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);
```

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`

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;
```

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;
}
```