Statische Typüberprüfung

M. Anton Ertl
TU Wien

Problem

- Fehler in Programmen
- Forth-Programmierer schreiben Tests
- Könnte nicht einen Teil der Fehler der Computer finden?
  Am besten gleich beim Compilieren?

Lösung?

- Typprüfung in anderen Programmiersprachen seit Jahrzehnten
- Typinferenz (Milner 1978) braucht keine Deklarationen
- oder Typüberprüfung zur Laufzeit

Gefahren

> As programmers learned C with Classes or C++, they lost the ability to quickly find the "silly errors" that creep into C programs through the lack of checking. Further, they failed to take the precautions against such silly errors that good C programmers take as a matter of course. After all, "such errors don't happen in C with Classes." Thus, as the frequency of run-time errors caused by uncaught argument type errors goes down, their seriousness and the time needed to find them goes up.

Bjarne Stroustroup

- C verändert sich weg vom "'high-level assembler"'.
- Trotzdem besteht Interesse an so einem Prüfer

Ziele

- Neuen Code prüfen
- Änderungen in altem Code prüfen
- Sollte ohne Typdeklarationen brauchbar sein
  $\Rightarrow$ Typinferenz
  Mit Deklarationen bessere Überprüfung bieten
- wenig Fehlalarme
  kleine Änderung in großem Programm
  Neuer Code, der existierenden Code verwendet
  Fehlalarme durch einfache Annotationen abschalten
- Verständliche Fehlermeldungen
  Bei Typinferenz evtl. problematisch

Was gibt es bisher?

- StrongForth von Stefan Becher
  recht fern von Standard-Forth
  8086-System

- Papers und Vorträge von Peter Knaggs, Jaanus Pöial, Jürgen Pfitzenmaier
  Demo von Jaanus Pöial
  keine Produktionssysteme

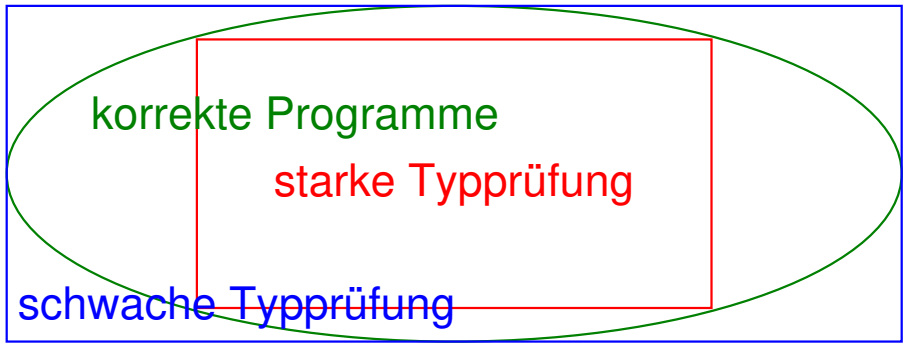
- Praktikum eines meiner Studenten
  nicht praxistauglich (zu viele Fehlalarme)
  keine einfache Verbesserungsmöglichkeit

- Diplomarbeit von Gregor Riegler
  konfigurierbares Typsystem

- Warum ist das so schwierig?

Allgemeines

- Typen für Overloading und Prüfung
- Statische Typüberprüfung
- Dynamische Typüberprüfung
- Forth hat gar keine Typüberprüfung
  Programmierdisziplin statisch
- Stack-Effekt-Kommentare entsprechen Deklarationen
  leider unbrauchbar



Gelöst: Stacktiefe

- Prüfer machbar (Glass 1983)
- Änderung der Stacktiefen mitzählen
- Bei Zusammenfluss müssen Tiefen gleich sein
  inkl. mehrere EXITs in Definition
- Return-Stack-Tiefe darf nicht negativ werden

```
?do i loop
if >r then
```

Stacktiefe: Probleme

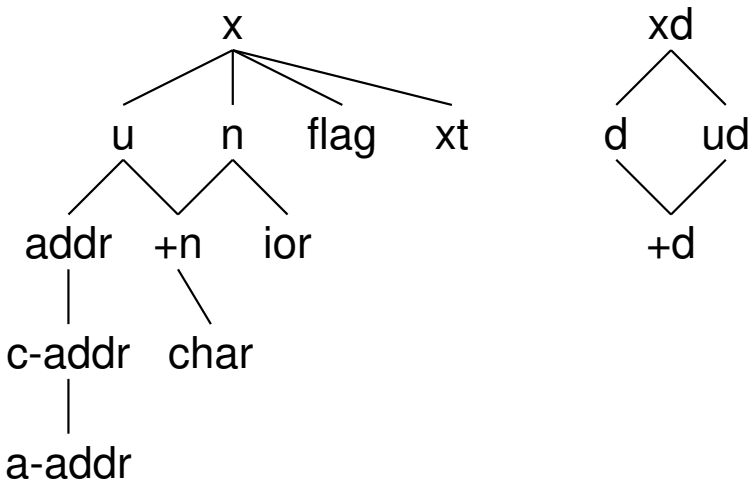
- `execute`, `defer`red words
  passenden Stack-Effekt annehmen (keine Prüfung)
  oder Annotation

- `?dup`
  `?dup if/while/until`
  `?dup 0= if/while/until`
  als Einheit betrachten

- `r> drop exit`
  komplexe Analyse
  oder Annotation

Und Typen?

Typhierarchie (im Standard)

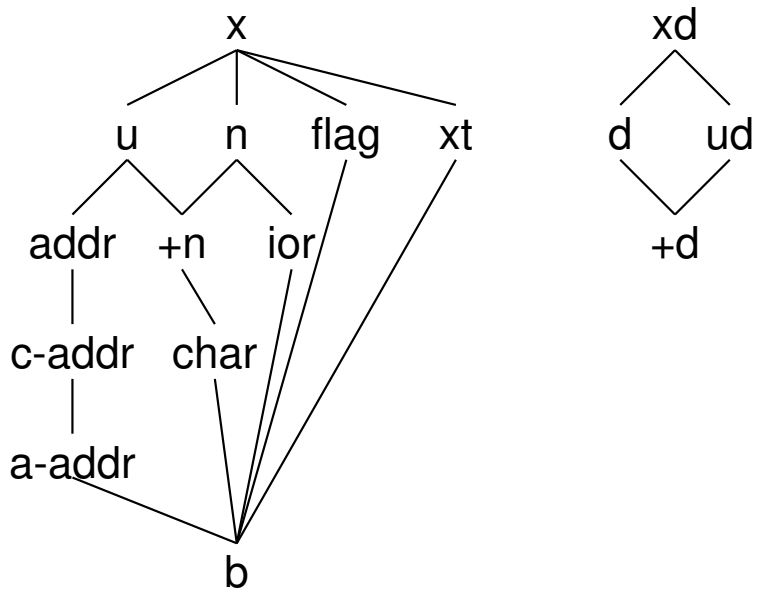


Kovarianz und Kontravarianz

```
! ( x a-addr -- )
@ ( a-addr -- x )
0< ( n -- flag )
a @ ( >=x und <=n ) 0< ( >=flag und <=x ) b !
```

Standard-Typregeln unbrauchbar

Typhierarchie (Variante)

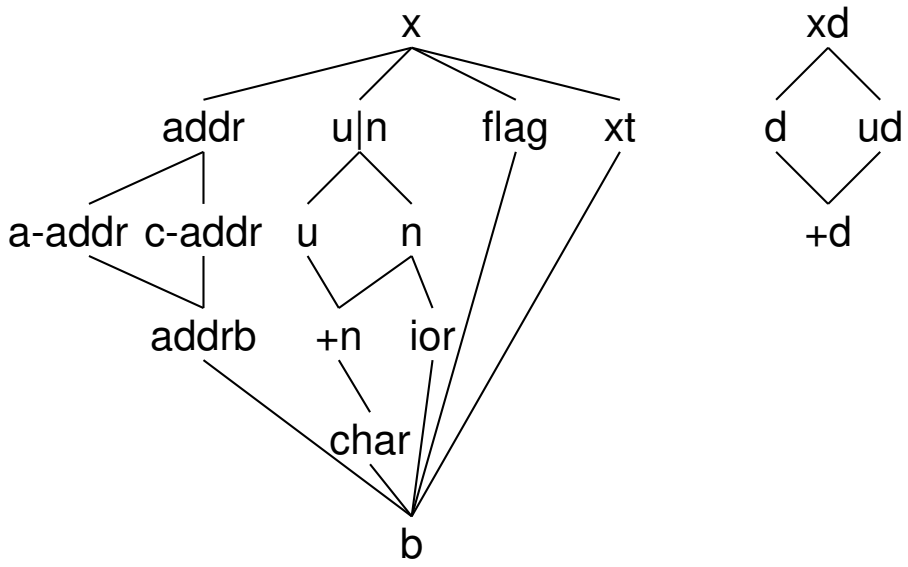



```
! ( x a-addr -- )
@ ( a-addr -- b )
0< ( n -- flag )
a @ ( >=b und <=n ) 0< ( >=flag und <=x ) b !
... negate ( >=n und <=x ) b ! b @ ( >=b und <=u ) 5 u<
```

Typregeln

```
Standard:
+ ( n|u n|u -- n|u )

Besser:
+ ( n|u n|u -- +n )
+ ( addr n|u -- addr )
+ ( a-addr a-n|u -- a-addr )
```



Typregeln überarbeiten, an realem Code testen (auf Fehlalarme)
Programme mit Typfehlern aus der Praxis sammeln

Typvariablen

```
@ ( V-addr -- V )
variable u ( -- u-addr )
u @ 5 <
```

Algorithmische Komplexität

- Alle Varianten von + probieren (schlimmstenfalls exponentiell)
- Allgemein: je nach Typsystem O(n)–unentscheidbar
- Typvariablen und Rekursion problematisch
- Aber auch für Fehlermeldungen
- Laxerer Checker oder mehr Deklarationen

Beispielprobleme

```
a end < if a @ ...
negate 5 u<
negate 0 5 within
create a 10 floats allot ... a ( a-addr ) 5 floats + f@
dup @ swap f@
if @ . else f@ f. then
... cfield c ... c c@
: cfield ... ;
```

Zusammenfassung

- Ziel: Einige Fehler beim Compilieren finden
  wenig Fehlalarme

- Oft versucht, für existierenden Code nicht erreicht

- Stacktiefe: gelöst

- Typechecking
  Kovarianz und Kontravarianz
  Typhierarchie überarbeiten
  Typregeln überarbeiten
  an realen Programmen testen

- Definitionswörter und Makros brauchen Deklarationen