

Workshop „Transformation of Legacy Software“

Source-to-Source-Transformationen

Björn Krellner

<bjk@informatik.tu-chemnitz.de>

TransBS



TECHNISCHE UNIVERSITÄT CHEMNITZ

15.02.2007

Inhalt

- 1 Reengineering-Ansätze
- 2 Zwischendarstellungen
- 3 Transformationen
- 4 TXL

Legacy Software

● Softwareinformationen

- Wissen: Entwickler, Dokumentation, ...
- Quellcode: Code Reuse / Wrapping / Source-to-Source-Transformationen, ...
- Quellcodeanalyse: Codemetriken/-statistiken, Clustering-Verfahren, ...
- GUI-Elemente
- Datenbank: Tabellen, Queries, ...
- Metadaten: Lastdaten, Zugriffsstatistiken, ...

● Zielsystem-Eigenschaften

- Unterstützte Plattform(en), verwendete Middleware(s)
- Konfiguration: Verteiltheit, Redundanz, Sicherheit
- Dimensionierung der Komponenten
- Welche Komponenten übernehmen welche Aufgabe?

Legacy Software

● Softwareinformationen

- Wissen: Entwickler, Dokumentation, ...
- Quellcode: Code Reuse / Wrapping / Source-to-Source-Transformationen, ...
- Quellcodeanalyse: Codemetriken/-statistiken, Clustering-Verfahren, ...
- GUI-Elemente
- Datenbank: Tabellen, Queries, ...
- Metadaten: Lastdaten, Zugriffsstatistiken, ...

● Zielsystem-Eigenschaften

- Unterstützte Plattform(en), verwendete Middleware(s)
- Konfiguration: Verteiltheit, Redundanz, Sicherheit
- Dimensionierung der Komponenten
- Welche Komponenten übernehmen welche Aufgabe?

Legacy Software

● Softwareinformationen

- Wissen: Entwickler, Dokumentation, ...
- Quellcode: Code Reuse / Wrapping / **Source-to-Source-Transformationen**, ...
- Quellcodeanalyse: Codemetriken/-statistiken, Clustering-Verfahren, ...
- GUI-Elemente
- Datenbank: Tabellen, Queries, ...
- Metadaten: Lastdaten, Zugriffsstatistiken, ...

● Zielsystem-Eigenschaften

- Unterstützte Plattform(en), verwendete Middleware(s)
- Konfiguration: Verteiltheit, Redundanz, Sicherheit
- Dimensionierung der Komponenten
- Welche Komponenten übernehmen welche Aufgabe?

Wrapping

- neue Schnittstelle, System bleibt unangetastet
- Übergangslösung, wenn Austausch geplant ist
- Bei fortlaufendem Subsystemtausch notwendig
- Überwiegend Anwendung, um konsolenbasierten Anwendungen mit einer GUI zu versehen

Zeitlicher Aufwand

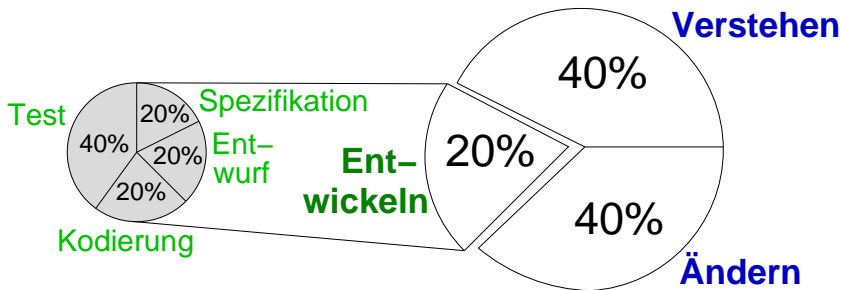


Abbildung: Aufwand im Softwarelebenszyklus [Nosek und Palvia, 1990]

Abstrakte Syntaxbäume I

- Syntaktische Dekomposition des Eingabeprogramms
- Syntaktische Struktur durch formale Beschreibung (BNF) festgelegt

Eigenschaften

- Vereinfachter Ableitungsbaum
- Syntaktische Kanten bilden Baum

- Nach semantischer Analyse auch Namensbindung und Typinformation verfügbar
- Semantische Kanten repräsentieren Verweise auf andere Knoten

Beispiel (Nutzen der graphischen Darstellung)

- Clon-Erkennung

Abstrakte Syntaxbäume II

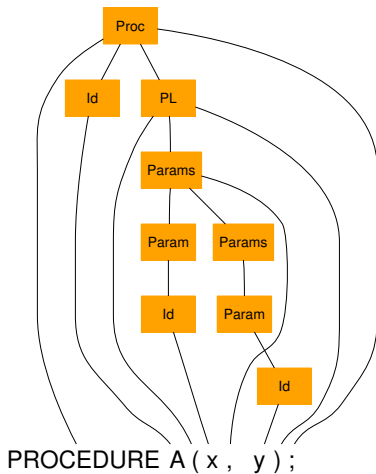


Abbildung: Ableitungsbaum

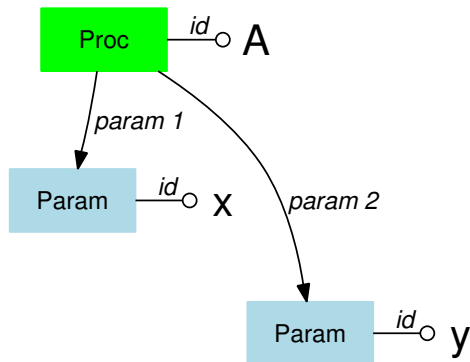


Abbildung: AST

Konflikt

möglichst detailliert

- alle möglichen Informationen extrahieren
- kompletter Code soll wieder generierbar sein



möglichst übersichtlich und vereinheitlicht

- wenige und allgemeine Konstrukte

Eigenschaften von Transformationsregeln

- (im Idealfall) automatisch und beliebig wiederholbar
- zugleich Dokumentation
- Spezifikation der Änderungen
- Repäsentation von Implementierungswissen
- anpassbar
- Vor- und Nachbedingungen prüfbar

Definition

$f : \text{Programm} \mapsto \text{Programm}$

auch: $f : \text{Programm} \mapsto \text{Spezifikation}, \dots$

Anwendung: Slicing

- Slicing bezüglich eines Programmpunktes P_1 entfernt Teile des Programms, die das Verhalten an P_1 nicht beeinflussen
- Dient u. a. der Reduzierung des Codes zur Verbesserung des Verständnisses
- Macht alle von einer Änderung betroffenen Stellen sichtbar, somit auch nützlich zur Restrukturierung

Allgemeines

- 1985 in Toronto als „Turing eXtender Language“ entwickelt
- generisches Transformationssystem
- Transformationsregeln: funktionale Programmiersprache, Patterns
- AST wird traversiert und Transformationen angewandt solange möglich

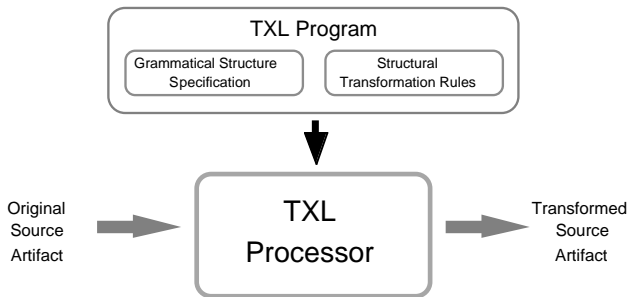


Abbildung: TXL-Prozessor [Cordy, 2001]

Beispiel: C-Grammatik

```
define program
  [repeat decl]
end define
```

```
define decl
  [var_decl]
  | [proc_decl]
end define
```

```
define proc_decl
  [opt type] [id] [header] [block]
end define
```

```
define block
  '{
    [repeat var_decl]
    [repeat statement]
  }
end define
```

% Kommentar

```
define statement
  %...
  | [if_statement]
  %...
  | [block]
end define

define if_statement
  'if [expression]
  [statement]
  'else [statement]
end define
```

Beispiel: Grammatik + Regeln

```
comments
```

```
    /*    */
```

```
end comments
```

```
define program
```

```
    [expression]
```

```
end define
```

```
define expression
```

```
    [multexpr]
```

```
    | [expression] '+' [multexpr]
```

```
end define
```

```
define multexpr
```

```
    [primary]
```

```
    | [multexpr] * [primary]
```

```
end define
```

```
define primary
```

```
    [number]
```

```
    | ( [expression] )
```

```
end define
```

```
include "PlusMult.Grammar"
```

```
rule main
```

```
    replace [expression]
```

```
        E [expression]
```

```
    construct NewE [expression]
```

```
        E [resolve_addition_expr]
```

```
    where not
```

```
        NewE [= E]
```

```
    by
```

```
        NewE
```

```
end rule
```

```
function resolve_addition_expr
```

```
    replace [expression]
```

```
        N1 [number] '+' N2 [number]
```

```
    by
```

```
        N1 [+ N2]
```

```
end function
```

Beispielschritte zur kompletten Transformation

- 1 Grammatiken für die Eingabe- und die Ausgabesprache
- 2 Nichtterminale so umbenennen, dass sie in einer kombinierten Grammatik eindeutig sind (z. B. *C_expression*)
- 3 Minimale Menge an Nichtterminalen suchen, die für die Beschreibung der Transformation ausreichen
- 4 Grammatik erstellen, die alle Zwischenzustände akzeptiert

```

define expression      redefine C_expression  define program
  [C_expression]      ...                  [C_program]
| [P_expression]     | [expression]       | [P_program]
end define            end define           end define

```

- 5 Eine `function`-Definition, die als `replace`-Konstrukt alle Regeln ausführt (z. B. `rule trans_for_statements`)

Vielen Dank für Ihr Interesse!

Verbundprojekt  **TransBS**

Prof. Dr. Gudula Rünger (*Koordinator*)
Professur Praktische Informatik
Technische Universität Chemnitz

09107 Chemnitz

Prof. Dr. Thomas Rauber
Lehrstuhl für Angewandte Informatik II
Universität Bayreuth

95440 Bayreuth

Dipl.-Math. Jürgen Berndt
Berndt und Brungs Software GmbH

40789 Monheim a. R.

TXL: Komplexere Regeln

```
rule resolveConstants
  replace [repeat statement]
    const C [id] '= V [expression];
    RestOfScope [repeat statement]
  by
    RestOfScope [replaceByValue C V]
end rule
```

```
rule replaceByValue ConstName [id] Value [expression]
  replace [primary]
    ConstName
  by
    ( Value )
end rule
```

TXL: In Aktion

```
$ ls
PlusMult.Grammar  R.Test  Test.Txl

$ cat R.Test
5 /* fuerf */ + 5 + (7 * (2 + 5))

$ txl R.Test
TXL v10.4c (15.3.06) (c)1988-2006 Queen's University at Kingston
Compiling Test.Txl ...
Parsing R.Test ...
Transforming ...
10 + (7 * (7))
```