Maschinenunabhängige Codeverbesserung

Übersicht

- Programmtransformationen
- Effizienzsteigernde Programmtransformationen
- Standardisierende Programmtransformationen
- Mustererkennung

Lernziel

- Die Grundideen der maschinen-unabhängigen Codeverbesserung wiedergeben können

Programmtransformationen

Darstellung von Programmen

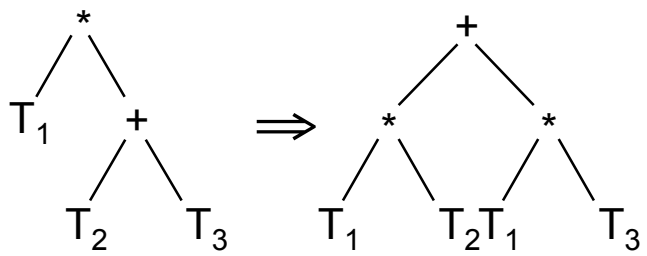
- Bäume, dekoriert mit semantischer Information

Programmtransformationen

- Übersetzung oder Transformation von Zwischendarstellungen von Programmen, beschrieben durch Menge von *Baumtransformationsregeln* ☛
- Ziel
  - Effizienzsteigerung
  - Standardisierung

Baumtransformationsregel

- p ⇒ e mit
  - p, e: Muster (= Terme mit Variablen an den Blättern)
  - p: Eingabemuster (Beschreibt syntaktische Anwendbarkeitsbedingung)
  - e: Ausgabemuster (Gibt an, wie der getroffene Teilbaum umgebaut werden muss)
- Beispiel

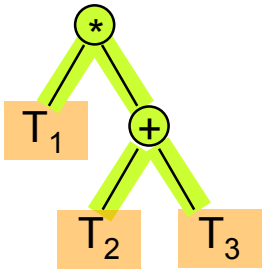


Anwendung von Transformationsregeln

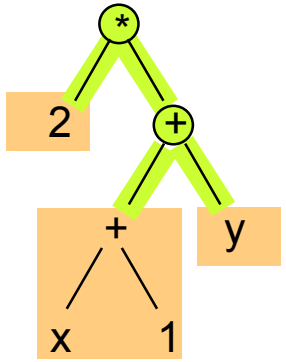
Muster passt auf Teilbaum (synonym: Muster trifft Teilbaum)

- Nichtvariable Teile des Musters müssen mit dem Teilbaum bezüglich Markierung der Knoten, Kinderzahl und Ordnung der Teilbäume übereinstimmen

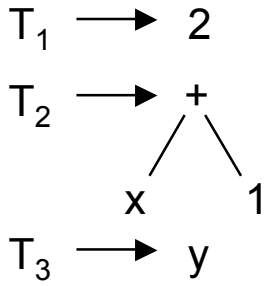
Beispiel



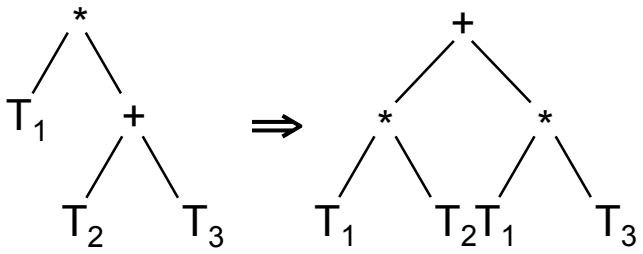
Muster



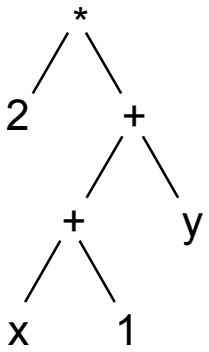
Baum



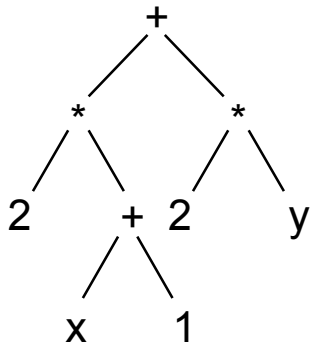
Hergestellte Bindung



Regel



Baum

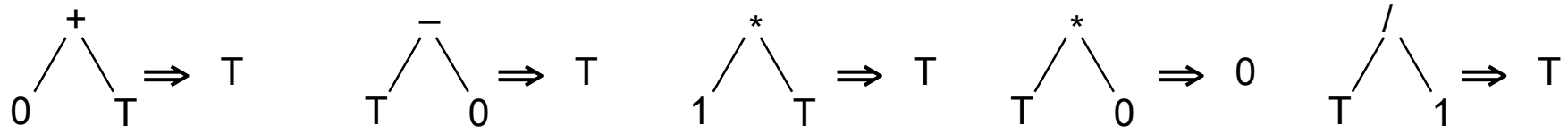


Resultat der Regelanwendung

Effizienzsteigernde Transformationen (1/2)

Algebraische Umformungen

- Vereinfachung arithmetischer Ausdrücke
- **Beispiele**



Kontextabhängige Transformationen

- Transformationen, die vom Kontext / globalen Programmeigenschaften abhängen
- In diesen Fällen
  - Einschränkung der Anwendbarkeit der Transformation durch Prädikate
  - Berechnung der Kontextinformation durch *abstrakte Interpretation*
  - Nach erfolgter Transformation: I.a. neue Analyse des transformierten Programms
- **Beispiel**

  assign(X, E) where notlive(X) ⇒ skip

Effizienzsteigernde Transformationen (2/2)

Vorziehen schleifeninvarianter Berechnungen

- Ein Ausdruck E (ohne Seiteneffekte), der sich beim Schleifendurchlauf nicht verändert, kann außerhalb der Schleife vor deren Ausführung vorab ausgewertet werden
- **Beispiel**
  - S_1.[while(B, S_3.[assign(X, E)].S_4)].S_2 **where** is_invar(E)
    $\Rightarrow$ S_1.[assign(t, E).while(B, S_3.[assign(X, t)].S_4)].S_2
  - Dabei
    S_1.[p].S_2: Vorkommen des Musters p in einer Liste (mit S_1 und S_2 als linkem/rechtem Kontext)

Konstantenfalten

- Ausrechnen (zur Übersetzungszeit) von Werten von Ausdrücken, deren sämtliche Variablen statisch bekannte Werte haben (berechnet durch abstrakte Interpretation)
- **Beispiele**
  - var(X) **where** isconst(X) $\Rightarrow$ value(X)
  - +(C_1, C_2) $\Rightarrow$ $C_1 + C_2$
  - if(true, S_1, S_2) $\Rightarrow$ S_1
  - while(false, S) $\Rightarrow$ skip

Zur Übersetzungszeit ausgewertete Summe

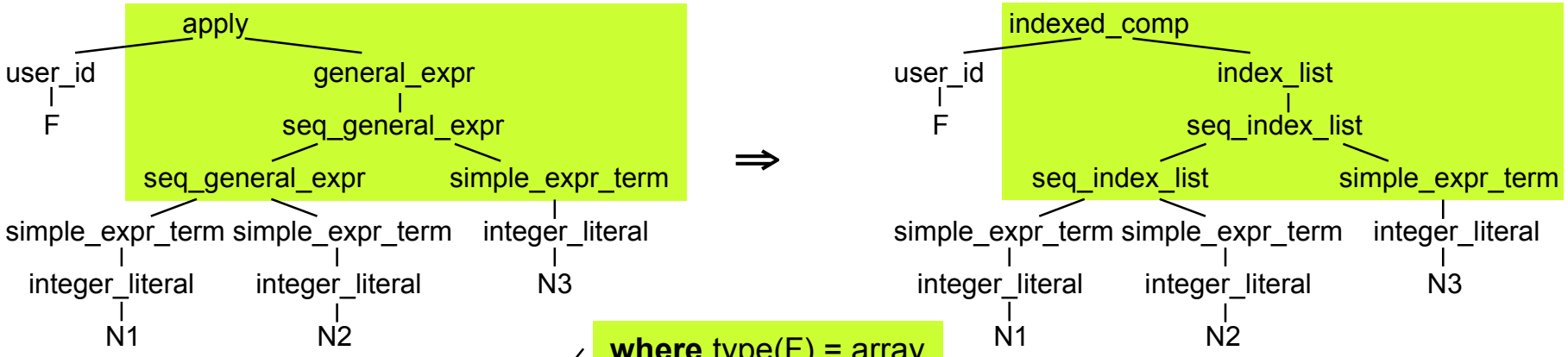
Standardisierende Transformationen

Ziel

- Einige Übersetzeraufgaben werden erleichtert, wenn ein Programm in einer **Standardform** vorliegt, d.h.
  - Eine ausgewählte Form für verschiedene Darstellungen desselben Konstrukts
  - Eindeutige Darstellung mehrdeutiger (syntaktischer) Terme durch Ausnutzung von Kontextinformation

Beispiel

- Term $F(E_1, ..., E_n)$ kann in Ada u.a. sein:
  - Prozedur- oder Funktionsaufruf
  - Indizierte Variable



Durch Kontextinformation garantiert

where type(F) = array

Mustererkennung

Mustererkennung auf Bäumen

- Feststellung, an welchen Knoten (in der Baumdarstellung eines Programms) welche Regeln anwendbar sind

Beschreibung der Mustererkennung

- Durch ***endliche Baumautomaten***
  (die aus einer gegebenen Menge aller Muster berechnet werden)
- Einzelheiten dazu: siehe Wilhelm, Maurer

***Grundidee**:*
Knoten im Baum besuchen und dabei jeweils Zustands-übergänge ausführen (analog zu endlichen Automaten)