HANSER

Taschenbuch Programmiersprachen

Herausgegeben von Peter A. Henning, Holger Vogelsang

ISBN-10: 3-446-40744-8
ISBN-13: 978-3-446-40744-2

Leseprobe

1 Grundlagen

Peter A. Henning
Dirk W. Hoffmann
Holger Vogelsang

1.1 Einführung

Ein Programm ist eine Abfolge von Daten und Befehlen an einen Prozessor, um diese Daten in andere Daten umzuwandeln. Es wird in einer formal definierten Sprache verfasst, der **Programmiersprache**.

Die Definition einer Programmiersprache besteht aus zwei Teilen:

- Die **Syntax** einer Sprache wird formal definiert. Sie beschreibt die gültigen Folgen von Zeichenketten mit Hilfe so genannter kontextfreier Grammatiken. Dabei kann das Darstellungsmittel sehr unterschiedlich ausfallen. Verbreitet sind die ↑Backus-Naur-Form (BNF) und ↑Syntaxdiagramme.[1]
- Die **Semantik** beschreibt, welche Bedeutung die einzelnen Sprachelemente der Syntaxdefinition besitzen. Die Semantik wird in der Regel textuell beschrieben.

1.2 Generationen der Programmiersprachen

Die Anfänge der Programmierung gehen bis in das 19. Jahrhundert zurück, als Ada Augusta Byron eine „Rechenvorschrift für die Analytische Maschine“ des Britischen Mathematikers Charles Babbage formulierte. Die Analytische Maschine konnte auf Grund ihrer mechanischen Komplexität nie fertiggestellt werden, so dass auch das erste Programm der Weltgeschichte niemals zur Ausführung kam.

Die formale Definition von Programmiersprachen erfolgte zeitgleich mit dem Bau der ersten elektromechanischen Computer. So entwickelte Konrad Zuse (1910 – 1995) Mitte der Vierzigerjahre mit dem ↑Plankalkül die erste höhere Programmiersprache der Welt. Das vollständige Manuskript wurde jedoch erst 1972 veröffentlicht [Z72].

Die Entwicklung der Programmiersprachen lässt sich in fünf Generationen einteilen.

[1] In diesem Buch sind Hinweise auf das Glossar mit einem ↑gekennzeichnet.

1.2.1 Erste Generation

Elektronische Computer waren zu Beginn nur in ↑**Maschinensprache** programmierbar. Dabei werden binäre, prozessorspezifische Codes, meist in Kombination mit den zu bearbeitenden Daten, an den Prozessor gesandt. Typischerweise handelt es sich dabei um Operationen zum Verschieben von Daten aus dem Speicher in Register oder um einfache Arithmetik mit den Inhalten der Register.

1.2.2 Zweite Generation

Sprachen der 1. Generation wurden sehr bald durch **Assemblersprachen** (↑Assembler) abgelöst, bei denen die Operationen des Prozessors in lesbarem Klartext formuliert werden müssen, aber dennoch registerorientiert sind. Die hexadezimalen Codes der Maschinensprache werden darin durch einfache, möglichst aussagekräftige Abkürzungen, so genannte Mnemonics, ersetzt.

Das Assemblerprogramm wird durch einen Übersetzer – häufig ebenfalls als Assembler bezeichnet – in Maschinensprache übersetzt.

1.2.3 Dritte Generation

Als Sprachen der 3. Generation bezeichnet man unterschiedliche Systeme, die etwa ab Anfang der 1960er-Jahre entstanden. Hier trat ein gewaltiger Sprung in der Abstraktion von der konkreten Hardware ein. Der Entwickler musste seine Programme nicht mehr direkt für eine spezielle Prozessorarchitektur mit dem Befehlssatz des Prozessors schreiben. Deshalb werden diese Sprachen auch häufig als „höhere Programmiersprachen“ bezeichnet.

Prozedurale Sprachen. Zunächst handelte es sich um problemorientierte Sprachen mit dem prozeduralen Programmierparadigma, vgl. Abschnitt 1.3.1. FORTRAN war ab 1954 die erste dieser Sprachen, die sich in breitem Rahmen durchsetzte und auch heute noch – nach massiven „Renovierungsarbeiten“ – weite Verbreitung findet. FORTRAN wird in Kapitel 9 ausführlich behandelt.

Die Blockorientierung von Programmen kam erst mit der Sprache ALGOL auf, die heute nicht mehr verwendet wird. Um 1970 begann sich mit Pascal die so genannte strukturierte Programmierung durchzusetzen. Damit setzte eine Abkehr vom Spaghetti-Code ein. Statt expliziter Verzweigungen („goto“) wurden saubere ↑Kontrollstrukturen geschaffen, die den Ablauf eines Programms direkt im Quelltext erkennen lassen.

Objektorientierte Sprachen. Parallel dazu entwickelte sich ab ca. 1970 das ↑objektorientierte Programmierparadigma, vgl. Abschnitt 1.3.1. Dieses basiert im Wesentlichen auf den Entwicklungsstufen der Sprache Smalltalk, die bereits in den Siebzigerjahren von der Learning Research Group des Pa-

lo Alto Research Centers (PARC) von Xerox entwickelt wurde. Die Sprache Smalltalk wird in Kapitel 10 ausführlich behandelt.

Ab ca. 1990 wurden objektorientierte Aspekte in die meisten Programmiersprachen übernommen.

Hybride Sprachen. Neben rein prozeduralen und rein objektorientierten Sprachen gibt es auch solche, die beide Konzepte verwenden. C++ ist ein typischer Vertreter. Es bietet Funktionen aus der prozeduralen und Klassen aus der objektorientierten Welt.

Domänenspezifische Sprachen. Unter diesen Sammelbegriff fallen alle Sprachen der dritten Generation, die nur in bestimmten Aufgabenbereichen (Domänen) eingesetzt werden und daher dafür optimiert sind.

Deklarative Sprachen. Die symbolische Programmierung zur Lösung nicht numerischer Probleme begann um 1960 mit LISP und führt heute zum ↑deklarativen Programmierparadigma, vgl. Abschnitt 1.3.1, das als Zusammenfassung des logischen und funktionalen Programmierparadigmas sowie der Constraintprogrammierung zu sehen ist.

1.2.4 Vierte Generation

Es gibt keine exakte Festlegung, welche Sprachen oder Paradigmen eine Sprache der vierten Generation ausmachen. Der Begriff der „4GL“-Sprache wurde häufig als Marketinginstrument verwendet, um ein neues Programm oder eine weitere Sprache durchzusetzen. Generelles Ziel dieser Sprachen aber ist ein höheres Abstraktionsniveau als in der dritten Generation. Damit soll es zur Formulierung einer Lösung nicht mehr erforderlich sein, die genauen Schritte zu formulieren. Vielmehr wird beschrieben, was gelöst werden soll.

Skriptsprachen. Ab 1960 entwickelte sich aus der Job Control Language JCL eine Vielzahl von Skriptsprachen, die eine einfache Integration anderer Programme und Abläufe ermöglichen. Die meisten dieser Skriptsprachen sind für spezielle Anwendungen gedacht, etwa SQL zur Datenbankabfrage oder PostScript als Seitenbeschreibungssprache. In den folgenden Kapiteln werden mehrere von ihnen behandelt.

Deskriptive Sprachen. Nach 1975 entstanden die deskriptiven Sprachen, die in reinem Zustand keine Programmlogik beinhalten. Dazu gehören XML-Anwendungen zur strukturierten und formatierten Beschreibung von Daten oder die Interface Definition Language (IDL) zur Beschreibung von Programmierschnittstellen. Eine gewisse Zwitterrolle spielt XML als Metasprache, denn es erlaubt z.B. die Definition von XSLT als nahezu funktionale Programmiersprache und kann deshalb nicht als rein deskriptiv gesehen werden [H04].

1.2.5 Fünfte Generation

Die fünfte Generation von Programmiersprachen (seit ca. 1980) umfasst teilweise die deklarativen sowie die logischen und die funktionalen Sprachen. Hier werden Probleme durch Rand- und Zwangsbedingungen beschrieben und nicht mehr explizit gelöst. Das System muss selbst in Gewissen Grenzen einen Lösungsweg finden. In diesem Buch wird PROLOG als Beispiel einer logischen Programmiersprache behandelt.

1.3 Klassifizierung von Programmiersprachen

Moderne Programmiersprachen lassen sich nach verschiedenen Kriterien klassifizieren. Die wichtigsten Unterscheidungsmerkmale sind

- das Programmierparadigma (siehe Abschnitt 1.3.1),
- der Abstraktionsgrad (siehe Abschnitt 1.3.2) sowie
- das Ausführungsschema (siehe Abschnitt 1.3.3)

der betrachteten Programmiersprache.

1.3.1 Programmierparadigmen

> Als Programmierparadigma einer Programmiersprache bezeichnet man die Sichtweise auf und den Umgang mit den zu verarbeitenden Daten und Operationen.

Imperatives Programmierparadigma. Imperative Sprachen bestehen aus einer Folge von Anweisungen, die streng sequenziell abgearbeitet werden. Beispiele solcher Anweisungen sind das Beschreiben einer ↑Variablen mit einem bestimmten Wert oder der Aufruf einer speziellen Unterfunktion. Prozedurale Programmiersprachen beruhen demnach hauptsächlich auf ↑Funktionen und ↑Prozeduren zur Kapselung und Wiederverwendung von Funktionalität. ↑Objekte und ↑Klassen existieren nicht. Der bekannteste Vertreter dieser Kategorie ist die Programmiersprache C, die bereits Anfang der Siebzigerjahre für die Programmierung der ersten UNIX-Systeme entwickelt wurde. Auch ist C immer noch eine der am häufigsten verwendeten Programmiersprachen im Bereich eingebetteter Systeme und der Programmierung von Betriebssystemen. C wird in Kapitel 2 genauer behandelt.

Objektbasiertes Programmierparadigma. Eine einheitliche Definition für objektbasierte Programmiersprachen zu finden ist nicht ganz einfach. Zwei Charakterisierungen sind aber häufig in der Literatur zu finden:

- Eine Programmiersprache ist dann objektbasiert, wenn sie ↑Objekte unterstützt, aber weiterführende Konzepte wie ↑Klassen und deren Beziehungen fehlen.

- Im „Handbuch Informatik“ [RePo06] wird als Abgrenzung die ↑Vererbung genannt. Sprachen, die ↑Klassen, ↑Objekte, ↑Methoden und ↑Attribute, aber keine Vererbung unterstützen, werden als objektbasiert eingestuft.

Ein Vertreter der objektbasierten Sprachen ist JavaScript, das in Kapitel 13 behandelt wird. Vererbung kann darin momentan (Stand 2006) nur indirekt nachgebaut werden.

Objektorientiertes Programmierparadigma. Im Gegensatz zu objektbasierten Sprachen verfügen objektorientierte Sprachen über weitergehende Konzepte, wie ↑Vererbung und ↑Polymorphie. Wegbereiter der objektorientierten Programmierung war die Programmiersprache Smalltalk. Obwohl die Sprache heute nur noch vereinzelt eingesetzt wird, finden sich viele ihrer Konzepte in den modernen Sprachen wieder. Vertreter dieser Sprachen sind Java sowie die diversen C-Derivate C++, C# und Objective-C.

Funktionales Programmierparadigma. Anders als imperative oder objektorientierte Sprachen kennen rein funktionale (oder applikative) Sprachen keine Wertzuweisungen. Ein funktionales Programm besteht damit ausschließlich aus einer Reihe von Funktionsdefinitionen, die eine Eingabe in eine entsprechende Ausgabe transformieren. Die Grundlage der funktionalen Programmierung reicht bis in die frühen Vierzigerjahre zurück und basiert auf dem 1941 von Alonzo Church (1903 – 1995) definierten λ-Kalkül.

Funktionale Programmiersprachen werden insbesondere im Bereich der künstlichen Intelligenz, im Compilerbau und der Computeralgebra eingesetzt. Der bekannteste Vertreter dieser Kategorie ist die Programmiersprache LISP, die in Kapitel 22 besprochen wird.

XSL Transformations (XSLT) ist eines der jüngsten Beispiele für funktionale Programmiersprachen mit industrieller Relevanz.

Logisches Programmierparadigma. Im Mittelpunkt der logischen (oder prädikativen) Programmierung steht der Aufbau einer Datenbasis, die aus Fakten und Regeln besteht. Fakten sind wahre Aussagen und entsprechen den Axiomen der klassischen Mathematik. Fakten werden in Form von Prädikaten angegeben, die eine beliebige Stelligkeit besitzen dürfen. Im Gegensatz zu den anderen Programmierparadigmen stellt die logische Programmierung die Problemformulierung und nicht den Lösungsalgorithmus in den Vordergrund.

Der bekannteste Vertreter aus dieser Kategorie ist die Programmiersprache PROLOG, die in Kapitel 23 genauer behandelt wird.

Deklaratives Programmierparadigma. Dieser Begriff wird zusammenfassend für das ↑funktionale und das ↑logische Programmierparadigma verwendet.

1.3.2 Abstraktionsgrade

Der Abstraktionsgrad einer Sprache definiert, wie weit sich die Semantik der einzelnen Sprachkonstrukte von den Grundbefehlen des Mikroprozessors unterscheidet. Den niedrigsten Abstraktionsgrad besitzt die ↑Maschinensprache. Die Maschinensprache wird direkt von dem auszuführenden Prozessor verstanden. Die nächste Abstraktionsebene bildet die Assemblersprache (↑Assembler, vgl. auch Abschnitt 1.2).

Typische Assemblerdialekte erlauben den Einsatz symbolischer Namen und Sprungmarken sowie die Definition von Makro-Blöcken. Der Assembler übersetzt den Quelltext in die Maschinensprache, indem die symbolischen Namen und Sprungmarken durch numerische Opcodes und Adressen ersetzt sowie die Makro-Blöcke aufgelöst werden.

Höhere Programmiersprachen abstrahieren deutlich von der zu Grunde liegenden Prozessorarchitektur. So besitzen prozedurale Sprachen Kontrollstrukturen, die durch den Compiler in elementare Prozessorbefehle umgesetzt werden müssen. Die Datenabstraktion wird durch die Definition neuer Datentypen wie Strukturen oder Arrays unterstützt.

Noch einen Schritt weiter gehen die objektorientierten Sprachen. Hier werden Daten und Methoden zu einer gedanklichen Einheit verschmolzen, die keine direkte Entsprechung auf Prozessorebene mehr besitzt. Den größten Abstraktionsgrad weisen die deklarativen Sprachen auf, die keine direkte Unterscheidung mehr zwischen Daten und Operationen kennen.

1.3.3 Ausführungsschemata von Programmiersprachen

Compilierende Programmiersprachen. Die Quelltexte des Programms werden vor der ersten Ausführung zunächst durch einen Compiler in ein Maschinenprogramm übersetzt. Durch die Vorübersetzung wird eine besonders hohe Ausführungsgeschwindigkeit erreicht. Des Weiteren können die Quelltexte während der Compilierung durch den Compiler auf Fehler untersucht werden.

Interpretierende Programmiersprachen. Die Quelltexte des Programms werden zur Laufzeit von einem Interpreter eingelesen und Befehl für Befehl abgearbeitet. Die Abarbeitung eines interpretierten Programms geschieht langsamer als die eines compilierten Programms, bietet jedoch ein größeres Maß an Flexibilität. Einerseits wird der aufwändige Edit-Compile-Test-Zyklus bei kleinen Programmänderungen überflüssig, andererseits kann der Interpreter die Ausführung des Programms während der Laufzeit überwachen.

Mischformen. Konnten die frühen Programmiersprachen eindeutig einer der beiden Klassen zugeordnet werden, so verwischt der Unterschied heute zunehmend. Moderne Programmiersprachen wie z.B. Java oder C# compi-

lieren das auszuführende Programm zunächst in einen Zwischencode, der später durch einen Interpreter in Form einer virtuellen Maschine ausgeführt wird. In Java beispielsweise werden sogar zur Laufzeit so genannte „Hotspots“ (häufig benutzte Programmsegmente) erkannt und während des Programmlaufs in ein Maschinenprogramm übersetzt. Auf diese Weise können Java- oder C#-Programme in einer überwachten Umgebung ausgeführt werden, ohne die Laufzeiteigenschaften im Vergleich zu vollständig compilierten Programmen drastisch zu verschlechtern.

1.4 Gebräuchliche Programmiersprachen

Heute existieren weit über 1000 verschiedene Programmiersprachen, die teilweise nur wenige Jahre en vogue sind.

Tabelle 1.1: Die wichtigsten Programmiersprachen

Sprache	Kurzbeschreibung
Ada	Sprache für sicherheitskritische Anwendungen, strukturierte Sprache im Echtzeitbereich, ab den 1970er-Jahren von Jean Ichbiah bei Honeywell Bull entworfen, wurde vom US-Verteidigungsministerium unterstützt, vgl. Kapitel 11
ALGOL	„Algorithmic Language“, ab 1960, Vorgänger vieler imperativer Sprachen
ANSYS	„Analysis Systems“, zur Beschreibung von Berechnungen mit der Finite-Elemente-Methode
APL	Programmiersprache für Großrechner der alten Generation, spezifiziert 1961 von Iverson
Apple Script	Universelle Scripting-Schnittstelle unter Mac OS X
APT	“Automatic Programmed Tools“, Sprache für Werkzeugmaschinen ab 1959
Assembler	Prozessornahe Programmiersprache, vgl. Glossar in Abschnitt 1.6
AutoCAD	4GL-Sprache für Computer Aided Design
awk	Stringverarbeitungssprache im Betriebssystem UNIX
BASIC	Für Anfänger konzipierte Allzweckprogrammiersprache, vgl. Kapitel 6
BCL	Vorläufer von C
C	Hardwarenahe Universalprogrammiersprache, vgl. Kapitel 2
C*	C-Dialekt zur Parallelprogrammierung
C#	C-Derivat der Firma Microsoft, vgl. Kapitel 4
C++	Objektorientierte Erweiterung von C, vgl. Kapitel 3
CHILL	CCITT High Level Language, ab 1976 für Telekommunikationszwecke entwickelt
CLOS	Common LISP Object System, ab 1988, Erweiterung von LISP um objektorientierte Sprachmerkmale
CLP	Constrained Logic Programming, ab 1990

Sprache	Kurzbeschreibung
COBOL	Common Business Oriented Language, ab ca. 1960, im Bankenbereich heute immer noch verbreitet
CSP	Communicating Sequential Processes, von Tony Hoare entwickelte formale Sprache zur Beschreibung paralleler Prozesse
Delphi	Objektorientierte Programmiersprache, vgl. Kapitel 7
DSL	Design System Language, Vorläufer von PostScript
Dylan	Dynamic Language, Abkömmling von Scheme, unter Führung von Apple 1992 für den Newton PDA entwickelt
Eiffel	Objektorientierte Programmiersprache, vgl. Kapitel 12, 1987
Erlang	Echtzeitfähige Sprache mit funktionalem Paradigma
Forth	Problemorientierte Sprache mit umgekehrter polnischer Notation zur Beschreibung von Ausdrücken, 1960 von Charles H. Moore entwickelt
FORTRAN	Erste höhere Programmiersprache ab 1954, dank leistungsfähiger Numerikbibliotheken noch heute im wissenschaftlichen Bereich eingesetzt, vgl. Kapitel 9
Haskell	Funktionale Programmiersprache, die auf dem Lambda-Kalkül sowie Miranda basiert, benannt nach dem amerikanischen Mathematiker Haskell Brooks Curry, ab 1990
HTML	Hypertext Markup Language, deskriptive Sprache für Web-Seiten, vgl. [H04]
IDL	Interface Definition Language, zur abstrakten Beschreibung von verteilten Objekten
Java	Objektorientierte Programmiersprache im universellen Einsatz, vgl. Kapitel 5
JavaScript	Skriptsprache, hauptsächlich zur Einbettung in HTML Seiten, vgl. Kapitel 13
JCL	Job Control Language, Skriptsprache, ab 1960
LISP	List Processor, symbolorientierte Sprache als Implementierung des Lambda-Kalküls, ab 1960, Einsatz im Bereich der künstlichen Intelligenz (KI) und der Computeralgebra, vgl. Kapitel 22
Logo	LISP-Abkömmling, ab 1970, entwickelt von Seymour Papert, kindgerechte Umsetzung einer Interpretersprache
Maple	Programmierbares Computeralgebrasystem, vgl. Kapitel 19
Mathematica	Programmierbares Computeralgebrasystem, vgl. Kapitel 20
MATLAB	MATrix LABoratory, mathematische Software, breiter Einsatz im Bereich der computergestützten Simulation, vgl. Kapitel 21
Miranda	Leicht zu erlernende funktionale Skriptsprache, ab 1985, von David Turner, wird nicht weiter gepflegt
Modula-2	Erweiterung von Pascal um modulare Konzepte, ab 1978 von Niklaus Wirth entwickelt
Mondrian	Funktionale Sprache, ähnlich Haskell
Oberon	Nachfolger von Modula-2 und Pascal, entwickelt von Niklaus Wirth und Jürg Gutknecht, Haupteinsatz ist die Lehre

Sprache	Kurzbeschreibung
Objective-C	Smalltalk-ähnliche Erweiterung von C, Mitte der 1980er-Jahre von Brad Cox entwickelt, objektorientierte Programmierschnittstelle von Mac OS X
Occam	Sprache für Parallelprogrammierung von Transputern, 1985
Pascal	Prozedurale Programmiersprache, ab 1968 von Niklaus Wirth entwickelt, benannt nach dem französischen Mathematiker Blaise Pascal, früher häufig in der Lehre eingesetzt
PEARL	Process and Experimental Automation Realtime Language, Sprache für Echtzeit- und Multitaskingsysteme
Perl	Universelle Scriptsprache, ab 1987 von Larry Wall entwickelt, Grundlage vieler Web-Anwendungen, vgl. Kapitel 14
PHP	PHP: Hypertext Preprocessor, dynamische Erstellung von Webseiten, vgl. Kapitel 15
PL/I	Programming Language I, Vorläufer von C, ab 1964 von IBM entwickelt
PostScript	Seitenbeschreibungssprache, seit 1984 von Adobe entwickelt, vgl. [H04]
PROLOG	Programming in Logic, Anfang der 1970er-Jahre von Alain Colmerauer entwickelt, Einsatz im Bereich der künstlichen Intelligenz, vgl. Kapitel 23
Python	Skriptsprache, die ursprünglich für das verteilte Betriebssystem Amoeba entworfen wurde, ab 1991, vgl. Kapitel 16
Ruby	Skriptsprache ähnlich Python, mit objektorientierten und funktionalen Elementen, ab 1993 von Yukihiro Matsumoto entwickelt, vgl. Kapitel 17
Scheme	LISP-Dialekt, 1975 am MIT entwickelt, unterstützt auch imperative Sprachelemente
Self	Objektorientierte und prototypenbasierte Sprache, hauptsächlich im akademischen Einsatz, ab 1986 von David Ungar und Randall B. Smith entwickelt
SGML	Standard Generalized Markup Language, deskriptive Sprache zur Markierung von Texten, ab 1978
Simula	Vorläufer von Smalltalk, 1966 von Ole-Johan Dahl und Kristen Nygaard entwickelt
Smalltalk	Objektorientierte Programmiersprache, Wegbereiter für viele andere objektorientierte Sprachen, in den 1970er-Jahren von Alan Kay, Daniel Ingalls und Adele Goldberg am XEROX Parc Forschungszentrum entwickelt, vgl. Kapitel 10
SNOBOL	String oriented Symbolic Language, Sprache zur Manipulation von Zeichenketten, 1965 von D. J. Farber, R. E. Griswold and F. P. Polensky bei AT&T entwickelt
SQL	Standard Query Language, Datenbankabfragesprache, seit 1980
Tcl	Tool command language, leicht zu erlernende Skriptsprache mit Einflüssen von C und LISP, seit 1988 von John Ousterhout entwickelt, vgl. Kapitel 18
VBasic	Visual Basic, auf BASIC basierende Sprache von Microsoft, entwickelt 1991 und ständig weiter entwickelt, vgl. Kapitel 6
VRML	Virtual Reality Modeling Language, eine deskriptive Sprache zur Modellierung dreidimensionaler Welten, vgl. [H04]
XML	eXtensible Markup Language, eine Metasprache zur Definition von deskriptiven Sprachen, XML ist eine Teilmenge von SGML, vgl. [H04]

Sprache	Kurzbeschreibung
XSLT	Funktionale Programmiersprache zur Transformation von XML-Dokumenten entweder wieder in XML oder in andere Formate, entwickelt von James Clark und Michael Kay, seit 1999

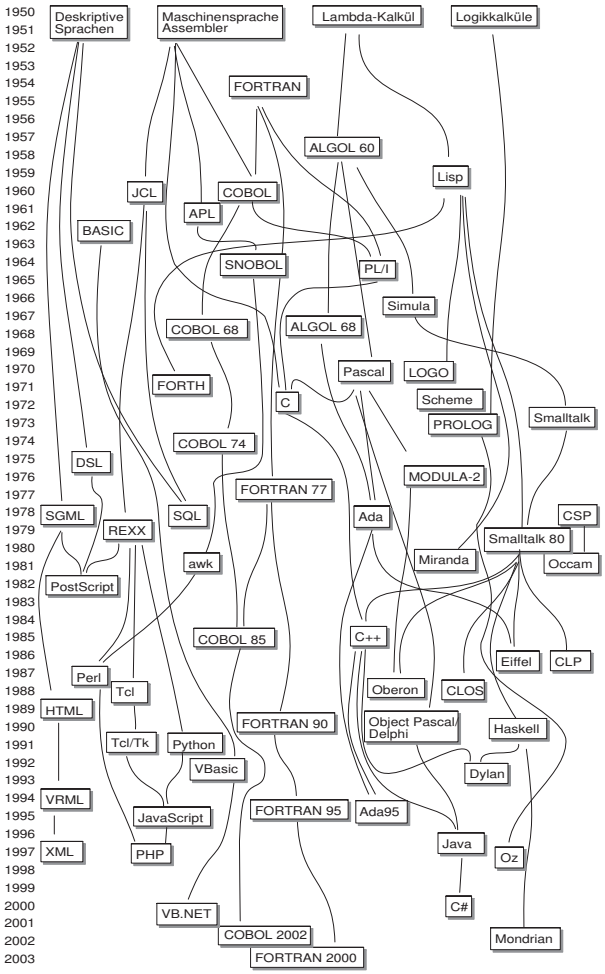

Abbildung 1.1: „Stammbaum“ der wichtigsten Programmiersprachen und ihrer Abkömmlinge