Zieltexte erzeugen

Werkzeug PTG: Pattern-based Text Generator

Erzeugung strukturierter Texte in beliebigen Sprachen. Anwendung als Berechnungen im Strukturbaum, und auch in beliebigen C-Programmen.

Prinzip:

1. Ausgabe-Pattern mit Einfügestellen spezifizieren:

```
ProgramFrame: $
              "void main () {\n"
              $
              "}\n"
Exit:         "exit (" $ int ");\n"
IOInclude:    "#include <stdio.h>"
```

2. PTG generiert Funktionen dazu. Aufrufe liefern Zielstruktur, bottom-up zusammensetzen:

```
PTGNode a, b, c;
a = PTGIOInclude ();
b = PTGExit (5);
c = PTGProgramFrame (a, b);
```

entsprechend mit Attributen im Strukturbaum

3. Ausgabe der Zielstruktur:

```
PTGOut (c);
PTGOutFile ("Output.c", c);
```

Praktikum Sprachimplementierung mit Werkzeugen WS 1999/ 2000 / Folie 701

Ziele:

Prinzip der Zieltexterzeugung mit PTG

im Vorlesungsteil:

Erläuterungen der Beispiele

nachlesen:

Eli-Dokumentation: PTG, Introduction

Verständnisfragen:

Geben Sie Möglichkeiten für PTG-Anwendungen ausserhalb von Eli-erzeugten Übersetzern an.

PTG-Muster: Wichtige Techniken

Indizierte Einfügestellen

Anwendung: **gleicher Text wird mehrfach eingefuegt**:

```
Module: "module " $1 "begin" $2 "end " $1 ";\n"
```

Aufruf `PTGModule (id, body);`

Anwendung: **Muster ändern - Aufruf beibehalten**:

heute `Decl: $1 /*type*/ " " $2 /*names*/ ";"`

morgen `Decl: $2 /*names*/ ": " $1 /*type*/ ";"`

immer Aufruf `PTGDecl (tp, ids);`

Typisierte Einfügestellen, Muster für Blatt-Texte:

```
CarId: $ string "-" $ string "-" $ int
```

Aufruf `PTGCarId ("PB", "AB", 127);`

String aus Csm-String-Speicher einsetzen:

```
CLASS SYMBOL IdOcc COMPUTE
  SYNT.Ptg = PTGId (TERM);
END;
```

Optionale Pattern-Elemente

Anwendung: Listen mit Trenner:

```
CommaSeq: $ { ", " } $
```

Aufruf `PTGCommaSeq (a, b);`

setzt Trenner nur ein, wenn weder a noch b gleich `PTGNULL` ist

Einige nützliche Pattern sind verfügbar in

`$/Output/PtgCommon.fw`

Praktikum Sprachimplementierung mit Werkzeugen WS 1999/ 2000 / Folie 702

Ziele:

Pattern-Formulierung lernen

im Vorlesungsteil:

Erläuterung der Techniken

nachlesen:

PTG-Dokumentation: Indexed Insertion Points, Typed Insertion Points, Marking Patterns as optional Modlib-Dokumentation: PtgCommon

Übungsaufgaben:

Suchen Sie im Abschnitt "Transformation in C-Programme" der Core-Spezifikation das PTG-Muster für den Programmrahmen und seine Anwendung. Erläutern Sie beide.

Verständnisfragen:

Warum ist die Technik des mehrfachen Einfügens in LIDO besonders nützlich?

Zieltext im Strukturbaum zusammensetzen

Benachbarte Kontexte:

```
ATTR Ptg: PTGNode;
RULE: Statement ::= UseIdent ':=' Expression
COMPUTE
  Statement.Ptg =
    PTGAssign (UseIdent.Ptg, Expression.Ptg);
END;
```

Zusammenfassen bei ähnlichen Quell- und Zielstrukturen:

```
SYMBOL Statements COMPUTE
  SYNT.Ptg =
    CONSTITUENTS Statement.Ptg
    WITH (PTGNode, PTGSeq, IDENTICAL, PTGNull);
END;
```

Unregelmäßiges Zusammenfassen in link-rechts Ordnung:

```
CHAIN ConstPtg: PTGNode;
SYMBOL Block COMPUTE
  CHAINSTART HEAD.ConstPtg = PTGNULL;
END;
RULE: Declaration::= DefIdent '=' IntLiteral ';'
COMPUTE
  Declaration.ConstPtg =
    PTGSeq (Declaration.ConstPtg,
            PTGConstDecl
              (DefIdent.Ptg, IntLiteral.Ptg));
END;
```

Praktikum Sprachimplementierung mit Werkzeugen WS 1999/ 2000 / Folie 703

Ziele:

Techniken fuer Berechnungen im Baum

im Vorlesungsteil:

Vergleich der Techniken Zusammenfassung durch CONSTITUENTS und durch CHAIN

nachlesen:

PTG-Dokumentation: A Complete Example

Übungsaufgaben:

Bearbeiten Sie alle I-Aufgaben der Core-Spezifikation im Abschnitt "Transformation in C-Programme".

Verständnisfragen:

- Wie unterscheiden sich PTGNULL und PTGNull?
- Geben Sie ein CONSTITUENTS-Konstrukt zur Zusammenfassung von Bezeichnerlisten in Deklarationen an.
- Welche Schemata würden Sie für die Berechnungen im Baum anwenden, wenn das Quellprogramm Deklarationen von Konstanten und Variablen in beliebiger Reihenfolge enthält, das Zielprogramm aber alle Deklarationen von Konstanten vor denen von Variablen?

Konsistente Umbenennung von Bezeichnern

Zu jedem bezeichneten Quellobjekt soll ein **eindeutiger Zielbezeichner** erzeugt werden, der gut **lesbar ist.**

Die **Bindung** der Zielbezeichner soll **unabhängig von den Regeln der Zielsprache** sein.

PTG-Muster hängt Nummer an den Quellbezeichner an:

```
TransId:  $ string /*src name*/ "_" $ int
```

PDL-Eigenschaft: Übersetzung des Programmobjektes:

```
TransId: PTGNode; "ptg_gen.h"
```

Berechnung zu allen Bezeichnerknoten **im Baum**:

```
CLASS SYMBOL IdOcc COMPUTE
  SYNT.Ptg = SetOrGetTransId (THIS.Key, TERM);
END;
```

C-Modul implementiert die Funktion `SetOrGetTransId`:

```
static int count = 1;
static PTGNode errid = PTGTransId ("_ERR", 0);

PTGNode SetOrGetTransId (DefTableKey k, int sym)
{ PTGNode result = GetTransId (k, PTGNULL);
  if (k == NoKey) return errid;
  if (result == PTGNULL)
  { result =
      PTGTransId (StringTable (sym), count++);
    ResetTransId (k, result);
  }
  return result;
}
```

Praktikum Sprachimplementierung mit Werkzeugen WS 1999/ 2000 / Folie 704

Ziele:

Zusammenwirken mehrerer Werkzeuge

im Vorlesungsteil:

- Erläuterung der Techniken
- Zusammensetzen zur Moduldefinition

Übungsaufgaben:

Vervollständigen Sie die Spezifikationen in einer .fw Datei als wiederverwendbaren Spezifikationsmodul.

Verständnisfragen:

- Welche Bedingung muessen die Quell- und Zielregeln für Bezeichner erfüllen?
- Warum ist die Dateiangabe in der .pdl-Spezifikation nötig?
- Erläutern Sie die Verwendung der Eigenschaft TransId in unserem Entwurfsschema für Setzen und Lesen von Eigenschaften in Berechnungen im Baum. Warum brauchen wir hier KEINE Reihenfolgeabhängigkeiten?