P. Thiemann, S. Wehr Wintersemester 2006/2007

Compilerbaupraktikum

http://proglang.informatik.uni-freiburg.de/teaching/compilerpraktikum/2006ws/

Aufgabenblatt 1

Lexing und Parsing

24.10.2006

Auf der Seite zum Praktikum steht ein Frontend für Waitomo zur Verfügung. Erweitern sie das Frontend um die im folgenden beschriebenen Features. Dabei sollen sie das Modul `Syntax.Dst`, falls nicht anders vermerkt, unverändert lassen.

- Fügen sie Unterstüzung für die folgenden Operatoren hinzu. Präzedenz und Assoziativität sollte so wie in Java gewählt werden.
  - Arithmetische Operatoren: `+` (Addition), `-` (Subtraktion), `*` (Multiplikation), `/` (ganzzahlige Division ohne Rest)
  - Logische Operatoren: `&&` (logisches Und) `||` (logisches Oder), `!` (logisches Nicht)
  - Vergleichsoperatoren: `<`, `<=`, `>`, `>=`, `==`, `!=`

  Sie müssen dazu den Lexer, den Parser, sowie die Module `Syntax.Ast`, `Syntax.Dst` und `Desugar` ändern.
- Fügen sie Stringliterale hinzu. Sie müssen dazu den Lexer, den Parser, sowie die Module `Syntax.Ast`, `Syntax.Dst` und `Desugar` ändern.
- Fügen sie Unterstützung für unqualifizierte Feldzugriffe und Methodenaufrufe hinzu. Im Moment muss auf ein Feld oder eine Methode der aktuelle Klasse (oder einer Superklasse) immer mittels `this.f` bzw. `this.m(...)` zugegriffen werden. Durch die Änderungen soll es möglich werden, das Feld `f` und die Methode `m` direkt anzusprechen, d.h. ohne das vorangestellte `this.`, vorausgesetzt `f` ist keine lokale Variable. Um das Feature zu implementieren müssen sie den Parser sowie die Module `Syntax.Ast` und `Desugar` erweitern.
- Fügen sie Unterstützung für die Sichtbarkeitsmodifikatoren `private`, `protected` und `public` hinzu. Dazu müssen sie den Lexer und Parser modifizeren. Außerdem müssen die die Module `Syntax.Ast`, `Syntax.Dst` und `Desugar` erweitern.
- Fügen sie Unterstützung für die folgenden syntaktischen Abkürzungen hinzu:
  - Eine Interfacedefinition der Form

```
interface I<X_1,...,X_n> where Q_1,...,Q_k {
  method_dec_1;
  ...
  method_dec_m;
}
```

ist eine Abkürzung für

```
interface I<X_1,...,X_n> [This] where Q_1,...,Q_k {
  class This {
    method_dec_1;
    ...
    method_dec_m;
  }
}
```

– Falls `I` eine Interface ist, dann kann der existenzielle Typ `exists X where X implements I<Y_1,...,Y_n> . X` zu `I<Y_1,...,Y_n>` abgekürzt werden.

– Sei folgende Deklaration gegeben:

```
implementation<X_1,...,X_n> C<X'_1,...,X'_n> of I<Y_1,...,Y_m>
  where Q_1,...,Q_p {
    static_method_dec_1;
    ...
    static_method_dec_l;
    method_dec_1;
    ...
    method_dec_k;
}
```

(Dabei seien die Methoden `method_dec_i` nicht-statisch, und `X'_1,...,X'_n` sei eine Permutation von `X_1,...,X_n`.) In ausgeschriebener Form ergibt sich folgendes:

```
extension C<X'_1,...,X'_n> where Q_1,...,Q_p {
    method_dec_1;
    ...
    method_dec_k;
}
implementation<X_1,...,X_n> C<X'_1,...,X'_n> of I<Y_1,...,Y_m>
  where Q_1,...,Q_p {
    static_method_dec_1;
    ...
    static_method_dec_l;
}
```

Erweitern sie dazu das Modul `Syntax.Ast` und wandeln sie mittels der Transformation im Module `Desugar` die neuen Konstrukte in einen Datentyp des Modules `Syntax.Dst` um.

Wichtig: Sie müssen die Lauffähigkeit ihrer Implementierung durch Tests belegen. Das auf der Homepage verfügbare Frontend enthält auch eine Testumgebung. Sie können die Tests mittels des Befehls `make check` aufrufen (im Wurzelverzeichnis). Um selber Tests anzulegen sollten sie wie folgt vorgehen:

- Das Verzeichnis `tests` enthält Unterverzeichnisse, welche die Testfälle in Kategorien (Bsp.: `parsing`) einteilen. (Das Unterverzeichnis `driver` gehört nicht dazu.) Entscheiden sie sich, in welche Kategorie ihr Test gehört und legen sie ggf. ein neues Unterverzeichnis an. Jedes der Unterverzeichnisse kann eine Datei mit Namen `Flag` enthalten; die erste Zeile dieser Datei gibt die Kommandozeilenoptionen an, mit der der Compiler für die Testfälle der Kategorie aufgerufen werden sollen.
- Schreiben sie den Sourcecode ihres Testfalls in eine Datei mit der Endung `.wtm` im Verzeichnis der gewählten Kategorie.
- Haben sie einen Testfall `foo` in der Kategorie `bar` angelegt (d.h. im Verzeichnis `tests/bar` gibt es jetzt ein Waitomo Sourcefile `foo.wtm`), so können sie in der Datei `foo.out` (im selben Verzeichnis) die erwartete Ausgaben angeben. Falls sie ein Fehlschlagen des Testfalls erwarten, so müssen sie alternativ in die Datei `foo.err` die erwartete Fehlermeldung schreiben.

Deadline: 27.11.2006