Retargierbare Zwischenformattransformation mittels „Tree Matching and Dynamic Programming“ unter Berücksichtigung von Scheduling-Erfordernissen

Annett Königsmann

Vortrag zur Diplomarbeit

Betreuer: Dipl. -Inf. Jens Braunes

Institut für Technische Informatik

Fakultät Informatik

Technische Universität Dresden

Gliederung

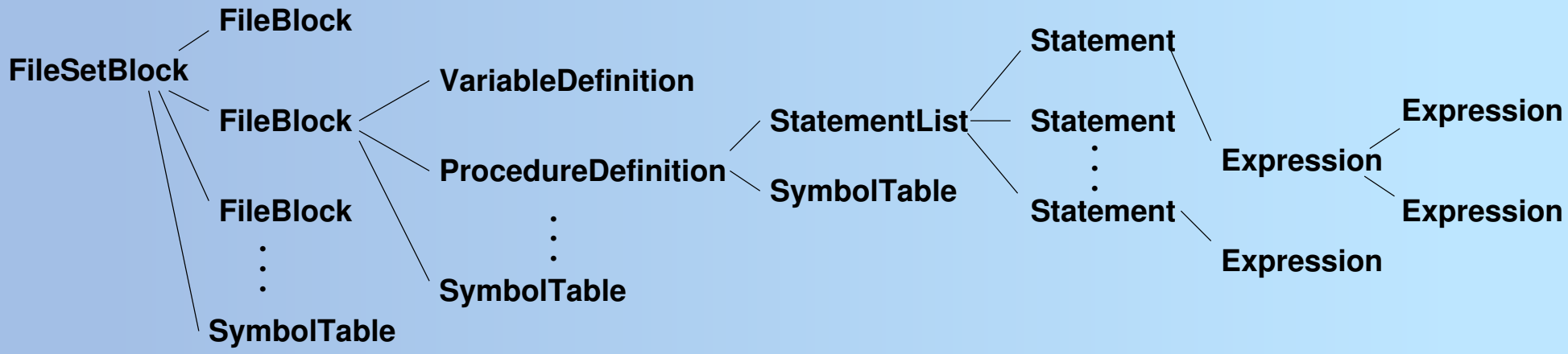
- Motivation
- Tree Matching - Vorbereitung und Ablauf
- Dynamic Programming - Generierung eines graphenorientierten Zwischenformats
- Beispiel zur Veranschaulichung
- Graphenbibliotheken - Bewertung und Auswahl
- Stand der Arbeit und weiterführende Schritte

Motivation

- Retargierbarkeit: Codegenerierung für verschiedene Zielarchitekturen
- Architekturen sind durch formale Beschreibung darstellbar
- Verändern, Anpassen von Compilern an neue Architekturen
- Ausgangspunkt: SUIF-Zwischenformat und Menge von möglichen Befehlen
- Ziel: Graphendarstellung des zu untersuchenden Programms, die alle Alternativen von Befehlsfolgen beinhaltet
- Spätere Verwendung des erzeugten Graphen bei Befehlscheduling-Verfahren

Ausgangspunkt - SUIF2

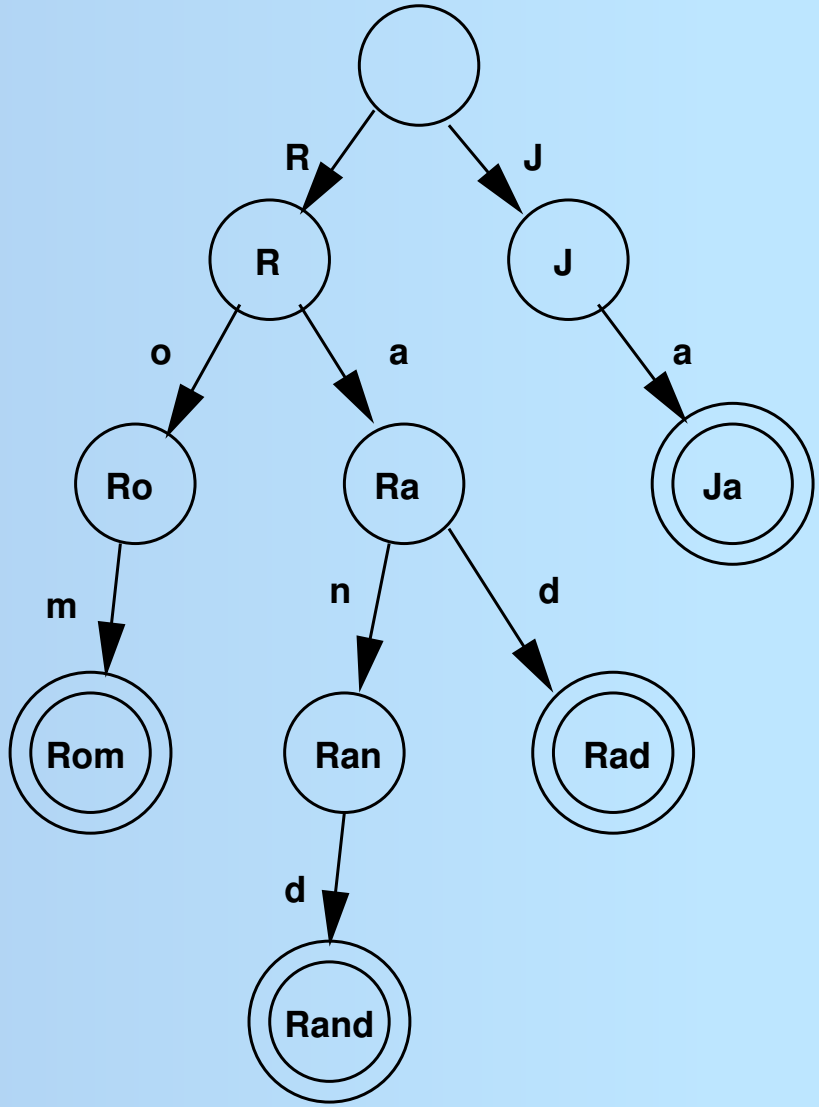
- Programmdarstellung in SUIF als „Ownership Tree“
- SUIF-Knoten sind in Baumform angeordnet
- Informationen zu Variablen, Label und Typen in Symboltabellen abgelegt, über Referenzierung von SUIF-Knoten aus erreichbar
- Berechnungen sind hierarchisch untergegliedert in *Procedure*-, *Statement*- und *Expression*-Knoten



Ausgangspunkt - Trie

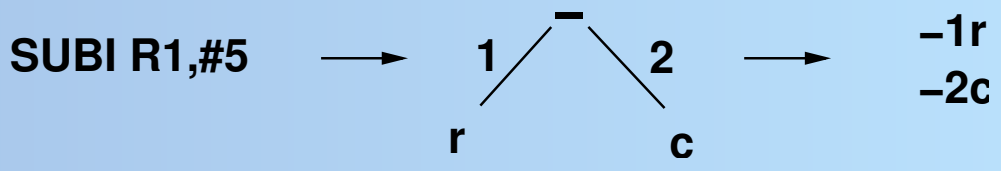
Definition: Trie

- *Retrieval* - Wiedergewinnung
- Datenstruktur oder Suchbaum zur gleichzeitigen Speicherung mehrerer Zeichenketten
- Zeichenkette wird durch einen Pfad durch den Trie dargestellt
- Einsatz bei Suche nach Schlüsselwörtern in einem Text



Erweiterung zum Tree Matching

- Ein Maschinenbefehl oder eine Sequenz von Befehlen kann durch einen Baum repräsentiert werden
- Der Wurzelknoten entspricht der letzten Operation, die Kindknoten den Quelloperanden
- Zweige zu den Kindknoten werden durchnummeriert
- Ein Baum ist dann bestimmbar durch die Menge aller enthaltener Pfade
- Bsp.:



Vorbereitung 1

- Durchlaufen aller Prozedurknoten einer SUIF-Datei
- Vereinfachungen durch einige Transformationen
- Ersetzen ausgewählter High-Level-Konstrukte durch Low-Level-Konstrukte
- Zerlegung des *Expression Trees* von jedem enthaltenen *Statement* in Einzelpfade
- Operanden werden durch Nummerierung gekennzeichnet
- Speicherung der Einzelpfade am jeweiligen Wurzel-*Statement*

Vorbereitung 2

Bsp.:

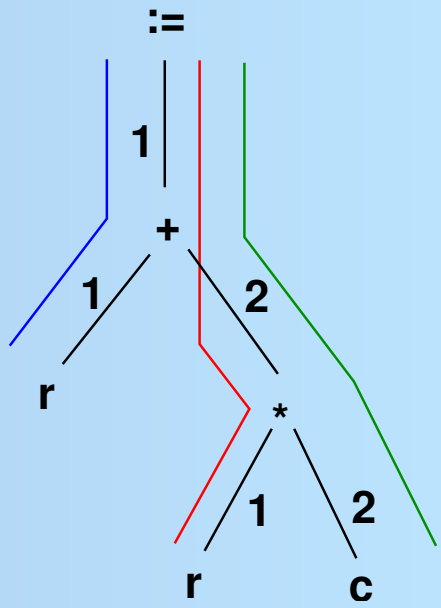
alle Pfade des *Expression Trees* für:

```
c := a + b * 2;
```

:= 1 + 1 r

:= 1 + 2 * 1 r

:= 1 + 2 * 2 c



Ablauf

- Darstellung aller möglichen Maschinenbefehle/Befehlssequenzen durch Bäume bzw. ihre zugehörigen Pfade
- Speicherung der einzelnen Bäume bzw. Pfade in einer Mustersammlung
- Betrachtung jedes *Expression Trees* der SUIF-Datei
- Vergleich von Teilbäumen des *Expression Trees* mit den Bäumen der Mustersammlung
- Abgleich erfolgt über die Untersuchung der enthaltenen Pfade
- Bei Übereinstimmung Markierung des Wurzelknotens im gerade betrachteten Teilbaum mit gefundenenem Muster

Graphenerstellung 1

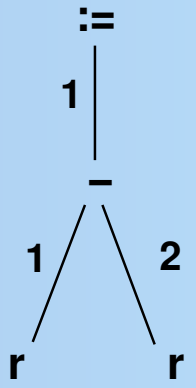
- Zu einem Wurzelknoten können mehrere Übereinstimmungen (*Cover*) existieren
- Jedes *Cover* ist mit bestimmten Kosten verbunden
- *Dynamic Programming* wählt den Code mit den optimalen Kosten aus
- Anstatt von Code soll jedoch graphenorientierte Zwischendarstellung generiert werden unter Beibehaltung aller gefundenen *Cover*

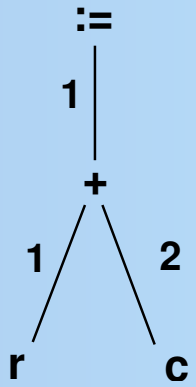
Graphenerstellung 2 - Ansatz

- Ausgehend von jedem Statement einer Prozedur werden alle Übereinstimmungen im *Expression Tree* bestimmt
- Pro *Cover* anlegen eines Knoten im Graphen, Benennung mit eindeutigem Label
- Falls absteigend im Baum mehr als ein *Cover* gefunden wird, Kante von Quellknoten zum Zielknoten als Alternativkante markieren
- Nach Generierung von den noch separaten Alternativbäumen, Untersuchung der SUIF-Datei auf Datenabhängigkeiten
- Zusammensetzen/Verbinden der Alternativbäume anhand der gewonnenen Informationen
- Ziel: Nutzung des erstellten Graphen bei späterem Scheduling

Graphenerstellung 3 - Beispiel

```
a := x - y;

   := 1 - 1 r
   := 1 - 2 r

b := a + 4;

   := 1 + 1 r
   := 1 + 2 c
```



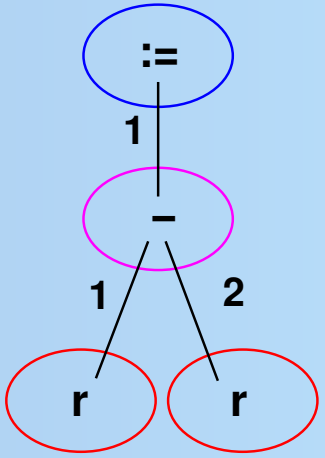


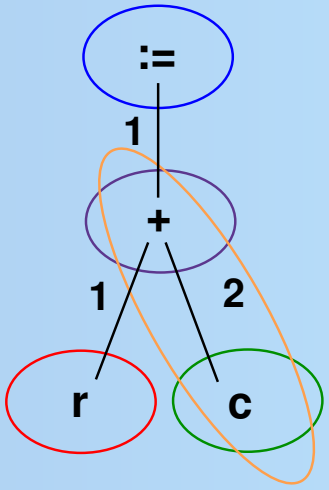
Mustersammlung:

add sub addi mov lod ldc

Graphenerstellung 3 - Beispiel

```
a := x - y;

   := 1 - 1 r
   := 1 - 2 r

b := a + 4;

   := 1 + 1 r
   := 1 + 2 c
```



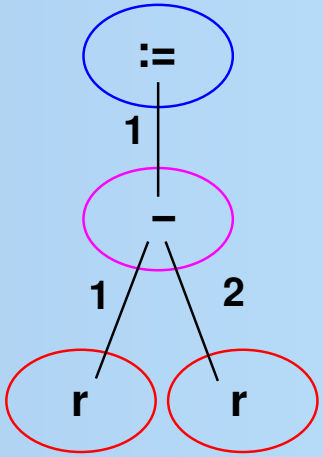


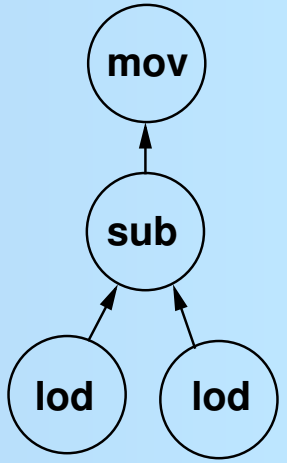
Mustersammlung:

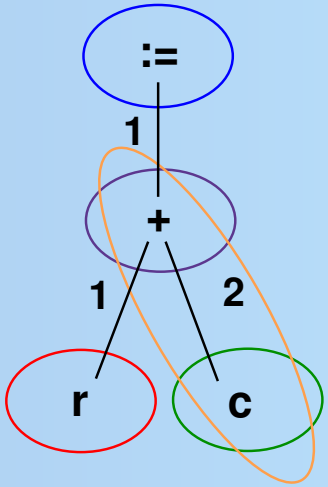
add sub addi mov lod ldc

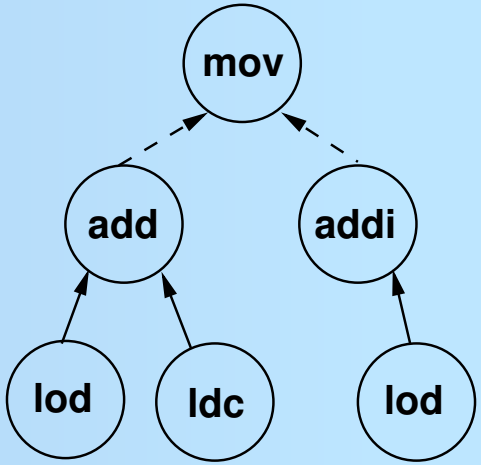
Graphenerstellung 3 - Beispiel

```
a := x - y;

   := 1 - 1 r
   := 1 - 2 r

b := a + 4;

   := 1 + 1 r
   := 1 + 2 c
```









Mustersammlung:

add sub addi mov lod ldc

Graphenbibliotheken

Bibliothek	besondere Voraussetzung	Information zu Knoten und Kanten	Funktions-umfang	Ausgabe-format
LEDA	keine	node-/edge_arrays, node-/edge_maps, parametrisierte Graphen	Zugriff Update Iteration	gw
GTL	keine	node-/edge_maps	ähnlich LEDA geringerer Umfang	gml
CFG-Library	Transformation in MachSUIF	Annotations	Zugriff/Update Walker	vcg
BOOST	keine	Property Maps	ähnlich LEDA	graphviz dot

Stand und weitere Schritte

- Derzeit Implementierung zum Anlegen der Alternativbäume für jedes einzelne *Statement*
- Danach Verbindung und Aktualisierung der Alternativbäume
- Erweiterung von SUIF um neue *Statements*, die DSP-typische *Expression Trees* aufnehmen (z.B. finite Schleifen)
- Erweiterung des Tree Pattern Matcher zum Erkennen solcher Strukturen
- Umsetzung dieser Befehlsmuster in Graphendarstellung