Vorlesung

TYPSYSTEME

Franz Puntigam

Technische Universität Wien
Institut für Computersprachen
Programmiersprachen und Übersetzer

www.complang.tuwien.ac.at/typsysteme.html

Inhalt

1. Begriffsbestimmungen und Überblick
2. Einfache Typmodelle (Grundlagen)
3. Typen in imperativen Sprachen (Ada)
4. Modelle polymorpher Typsysteme
   und Typen in funktionalen Sprachen
5. Typen in objektorientierten Sprachen

Was ist ein Typ? (1)

Typen sind Werkzeuge zur Klassifikation von Werten aufgrund ihrer Eigenschaften, ihres Verhaltens und ihrer möglichen Verwendungen.

Typen sind Werkzeuge des Softwareentwicklers zur Unterstützung der Softwareerstellung, -weiterentwicklung und -wartung (objektorientierte Programmierung).

Typen sind Schutzschilder gegen unbeabsichtigte (falsche) Interpretationen von rohen Daten.

Typen machen die statischen Teile eines Programms sichtbar.

Was ist ein Typ? (2)

Typen bestimmen Speicherauslegungen für Werte.

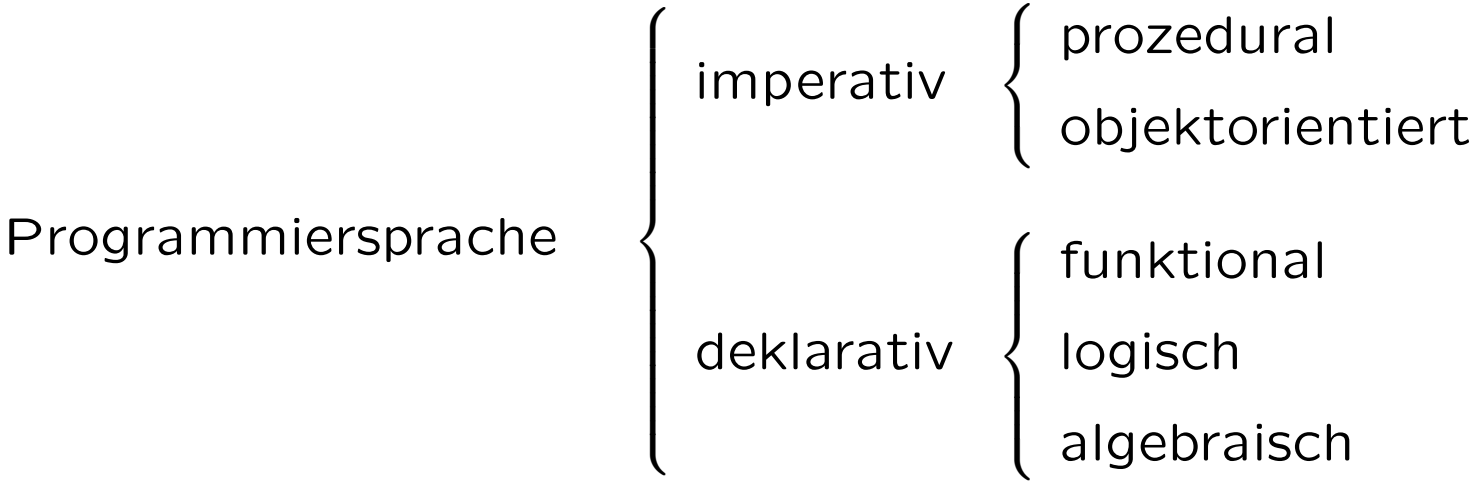
Typen sind syntaktische Einschränkungen auf Ausdrücken zur Zusicherung von Operator-Operand-Kompatibilitäten.

Typen legen Verhaltens-Invarianten fest, die alle Instanzen der Typen erfüllen müssen.

Ein Typsystem ist eine Menge von Regeln, die jedem Ausdruck einen eindeutigen, allgemeinsten Typ zuordnet. Dieser beschreibt die Menge aller Umgebungen, in denen der Ausdruck vorkommen kann und eine Bedeutung hat.

Ein Typ beschreibt eine Menge von Instanzen.

Einteilung von Programmiersprachen



Imperative Sprachen

Charakteristische Merkmale imperativer Sprachen:

- Imperative Befehle ändern den Programmzustand.
- Der wichtigste Befehl ist die destruktive Zuweisung.

Weitere Merkmale prozeduraler Sprachen:

- Wichtigster Abstraktionsmechanismus ist die Prozedur.
- Saubere Programme durch strukturierte Programmierung.

Weitere Merkmale objektorientierter Sprachen:

- Wichtigster Abstraktionsmechanismus ist das Objekt.
- Inkrementelle Programmerstellung wird unterstützt.
- Zustand und Verhalten eines Objekts kann (eher als in anderen Sprachklassen) unabhängig vom gesamten Programm betrachtet werden.

Deklarative Sprachen

Charakteristische Merkmale deklarativer Sprachen:

- Sie beruhen auf einem mathematischen Modell.
- Modelle sind oft statisch (zustandsfreie Programme).
- Grundlegende Sprachelemente sind Symbole.

Funktionale Sprachen beruhen auf dem Lambda-Kalkül.
Imperative Sprachen wurden davon wesentlich beeinflusst.

Logische Sprachen beruhen auf der Prädikatenlogik.

Algebren werden als Spezifikationssprachen und als formale Grundlage von Typen verwendet.

Einteilungen von Typen

Kriterium	einige Ausprägungen
Definierbarkeit	vordefiniert, definierbar
Art der Spezifikation	extensional, intensional
Typfestlegung	statisch/dyn., stark/schwach
Mächtigkeit	flexibel, sicher, einfach
Typzwang	konservativ, optimistisch
Durchlässigkeit	dicht, löchrig
Typäquivalenz	Struktur-, Namensgleichheit
Abstraktionsgrad	abstrakt, konkret
Ausdrücklichkeit	implizit, explizit
Typüberschneidungen	monomorph, polymorph

Definierbarkeit von Typen

In allen Sprachklassen gibt es Sprachen, in denen alle verwendbaren Typen vordefiniert sind.

Die meisten neueren Programmiersprachen (in allen Sprachklassen) erlauben dem Programmierer, Typen zu definieren.

Es gibt eine allgemeine Tendenz, dem Programmierer die Definition immer mächtigerer Typen zu erlauben.

Arten der Typspezifikation

Eine *extensionale* Spezifikation zählt alle Instanzen auf.

- Typen sind meist nicht strukturiert.
- Analogon bei Mengen: $M = \{2, 3, 5, 7, 11, 13\}$

Eine *intensionale* Spezifikation beschreibt die Instanzen durch die Instanzen anderer Typen mit bestimmten Eigenschaften.

- Typen sind oft strukturiert und/oder durch bestimmte Eigenschaften eingeschränkt.
- Analogon bei Mengen: $M = \{i \in 1..16 \mid p(i)\}$

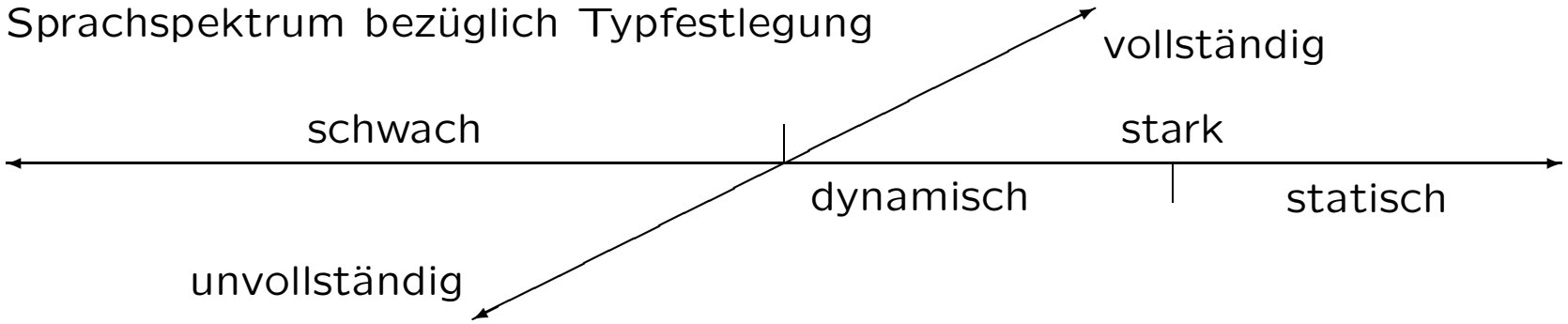
Intensionale Spezifikationen können mächtig sein, wesentlich mächtiger als in der Praxis verwendbar. In Programmiersprachen sind nur handhabbare Spezifikationen erlaubt.

Typfestlegung

Vollständig typisierte Sprache: Alle Ausdrücke sind typkonsistent. Typkonsistenz wird spätestens zur Laufzeit überprüft.

Stark typisierte Sprache: Die Sprache ist typisiert und alle Typfehler werden zur Übersetzungszeit erkannt.

Statisch typisierte Sprache: Der eindeutige Typ jedes Ausdrucks ist statisch festgelegt.



Typkonsistenz

Ein Programm ist typkonsistent wenn für jede Anwendung eines Operators auf einen Operanden der Operand einen vom Operator verlangten Typ hat.

In stark typisierten Sprachen wird Typkonsistenz vom Compiler (im wesentlichen anhand der Syntax) überprüft.

Der Begriff "Typkonsistenz" kann von Sprache zu Sprache stark divergieren.

Typkonsistenz ist kein Allheilmittel gegen Fehler.

Mächtigkeit von Typsystemen

Widersprüchliche Ziele:

- *Sicherheit*, die viele Fehlerarten ausschließt;
- *Flexibilität*, die dem Programmierer große Freiheit lässt;
- *Einfachheit* sowohl für den Programmierer als auch für den Compiler (bzw. Compilerschreiber).

Großer Entscheidungsspielraum für das Design von Sprachen.

Tendenz zu sicheren, flexiblen Typsystemen auf Kosten der Einfachheit und Verständlichkeit.

Typzwang

Mögliche Ergebnisse einer Typüberprüfung:

- Das Programm ist garantiert typkonsistent.
- Das Programm ist garantiert nicht typkonsistent.
- Weder Typkonsistenz noch Typinkonsistenz können zweifelsfrei festgestellt werden.

Eine Typüberprüfung heisst *konservativ*, wenn nur garantiert typkonsistente Programme übersetzt werden können.

Bei *optimistischer* Typüberprüfung kann das Programm auch übersetzt werden, wenn es nur möglicherweise typkonsistent ist. Tatsächliche Typfehler in übersetzten Programmen werden zur Laufzeit erkannt.

Durchlässigkeit

In vollständig typisierten Sprachen werden alle Typfehler spätestens zur Laufzeit erkannt.

Viele Sprachen sind nur unvollständig typisiert da

- der Programmierer nicht zu sehr eingeschränkt werden soll;
- aus Effizienzgründen auf Typüberprüfungen zur Laufzeit verzichtet wird.

Solche *Löcher* führen leicht zu unerkannten Fehlern.

In den Typsystemen neuerer Sprachen gibt es kaum Löcher.

Typäquivalenz

Zwei Typen sind äquivalent wenn sie

- gleich strukturiert sind (Strukturgleichheit) oder
- den gleichen Namen haben (Namensgleichheit).

Typäquivalenz aufgrund von Strukturgleichheit ist oft einfacher handhabbar und hat eine klarere Semantik.

Typäquivalenz aufgrund von Namensgleichheit erlaubt zusätzliche Abstraktionen.

Praktisch wird meist ein Mischsystem eingesetzt.

Abstrakte Datentypen (ADT)

Ein ADT "versteckt" die interne Darstellung seiner Instanzen (Datenkapselung). Nach aussen hin wird eine Instanz als abstraktes Objekt ohne innere Struktur repräsentiert.

Auf Instanzen sind nur die vom ADT exportierten Operationen anwendbar. (ADT = Daten + Operationen)

Meist beruht ein ADT auf Typäquivalenz aufgrund von Namensgleichheit.

Abstraktionsgrad von Typen

Abstrakte Typen sind unvollständig spezifizierte Typen.

Aus abstrakten Typen werden durch Angabe der fehlenden Spezifikationen *konkrete Typen* erzeugt.

Abstrakte Typen dienen als Strukturierungsmittel zur Erzeugung mehrerer gleich oder ähnlich strukturierter Typen.

Ausdrücklichkeit der Typdefinition

In den meisten neueren imperativen Sprachen werden Typen von Ausdrücken *explizit* (in einer Deklaration) angegeben.

In vielen deklarativen Sprachen sind die Typen von Ausdrücken *implizit* definiert.

Die Typen einiger grundlegender Konstanten (z.B. Zahlen) sind in den meisten Sprachen implizit festgelegt.

Typüberschneidungen

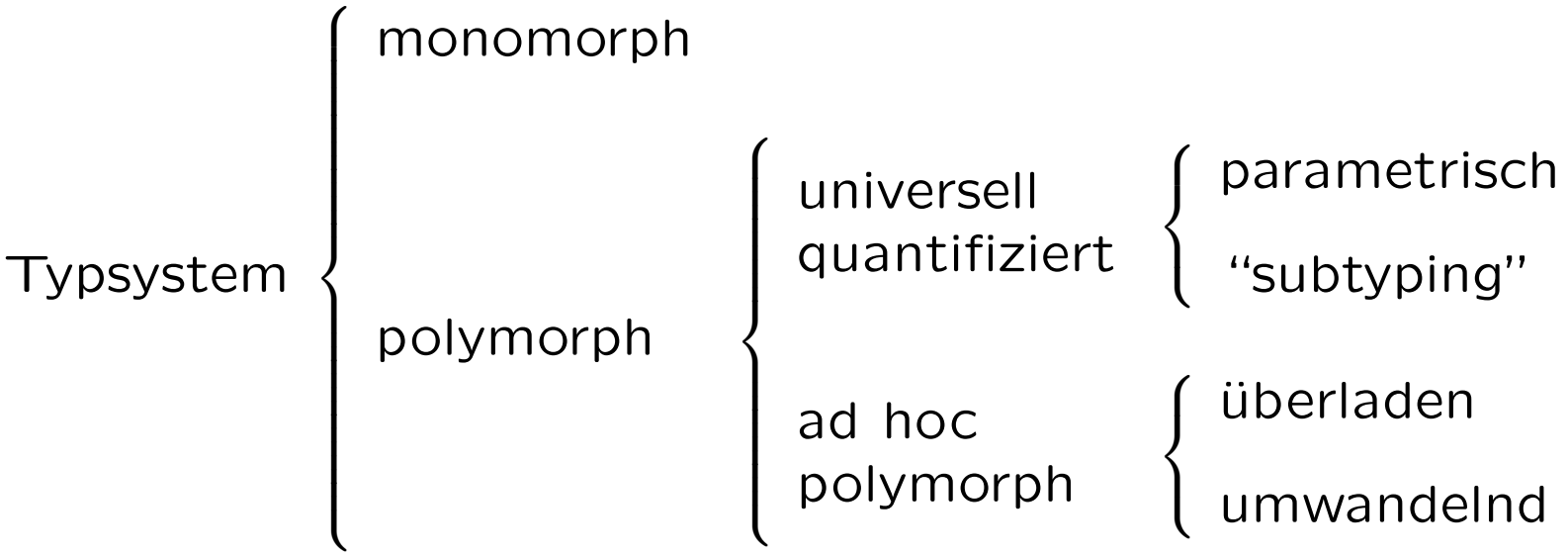
Der Typ eines Ausdrucks muss nicht immer eindeutig sein. Das heisst, ein Ausdruck kann mehrere Typen haben.

Beispiel: Der Ausdruck "`(a 1)`" in Lisp ist sowohl eine "Liste" als auch eine "zweielementige Liste von einem Symbol und einer ganzen Zahl".

In vielen neueren (funktionalen) Sprachen hat jedoch jeder Ausdruck einen eindeutig bestimmbaren *allgemeinsten* Typ.

Typsysteme, in denen Ausdrücke mehrere Typen haben können, sind *polymorph.* Alle anderen Typsysteme sind *monomorph.*

Polymorphe Typsysteme



Parametrische (generische) Typsysteme

Typausdrücke können *Typparameter* enthalten, für die wieder Typausdrücke eingesetzt werden.

Beispiel: "`List`$[a] \rightarrow a$" enthält den Typparameter a. Wenn `Integer` für a eingesetzt wird, erhält man den Typausdruck "`List[Integer]` $\rightarrow$ `Integer`".

Ein Typausdruck mit freien Typparametern bezeichnet die Menge aller Typausdrücke, die durch Einsetzen von Typausdrücken ohne freie Parameter generiert werden können.

Anders ausgedrückt: Typparameter werden implizit als "universell über die Menge aller Typausdrücke quantifizierte Variablen" betrachtet.

“Subtyping”

“Subtyping” (Vererbung) wird in objektorientierten Sprachen eingesetzt.

Ein Typ (ADT) beschreibt auch die Instanzen seiner *Untertypen.*

Beispiel: Der Typ `Person` hat die beiden Untertypen `Student` und `Angestellter`.

An eine Funktion mit einem Parameter vom Typ `Person` kann auch ein Argument vom Typ `Student` oder `Angestellter` übergeben werden.

Diese Funktion akzeptiert alle Argumente vom Typ a, wobei a ein “universell über `Person` und dessen Untertypen quantifizierter Typparameter” ist.

Statisches und dynamisches Binden

Wenn ein ADT eine Funktion exportiert, dann exportiert auch jeder seiner Untertypen eine Methode mit derselben (oder einer varianten) Schnittstelle.

Der Compiler kennt oft nur die Typen der Parameter, aber nicht die tatsächlichen Argumenttypen.

Wenn der Compiler nicht zur Übersetzungszeit feststellen kann welche konkrete Funktion ausgeführt werden muss, kann diese Funktion erst zur Laufzeit durch *dynamisches Binden* bestimmt werden.

Wenn der Compiler die auszuführende Funktion kennt, kann er die Funktion *statisch* an den Aufruf binden.

Überladene Typsysteme

Ein und derselbe Name bezeichnet verschiedene Funktionen, die sich durch ihre formalen Parameter unterscheiden und keinerlei Gemeinsamkeiten haben müssen.

Typen der übergebenen Argumente entscheiden, welche Funktion ausgeführt wird (vom Compiler feststellbar).

Überladen dient meist nur der syntaktischen Vereinfachung.

Beispiel: “/” bezeichnet sowohl die ganzzahlige als auch die Fließkomma-Division.

Typumwandlung

Im Gegensatz zum Überladen ist Typumwandlung eine semantische Operation.

Typumwandlungen erfolgen implizit oder explizit.

Beispiel: In C werden Instanzen von `char` und `short` bei der Argumentübergabe implizit in Instanzen von `int` umgewandelt.

Durch gleichzeitige Unterstützung von Überladen und impliziter Typumwandlung können sich diffizile semantische Unklarheiten ergeben.