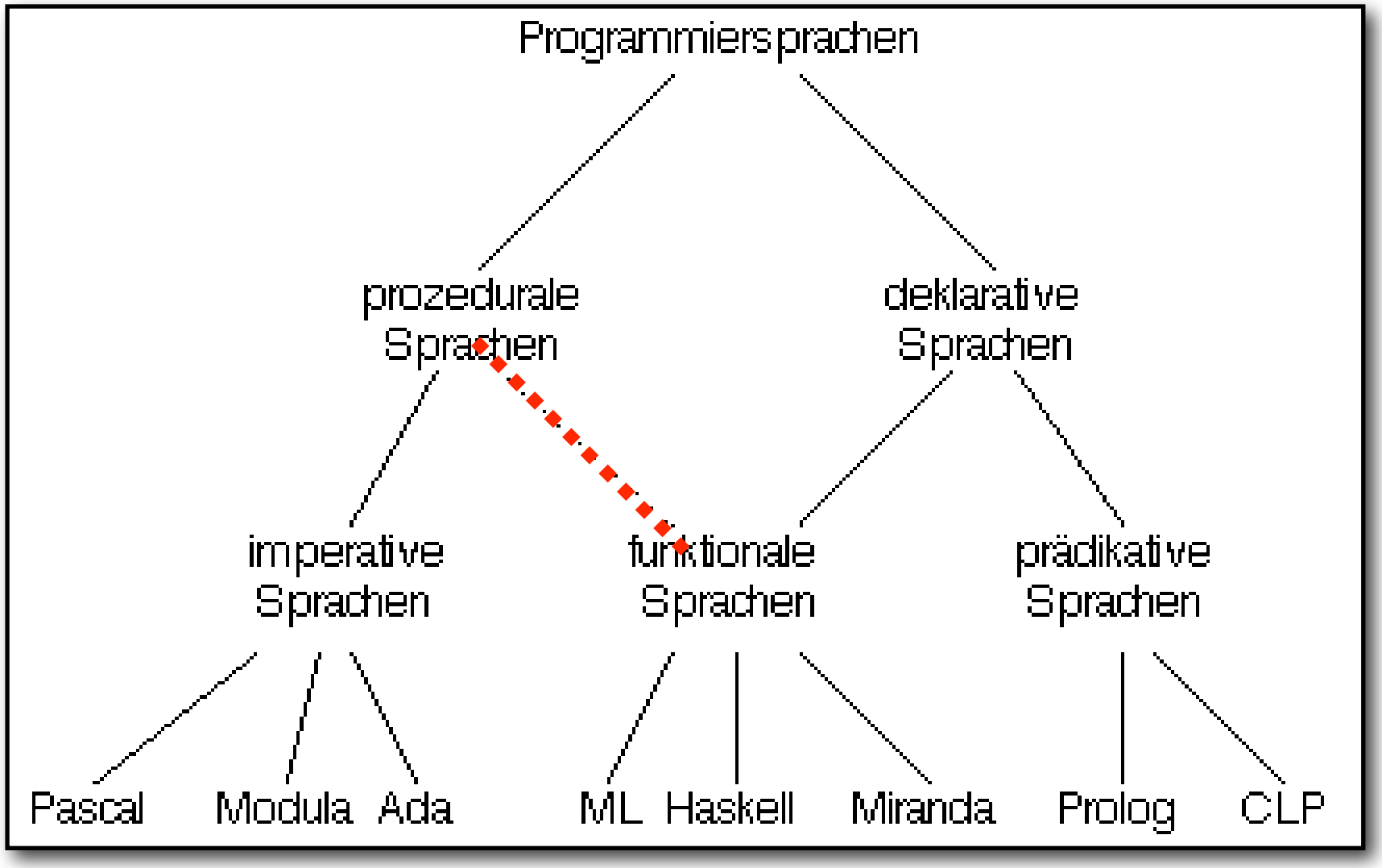
Programmierstile
Übergeordnete Kategorien



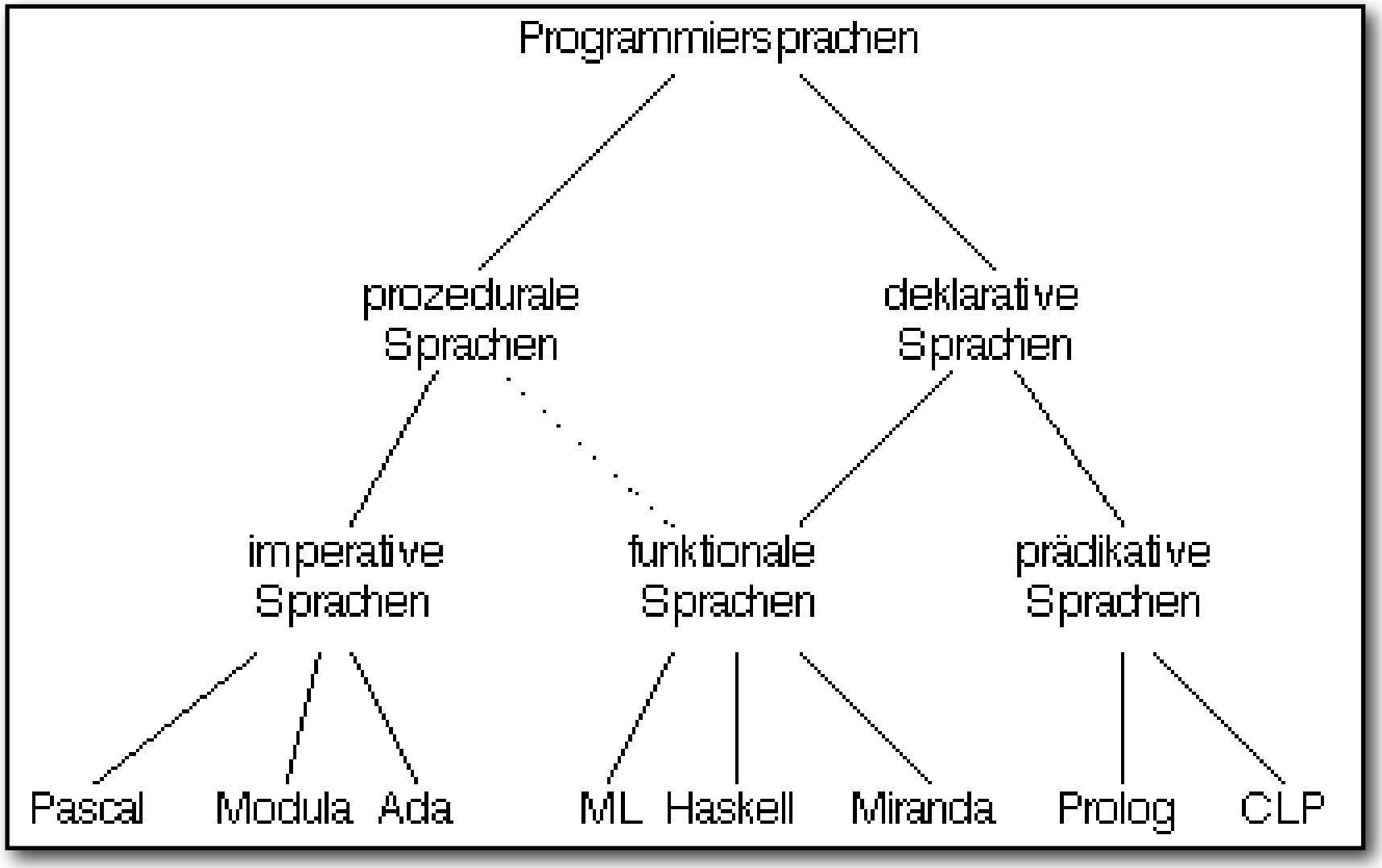
Programmierstile
Übergeordnete Kategorien



Programmierstile
Übergeordnete Kategorien



Programmierstile
Übergeordnete Kategorien



Programmierstile
... und objektorientierte Programmierg.

- 4. Programmierstil ????
- leistungsfähige Typsysteme
- Datenabstraktion, -kapselung und Vererbung
- Objekt: Zusammenfassung von Daten und Operationen (Methoden) zu einer Einheit
- Vererbungsbeziehungen zwischen Objekten ("ist Spezialisierung von", "ist Verallgemeinerung von")
- Nachrichtenaustausch
- Objekte mit gemeinsamen Eigenschaften: Klassen

Programmierstile
... und objektorientierte Programmierg.

- Sprachvertreter: Smalltalk-80 (als Reinform), Oberon, Java, Simula (als Urahn), C++, Eiffel
- die meisten OO-Sprachen sind derzeit im Kern imperativ
- Andererseits: OO-Konzepte können mit allen drei Stilen kombiniert werden, ohne deren Charakter zu stören (akt. Forschung)
- Logisch gesehen: OOP und strukturierte Programmierung $\in$ Softwareentwurfsmethoden

Programmierstile
Prozedurale Programmierung

Zweckbindung:

- Lösungsweg
- gegebene Größen
- gesuchte Größen
- Weg, um gesuchte aus gegebenen Größen zu gewinnen

Programmierstile
Deklarative Programmierung

Zweckfreiheit:

- Beschreibung allgemeiner Eigenschaften gewisser Objekte
- Beziehungen (*Wissen*)
- Wissen induziert kein Problem
- kein Lösungsweg
- keine bekannten, keine gesuchten Größen
- Zweck→Problembeschreibung mit Angabe bekannter und gesuchter Größen
- Aufgabe des Computers: Wissen zur Lösung des Problems nutzen

Programmierstile
Deklarative Programmierung

Drawback

- es gibt (bisher) keine deklarativen Programmiersprachen im engeren Sinne.
- Auch prädikative Sprachen sind mit Ausnahme von kleinen "Spielbeispielen" nicht deklarativ.

- Hilfsweise: natürliche Sprache

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

- "zweckfreies" (naives) Wissen über Zusammenhang zwischen Geldbetrag, Kaufkraft, Preissteigerungsrate
- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Problem/Zweck1: zu einem gegebenen Geldbetrag G zu Beginn des Jahres und zu gegebener Preissteigerungsrate P die Kaufkraft K am Schluß des Jahres berechnen

Preissteigerungsrate

- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

- "zweckfreies" (naives) Wissen über Zusammenhang zwischen Geldbetrag, Kaufkraft, Preissteigerungsrate
- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

Problem/Zweck2: zu G und K nach Ablauf eines Jahres P ermitteln

zwischen Geldbetrag, Kaufkraft, Preissteigerungsrate

- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

- "zweckfreies" (naives) Wissen über Zusammenhang zwischen Geldbetrag, Kaufkraft, Preissteigerungsrate
- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

Problem/Zweck3: zu K eine Tabelle von Paaren (G,P) liefern, so daß G vermindert um P der Kaufkraft K entspricht

Preissteigerungsrate

- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

- "zweckfreies" (naives) Wissen über Zusammenhang zwischen Geldbetrag, Kaufkraft, Preissteigerungsrate
- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

Problem/Zweck4: alle Tripel (G,P,K) mit der spezifizierten Eigenschaft ausgeben

- zwischen Geldbetrag, Kaufkraft, Preissteigerungsrate
- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiel

Deklaratives "Programm":

Die Kaufkraft eines Geldbetrages sinkt nach Ablauf eines Jahres um die Preissteigerungsrate.

Merkmale:

- "zweckfreies" (naives) Wissen über Zusammenhang zwischen Geldbetrag, Kaufkraft, Preissteigerungsrate
- *kein* Problem, *keine* gesuchten, *keine* gegebenen Größen
- Zweck dieses Wissens: Programmierer

Programmierstile
Deklarative Programmierung - Beispiele

Beispiele für deklarative Programme:

- Katzen trinken Milch.
- Wenn A und B dieselbe Mutter haben, dann sind A und B Geschwister.
- Das Quadrat einer geraden/ungeraden Zahl ist gerade/ungerade.

Programmierstile
Prozedurale Programmierung - Beispiel

Eine mögliche prozedurale Darstellung des deklarativen Programms:

> *Die Kaufkraft eines Geldbetrages nach Ablauf eines Jahres erhält man, indem man den Geldbetrag um die Preissteigerungsrate vermindert.*

- wohldefinierter Zweck: Problem, Kaufkraft eines Geldbetrages nach Ablauf eines Jahres zu ermitteln, und zugehörigen maschinell nachvollziehbaren Lösungsweg.
- Gegebene Größen (Geldbetrag, Preissteigerungsrate), gesuchte Größen (Kaufkraft nach Ablauf eines Jahres)

Programmierstile
Prozedurale Programmierung - Beispiele

Beispiele für prozedurale Programme:

- Zum Öffnen Lasche anheben, zusammendrücken und farbige Ecke abreißen.
- Falls Sie weitere Informationen wünschen, brauchen Sie nur den ausgefüllten Coupon zurückzusenden.
- Zur Installation der Software müssen Sie mindestens die Dateien X und Y auf Ihre Festplatte kopieren. Alles weitere entnehmen Sie der Datei "Liesmich".

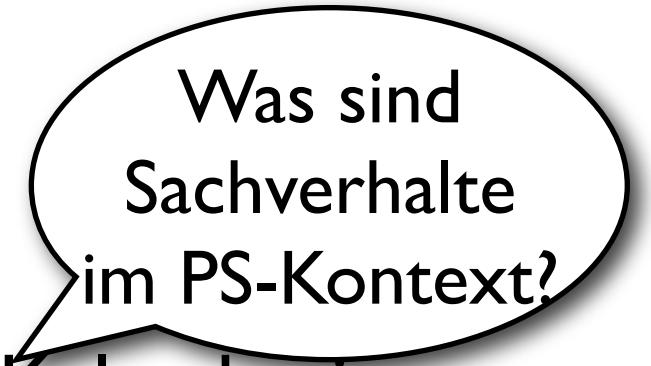
Programmierstile
Beschreibungsmächtigkeit

Analogie: natürliche Sprachen

> Sprachen, zumindest unseres Kulturkreises, sind gleichmächtig: Beliebige Sachverhalte lassen sich gleichermaßen in Deutsch, Englisch, Französisch, Italienisch usw. formulieren.

Gelten ähnliche Aussagen auch im Bereich der Programmierstile?

Programmierstile
Beschreibungsmächtigkeit



Analogie: natürliche Sprachen

Sprachen, zumindest unseres Kulturkreises, sind gleichmächtig: Beliebige Sachverhalte lassen sich gleichermaßen in Deutsch, Englisch, Französisch, Italienisch usw. formulieren.

Gelten ähnliche Aussagen auch im Bereich der Programmierstile?

Programmierstile
Beschreibungsmächtigkeit

- Hier: Sachverhalte=berechenbare Funktionen
- alle genannten Programmierstile sind in diesem Sinne gleich leistungsfähig und besitzen die gleiche Ausdruckskraft:

  Jede berechenbare Funktion läßt sich durch ein Programm jedes Programmierstils beschreiben.

- Hat man zu irgendeinem Problem ein Programm im Programmierstil A erstellt hat, dann kann man auch ein Programm in irgendeinem der anderen Stile schreiben, das die gleiche Aufgabe löst.

Programmierstile

Beschreibungsmächtigkeit - “Beweis”

Beweismethode: Ringschluß

- Es gibt ein imperatives Programm, das funktionale Programme simuliert
- es gibt ein funktionales Programm, das prädikative Programme nachvollzieht
- es gibt ein prädikatives Programm, mit dem man imperative Programme simulieren kann

- Exakter Beweis: andere Veranstaltungen

Programmierstile
Detailanalyse - Merkmale

1. *Kalkül* oder mathematisches *Modell*:

Jedem Programmierstil liegt ein bestimmtes mathematisches Modell oder ein Kalkül zugrunde.

Programmiersprache=Kalkül + syntactic sugar

Programmierstile
Detailanalyse - Merkmale

1. Kalkül od *Modell*:

Jedem P timmtes
mathem alkül
zugrunde.

Sprachelemente, die Lesbarkeit von Programmen und Benutzungsfreundlichkeit verbessern, nicht aber Ausdruckskraft erhöhen

Programmiersprache=Kalkül + syntactic sugar

Programmierstile
Detailanalyse - Merkmale

1. *Kalkül* oder mathematisches *Modell*:

Jedem Programmierstil liegt ein bestimmtes mathematisches Modell oder ein Kalkül zugrunde.

Programmiersprache=Kalkül + syntactic sugar

Programmierstile
Detailanalyse - Merkmale

2. Begriff des *Programms*
3. Begriff des *Befehls* (besser: elementaren Bausteins) zur Beschreibung von Programmen und der *Konstruktoren*, um elementare Bausteine zu komplexeren zusammenzusetzen
4. *Variablenkonzept*: Je Programmierstil bestimmte Vorstellung vom Begriff der Variablen und darauf möglichen Operationen

Programmierstile
Detailanalyse - vergleichendes Beispiel

Mischen/Verschmelzen von Zahlenfolgen:

> *Gegeben sind zwei aufsteigend sortierte Folgen von ganzen Zahlen. Gesucht ist ein Algorithmus, der die beiden Folgen zu einer aufsteigend sortierten Folge mischt.*

Je Stil

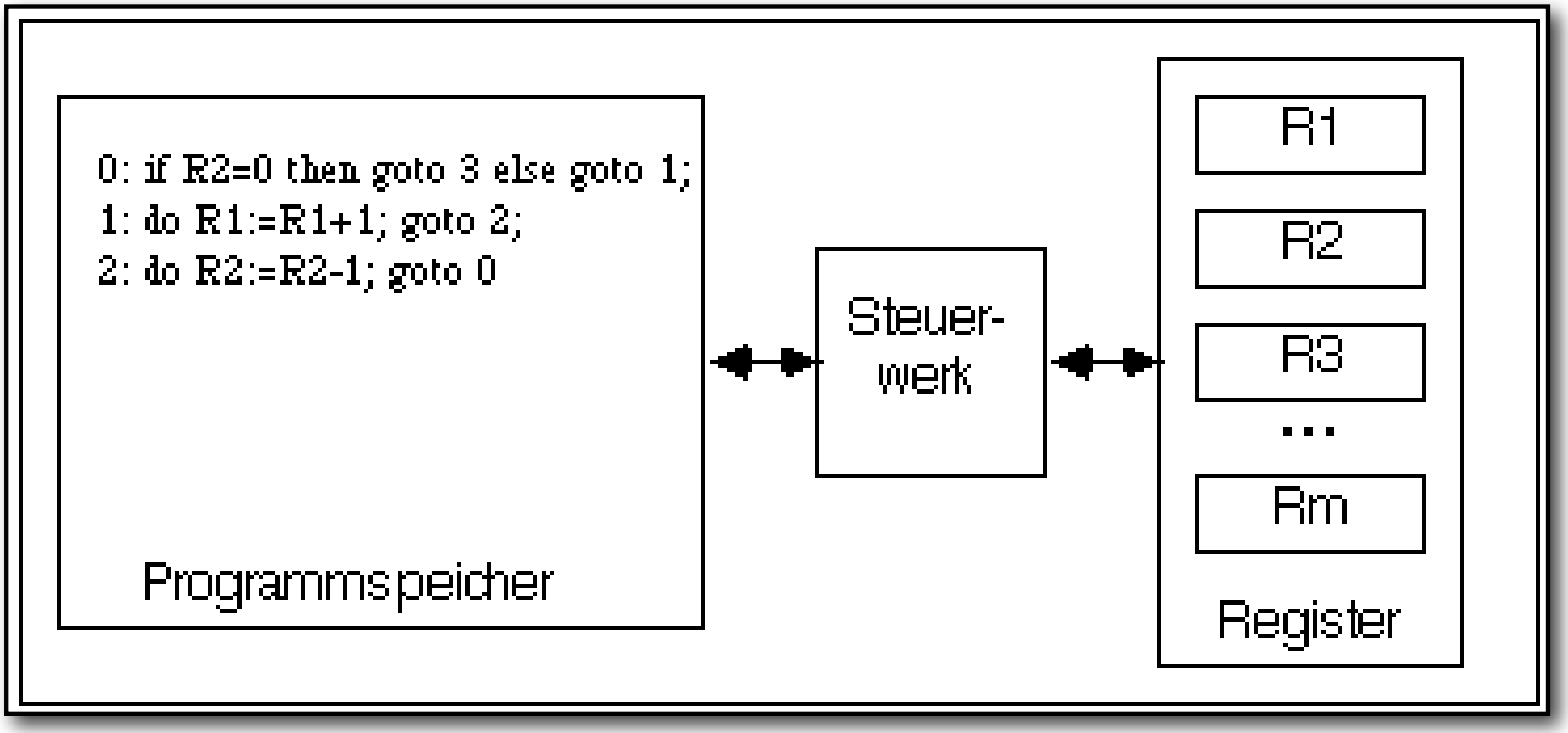
- Programm in fiktiver Programmiersprache
- Programm in konkreter Programmiersprache

Darstellung in fiktiver Sprache läßt Merkmale der Stile deutlicher hervortreten.

Programmierstile

Imperative Programmierung - Modell

mathematisches Modell:
Registermaschine.

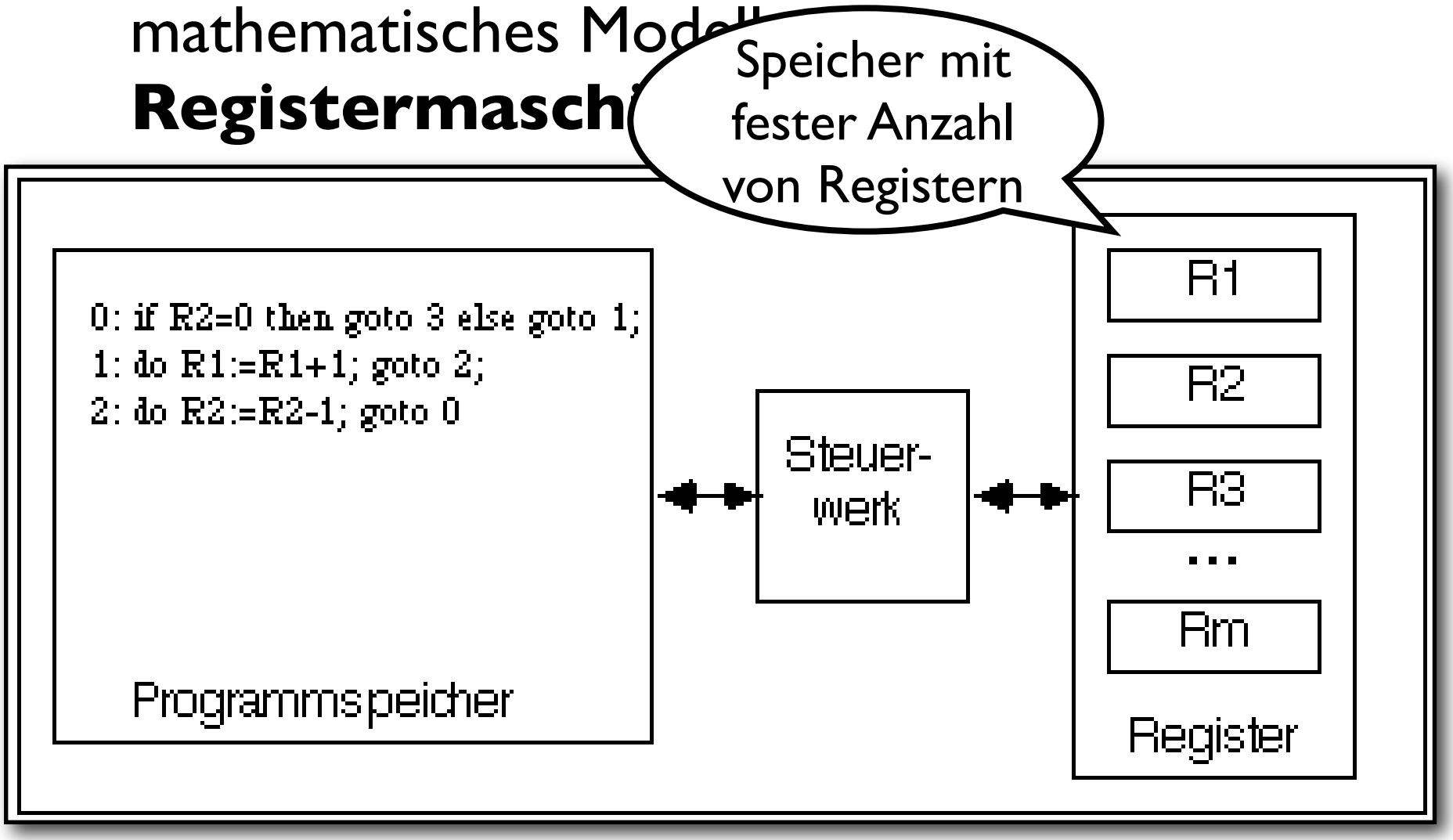


Programmierstile

Imperative Programmierung - Modell

mathematisches Mod

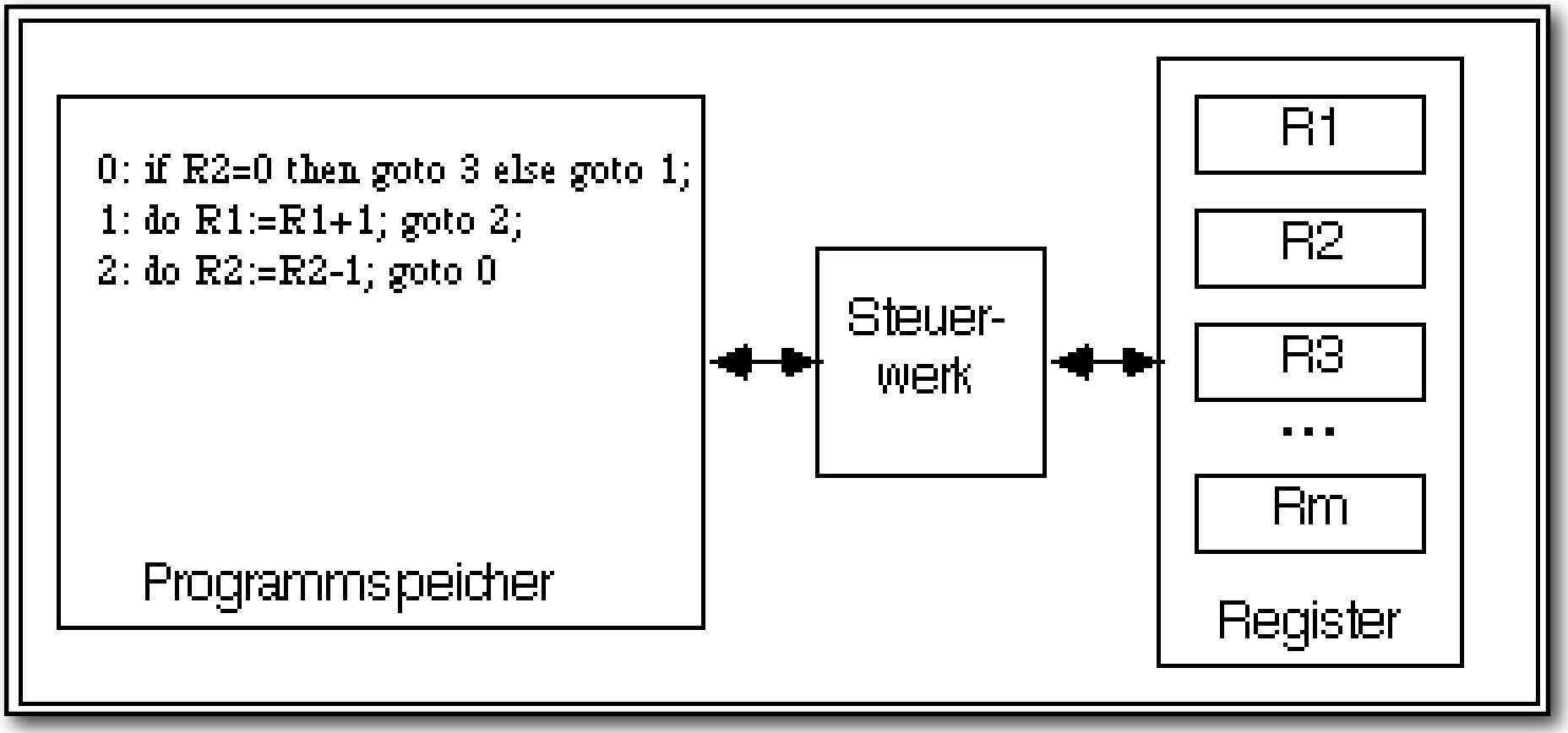
Registermasch



Programmierstile

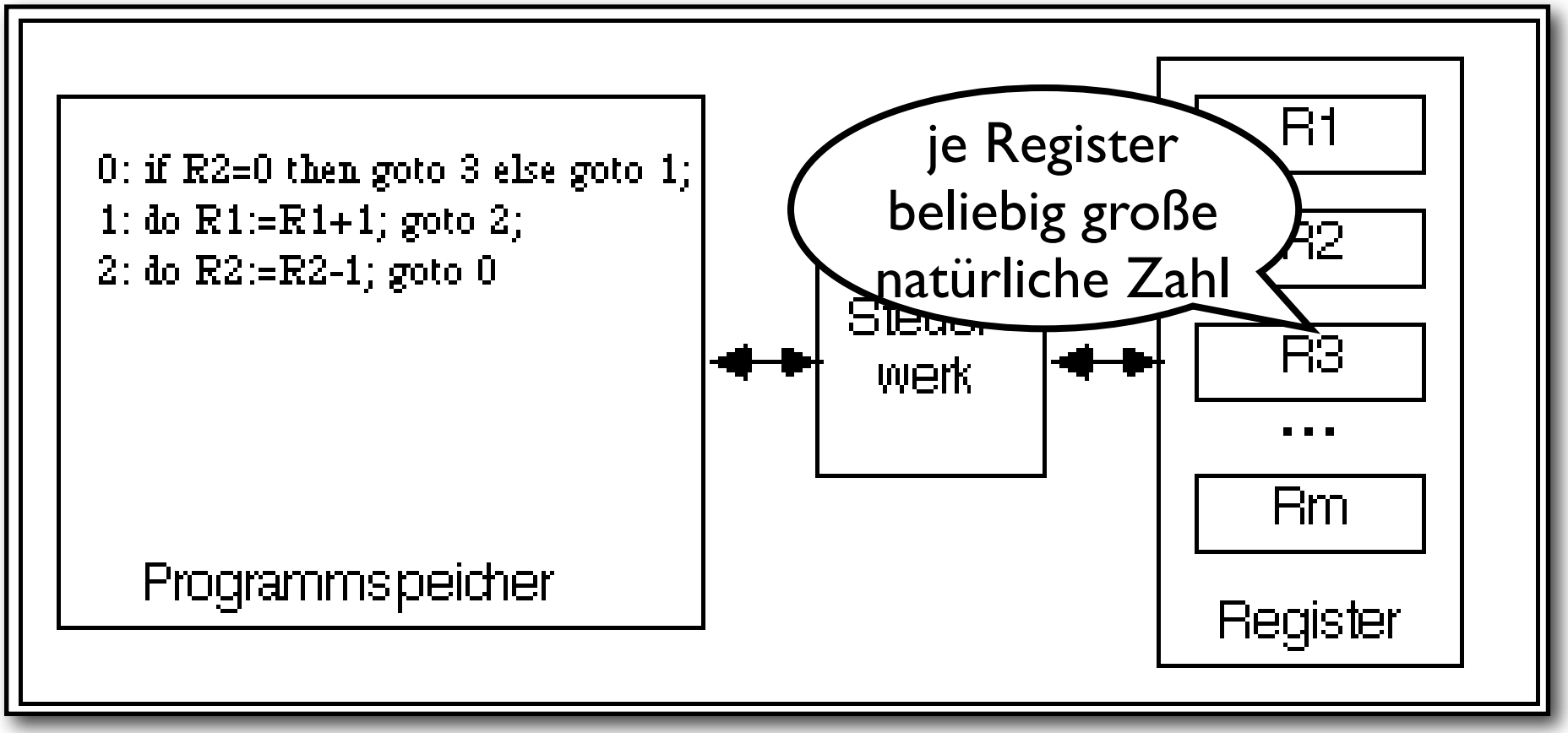
Imperative Programmierung - Modell

mathematisches Modell:
Registermaschine.



Programmierstile
Imperative Programmierung - Modell

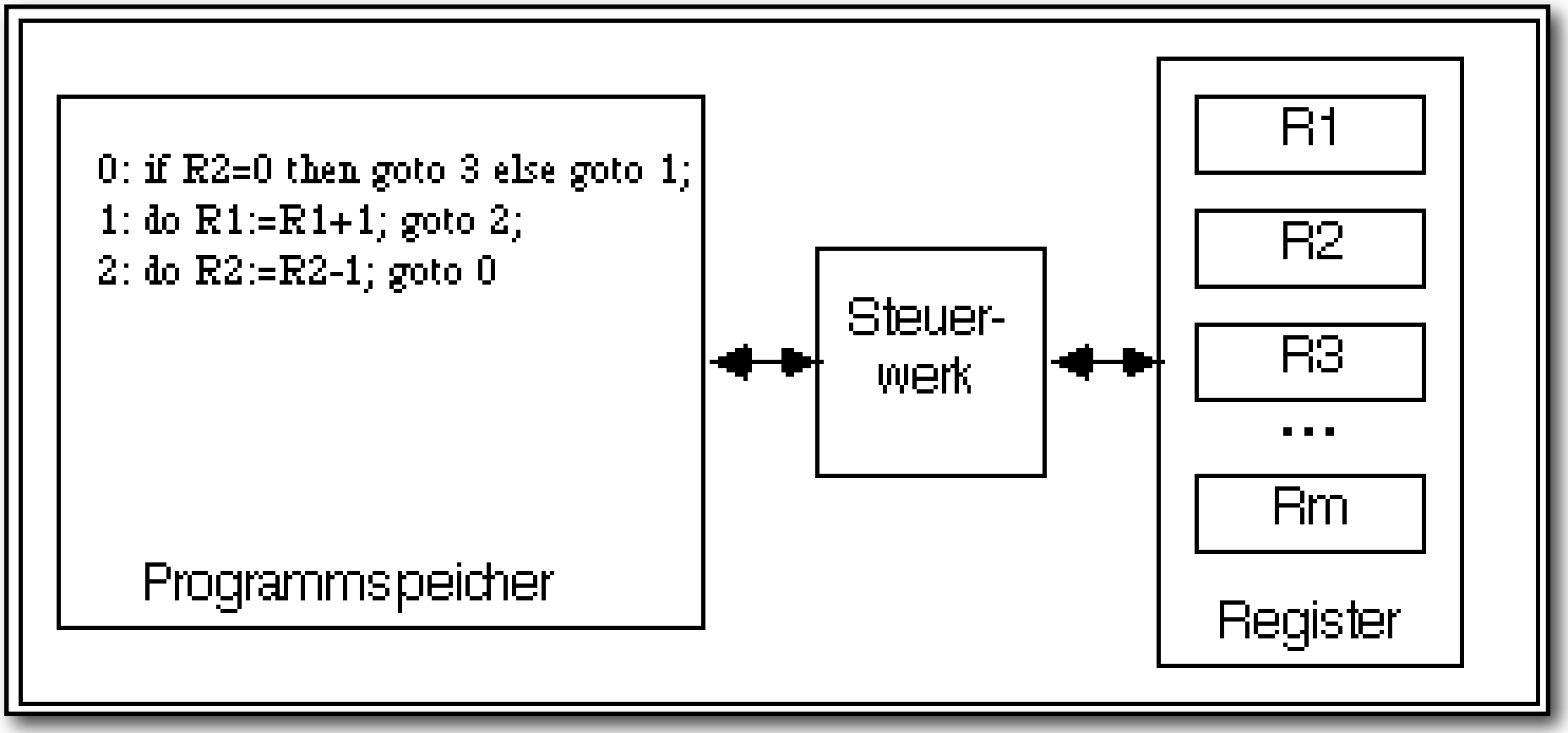
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

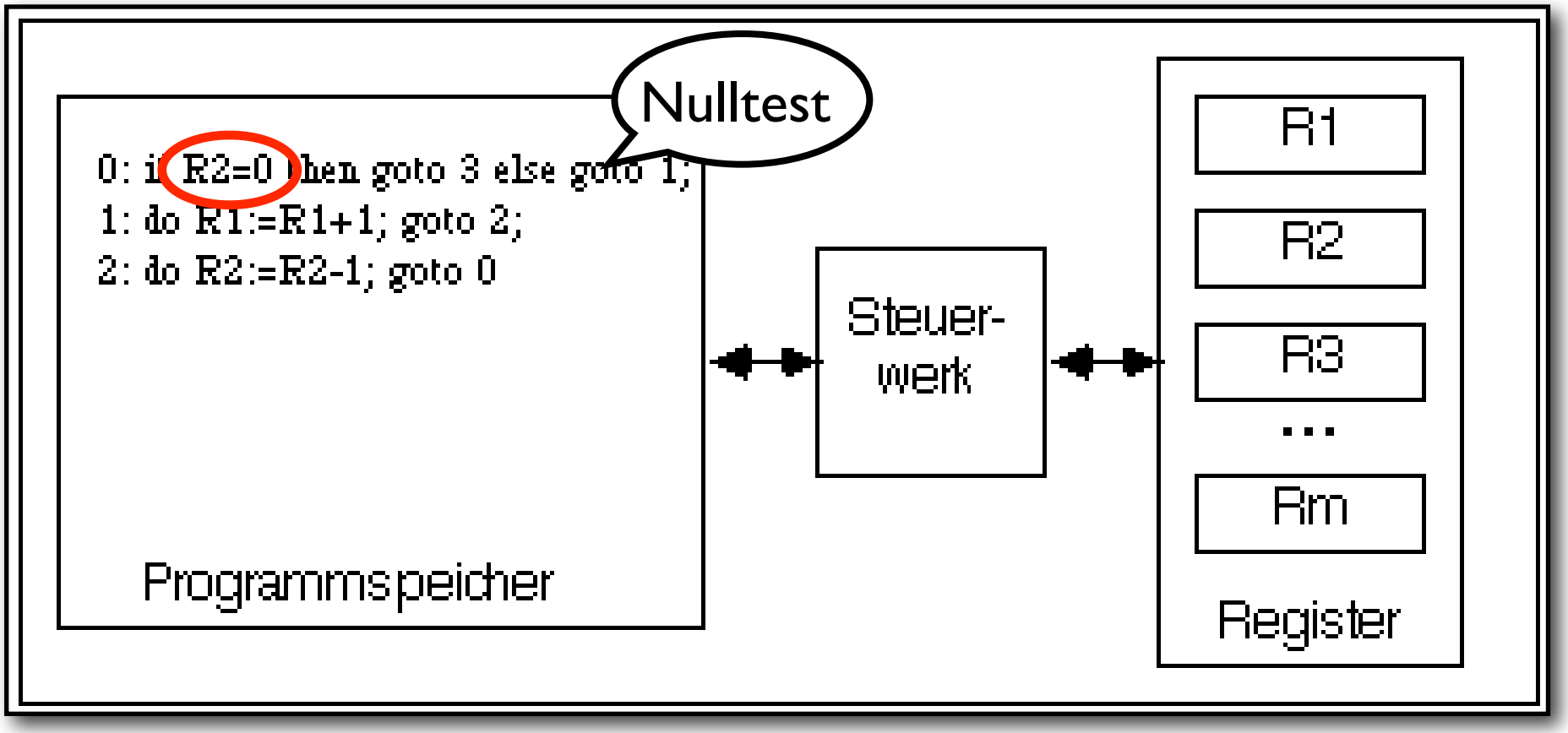
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

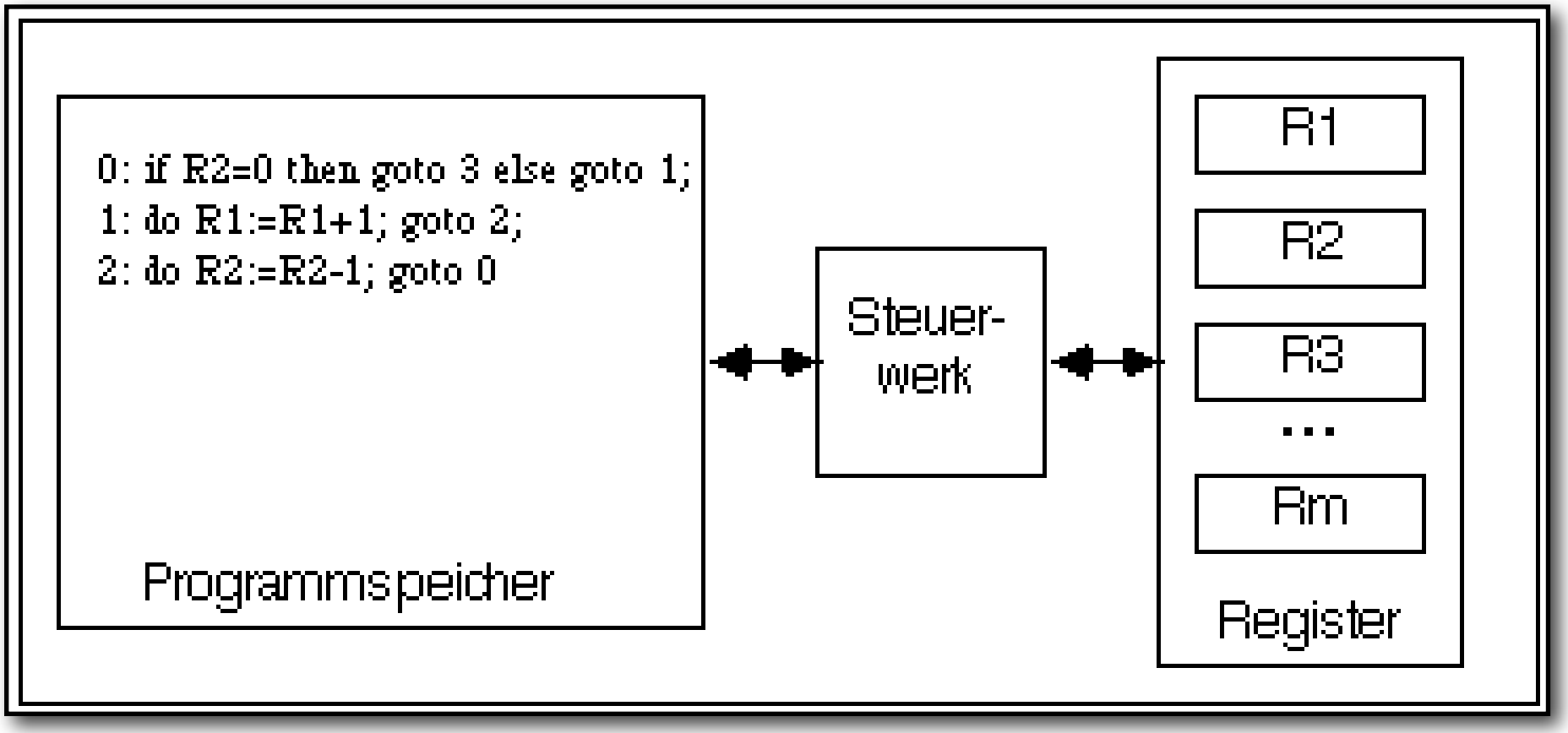
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

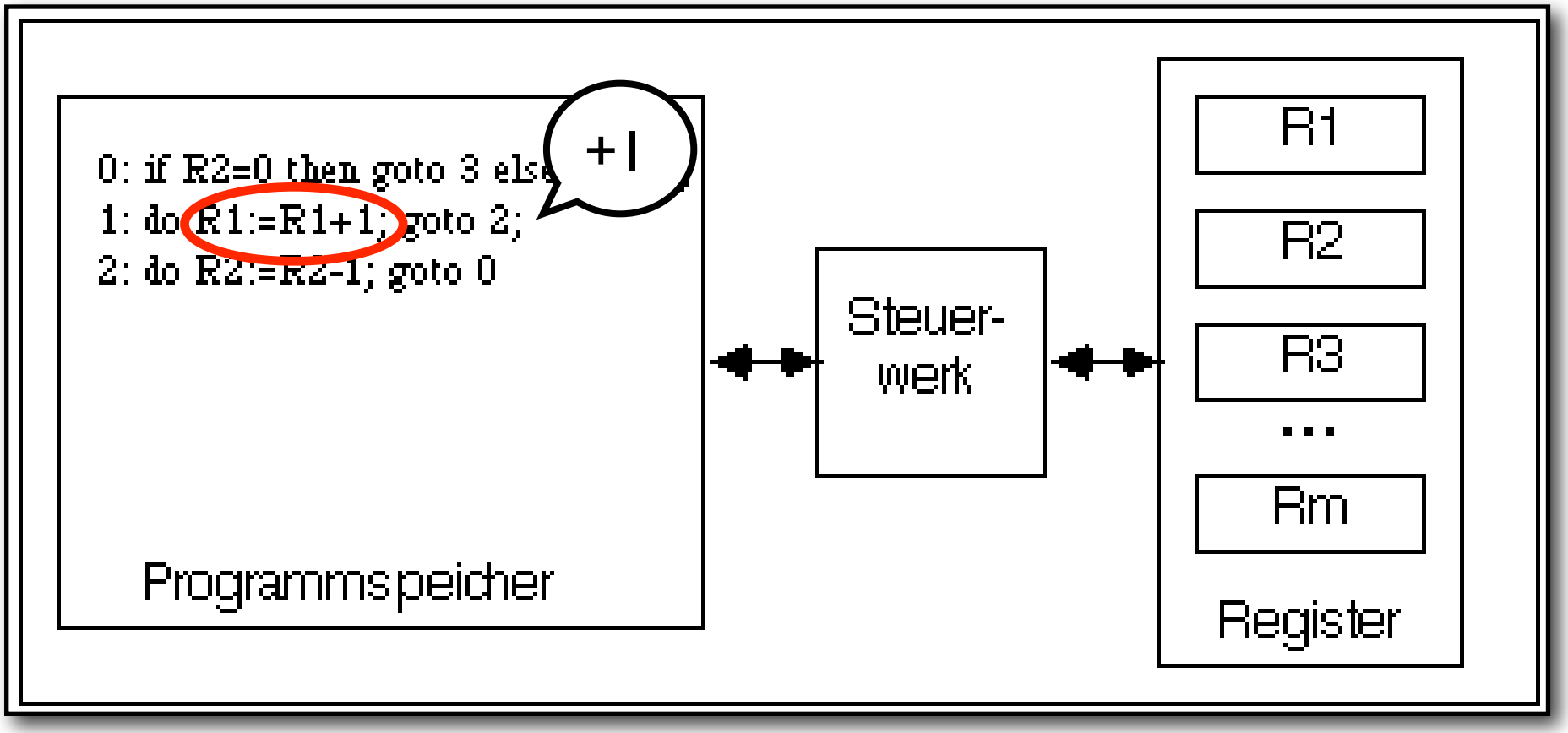
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

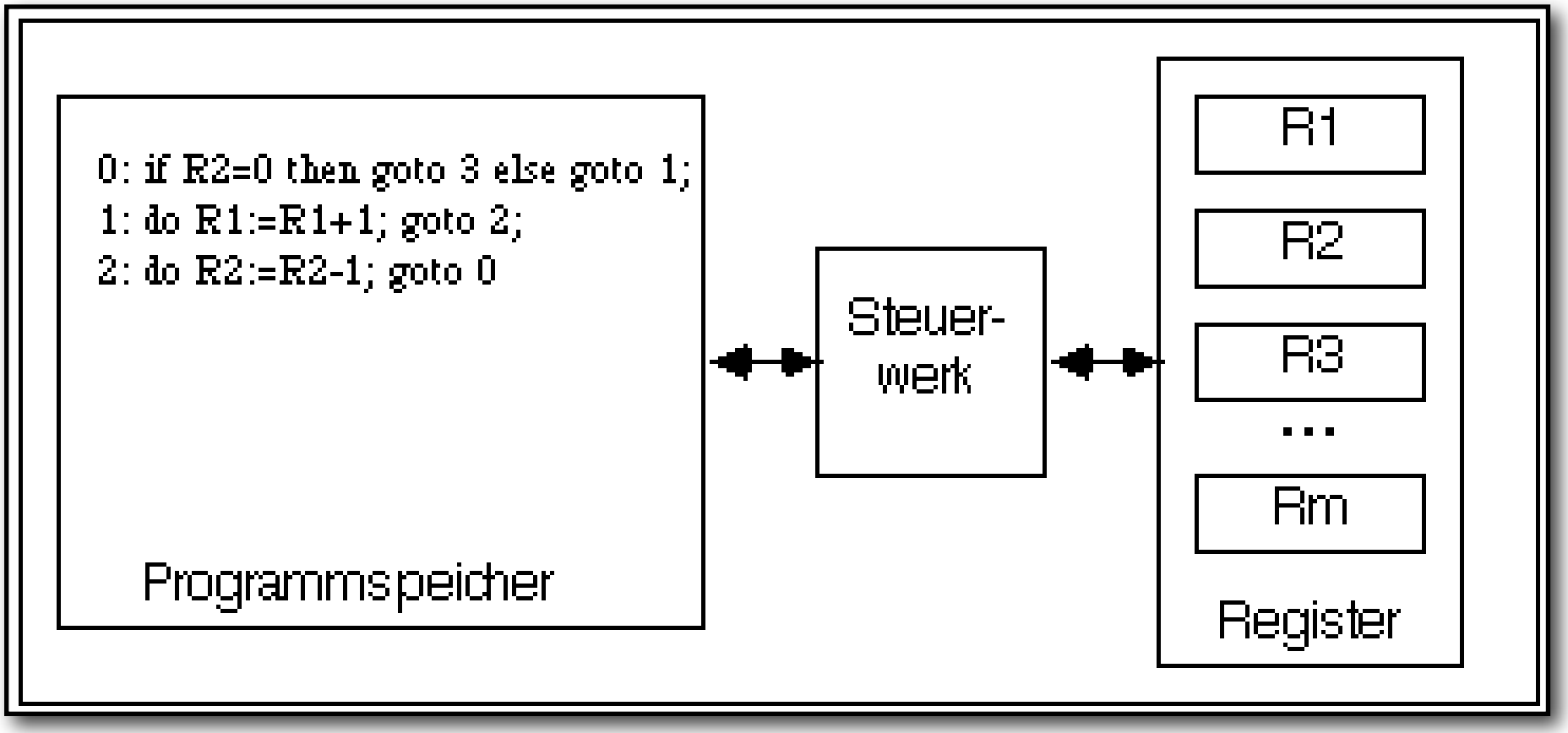
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

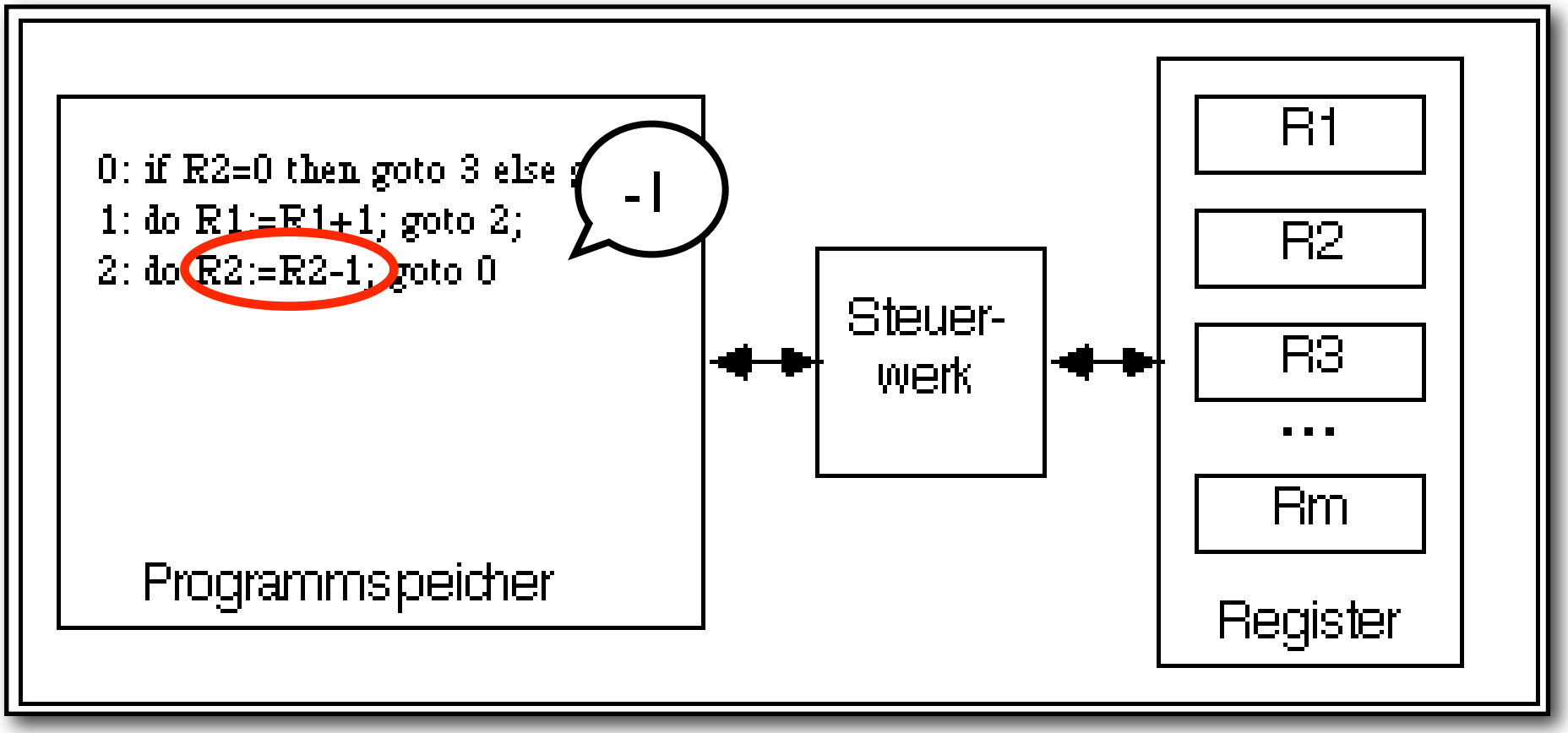
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

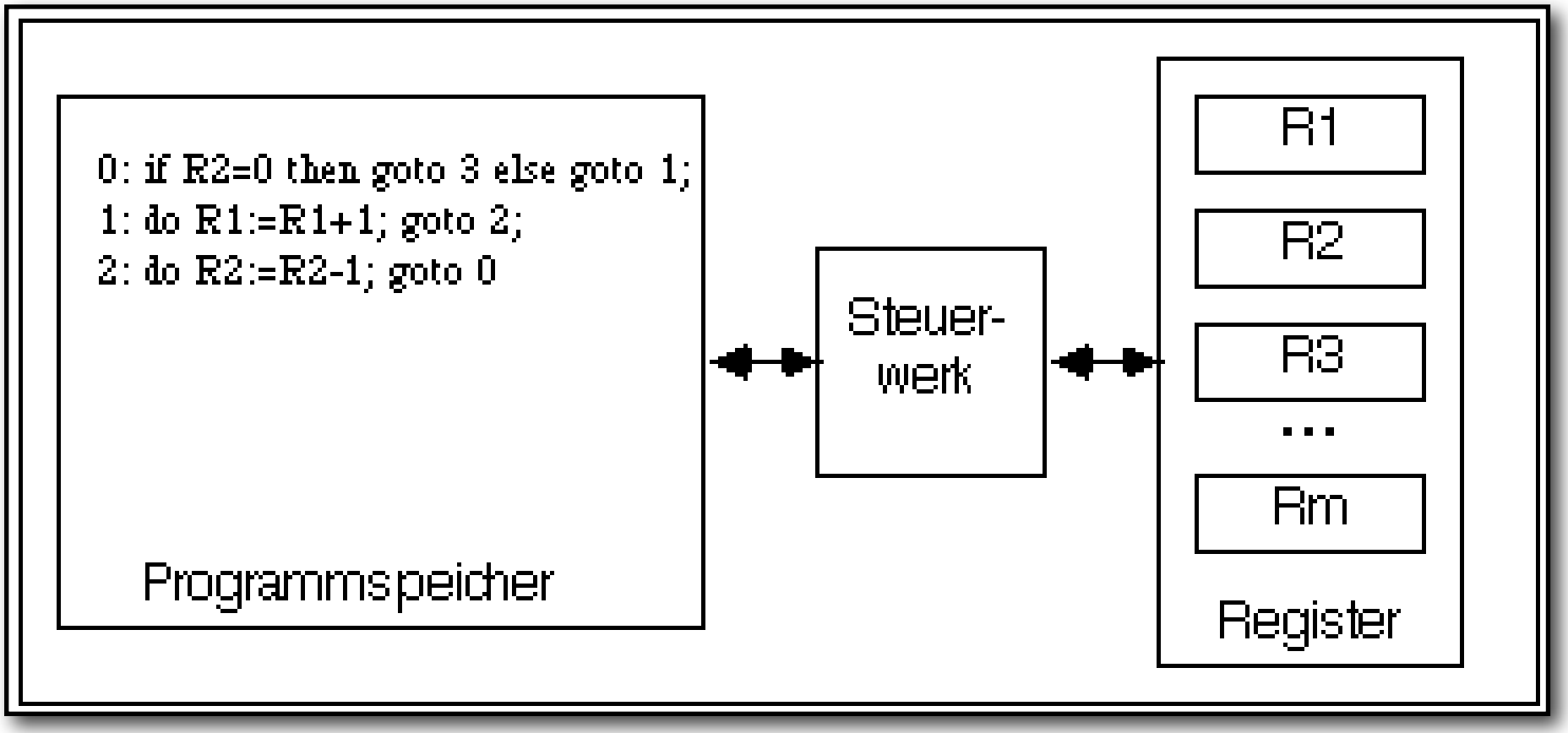
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

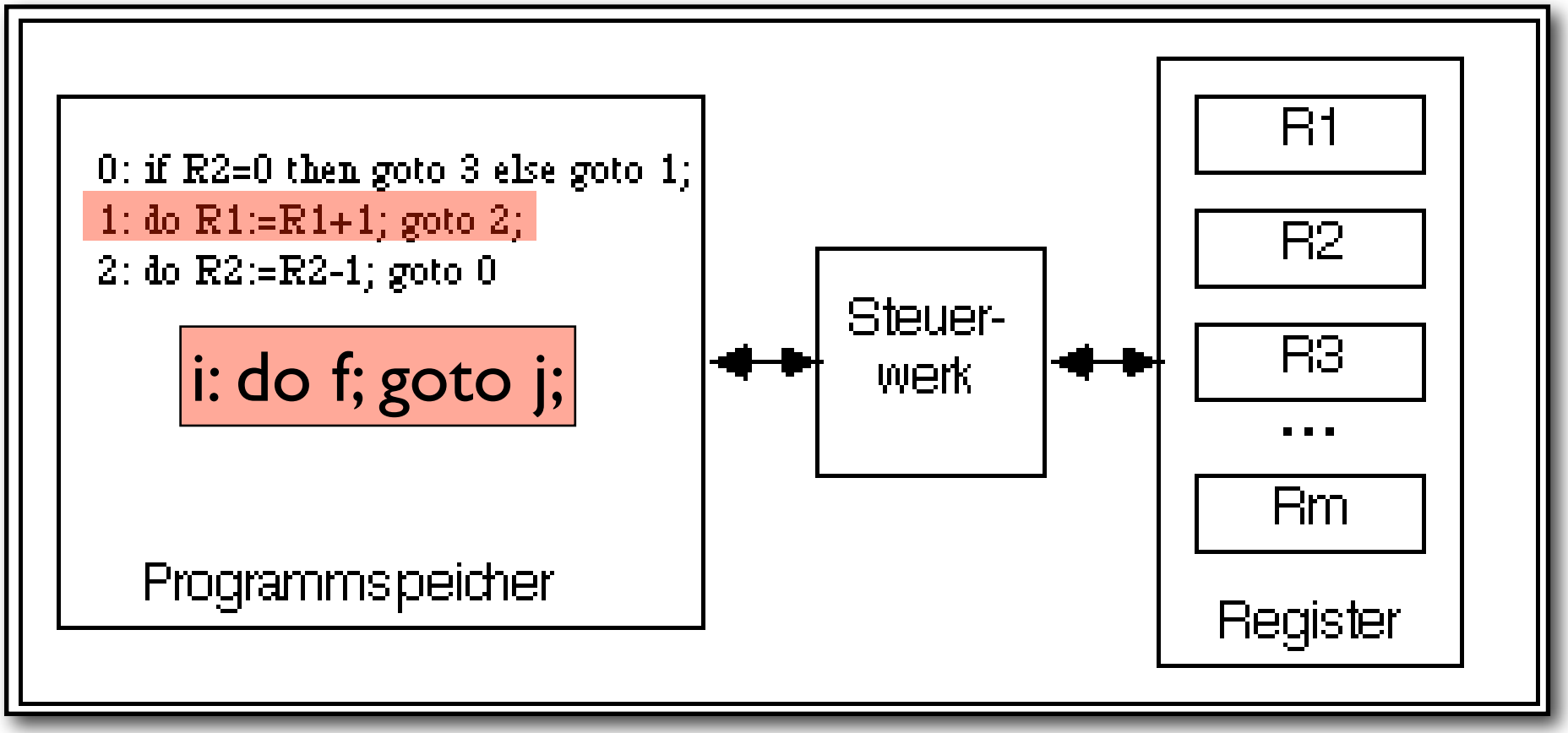
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

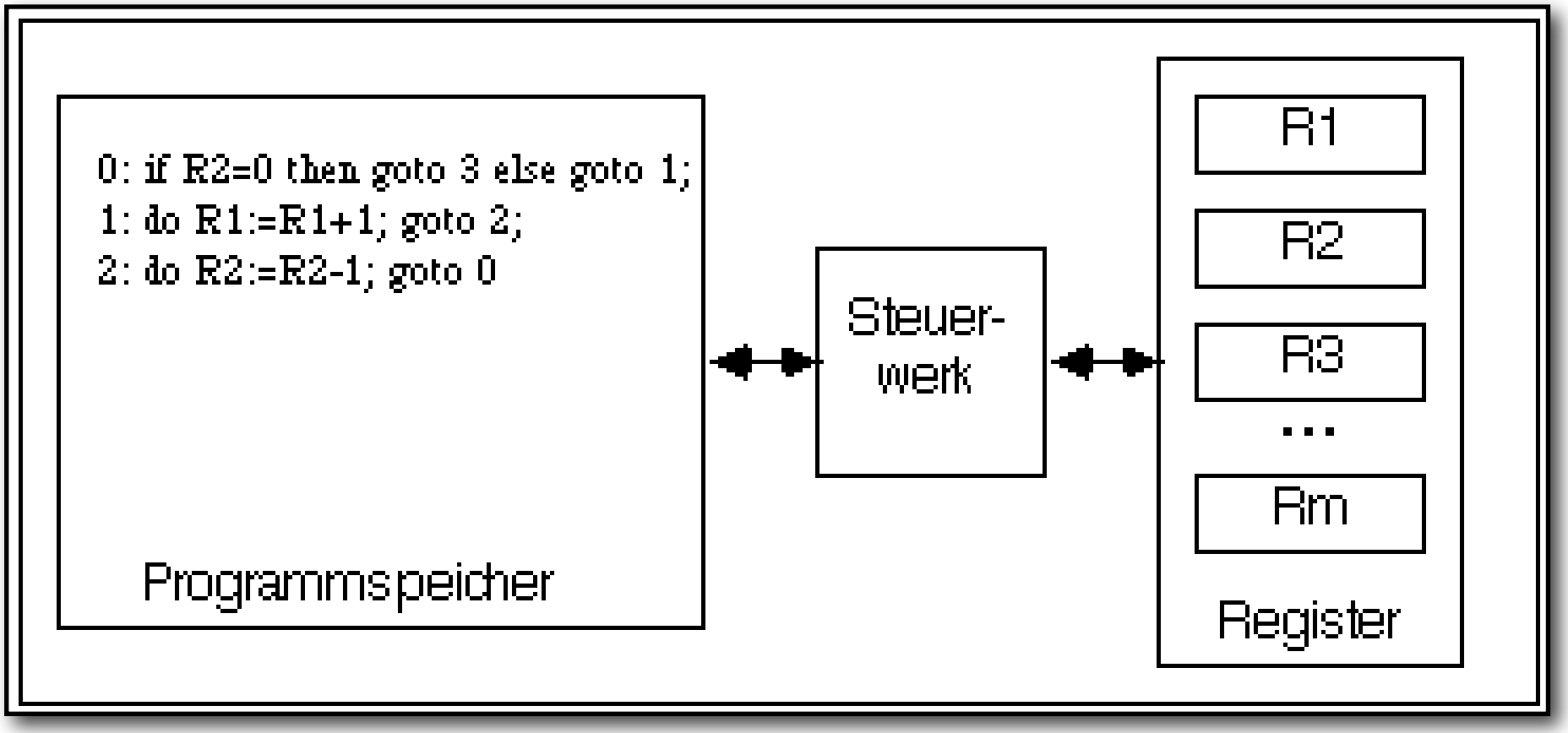
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

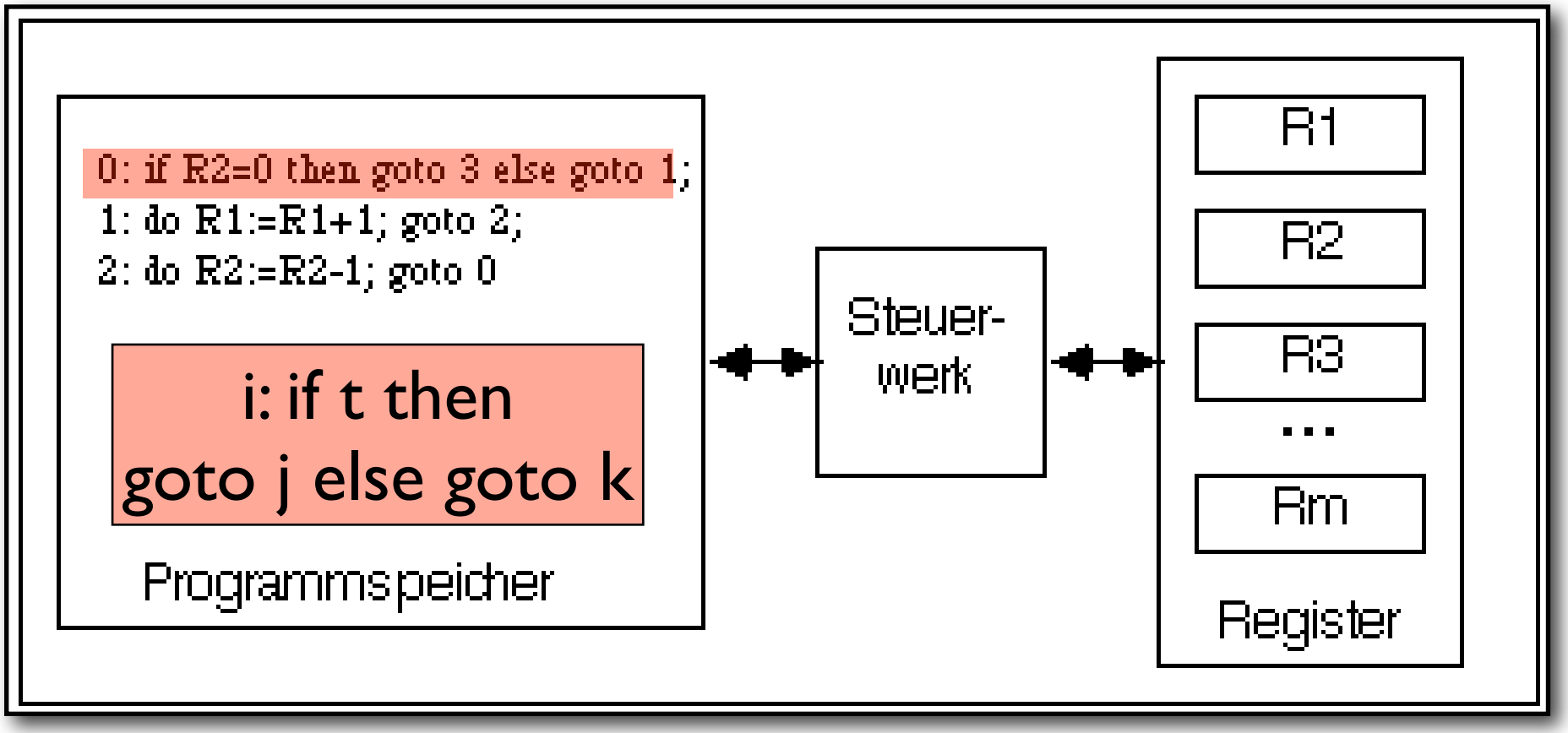
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

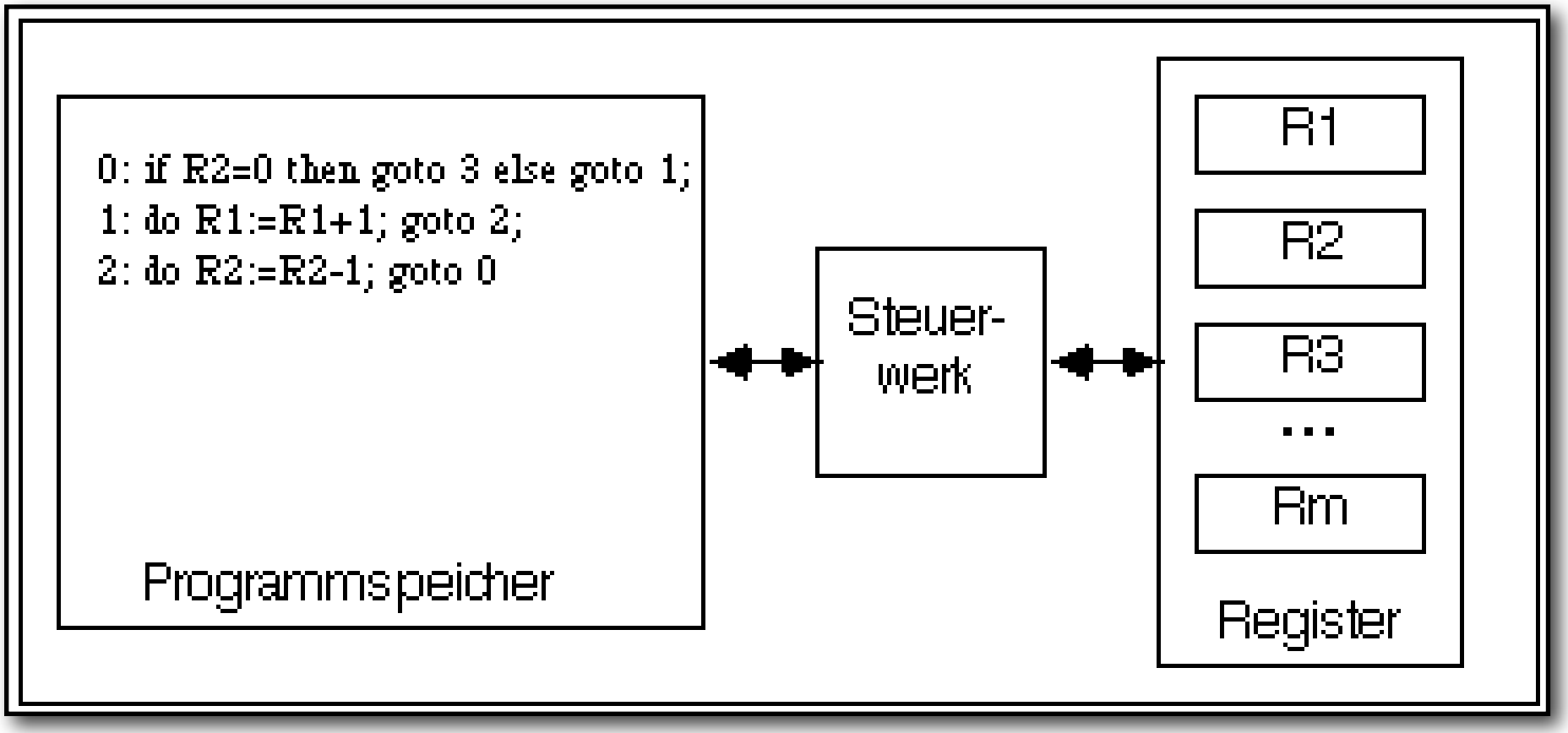
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

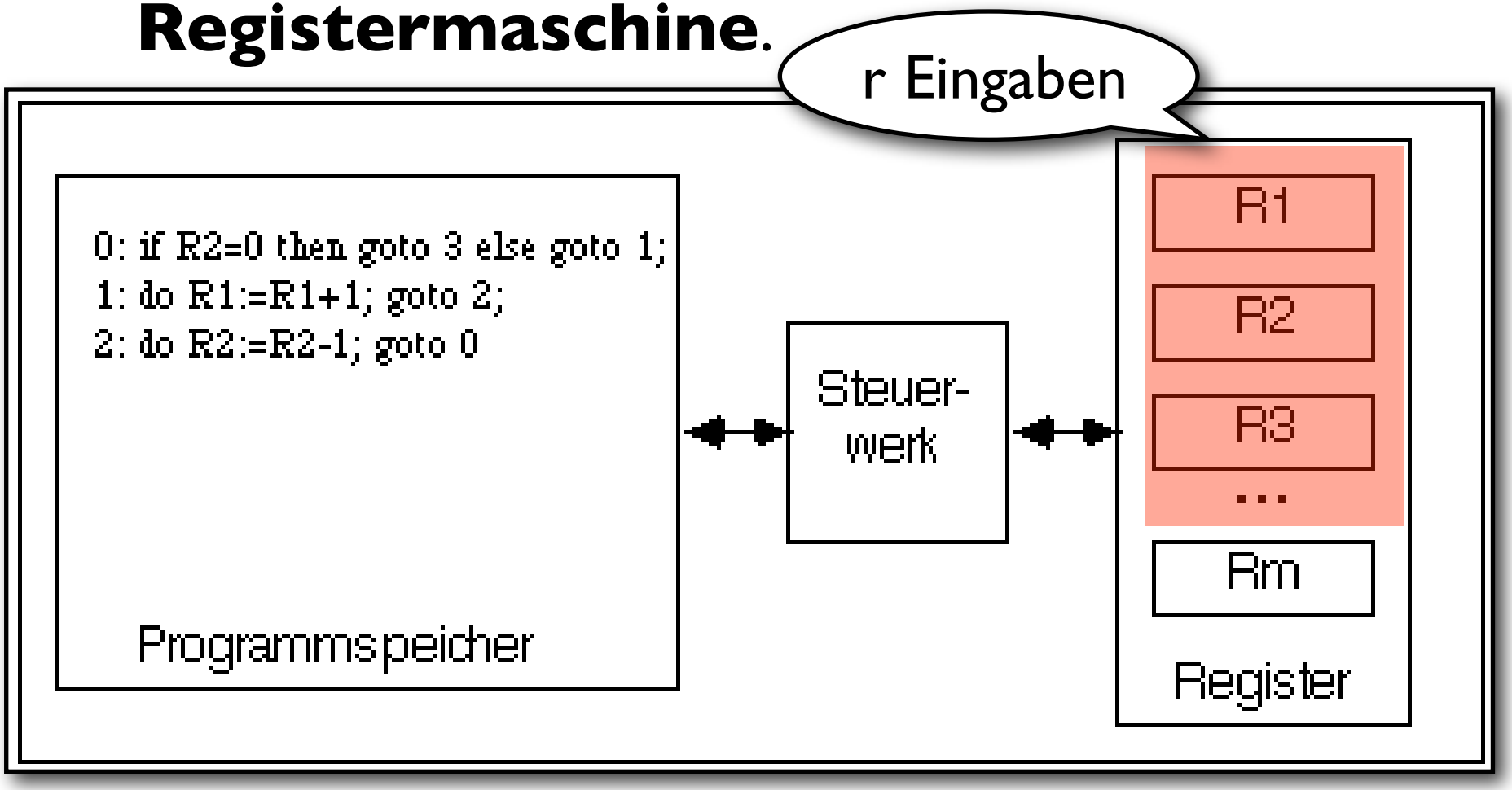
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

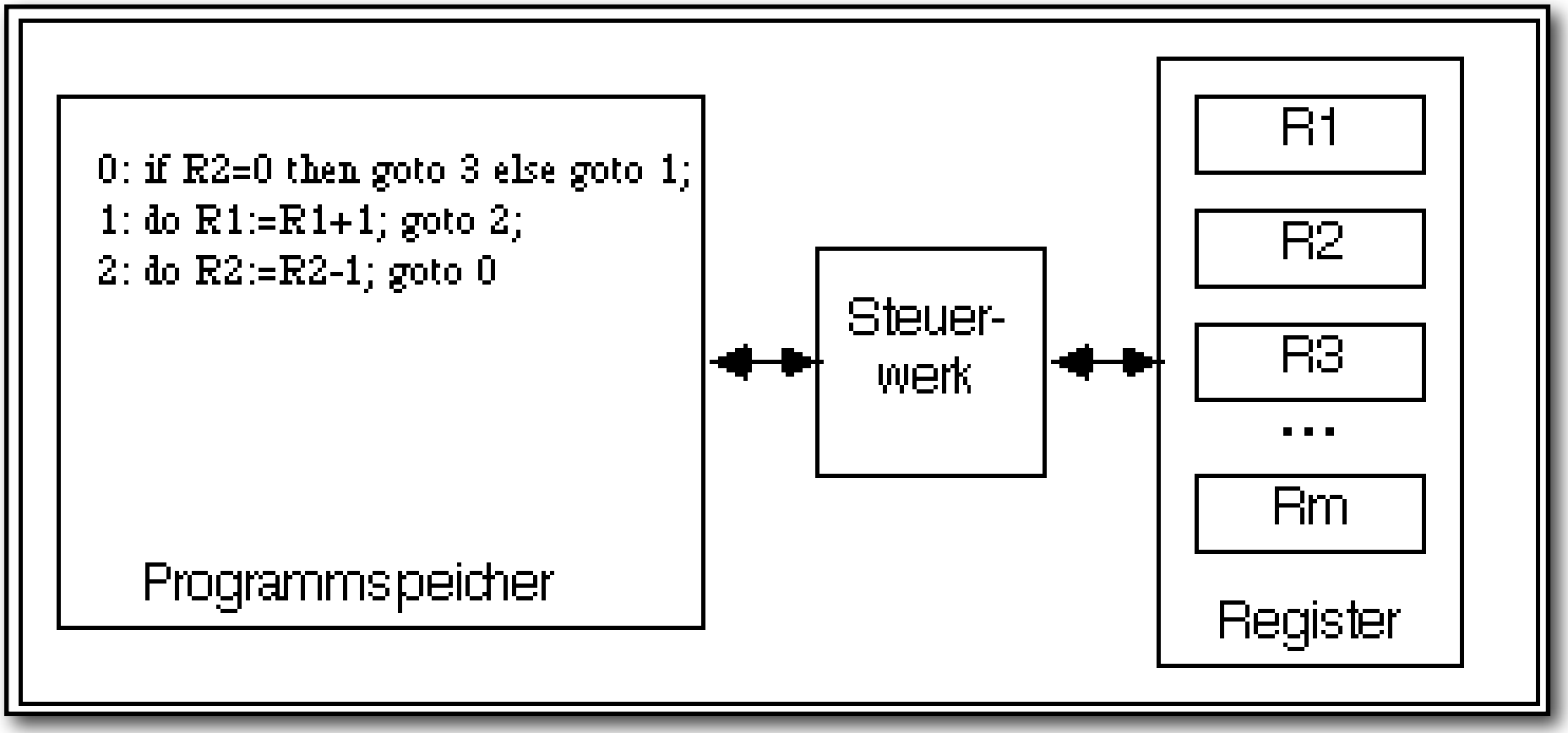
mathematisches Modell:
Registermaschine.



Programmierstile

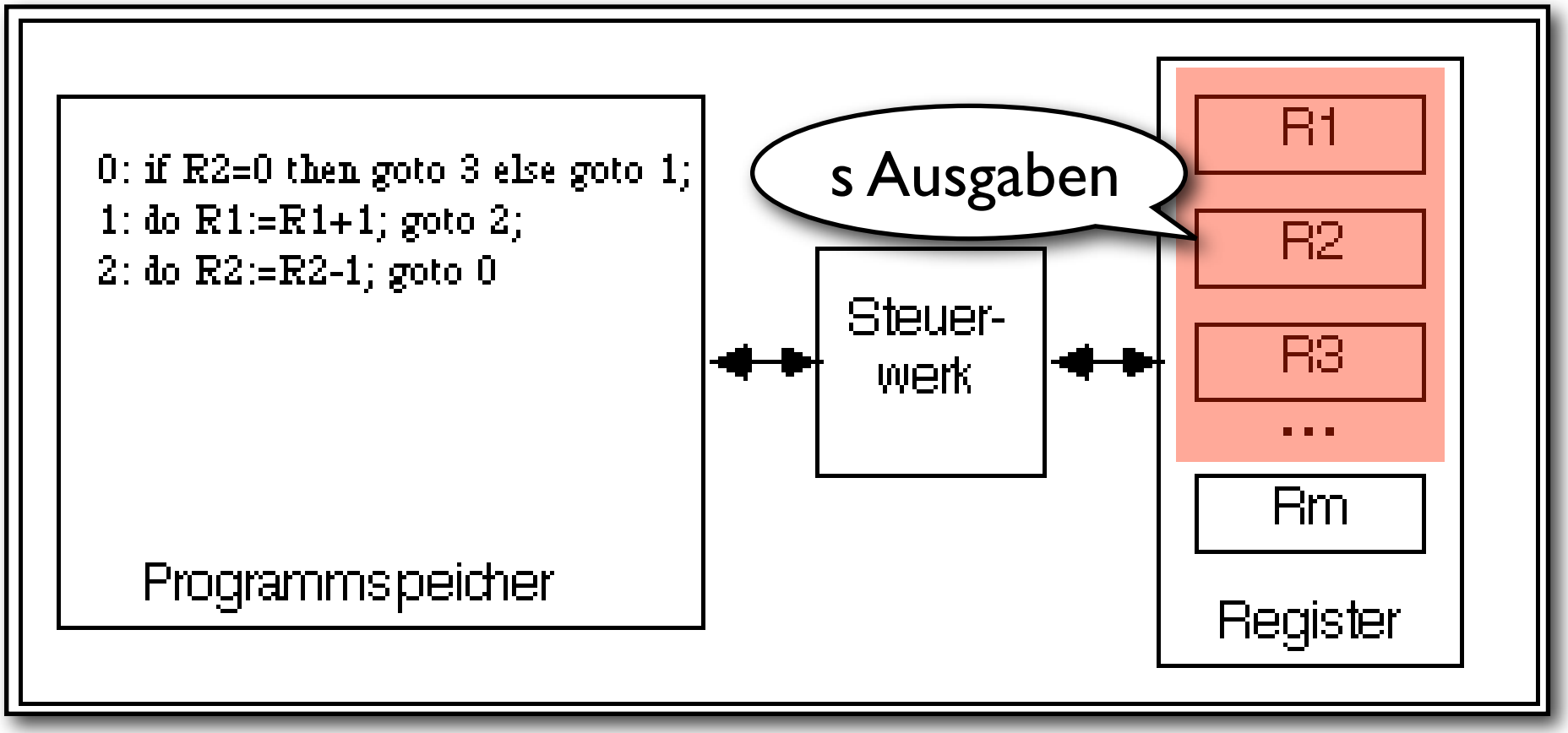
Imperative Programmierung - Modell

mathematisches Modell:
Registermaschine.



Programmierstile
Imperative Programmierung - Modell

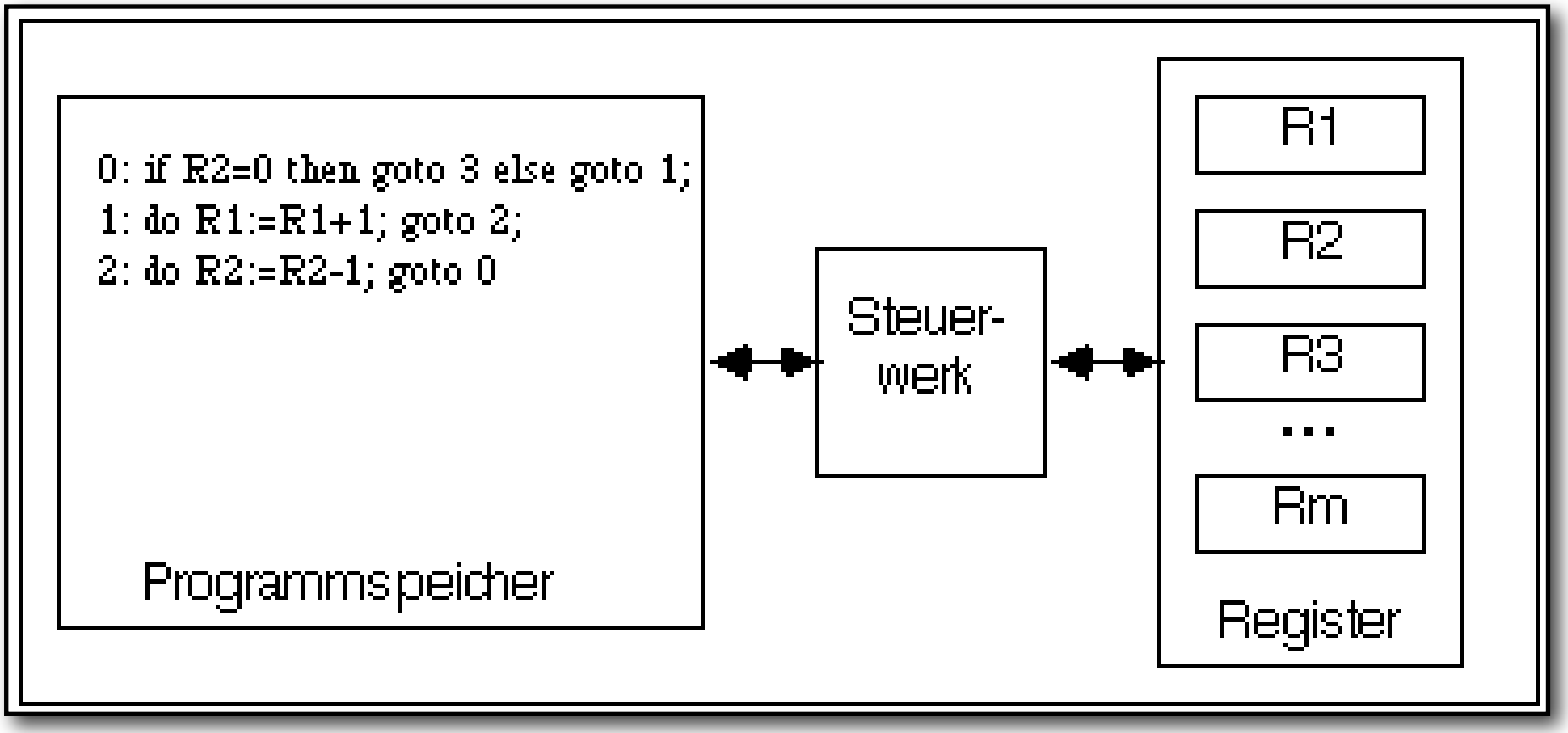
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

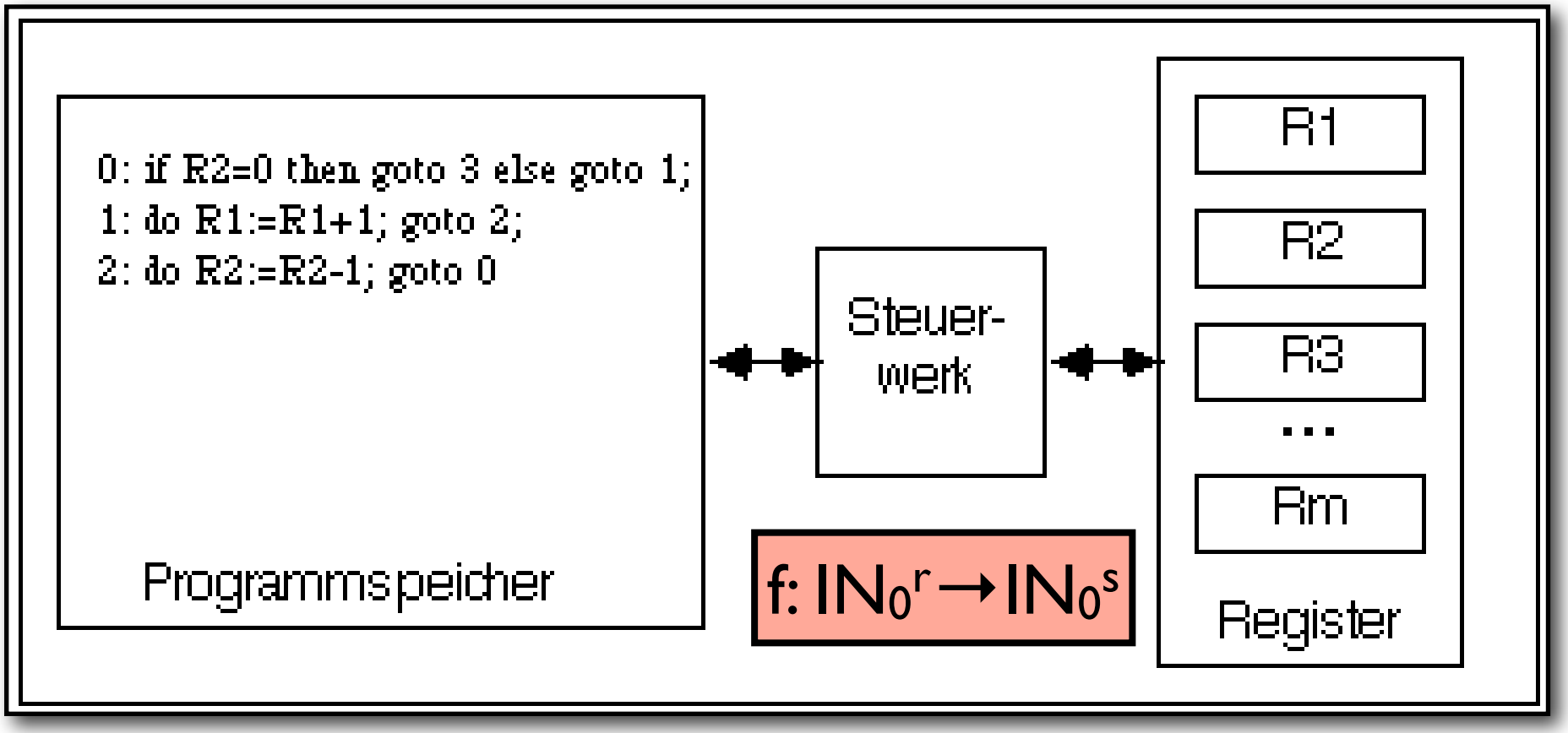
mathematisches Modell:
Registermaschine.



Programmierstile

Imperative Programmierung - Modell

mathematisches Modell:
Registermaschine.



Programmierstile
Leistungsfähigkeit der Registermaschine

Registermaschine unerwartet leistungsfähig

Berechnungsvollständigkeit:

Man kann zeigen: zu jeder berechenbaren Funktion f gibt es eine Registermaschine mit höchstens drei (!) Registern, die f berechnet.

Programmierstile
imperat. Programmierung - Programm

- **Programm**: nach bestimmten Konstruktionsprinzipien aufgebaute Zusammenstellung von Befehlen (lat. imperare=befehlen) oder Anweisungen
- **Start eines Programms**: Ausführung des ersten Befehls

Programmierstile
imperat. Programmierung - Anweisung

- **Anweisung**: Aufforderung an den Computer, eine Handlung auszuführen.
  - elementare Anweisungen: Zuweisung, ..
  - strukturierte Anweisungen vermöge Konstruktoren: Konkatenation, Alternative, Iteration, ...

Programmierstile
imperat. Programmierung - Variablen

- **Variablenkonzept**: Variable=Behälter mit Bezeichner und Typ; Operationen: *Lesen* und *Schreiben*.
- Ursprung des Variablenkonzepts: Rechnerarchitektur (Von Neumann-Architektur) mit Speichern aus einzelnen Speicherzellen; Speicherzelle=Standardbehälter für Variablen.

Programmierstile
imperat. Programmierung - Historie

- 1950: Maschinen- und Assemblersprachen
- unstrukturierte Programmiersprachen: FORTRAN (1954), COBOL (1959) und BASIC (1962)
- strukturierte Sprachen: ALGOL60 (1958), PASCAL (1970), C (1974), MODULA-2 (1977), ADA (1979) - gewisser Abschluß der Entwicklung
- spätere Sprachen nahezu ausschließlich objektorientierte Erweiterungen.

Programmierstile

imperat. Programmierung - Mischen

Mischen in PRO:

```
def s1, s2 s3: Zahlenfolge;
solange s1 ≠ [ ] und s2 ≠ [ ] tue
    wenn erstes(s1) < erstes(s2) dann
        s3 ← s3 • [ erstes(s1) ];
        s1 ← rest(s1)
    sonst
        s3 ← s3 • [ erstes(s2) ];
        s2 ← rest(s2)
    ende
ende;
s3 ← s3 • s1 • s2.
```

Programmierstile

imperat. Programmierung - Mischen

Mischen in PRO:

```
def s1, s2 s3: Zahlenfolge;
solange s1 ≠ [ ] und s2 ≠ [ ] tue
    wenn erstes(s1) < erstes(s2) dann
        s3 ← s3 • [ erstes(s1) ];
        s1 ← rest(s1)
    sonst
        s3 ← s3 • [ erstes(s2) ];
        s2 ← rest(s2)
    ende
ende;
s3 ← s3 • s1 • s2.
```

Start des Programms: run misch.

Programmierstile

imperat. Programmierung - Mischen

imper schen

```
program misch(f,g,h);
var f,g,h: file of integer;
begin
   reset(f); reset(g); rewrite(h);
   while not (eof(f) or eof(g)) do
   begin
        if f↑ < g↑ then
        begin
             h↑:=f↑; put(h); get(f)
        end else
        begin
             h↑:=g↑; put(h); get(g)
        end
   end;
   while not eof(f) do
   begin
        h↑:=f↑; put(h); get(f)
   end;
   while not eof(g) do
   begin
        h↑:=g↑; put(h); get(g)
   end
end.
```

Programmierstile
funkt. Programmierung - Kalkül

- λ-Kalkül
- Anfang der 30er Jahre von A. Church entwickelt
- Zweck: Präzisierung des Begriffs der Berechenbarkeit
- mathematischer Formalismus zur Beschreibung von Definition und Anwendung von Funktionen, die durch *Rechenvorschriften* gegeben sind
- sehr elementare, aber überaus mächtige Operationen

Programmierstile
funkt. Programmierung - λ-Kalkül

- *Formeln* (λ-*Ausdrücke*, λ-*Terme*)=”Programme”, induktiv definiert
- $X=\{x_1,x_2,x_3,...\}$ Menge von Variablen

Programmierstile
funkt. Programmierung - λ-Terme

1. Jede Variable $x \in X$ ist ein λ-Term.
2. M und N λ-Terme, dann auch (MN)
   - **Anwendung** des λ-Terms M auf den λ-Term N als Argument; bekannter M(N)
3. $x \in X$ Variable, M λ-Term, dann auch (λx.M)
   - **Abstraktion**: Übergang von einem λ-Term M zu einer Funktion
     - x formaler Parameter
     - M Funktionsrumpf
     - λ=function

Programmierstile

funkt. Programmierung - λ-Terme

λ	x	.	M
function f	x	=	M

Programmierstile

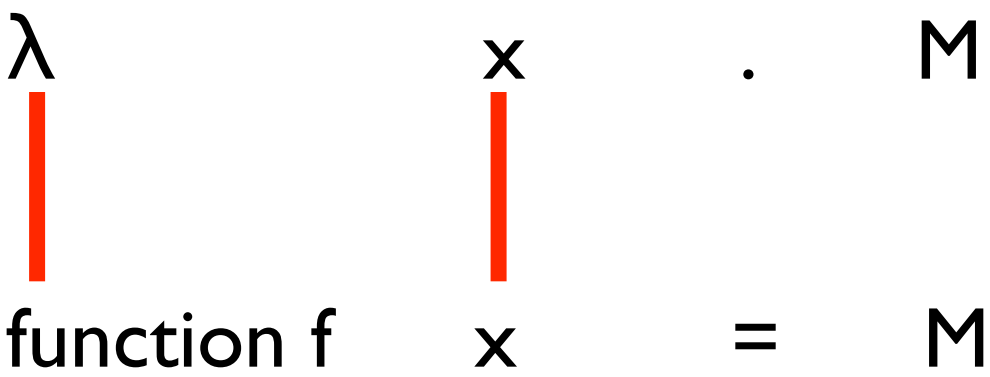
funkt. Programmierung - λ-Terme

λ x . M

function f x = M

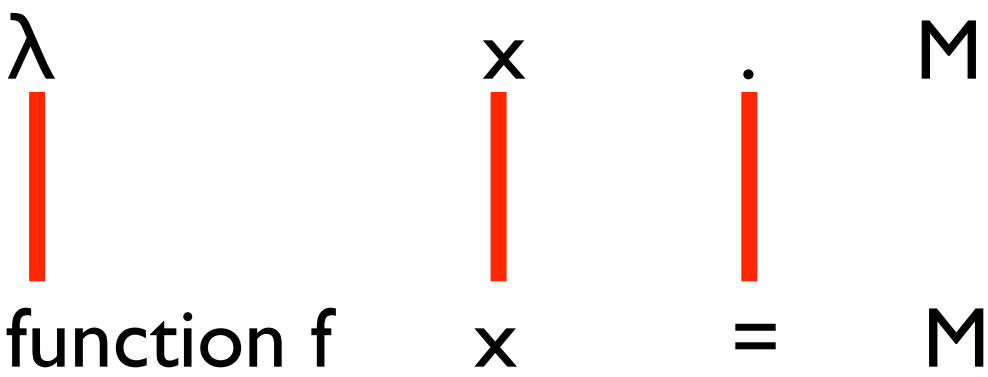
Programmierstile

funkt. Programmierung - λ-Terme



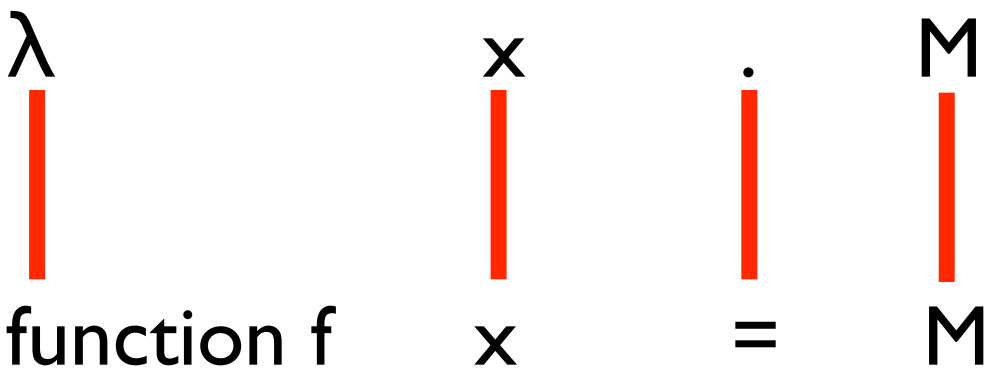
Programmierstile

funkt. Programmierung - λ-Terme



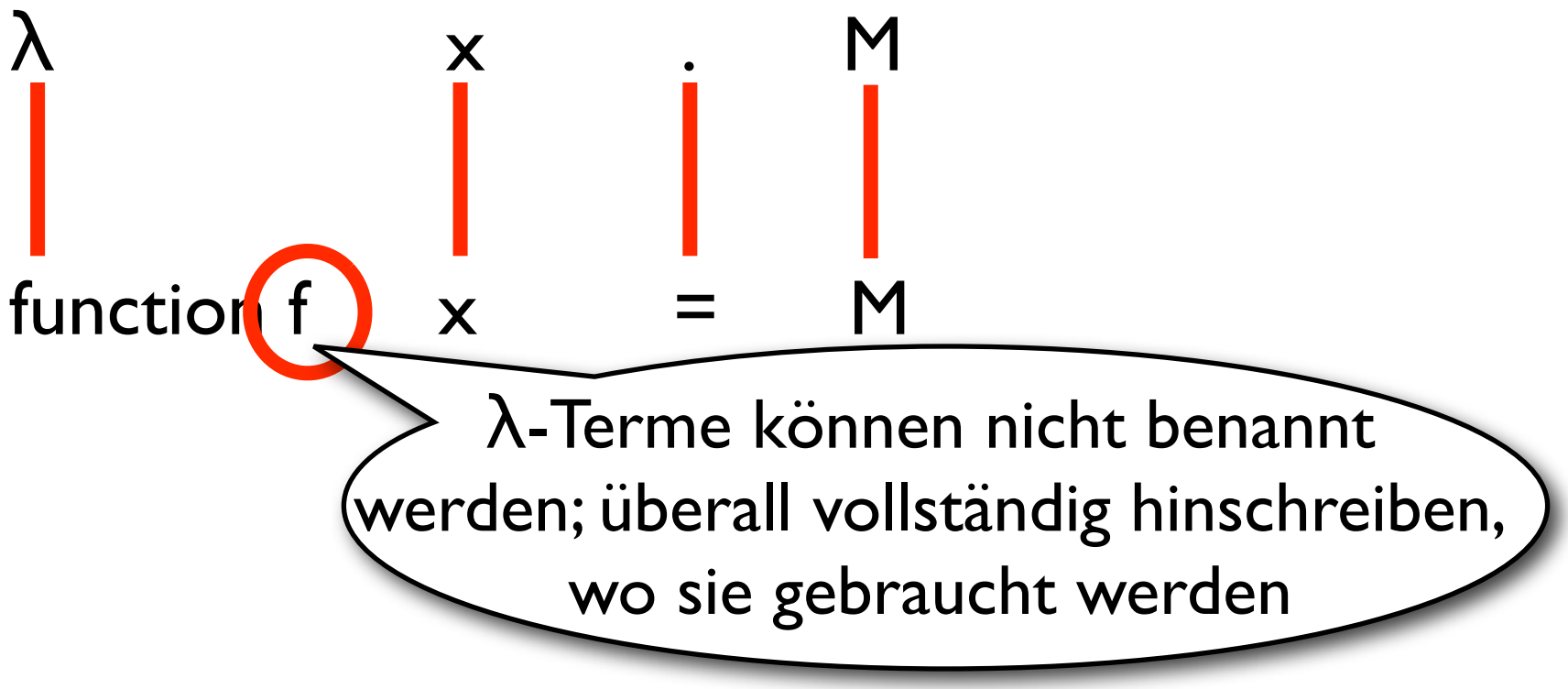
Programmierstile

funkt. Programmierung - λ-Terme



Programmierstile

funkt. Programmierung - λ-Terme



Programmierstile
funkt. Programmierung - Programm

- funktionales Programm = Folge von benannten λ-Termen

$$f_1=T_1, ..., f_n=T_n.$$

- Hier: $f_1, ..., f_n$ Funktionsbezeichner.

Programmierstile
λ-Terme - Bindung

Bezeichnungen:

- *gebundenes* Auftreten von x: Auftreten von x in M im λ-Term (λx.M)
- *freies* Auftreten von x: Auftreten von x im Term N außerhalb einer λ-Abstraktion (λx.M)

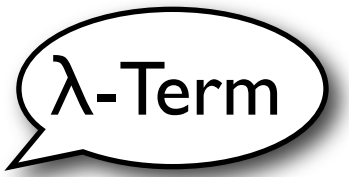
Programmierstile
λ-Terme - Funktionsweise

- Wie rechnet man mit λ-Termen?
- Man wendet sie auf Argumente an: **β-Regel**
- die β-Regel beschreibt Ersetzung formaler durch aktueller Parameter
- Übergabeart: call-by-name (textuelle Ersetzung)
- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.

Programmierstile
λ-Terme - β-Regel

- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.

Programmierstile
λ-Terme - β-Regel



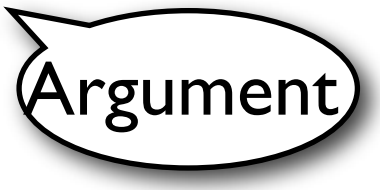
- β-Regel: ((λx.M)N)=M[x←N].

Programmierstile
λ-Terme - β-Regel

- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.

Programmierstile
λ-Terme - β-Regel

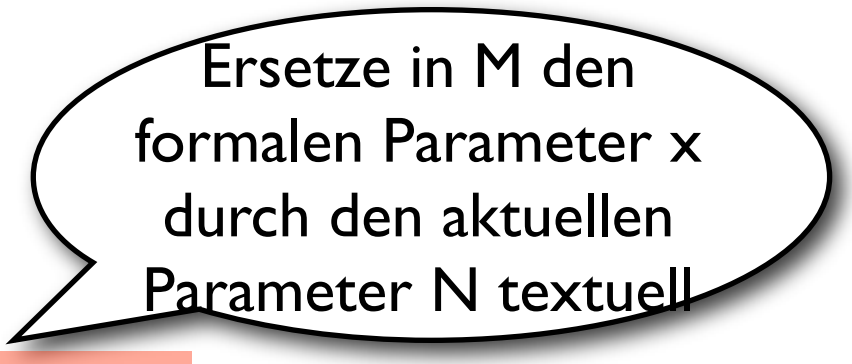
- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.



Programmierstile
λ-Terme - β-Regel

- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.

Programmierstile
λ-Terme - β-Regel



- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.

Programmierstile
λ-Terme - β-Regel

- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.

Programmierstile
λ-Terme - β-Regel

- β-Regel: $((\lambda x.M)N)=M[x \leftarrow N]$.

Die β-Regel ist zulässig, wenn durch $M[x \leftarrow N]$ keine in N freie Variable gebunden wird. Anderenfalls muß man Umbenennungen vornehmen, sog. α-*Konversionen*.

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

((λx.((xd)((λy.(xy))a)))b)

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a \rightarrow (xa)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a \rightarrow (xa)$$

$$((\lambda x.((xd)(xa)))b)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a \rightarrow (xa)$$

$$((\lambda x.((xd)(xa)))b) \rightarrow (bd)(ba)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a \rightarrow (xa)$$

$$((\lambda x.((xd)(xa)))b) \rightarrow (bd)(ba)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a \rightarrow (xa)$$

$$((\lambda x.((xd)(xa)))b) \rightarrow (bd)(ba)$$

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a \rightarrow (xa)$$

$$((\lambda x.((xd)(xa)))b) \rightarrow (bd)(ba)$$

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$\rightarrow (((bd)((\lambda y.(by))a)))$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$(\lambda y.(xy))a \rightarrow (xa)$$

$$((\lambda x.((xd)(xa)))b) \rightarrow (bd)(ba)$$

$$((\lambda x.((xd)((\lambda y.(xy))a)))b)$$

$$\rightarrow (((bd)((\lambda y.(by))a)))$$

$$\rightarrow (bd)(ba)$$

Programmierstile
λ-Terme - Beispiel

Beispiel:

$((\lambda x.((xd)((\lambda y.(xy))a)))b)$

$(\lambda y.(xy))a \rightarrow (xa)$

$(xa)))b) \rightarrow (bd)(ba)$

Konfluenz
Church-Rosser-Eigenschaft

$((\lambda x.((xd)((\lambda y.(xy))a)))b)$

$\rightarrow (((bd)((\lambda y.(by))a)))$

$\rightarrow (bd)(ba)$

Programmierstile
λ-Terme - Church-Rosser-Eigenschaft

- Beachte: fortlaufender Ersetzungsprozeß = Berechnung eines Funktionsergebnisses
- *Konfluenz* und *Church-Rosser-Eigenschaft*: Funktionsergebnis eindeutig (bis auf Umbenennung), egal in welcher Reihenfolge man die Auswertung eines λ-Terms durchführt

Programmierstile
λ-Kalkül - Mächtigkeit

- A. Church (1941): λ-Kalkül ist vollwertige Programmiersprache
- zu jeder berechenbaren Funktion ex. ein λ-Ausdruck, der sie beschreibt
- Beweisidee: Simulation von Registermaschinen durch λ-Ausdrücke: Zahlen, Register, +1, -1, =0, ...

Programmierstile

λ-Kalkül - Mächtigkeit - Beweisansatz

Simulation natürlicher Zahlen:

Wir definieren (zur besseren Lesbarkeit Klammern weitgehend weggelassen):

0	λ-Term	λf.λx.x,	
1	λ-Term	λf.λx.fx,	
2	λ-Term	λf.λx.f(fx),	
3	λ-Term	λf.λx.f(f(fx))	usw.

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

Nachfolgerfunktion +1: Konstruiere einen λ-Term N, der auf den λ-Termen für nat. Zahlen +1 realisiert:

$$N: \quad \lambda n.\lambda f.\lambda x.f(nfx)$$

Probe: Berechne

$$N(2): \quad (\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx))$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

Nachfolgerfunktion +1: Konstruiere einen λ-Term N, der auf den λ-Termen für nat. Zahlen +1 realisiert:

$$N: \quad \lambda n.\lambda f.\lambda x.f(nfx)$$

Probe: Berechne

$$N(2): \quad (\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx))$$

$$N$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

Nachfolgerfunktion +1: Konstruiere einen λ-Term N, der auf den λ-Termen für nat. Zahlen +1 realisiert:

$$N: \quad \lambda n.\lambda f.\lambda x.f(nfx)$$

Probe: Berechne

$$N(2): \quad (\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx))$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

Nachfolgerfunktion +1: Konstruiere einen λ-Term N, der auf den λ-Termen für nat. Zahlen +1 realisiert:

$$N: \quad \lambda n.\lambda f.\lambda x.f(nfx)$$

Probe: Berechne

$$N(2): \quad (\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx))$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

Nachfolgerfunktion +1: Konstruiere einen λ-Term N, der auf den λ-Termen für nat. Zahlen +1 realisiert:

$$N: \quad \lambda n.\lambda f.\lambda x.f(nfx)$$

Probe: Berechne

$$N(2): \quad (\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx))$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx))$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx))$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

$$\rightarrow \lambda f.\lambda x.f((\lambda x.f(fx))x)$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

$$\rightarrow \lambda f.\lambda x.f((\lambda x.f(fx))x)$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

$$\rightarrow \lambda f.\lambda x.f((\lambda x.f(fx))x) \rightarrow \lambda f.\lambda x.f(f(fx))$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

$$\rightarrow \lambda f.\lambda x.f((\lambda x.f(fx))x) \rightarrow \lambda f.\lambda x.f(f(fx))$$

Programmierstile

λ-Kalkül - Mächtigkeit - Nachfolgerfkt.

$$(\lambda n.\lambda f.\lambda x.f(nfx))(\lambda f.\lambda x.f(fx)) \rightarrow \lambda f.\lambda x.f((\lambda f.\lambda x.f(fx))fx)$$

$$\rightarrow \lambda f.\lambda x.f((\lambda x.f(fx))x) \rightarrow \lambda f.\lambda x.f(f(fx))$$

3

Programmierstile
funkt. Programmierung - Programm

- Programm: Menge von Funktionsdefinitionen (*Funktionalgleichungen*), die nach bestimmten Grundprinzipien aufgebaut sind
- Start eines Programms: Aufruf einer Funktion mit Parametern

Programmierstile
funkt. Programmierung - Funktion

- Funktion=Funktion im mathematischen Sinne, also Abb. $f: A \rightarrow B$, durch Rechenvorschriften definiert
- Bildungsprinzipien (Konstruktoren)
  - elementare Funktionen: Addition, Subtraktion, Operationen auf linearen Listen, Alternative usw.
  - Konstruktoren: Komposition/Substitution, Applikation, Abstraktion

Programmierstile
funkt. Programmierung - Variablenkonz.

- Variable = Bezeichner, der an konkretes Objekt gebunden ist
- keine Zuordnung zu Speicherzellen oder dynamische Veränderung von Werten
- keine Seiteneffekte von Funktionen
- **referenzielle Transparenz**
- Operation: Substitution gem. Substitutionsregeln Call-by-value, Call-by-name, Call-by-need (lazy evaluation)

Programmierstile
funkt. Programmierung - ref. Transp.

Eine Programmiersprache heißt **referenziell transparent**, wenn alle möglichen Ausdrücke folgenden Prinzipien genügen:

- *Extensionalitätsprinzip*: Das einzig wichtige Merkmal eines Ausdrucks ist sein Wert; unwichtig, wie dieser Wert berechnet wird (intensionale Sicht).
- *Leibnizsches Prinzip* der Ersetzung von Identitäten: Wert eines Ausdrucks ändert sich nicht, wenn man Teilausdrücke durch andere gleichen Werts ersetzt.
- Prinzip der *Definitheit*: Wert eines Ausdrucks ist innerhalb eines gewissen Kontexts immer gleich, unabhängig von der Stelle, an der der Ausdruck im Kontext auftritt.

Programmierstile
funkt. Programmierung - Beispiel

- Beispiel: $(2ax+b) \cdot (2ax+c)$
  - ref. Transparenz: $2ax=2ax$
  - keine ref. Transparenz: $2ax \neq 2ax \neq 2ax$
- imperative Sprachen sind nicht referenziell transparent (ihre größte Schwäche)

Programmierstile
funkt. Programmierung - Historie

- 1930: A. Church: Entwicklung des λ-Kalküls
- 1958: J. McCarthy: LISP auf der Basis des λ-Kalküls. Weiterentwicklungen in der Folgezeit (z.B. SCHEME).
- 1965: P. Landin: Sprache ISWIM (Vorläufer von ML). In der Folge: Entwicklung zentraler Ideen bei Anwendung, Notation, Implementierung funktionaler Sprachen.
- 1978: J. Backus: Sprache FP orientiert am Kombinatoren-Kalkül (dem λ-Kalkül vergleichbarer Kalkül). In FP gibt es keine Variablen.
- 1978: R. Milner: ML (mittlerweile standardisiert); leistungsfähiges Typkonzept und Sprachelemente für die Definition von abstrakten Datentypen.

Programmierstile
funkt. Programmierung - Mischen

- Mischen in FUN:

```
funktion misch f:intlist g:intlist → intlist ≡
wenn f=leer dann g sonst
   wenn g=leer dann f sonst
      wenn (erstes f)<(erstes g) dann (erstes
      f,misch (rest f) g)
         sonst (erstes g,misch f (rest g))
      ende
   ende
ende .
```

- Aufruf: misch [3,7,19,54] [2,3,5,16,74].

Programmierstile
funkt. Programmierung - Beispiel

- Mischen in ML:

```
fun    misch [ ] g  = g: int list  |
       misch f [ ] = f   |
       misch (x::r)(x'::r')=
           if x<x' then [x]@(misch r (x'::r')) else
                                [x]@(misch (x::r) r').
```

Programmierstile
prädikative Programmierung - Kalkül

- *Prädikatenlogik*
- Formalisierung von Sachverhalten und logischen Argumenten
- Präzisierung von mathematischen Begriffen wie Definition, Satz, Beweis, Folgerung, wahr, falsch usw.

Programmierstile
präd. Programmierung - Präd.-kalkül

- formale Sprache (*Prädikatenkalkül*) der korrekten Zeichenreihen (*Formeln*) zur Darstellung dieser Sachverhalte
- Formel: sinnleere Folge von Symbolen (Buchstaben und Zeichen)
- Formeln gehen durch Interpretation (Zuordnung von Symbolen zu realen Objekten) in Aussagen (*Prädikate*) über, deren Wahrheitsgehalt (*wahr* oder *falsch*) man bestimmen kann
- Prädikatives Programmieren: Aufstellen von Behauptungen und Beweisen in einem System von *gültigen* (d.h. immer wahren) Aussagen (Beschränkung auf *Hornformeln*)

Programmierstile
Prädikatenkalkül - Beispiel

Beispiel: Formel des Prädikatenkalküls:

$$(\exists f)\ (f(a)=b \wedge (\forall x)\ (p(x) \Rightarrow f(x)=g(x,f(h(x)))))).$$

Interpretation ordnet der sinnleeren Zeichenfolge eine Bedeutung zu. Dazu benötigt man

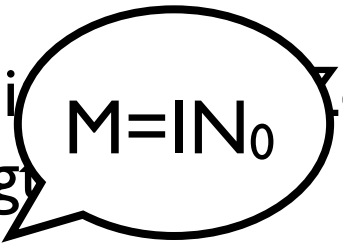
- eine nichtleere Menge M
- Zuordnung von a und b zu Elementen von M
- Zuordnung von p zu einstelligem Prädikat über M, also zu Abbildung p: $M \rightarrow \{wahr, falsch\}$
- Zuordnung von g zu zweistelliger Funktion g: $M \times M \rightarrow M$
- Zuordnung von h zu einstelliger Funktion h: $M \rightarrow M$

Programmierstile
Prädikatenkalkül - Beispiel

Beispiel: Formel des Prädikatenkalküls:

$$(\exists f)\ (f(a)=b \wedge (\forall x)\ (p(x) \Rightarrow f(x)=g(x,f(h(x))))).$$

Interpretation ordnet der si[illegible]Zeichenfolge eine Bedeutung zu. Dazu benötigt



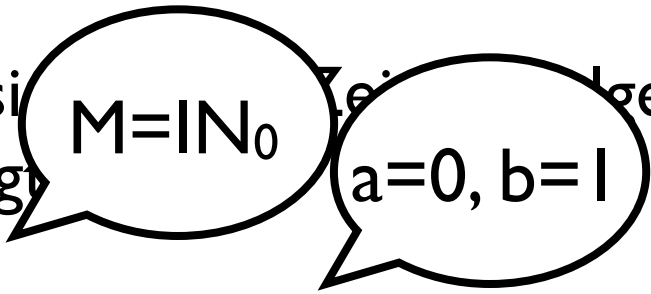
- eine nichtleere Menge M
- Zuordnung von a und b zu Elementen von M
- Zuordnung von p zu einstelligem Prädikat über M, also zu Abbildung p: M→{wahr, falsch}
- Zuordnung von g zu zweistelliger Funktion g: M×M→M
- Zuordnung von h zu einstelliger Funktion h: M→M

Programmierstile
Prädikatenkalkül - Beispiel

Beispiel: Formel des Prädikatenkalküls:

$$(\exists f)\ (f(a)=b \wedge (\forall x)\ (p(x) \Rightarrow f(x)=g(x,f(h(x)))))).$$

Interpretation ordnet der si[illegible]ge eine Bedeutung zu. Dazu benötigt



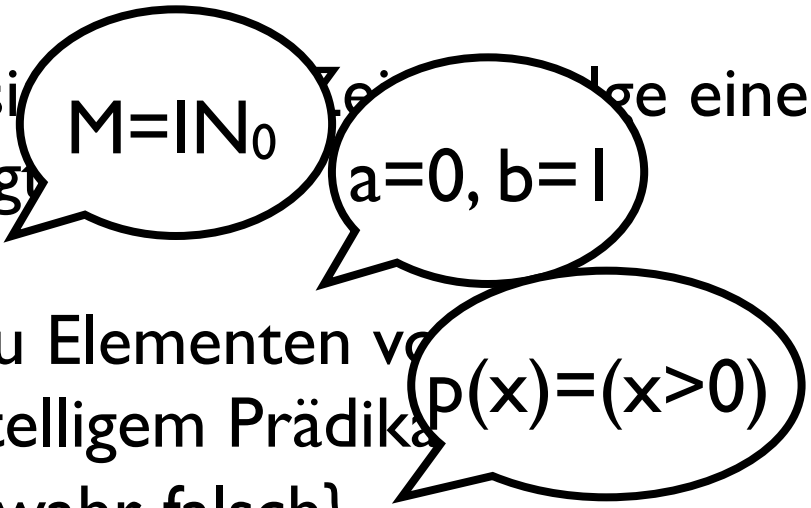
- eine nichtleere Menge M
- Zuordnung von a und b zu Elementen von M
- Zuordnung von p zu einstelligem Prädikat über M, also zu Abbildung p: M→{wahr, falsch}
- Zuordnung von g zu zweistelliger Funktion g: M×M→M
- Zuordnung von h zu einstelliger Funktion h: M→M

Programmierstile
Prädikatenkalkül - Beispiel

Beispiel: Formel des Prädikatenkalküls:

$$(\exists f)\ (f(a)=b \wedge (\forall x)\ (p(x) \Rightarrow f(x)=g(x,f(h(x)))))).$$

Interpretation ordnet der si[illegible]ge eine Bedeutung zu. Dazu benötigt



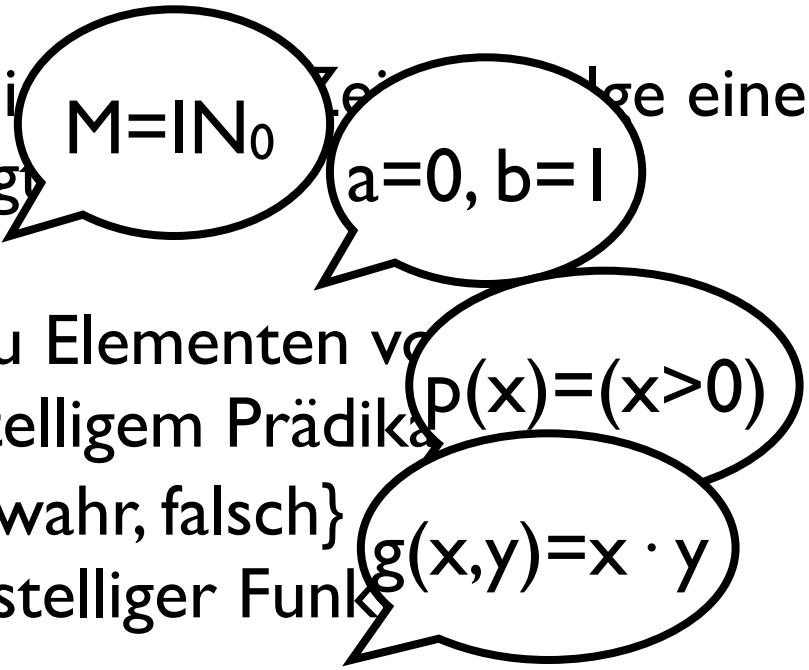
- eine nichtleere Menge M
- Zuordnung von a und b zu Elementen v[illegible]
- Zuordnung von p zu einstelligem Prädika[illegible]
  also zu Abbildung p: M→{wahr, falsch}
- Zuordnung von g zu zweistelliger Funktion g: M×M→M
- Zuordnung von h zu einstelliger Funktion h: M→M

Programmierstile
Prädikatenkalkül - Beispiel

Beispiel: Formel des Prädikatenkalküls:

$$(\exists f)\ (f(a)=b \wedge (\forall x)\ (p(x) \Rightarrow f(x)=g(x,f(h(x)))))).$$

Interpretation ordnet der si[illegible]ge eine Bedeutung zu. Dazu benötigt



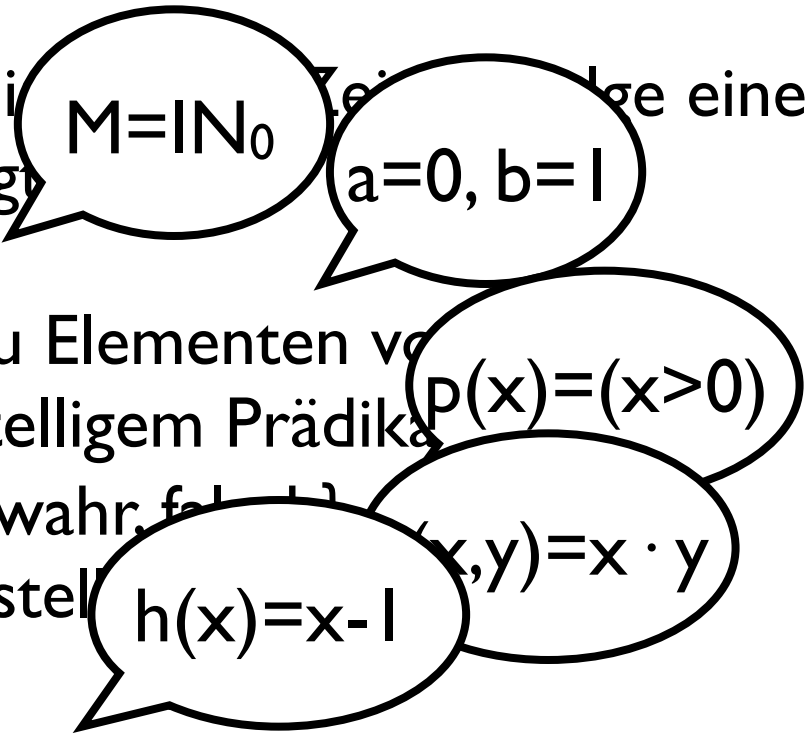
- eine nichtleere Menge M
- Zuordnung von a und b zu Elementen v[illegible]
- Zuordnung von p zu einstelligem Prädika[illegible] also zu Abbildung p: M→{wahr, falsch}
- Zuordnung von g zu zweistelliger Funk[illegible] M×M→M
- Zuordnung von h zu einstelliger Funktion h: M→M

Programmierstile
Prädikatenkalkül - Beispiel

Beispiel: Formel des Prädikatenkalküls:

$$(\exists f)\ (f(a)=b \wedge (\forall x)\ (p(x) \Rightarrow f(x)=g(x,f(h(x)))))).$$

Interpretation ordnet der si[illegible]ge eine Bedeutung zu. Dazu benötig[illegible]



- eine nichtleere Menge M
- Zuordnung von a und b zu Elementen v[illegible]
- Zuordnung von p zu einstelligem Prädika[illegible] also zu Abbildung p: M→{wahr, fal[illegible]}
- Zuordnung von g zu zweistel[illegible] M×M→M
- Zuordnung von h zu einstelliger Funktion h: M→M

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.

Programmierstile
Prädikatenkalkül - Beispiel

Interpretation liefert Prädikat

$$(\exists f)\ (f(0)=1 \wedge (\forall x)\ (x>0 \Rightarrow f(x)=x \cdot f(x-1))),$$

das wahr oder falsch sein kann.

Setze f = Fakultät, dann Prädikat *wahr*.
Prädikat ist nicht *gültig*, also nicht bei jeder Interpretation wahr (geeignete Interpretation selbst suchen).

Programmierstile
präd. Programmierung - Programm

- Programm: Menge von wahren Aussagen(**Fakten**) und von logischen Schluß**regeln**, mit denen aus Fakten neue hergeleitet werden können:

  Wissen=Fakten+Regeln

- Start eines Programms: Eingabe einer Behauptung; Computer versucht, Behauptung mit Regeln und Fakten zu beweisen.

Programmierstile
präd. Programmierung - Faktum

- **Faktum**: wahre Aussage über Eigenschaften oder Beziehungen zwischen Objekten.
- Beispiel: Objekte $a_1,...,a_n$ und Beziehung e:

$$\text{Faktum } e(a_1,...,a_n)$$

- Beispiel: misch(X,Y,Z): Folge Z ist gemischte Version der Folgen X und Y:

$$(\forall Y)\ (\text{misch}([\ ],Y,Y))$$

Programmierstile
präd. Programmierung - Faktum

- **Regel**: allgemeine Form:

  Prämisse ⇒ Konklusion

- Bedeutung: Ist Prämisse erfüllt, so gilt Konklusion. Die Prämisse wird gebildet aus Termen durch logisches UND verknüpft.
- Beispiel: sinnvolle Regel für Mischen zweier Folgen der Länge 2 zu Ergebnisfolge, die bereits drei Elemente enthält:

  (∀A,B,C,D,E,F,G) ((A≤C ^ misch([A,B],[C,D],[E,F,G]))
  ⇒ misch([B],[C,D],[A,E,F,G]))

Programmierstile
präd. Programmierung - Variablen

- Variablen = Unbestimmte
- Operation: **Unifikation**
  - “Gleichmachen” zweier Ausdrücke (*Terme*) durch konsistente Ersetzung von Variablen durch irgendwelche Terme

Programmierstile
präd. Programmierung - Unifikation

- Motivation:
  - Fakten und Regeln beschreiben allgemeine Beziehungen: $\forall x: x>0 \Rightarrow x+1>0$
  - Behauptungen beschreiben konkrete Objekte: $3>0$
  - Assoziiere Objekte, die in Behauptungen vorkommen, mit denen aus der Fakten-/ Regelmenge: $2>0 \Rightarrow 2+1=3>0$

Programmierstile
präd. Programmierung - Unifikation

Beispiel 1:

- Faktum von oben:

$$(\forall Y)\ (misch([\ ], Y, Y))$$

- Behauptung:

$$misch([\ ], [1,3], [1,3])$$

- Unifiziere Variable Y mit Folge [1,3].

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

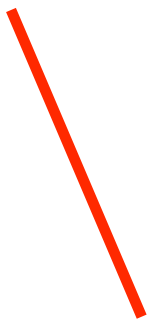
- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))



- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

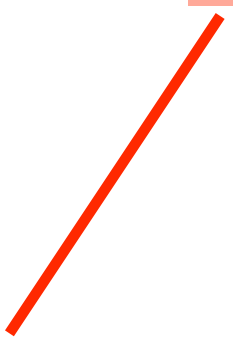
- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))



- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

  H=a(b(D))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

  H=a(b(D))

  F=H

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

  H=a(b(D))

  F=H

  K=d(e(F))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Unifikation

Beispiel 2:

- Faktum: (∀D,F,H) (z(a(b(D)),d(e(F)),g(H)))

  H=a(b(D))

  F=H D, H, F beliebig

  K=d(e(F))

- Behauptung: (∃H,K,F) (z(H,K,g(F)))

Programmierstile
präd. Programmierung - Historie

- 70er Jahre: Arbeiten über automatisches Beweisen und Künstliche Intelligenz
- 1965: J.A. Robinson: *Resolutionsprinzip*
- 1972: R.A. Kowalski: Urheber der Idee, Logikkalküle als Programmiersprachen zu verwenden
- 1973: A. Colmerauer: Sprache PROLOG
- aktuelle Forschungen: Ergänzung von PROLOG um Datentypsystem und funktionale Elemente

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

(∀Y) (misch([ ],Y,Y)).

1. oder 2. Liste leer

(∀Y) (misch(Y,[ ],Y)).

(∀A,B,X,Y,Z) (A≤B ^ misch(X,[B|Y],Z) ⇒

misch([A|X],[B|Y],[A|Z])).

(∀A,B,X,Y,Z) (B<A ^ misch([A|X],Y,Z) ⇒

misch([A|X],[B|Y],[B|Z])).

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)$ (misch([ ],Y,Y)).

$(\forall Y)$ (misch(Y,[ ],Y)).

$(\forall A,B,X,Y,Z)$ ($A \leq B$ ^ misch(X,[B|Y],Z) $\Rightarrow$

misch([A|X],[B|Y],[A|Z])).

$(\forall A,B,X,Y,Z)$ ($B<A$ ^ misch([A|X],Y,Z) $\Rightarrow$

misch([A|X],[B|Y],[B|Z])).

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\,],Y,Y)).$

$(\forall Y)\ (misch(Y,[\,],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)$ (misch([ ],Y,Y)).

$(\forall Y)$ (misch(Y,[ ],Y)).

$(\forall A,B,X,Y,Z)$ $(A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

misch([A|X],[B|Y],[A|Z])).

$(\forall A,B,X,Y,Z)$ $(B<A \wedge misch([A|X],Y,Z) \Rightarrow$

misch([A|X],[B|Y],[B|Z])).

B=1. Elem.
Y=Rest

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B<A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\,],Y,Y)).$

$(\forall Y)\ (misch(Y,[\,],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)$ (misch([ ],Y,Y)).

$(\forall Y)$ (misch(Y,[ ],Y)).

$(\forall A,B,X,Y,Z)$ ($A \leq B$ ^ misch(X,[B|Y],Z) $\Rightarrow$

misch([A|X],[B|Y],[A|Z])).

$(\forall A,B,X,Y,Z)$ ($B < A$ ^ misch([A|X],Y,Z) $\Rightarrow$

misch([A|X],[B|Y],[B|Z])).

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)$ (misch([ ],Y,Y)).

$(\forall Y)$ (misch(Y,[ ],Y)).

$(\forall A,B,X,Y,Z)$ $(A \leq B \wedge$ misch(X,[B|Y],Z) $\Rightarrow$

misch([A|X],[B|Y],[A|Z])).

$(\forall A,B,X,Y,Z)$ $(B < A \wedge$ misch([A|X],Y,Z) $\Rightarrow$

misch([A|X],[B|Y],[B|Z])).

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)\ (misch([\ ],Y,Y)).$

$(\forall Y)\ (misch(Y,[\ ],Y)).$

$(\forall A,B,X,Y,Z)\ (A \leq B \wedge misch(X,[B|Y],Z) \Rightarrow$

$misch([A|X],[B|Y],[A|Z])).$

$(\forall A,B,X,Y,Z)\ (B < A \wedge misch([A|X],Y,Z) \Rightarrow$

$misch([A|X],[B|Y],[B|Z])).$

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)$ (misch([ ],Y,Y)).

$(\forall Y)$ (misch(Y,[ ],Y)).

$(\forall A,B,X,Y,Z)$ ($A \leq B$ ^ misch(X,[B|Y],Z) $\Rightarrow$

misch([A|X],[B|Y],[A|Z])).

$(\forall A,B,X,Y,Z)$ ($B < A$ ^ misch([A|X],Y,Z) $\Rightarrow$

misch([A|X],[B|Y],[B|Z])).

misch([3,7,19,54],[2,3,5,16,74],[2,3,3,5,7,16,19,54,74])?

Programmierstile
präd. Programmierung - Mischen

- Mischen fiktiv:

$(\forall Y)$ (misch([ ],Y,Y)).

$(\forall Y)$ (misch(Y,[ ],Y)).

$(\forall A,B,X,Y,Z)$ ($A \leq B$ ^ misch(X,[B|Y],Z) $\Rightarrow$

misch([A|X],[B|Y],[A|Z])).

$(\forall A,B,X,Y,Z)$ ($B < A$ ^ misch([A|X],Y,Z) $\Rightarrow$

misch([A|X],[B|Y],[B|Z])).

misch([3,7,19,54],[2,3,5,16,74],[2,3,3,5,7,16,19,54,74])?

wahr

Programmierstile

präd. Programmierung - Mischen

Programmierstile
präd. Programmierung - Mischen

- deklarativ (!!): geg. u. gesuchte Größen frei:

Programmierstile
präd. Programmierung - Mischen

- deklarativ (!!): geg. u. gesuchte Größen frei:

$$(\exists Y)\ (\text{misch}([2,5],[1,2,8],Y))?$$

Programmierstile
präd. Programmierung - Mischen

- deklarativ (!!): geg. u. gesuchte Größen frei:

(∃Y) (misch([2,5],[1,2,8],Y))?

(∃Y) (misch(Y,[1,2,8],[1,2,4,6,8]))?

Programmierstile
präd. Programmierung - Mischen

- deklarativ (!!): geg. u. gesuchte Größen frei:

(∃Y) (misch([2,5],[1,2,8],Y))?

(∃Y) (misch(Y,[1,2,8],[1,2,4,6,8]))?

(∃X,Y) (misch(X,Y,[1,2,5]))?

Programmierstile
präd. Programmierung - Mischen

- deklarativ (!!): geg. u. gesuchte Größen frei:

(∃Y) (misch([2,5],[1,2,8],Y))?

(∃Y) (misch(Y,[1,2,8],[1,2,4,6,8]))?

(∃X,Y) (misch(X,Y,[1,2,5]))?

im Prinzip: (∃X,Y,Z) (misch(X,Y,Z))?

Programmierstile

präd. Programmierung - Mischen

- deklarativ (!!): geg. u. gesucht

(∃Y) (misch([2,5],[

(∃Y) (misch(Y,[1,2,8],

(∃X,Y) (misch(X,Y,[1,2,5]))?

im Prinzip: (∃X,Y,Z) (misch(X,Y,Z))?

X=[ ],Y=[1,2,5]
X=[1],Y=[2,5]
X=[2],Y=[1,5]
X=[5],Y=[1,2]
X=[1,2],Y=[5]
...
X=[1,2,5],Y=[ ]

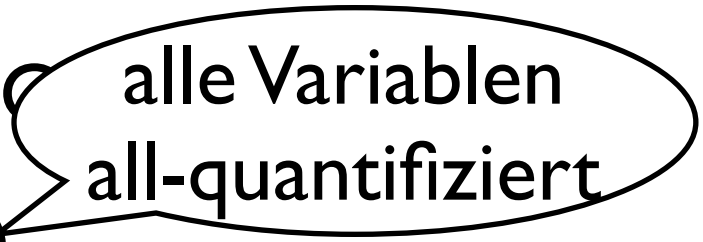
Programmierstile
präd. Programmierung - Mischen

- Mischen in PROLOG:

```
misch([ ],Y,Y).
misch(Y,[ ],Y).
misch([A|X],[B|Y],[A|Z]) :–
        A≤B, misch(X,[B|Y],Z).
misch([A|X],[B|Y],[B|Z]) :–
        B<A, misch([A|X],Y,Z).
```

Programmierstile
präd. Programmierung - Mischen

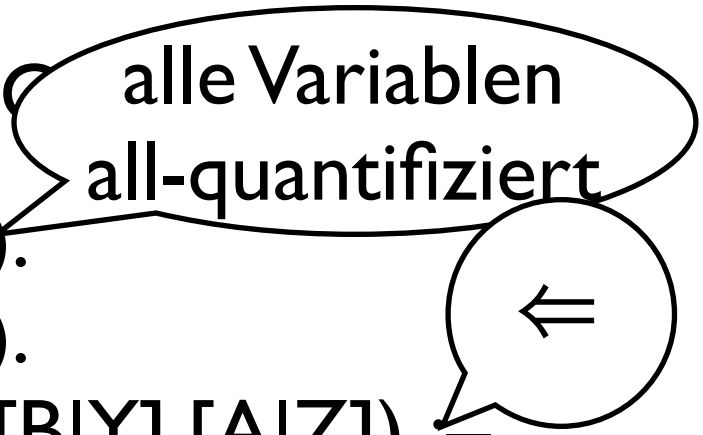
- Mischen in PROLOG




```
misch([ ],Y,Y).
misch(Y,[ ],Y).
misch([A|X],[B|Y],[A|Z]) :–
        A≤B, misch(X,[B|Y],Z).
misch([A|X],[B|Y],[B|Z]) :–
        B<A, misch([A|X],Y,Z).
```

Programmierstile
präd. Programmierung - Mischen

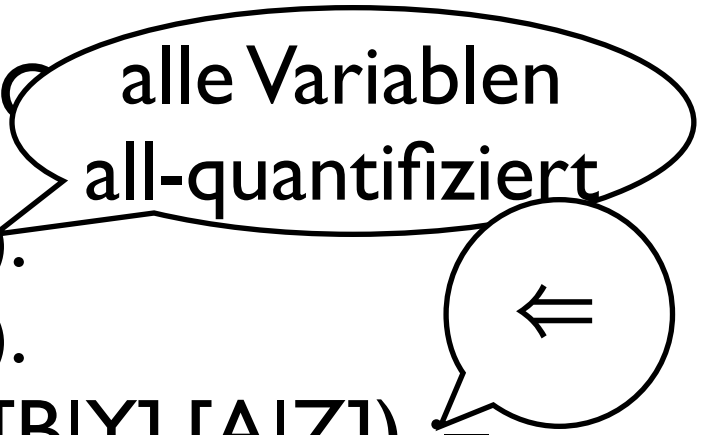
- Mischen in PROLOG



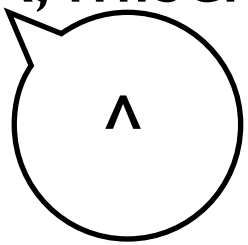

```
misch([ ],Y,Y).
misch(Y,[ ],Y).
misch([A|X],[B|Y],[A|Z]) :–
        A≤B, misch(X,[B|Y],Z).
misch([A|X],[B|Y],[B|Z]) :–
        B<A, misch([A|X],Y,Z).
```

Programmierstile
präd. Programmierung - Mischen

- Mischen in PROLOG

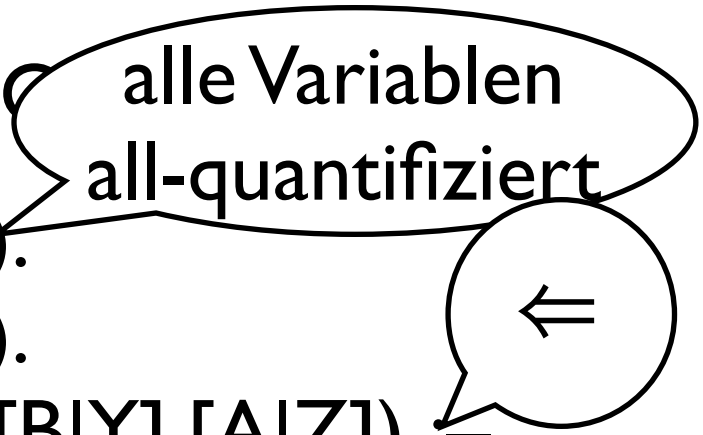



```
misch([ ],Y,Y).
misch(Y,[ ],Y).
misch([A|X],[B|Y],[A|Z]) :–
        A≤B, misch(X,[B|Y],Z).
misch([A|X],[B|Y],[B|Z]) :–
        B<A, misch([A|X],Y,Z).
```



Programmierstile
präd. Programmierung - Mischen

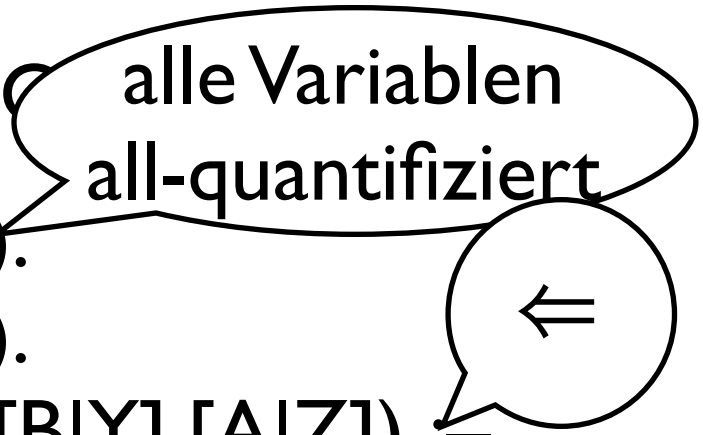
- Mischen in PROLOG




```
misch([ ],Y,Y).
misch(Y,[ ],Y).
misch([A|X],[B|Y],[A|Z]) :–
        A≤B, misch(X,[B|Y],Z).
misch([A|X],[B|Y],[B|Z]) :–
        B<A, misch([A|X],Y,Z).
```

Programmierstile
präd. Programmierung - Mischen

- Mischen in PROLOG

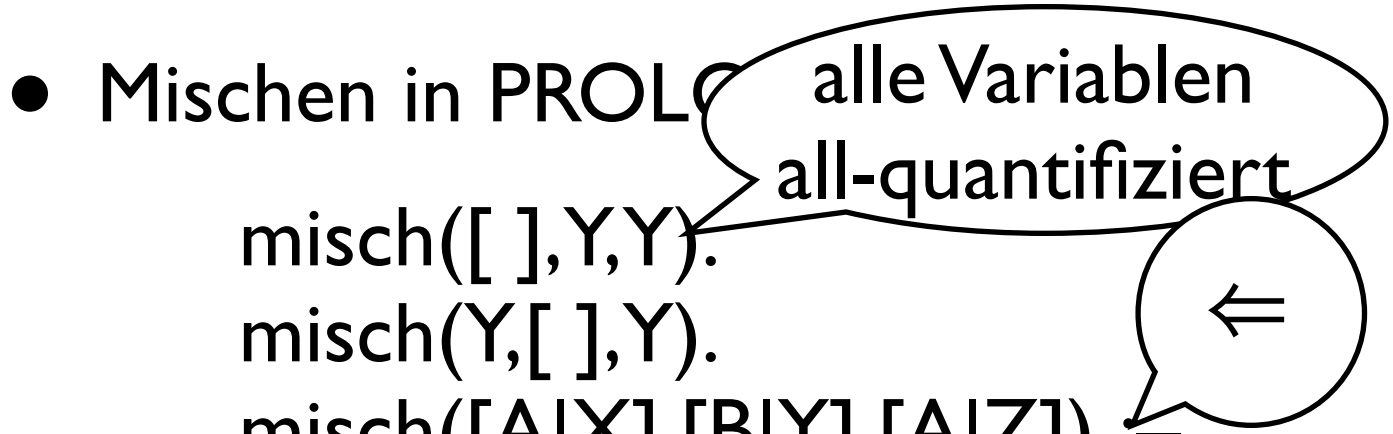


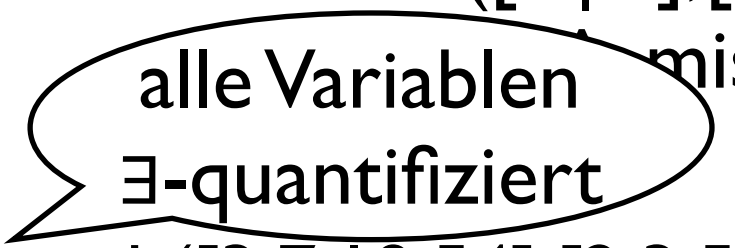

```
misch([ ],Y,Y).
misch(Y,[ ],Y).
misch([A|X],[B|Y],[A|Z]) :–
        A≤B, misch(X,[B|Y],Z).
misch([A|X],[B|Y],[B|Z]) :–
        B<A, misch([A|X],Y,Z).
```

```
?- misch([3,7,19,54],[2,3,5,16,74],[2,3,3,5,7,16,19,54,74]).
```

Programmierstile
präd. Programmierung - Mischen

- Mischen in PROLOG



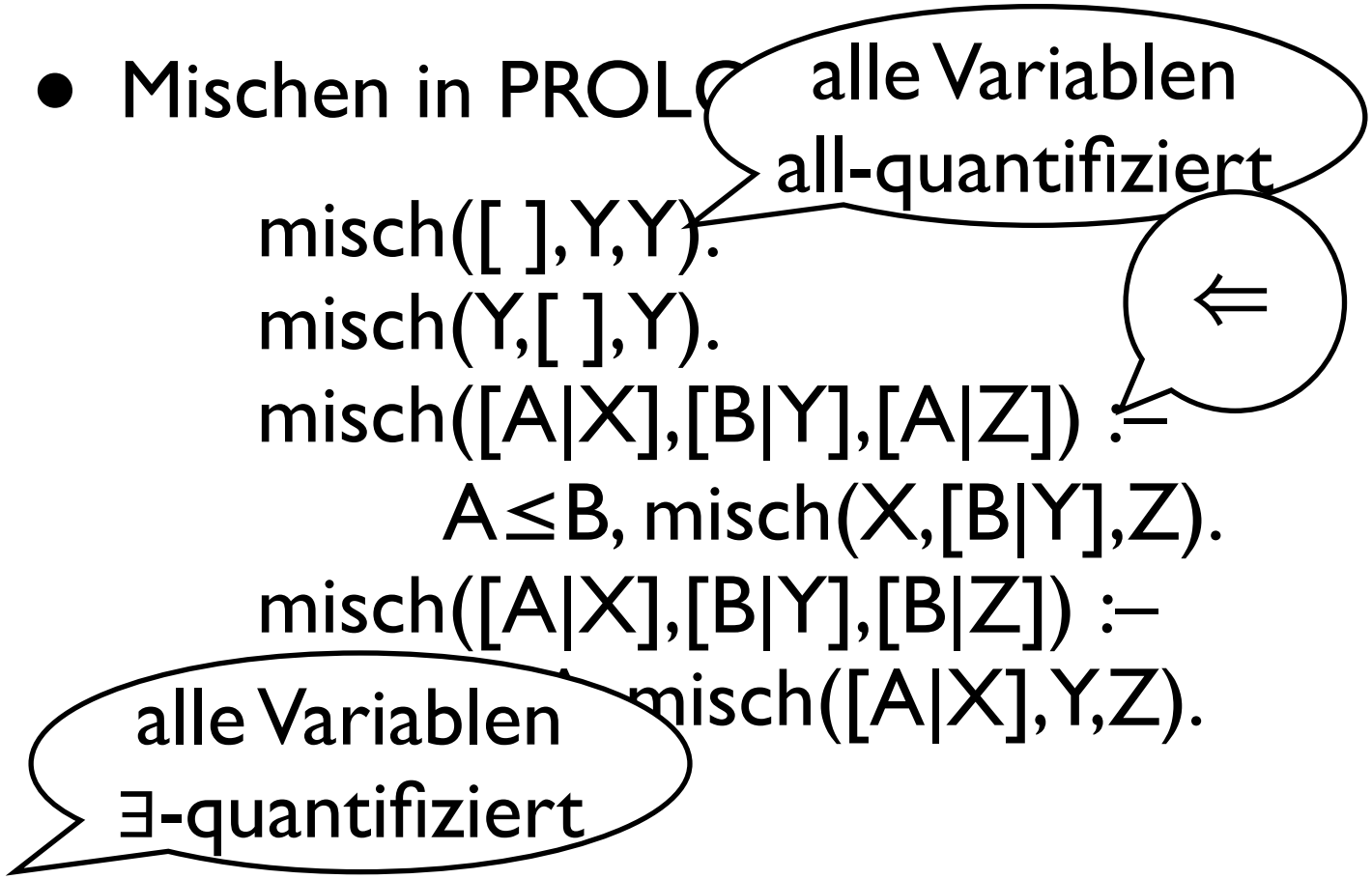



```
?- misch([3,7,19,54],[2,3,5,16,74],[2,3,3,5,7,16,19,54,74]).
```

Programmierstile

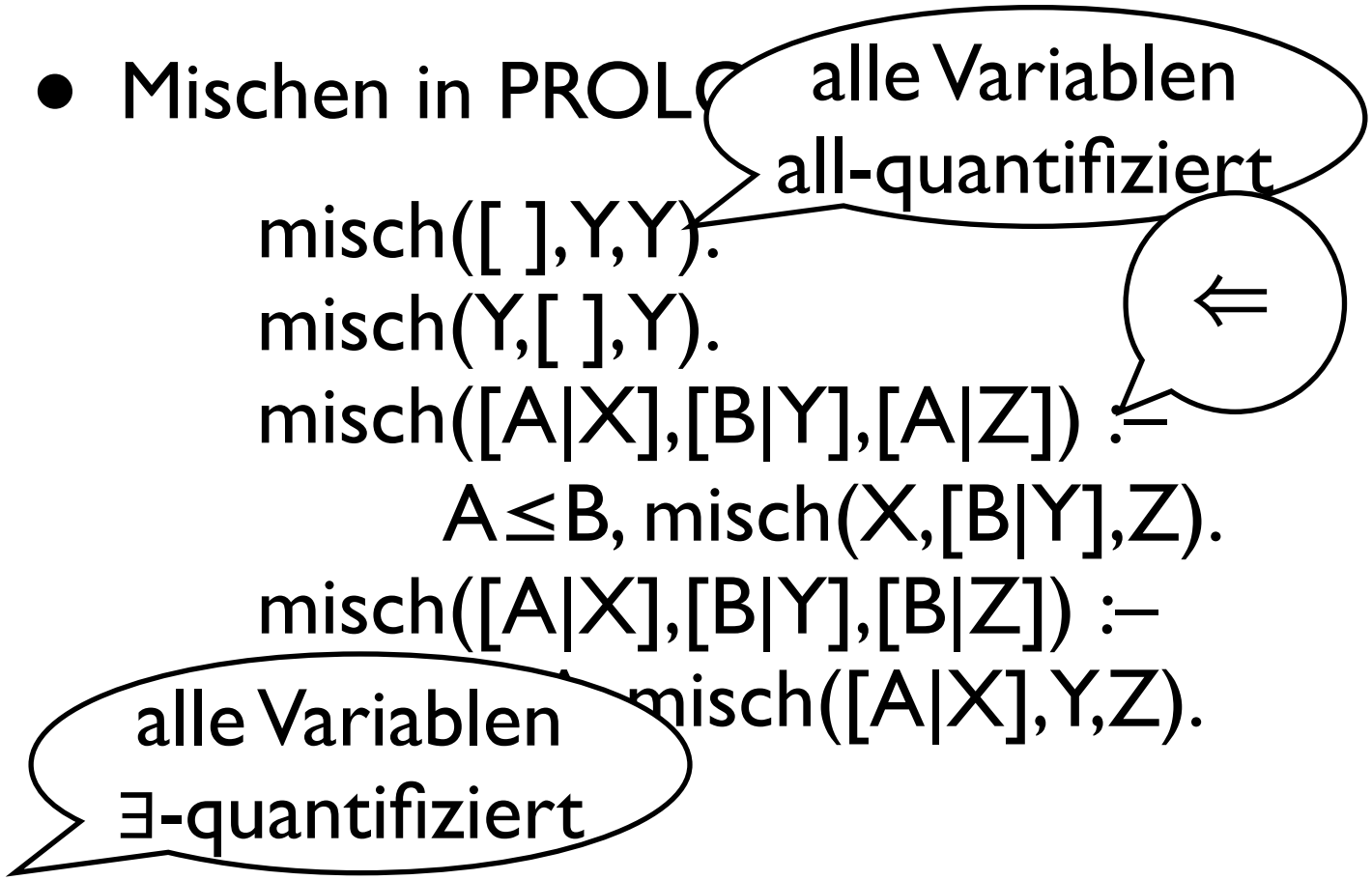
präd. Programmierung - Mischen

- Mischen in PROLOG



Programmierstile
präd. Programmierung - Mischen

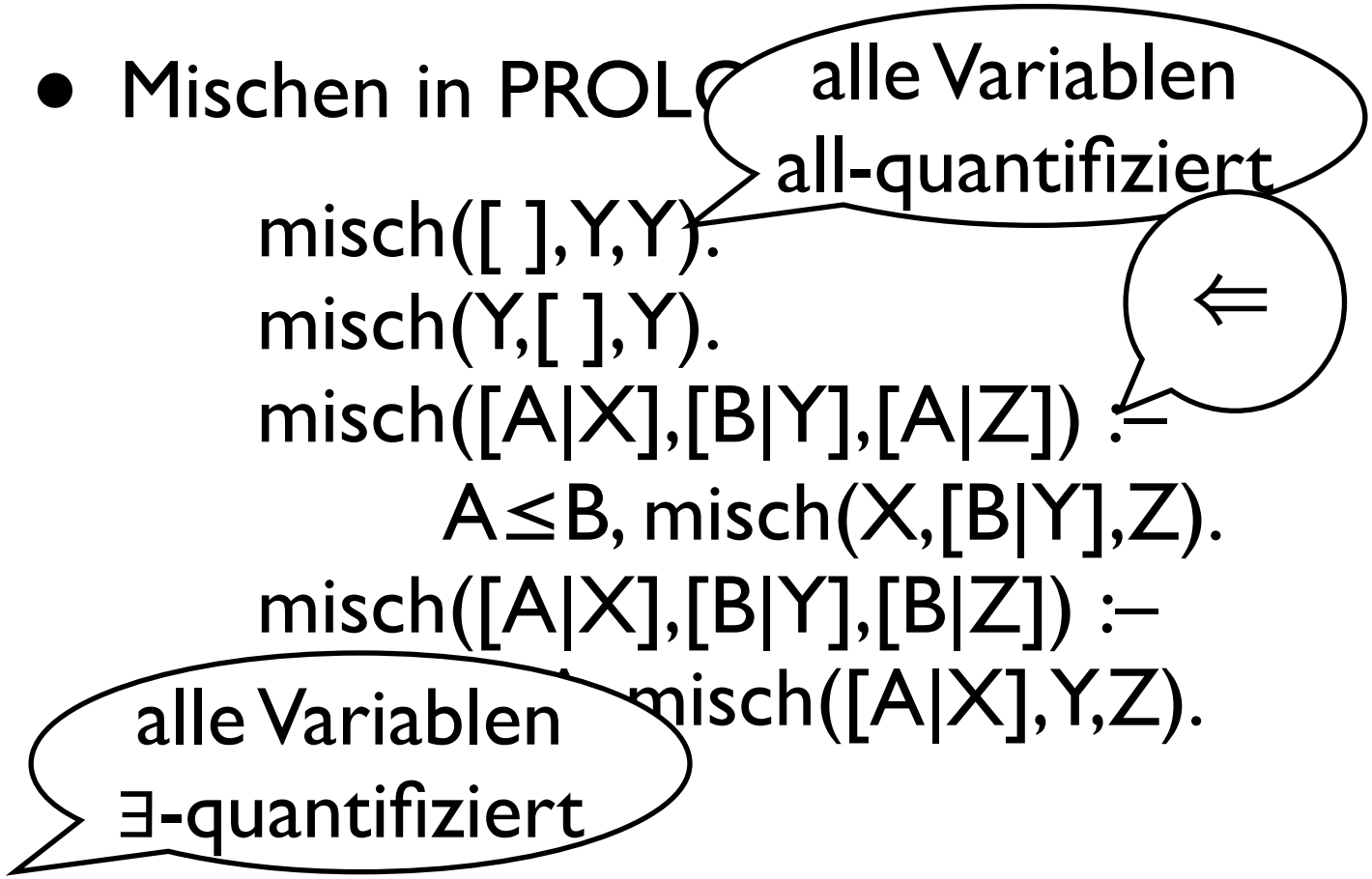
- Mischen in PROLOG



?- misch([2,5],[1,2,8],Y).

Programmierstile
präd. Programmierung - Mischen

- Mischen in PROLOG




```
?- misch([2,5],[1,2,8],Y).
Y=[1,2,2,5,8].
```

Programmierstile
präd. Programmierung - Drawback

- derzeit *keine* rein deklarativen Sprachen
- PROLOG im strengen Sinne *nicht* deklarativ
- zahlreiche nicht-deklarative Elemente.
- Hintergrund: imperative *deterministische* Arbeitsweise des Von Neumann-Rechners
- fixe Reihenfolge, in der Fakten und Regeln zum Beweis herangezogen werden
- unerwünschte Situation: Behauptung "nicht beweisbar", obwohl Beweis existiert
- **prozedurale** Semantik von PROLOG-Programmen versus **deklarative** Semantik des zugrundeliegenden Prädikatenkalküls