Übersicht

- Organisation der Vorlesung
- Struktur eines Übersetzers
- Quellsprache *TassKaf*
- Scanner-Generator *Jlex*
- Parsergenerator *Cup*
- Zwischendarstellung
- Zielsprache *Jasmin*-Assembler
- Schnittstelle zum Frontend

Organisation

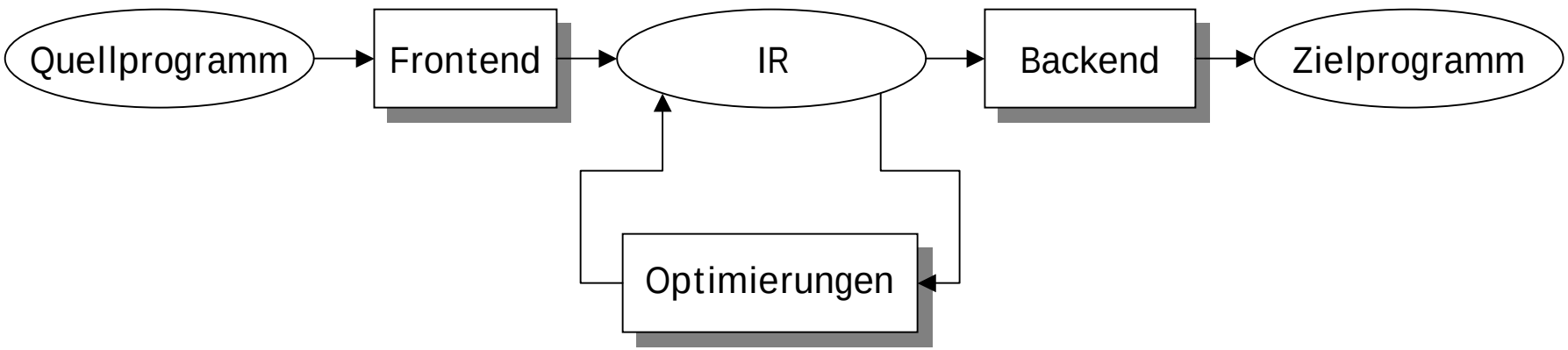
- Voraussetzungen zur Scheinvergabe
  - 50% der Übungspunkte
  - Erfolgreicher Projektabschluß
- Notenvergabe
  - 30% Übungspunkte
  - 70% Projekt: Beispielprogramme + „Kür“

Organisation (2)

- Implementierung in Abschnitten:
  - Codeerzeugung bis 18. Mai
  - Syntaktische Analyse bis 8. Juni
  - Semantische Analyse bis 29. Juni
- Arbeit in Zweiergruppen
- Mailingliste: `compilerbau@cs.uni-sb.de`
- Webseite:

  `http://www.cs.uni-sb.de/~mlangen/TassKaf`

Struktur eines Übersetzers



TassKaf

- Teilsprache von Java
- Nicht enthalten:
  - Explizite Typumwandlung (*type casts*)
  - Selbstdefinierte Konstruktoren
  - Schnittstellen (*interfaces*)
  - Leichtgewichtige Prozesse *(threads)*
  - Ausnahmen *(exceptions)*
  - Importmechanismus
  - …

TassKaf (2)

```
class Example extends java.lang.Object
{
  int i=3;

  Example ex1=null;
  Example ex2=new Example();

  int a[][]=new int[20][];

  static int si=3;
  static int sa[]=null;
```

Vererbung

Objektfeld und Initialisierung

Objektreferenz

Objekterzeugung

Feldreferenz

Klassenfelder

TassKaf (3)

```
void m1(int i, Example e, int a[])
{
  int j=3;

  e.i=e.i-1;

  Example.sa=new int[815];

  e.m1(i-1, this, a);
}
```

Methoden

Lokale Variablen

Zugriff auf Objektfelder

Zugriff auf Klassenfelder

Aufruf von Methoden

TassKaf (4)

Überladung

```
int m1(int i)
{
   return i-1;
}
```

Klassenmethoden

```
static Example m2(Example e)
{
   return e;
}
}
```

JLex

- Scanner-Generator
- Erzeugt Java-Klassendefinition `Yylex`
- Wandelt Zeichenstrom nach Tokenstrom
- Regeln, die regulären Ausdrücken Aktionen zuweisen:

```
"class"             { return new Symbol(sym.CLASS); }
[A-Z][a-zA-Z0-)_]* { return new Symbol(sym.UIDENT, yytext()); }
```

Beispiel

- Ausrücke in polnischer Notation
- Operator steht vor Operanden

$$1 + 2 * 3 \quad \cong \quad +1 * 2\ 3$$

- Grammatik:

$$\begin{aligned} \textit{expr} &\rightarrow + \textit{expr}\ \textit{expr} \\ &\mid * \textit{expr}\ \textit{expr} \\ &\mid \textit{const} \end{aligned}$$

Beispiel (2)

```
import sym;
import java_cup.runtime.*;
%%
%function nextToken
%type Symbol
%eofval {
  { return new Symbol(sym.EOF);}
%eofval}
```

Beispiel (3)

```
DIGIT=[0-9]
WHITESPACE=[\n\t\f\r ]
%%
"+"          { return new Symbol(sym.PLUS); }
"*"          { return new Symbol(sym.MUL); }
{DIGIT}+     { return new Symbol(sym.CONST,
                         new Integer(yytext()); }
{WHITESPACE}+     {}

. { System.err.println("Unexpected character: "+
                         yytext()); }
```

Cup

- *Constructor of useful parsers*
- Parser-Generator
- Erzeugt LALR(1)-Parser als Java-Klassendefinition `parser`
- Verwendet Tokenstrom und Regeln zum Aufbau eines Syntaxbaums:

```
expr ::= expr:e1 PLUS expr:e2 {: Anweisungen :}
```

Beispiel (4)

```
import java_cup.runtime.*;
import Yylex;
parser code {: static Yylex scanner;
               public parser(Yylex yyl) {
                    scanner=yyl;
               }
            :};
scan with {: Symbol s=scanner.nextToken();
             return s; :}
```

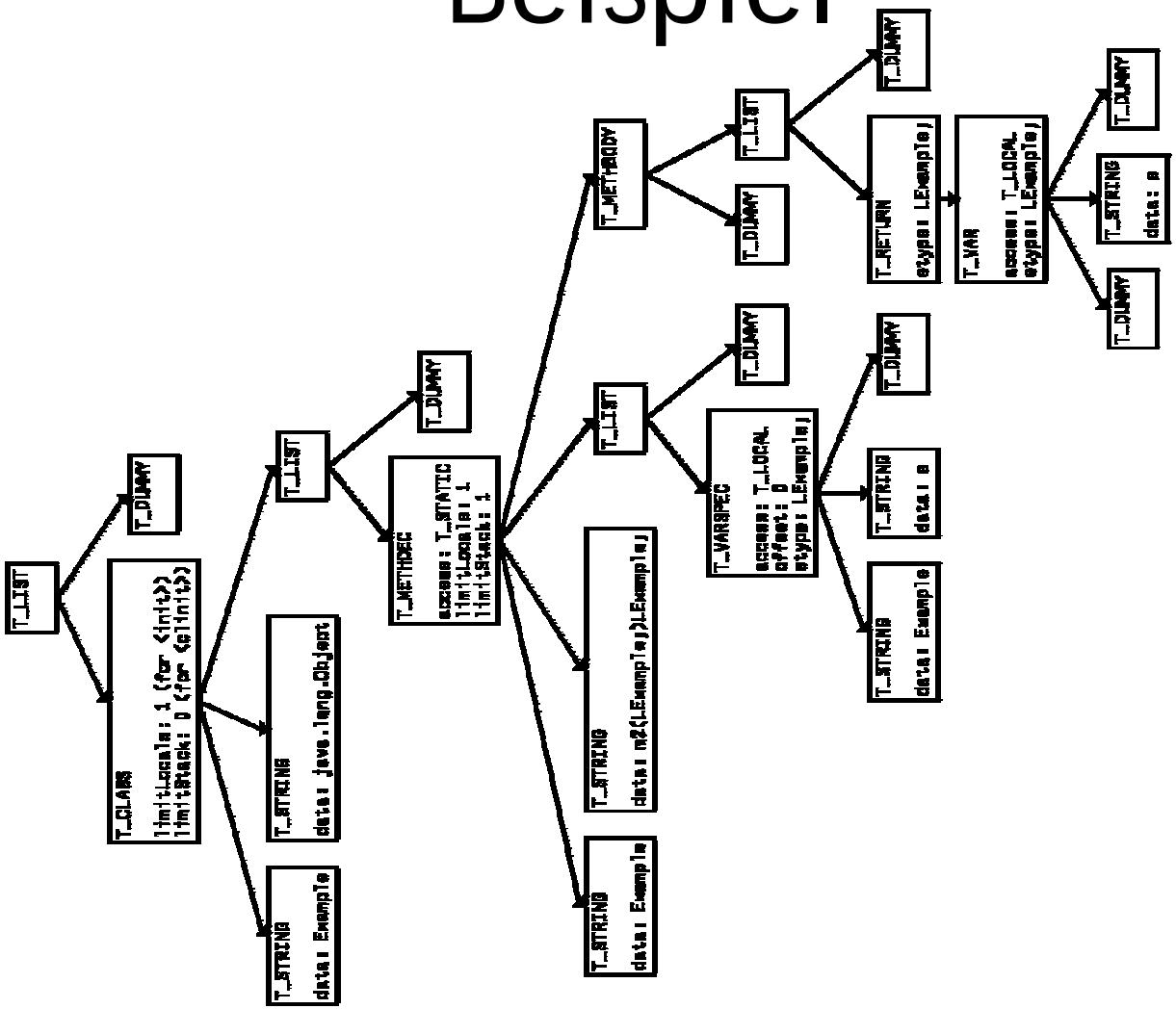
Beispiel (5)

```
terminal Symbol PLUS, MUL;
terminal Integer CONST;
non terminal Integer expr;

start with expr

expr ::= PLUS expr:e1 expr:e2
         {: RESULT=new Integer(e1.intValue()+e2.intValue()); :}
       | MUL expr:e1 expr:e2
         {: RESULT=new Integer(e1.intValue()*e2.intValue()); :}
       | CONST
       ;
```

Zwischendarstellung

- Annotierter Syntaxbaum
- Knoten besitzen:
  - Typ (Klasse, Variable, Methodenaufruf, ...)
  - Attribute (rel. Adresse, Deklarationsverweis, ...)
  - Kinder
- Knoten sind Objekte der Klasse `lib.Node`

Beispiel


T_LIST
T_DUMMY
T_CLASS
limit:Locals: 1 (for <init>)
limit:Stack: 0 (for <clinit>)
T_LIST
T_DUMMY
T_METHODDEC
access: T_STATIC
limit:Locals: 1
limit:Stack: 1
T_STRING
data: java.lang.Object
T_STRING
data: Example
T_METHODDY
T_LIST
T_DUMMY
T_LIST
T_DUMMY
T_VARSPEC
access: T_LOCAL
offset: 0
etype: LExample;
T_DUMMY
T_STRING
data: a
T_STRING
data: Example
T_STRING
data: n2(LExample;)LExample;
T_STRING
data: Example
T_DUMMY
T_RETURN
etype: LExample;
T_VAR
access: T_LOCAL
etype: LExample;
T_DUMMY
T_STRING
data: a
T_DUMMY

Jasmin

- Assembler
- Erzeugt Java-Klassendateien
- Zielsprache für Backend
- Vorteile
  - Abstraktion von Klassendateien
  - Textdateien erleichtern Fehlersuche
- Eine Klasse pro Assemblerdatei

Jasmin: Felder

TassKaf

```
class Example
   extends java.lang.Object
{
   int i=3;

   Example ex1=null;
   Example ex2=new Example();

   int a[][]=new int[20][];

   static int si=3;
   static int sa[]=null;
```

Jasmin

```
.class public Example
.super java/lang/Object

.field public i I

.field public ex1 LExample;
.field public ex1 LExample;

.field public a [[I

.field public static si I
.field public static si [I
```

Jasmin: Konstruktoren

Objektkonstruktor

```
.method public <init>()V
   aload_0
   invokespecial java/lang/Object/<init>(V)
   ...
   return
.end method
```

Klassenkonstruktor

```
.method public <clinit>()V
   ...
.end method
```

Jasmin: Methoden

TassKaf

```
void m1(int i, Example e,
        int a[])
{
   int j=3;

   e.i=e.i-1;
   Example.a=new int[47][11];

   e.m1(i-1, this, a);
}
```

Jasmin

```
.method public m1(ILExample;[I)V
.limit locals 5

   ldc 3
   istore 4
   .
   .
   .
   aload 3
   invokevirtual Example/m1...
   return
.end method
```

Erster Projektabschnitt

- Codeerzeugung für gegebene Zwischendarstellung
- Anwendung der Codeschemata zur Erzeugung von *Jasmin*-Assemblerdateien
- Zugriff auf Frontend über Klassen:

```
import java_cup.runtime.*;
import frontend.Yylex;
import frontend.parser;
import frontend.Analyzer;
import lib.Node;
import lib.GDLVisualizer;
```

Beispieltreiber

```
public class Main {
   public static void main(String argv[]) {
        ...
        Yylex lex=new Yylex(System.in);
        parser parse_obj=new parser(lex);
        Node root=(Node)parse_obj.parse().value;
        Analyzer a=new Analyzer(root);
        a.analyze();
        ...
        // emit jasmin assembler code
        ...
   }
}
```