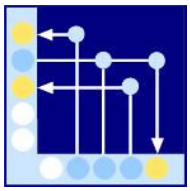
Fakultät Elektronik und Informatik
Studienbereich Informatik



Compilerbau-Praktikum

Lehrveranstaltung im Wintersemester 2022/2023
Prof. Dr. habil. Christian Heinlein

10. Aufgabenblatt (13. Dezember 2022)

Aufgabe 12: Verzweigte Operatoren

Verallgemeinern Sie die Deklaration dynamischer und statischer Operatoren dahingehend, dass ihre Signatur auch „Verzweigungen“ in Form runder, eckiger und geschweifter Klammern mit beliebig vielen Alternativen enthalten kann, zum Beispiel:

```
calc [minus] (x) { (plus|minus) (y) } end -> (
    res := uniq;
    res =! [-x || x];
    { res +! (y || -y) };
    ?res
);

calc minus 1 plus 2 minus 3 end
```

Klammern können bei Bedarf benannt und beliebig verschachtelt werden, zum Beispiel:

```
dnf [n1: not] (x1) { and [n2: not] (x2) }
{ or [n3: not] (x3) { and [n4: not] (x4) } } end -> (
    res := uniq;
    res =! [~x1 || x1];
    { res &! [n2: ~x2 || x2] };
    {
        tmp := uniq;
        tmp =! [n3: ~x3 || x3];
        { tmp &! [n4: ~x4 || x4] };
        res |! ?tmp
    };
    ?res
);

dnf 1<2 and not 2=3 and 3>2
 or not 4<5
 or 5>=6 and 6<=7
end
```

Notwendige Maßnahmen zur Übersetzungszeit

- Damit Klammern in der Signatur eines Operators beliebig verschachtelt werden können, kann der Operatordeklarationsoperator wie folgt definiert werden:

```
// Ein Name (mit oder ohne Anführungszeichen).
Part name = Part(name_, ".....")(reg_, true);

// Ein oder mehrere Namen.
Part names = rep(name)(opt_, false);

// Signatur bzw. Teilsignatur einer Operatordeklaration.
Par p1, p2, ...;
Part opersig = alts(
    name,                       // Name.
    sig("(", names, ")"),       // Parameterdeklaration.
                                // Runde Klammer mit optionaler Benennung
                                // und mindestens zwei Alternativen.
    sig("(", opt(names, ":"), p1 = par("p1"),
                        rep("|", p2 = par("p2"))(opt_, false), ")"),
    sig("[", ......, "]"),      // Eckige und geschweifte Klammern analog
    sig("{", ......, "}")       // mit mindestens einer Alternative.
)(rep_, true);

// Für die Parameter p1, p2, ..., die jeweils einer Alternative einer
// Klammer entsprechen, darf nur eine Anwendung des Hilfsoperators alt
// eingesetzt werden (der nicht global bekannt sein soll).
Oper alt = oper(opersig)
for (Par p : { p1, p2, ... }) {
    impt(p, opers(All, alt));
    excl(p, opers(AllBut, alt), "MT");
}

// Operatordeklarationsoperator.
Oper operdecl = oper(opersig, alts("->", "=>"), "(", par("impl"), ")");
nofix += operdecl;
```

- Die Prolog- oder Epilogfunktion zur Erzeugung des deklarierten Operators erzeugt wie bisher auch die in der Signatur deklarierten Parameter des Operators, fügt zum Exportkatalog des Operatordeklarationsausdrucks aber nur diejenigen hinzu, die auf „oberster Ebene“, d. h. außerhalb aller Klammern deklariert sind.
- Außerdem erzeugt sie für jede Klammer in der Signatur des Operators einen zugehörigen Klammeroperator und speichert ihn in einem zusätzlichen Attribut

```
ATTR1(boper_, Part, Oper)       // bracket operator
```

  des Signaturteils, das die Klammer repräsentiert. (Das bereits vorhandene Attribut `par` sollte hierfür nicht verwendet werden, weil der Parser das Signaturteil sonst für einen Parameter halten könnte.) Klammeroperatoren, die zu Klammern auf oberster Ebene gehören, werden außerdem zum Exportkatalog des Operatordeklarationsausdrucks hinzugefügt.

- Die Signatur eines solchen Klammeroperators enthält:
  - Eine öffnende Klammer (je nach Art der Klammer `"("`, `"["` oder `"{"`).
  - Wenn die Klammer eine Benennung besitzt, ein optionales Signaturteil, das aus den entsprechenden Namen und einem Doppelpunkt besteht. (Das heißt, wenn eine Klammer eine Benennung besitzt, kann diese bei der Verwendung des Klammeroperators optional angegeben werden.)
  - Einen Parameter für die erste Alternative der Klammer.
  - Für jede weitere Alternative zwei Trennstriche (`"||"`) und einen weiterer Parameter.
  - Bei einer eckigen Klammer zusätzlich ein optionales Signaturteil, das aus weiteren zwei Trennstrichen und einem weiteren Parameter besteht. (Ein zugehöriger Operand wird zur Laufzeit genau dann ausgewertet, wenn die eckige Klammer nicht durchlaufen wurde.)
  - Eine schließende Klammer (je nach Art der Klammer `")"`, `"]"` oder `"}"`).

  Beispielsweise entstehen aus den ersten beiden Klammern des Operators `dnf` Klammeroperatoren mit folgenden Signaturen:

  ```
  // Signatur des Klammeroperators zur Klammer [n1: not]
  "[", opt("n1", ":"), par("p1"), opt("||", par("p2")), "]"

  // Signatur des Klammeroperators zur Klammer { and [n2: not] (x2) }
  "{", par("q"), "}"
  ```

- Jeder Parameter eines solchen Klammeroperators außer dem optionalen Parameter bei eckigen Klammern erhält eine Importangabe, die neben dem Pseudo-Operator `All` alle Parameter und Klammeroperatoren enthält, die zu der entsprechenden Alternative der Klammer gehören.

  Beispielsweise enthält die Importangabe des oben mit `p1` bezeichneten Parameters nur den Pseudo-Operator `All`, während die Importangabe des mit `q` bezeichneten Parameters zusätzlich den Klammeroperator für die Klammer `[n2: not]` sowie den Parameter `x2` enthält. Der mit `p2` bezeichnete Parameter erhält keine Importangabe.

Notwendige Maßnahmen zur Laufzeit

- Die Auswertungsfunktion des Operatordeklarationsoperators bleibt unverändert.
- Die Auswertungsfunktion eines benutzerdefinierten Operators erzeugt wie bisher einen neuen Kontext (der im folgenden als Hauptkontext bezeichnet wird) mit dem passenden umschließenden Kontext und wertet alle Operanden des Ausdrucks von links nach rechts aus.

  Im Hauptkontext werden jedoch nur die Werte derjenigen Parameter gespeichert, die auf oberster Ebene der Signatur des Operators definiert sind. Außerdem wird in diesem Kontext für jede Klammer auf oberster Ebene der Signatur der zugehörige Klammeroperator (der im Attribut `boper` gespeichert ist) zusammen mit einem Wert gespeichert, der folgendes enthält:
  - Das zusätzliche Attribut

    ```
    ATTRN(choices_, Value, posA)
    ```

    enthält für jeden Durchlauf durch die Klammer die im Attribut `choice` des Durchlaufs gespeicherte Position der durchlaufenen Alternative.

    Achtung: Eine eckige Klammer kann einen Pseudodurchlauf mit `choice` gleich `0*A` besitzen, der ignoriert werden muss, weil die Klammer in diesem Fall tatsächlich nicht durchlaufen wurde.
  - Das zusätzliche Attribut

    ```
    ATTRN(contexts_, Value, Context)
    ```

enthält für jeden Durchlauf durch die Klammer einen eigenen neuen Kontext, der – analog zum Hauptkontext – für die Parameter und Klammern der durchlaufenen Alternative jeweils den zugehörigen Wert enthält. Jeder dieser Kontexte besitzt als umschließenden Kontext entweder den Hauptkontext – wenn sich die Klammer auf oberster Ebene der Signatur befindet – oder den Kontext, der zum aktuellen Durchlauf der umschließenden Klammer gehört.

Für den Ausdruck `calc minus 1 plus 2 minus 3 end` enthält der Hauptkontext zum Beispiel folgendes:

- Einen Eintrag für den Klammeroperator, der zur Klammer `[minus]` gehört, dessen Wert folgendes enthält:
  - Das Attribut `choices` enthält den Wert `A`, weil die erste (und einzige) Alternative der Klammer durchlaufen wurde.
  - Das Attribut `contexts` enthält einen leeren Kontext, weil es in dieser Alternative weder Parameter noch geschachtelte Klammern gibt.

  (Wenn das Wort `minus` nach `calc` im Ausdruck fehlen würde, würden `choices` und `contexts` jeweils nichts enthalten, weil es dann keinen Durchlauf durch die Klammer gäbe.)
- Einen Eintrag für den Parameter `x` mit Wert 1.
- Einen Eintrag für den Klammeroperator, der zur Klammer `{ (plus|minus) (y) }` gehört, dessen Wert folgendes enthält:
  - Das Attribut `choices` enthält zweimal den Wert `A`, weil in beiden Durchläufen durch die Klammer die erste (und einzige) Alternative durchlaufen wurde.
  - Das Attribut `contexts` enthält für jeden dieser Durchläufe einen Kontext, der jeweils folgendes enthält:
    - Einen Eintrag für den Klammeroperator, der zur geschachtelten Klammer `(plus|minus)` gehört, dessen Wert folgendes enthält:
      - Das Attribut `choices` enthält im Kontext für den ersten bzw. zweiten Durchlauf durch die geschweifte Klammer den Wert `A` bzw. `2*A`, weil hier die erste bzw. zweite Alternative der runden Klammer durchlaufen wurde.
      - Das Attribut `contexts` enthält jeweils einen leeren Kontext, weil es in beiden Alternativen der runden Klammer weder Parameter noch geschachtelte Klammern gibt.
    - Einen Eintrag für den Parameter `y` mit Wert 2 im ersten Durchlauf und Wert 3 im zweiten Durchlauf.

Schließlich wird bei einem dynamischen Operator wie bisher die Implementierung des Operators mit dem Hauptkontext als aktuellem Kontext ausgewertet und der resultierende Wert zurückgegeben, nachdem der vorige aktuelle Kontext wiederhergestellt wurde.

Bei einem statischen Operator wird als Tabellenschlüssel die Sequenz aller Operandenwerte und Wörter des Ausdrucks verwendet. Wörter können hierfür in einem zusätzlichen Attribut

```
ATTR1(word_, Value, str)
```

gespeichert werden und müssen dann sowohl beim Vergleich von `Value`-Objekten als auch bei der Berechnung von Streuwerten geeignet berücksichtigt werden. Der Streuwert einer Zeichenkette `s` des Typs `str` kann hierfür mittels `std::hash<CH::str>()(s)` berechnet werden. Beim Vergleich von Schlüsseln ist zu beachten, dass die beiden Sequenzen unterschiedlich lang sein können und korrespondierende `Value`-Objekte von unterschiedlicher Art sein können.

- Die Auswertungsfunktion eines Klammeroperators sucht den Eintrag ihres Operators mit dem zugehörigen Wert auf die gleiche Weise wie die Auswertungsfunktion eines Konstantenoperators.

  Dann verwendet sie den Inhalt der Attribute `choices` und `contexts` dieses Werts wie folgt, um für jeden Durchlauf durch die Klammer, zu der dieser Klammeroperator gehört, den passenden Operanden des Klammerausdrucks auszuwerten:

◦ Der Wert des Attributs `choices`, der zu diesem Durchlauf gehört, bestimmt den passenden Operanden.

◦ Der Wert des Attributs `contexts`, der zu diesem Durchlauf gehört, bestimmt den Kontext, der während der Auswertung dieses Operanden zum aktuellen Kontext gemacht wird.

Wenn `choices` und `contexts` bei einer eckigen Klammer nichts enthalten, wird – im aktuellen Kontext – der Operand ausgewertet, der zu dem optionalen Parameter am Ende der Signatur des Klammeroperators gehört, sofern dieser Operand vorhanden ist. Andernfalls wird nichts ausgewertet. (Wenn `choices` und `contexts` bei einer geschweiften Klammer nichts enthalten, wird grundsätzlich nichts ausgewertet.)

Als Resultatwert wird abhängig von der Art der Klammer folgendes geliefert:

◦ Bei einer runden Klammer der Wert des ausgewerteten Operanden.

◦ Bei einer eckigen Klammer ebenso, sofern ein Operand ausgewertet wurde, andernfalls nil.

◦ Bei einer geschweiften Klammer die Anzahl der Durchläufe durch diese Klammer.

Wenn für den Ausdruck `calc minus 1 plus 2 minus 3 end` die Implementierung seines Operators ausgewertet wird, werden dabei zum Beispiel folgende Anwendungen von Klammeroperatoren wie folgt ausgewertet:

◦ Beim Ausdruck `[-x || x]` wird der erste Operand `-x` ausgewertet, weil die (erste und einzige Alternative der) Klammer `[minus]` durchlaufen wurde.

◦ Beim Ausdruck `{ res +! (y || -y) }` wird der Operand `res +! (y || -y)` zweimal ausgewertet, weil die (erste und einzige Alternative der) Klammer `{ (plus|minus) (y) }` zweimal durchlaufen wurde.

◦ Bei der ersten bzw. zweiten Auswertung, bei der der Parameter `y` den Wert 2 bzw. 3 besitzt, wird beim Ausdruck `(y || -y)` der erste bzw. zweite Operand ausgewertet, weil hier die erste bzw. zweite Alternative der Klammer `(plus|minus)` durchlaufen wurde.