

Projektbericht: Erweiterung von SL um ein Modulsystem

Benjamin Bisping, Rico Jasper, Sebastian Lohmeier, Friedrich Moritz Psiorz

Compilerbauprojekt SoSe 2013, Technische Universität Berlin

1 Einleitung

TODO (* Hällü Wörlö *)

2 Überblick

TODO (* Alle/Ben: Kurz die neuen Features bzw. den Ausgangspunkt beschreiben *)

3 Syntax und Parser

TODO (* Fritz: Syntaxanpassungen, Schwierigkeiten, Designentscheidungen *)

3.1 Qualifizierte Bezeichner

3.2 Grammatik

4 Semantische Analyse

TODO (* Rico: Methode, Schwierigkeiten, Designentscheidungen *)

4.1 Auflösung von Importen

4.2 Type-Checking

5 Codegenerierung

TODO (* Sebastian: Beispiele, Schwierigkeiten, Designentscheidungen *)

TODO (* Aufruf des Compilers, aus binary und in sbt, durchgehendes Beispiel *)

Die Ausführung des generierten JavaScript-Codes wird in node.js¹, Firefox², Chrome³ und Internet Explorer⁴ unterstützt.

Bei Aufruf des Compilers mit

```
$> <PROGRAMM-NAME> -d <outputDirectory> -cp <classpathDirectory>  
[ -sourcepath <sourceDirectory> ] <moduleFile>
```

werden aus dem <classpathDirectory> die Signatur-Dateien bereits kompilierter Module geladen, sowie das angegebene <moduleFile> sowie alle von diesem transitiv verwendeten Module, die noch nicht kompiliert im <outputDirectory> vorhanden sind bzw. deren **TODO (* keine Metonymie! *)**

Modifikationsdatum im <outputDirectory> vor dem Modifikationsdatum der SL-Moduldatei im <sourceDirectory> liegt, kompiliert. Dabei werden Signaturen (siehe Abschnitt 5.1), sowie JavaScript-Dateien (siehe Abschnitt 5.3) für alle kompilierten Module erstellt, wobei require.js (siehe Abschnitt 5.2) verwendet wird, um die JavaScript-Dateien der Module zu laden. Sofern das beim Aufruf des Compilers angegebene <moduleFile> eine Funktion namens **main** deklariert, werden für dieses noch eine **main.js**-Datei und eine **index.html**-Datei erstellt, die den Aufruf der **main**-Funktion in node.js und im Browser erlauben (siehe Abschnitt 5.3).

TODO (* Ausführung des Codes in (unterstütztem) Browser und node.js, Voraussetzungen dafür *)

5.1 Signaturen

TODO (* Das ist wohl eher für Rico... *)

5.2 require.js

Um die Module zur Laufzeit in JavaScript zu laden, wurde require.js⁵ statt CommonJS⁶ ausgewählt, da es im Gegensatz zum Modulsystem von node.js auch im Browser verfügbar ist, jedoch auch in node.js genutzt werden kann **TODO (* naja, das stimmt noch nicht ganz *)**

¹ <http://nodejs.org/> - getestet mit Version 0.10.10 (**TODO (* aktualisieren auf 0.10.13 *)**)

² **TODO (* URL, zum Testen benutzte Version und OS *)**

³ **TODO (* URL, zum Testen benutzte Version und OS *)**

⁴ **TODO (* URL, zum Testen benutzte Version und OS *)**

⁵ <http://requirejs.org/> v. 2.1.6 **TODO (* updaten auf 2.1.8 *)**

⁶ <http://www.commonjs.org/>

In node.js stehen zwei Wege zur Verfügung, um Abhängigkeiten zwischen Modulen zu deklarieren und zur Laufzeit aufzulösen, siehe Listings ?? und ??.
TODO (* AMD besprechen? *)

Die Moduldefinition mit einem Array von Abhängigkeiten (siehe Beispiel im Listing ??) erlaubt den Zugriff auf verwendete Module, können jedoch keine zirkulären statischen Abhängigkeiten auflösen, da für die Erstellung der gegenseitig abhängigen Module jeweils das andere Modul-Objekt als Parameter bei Erstellung des Moduls übergeben werden muss. Dieses Problem wird in require.js mittels Export-Objekten gelöst, die beim Erstellen eines Moduls übergeben und zur Laufzeit verwendet werden (siehe Beispiel im Listing ??). Die Moduldefinition mit Export-Objekten wurde in SL2 gewählt, um später statische zirkuläre Abhängigkeiten auflösen zu können, auch wenn die bisherige Implementierung des Compilers dies nicht erlaubt.

```
define(["modules/B"], function(b) {
    return {
        "a" : function() { return "A.a"; },
        "b" : function() { return b.b(); }
    };
});

define(function(require, exports, module) {
    var b = require("modules/B");
    exports.a = function() { return "A.a"; };
    exports.b = function() { return b.b(); };
});
```

TODO (* Kompilierung der Module *)

TODO (* Kompilierung der main-Funktion *)

TODO (* Designentscheidung für require.js-Verwendung, die theoretisch auch statisch zirkuläre Abhängigkeiten auflösen kann *)

TODO (* require.js wird mitgeliefert, sodass es für Ausführung im Browser nicht installiert werden muss *)

TODO (* Installation von require.js in node.js – im lokalen Verzeichnis oder global? in Systemvoraussetzungen für SL2 beschreiben *)

5.3 Build-Prozess

TODO (* implizit unqualifiziert importiertes prelude aus dem resources-Verzeichnis der SL2-Distribution, Zugriffe darauf werden nach dem Typcheck qualifiziert mit /lib/prelude – bzw. mit /lib/prelude *)

TODO (* Übersetzung der / (oder aller nicht-zugelassenen Zeichen) zu \$ in JavaScript? *)

TODO (* Ort, an dem die Templates, prelude, und require.js (im Distributable) gespeichert sind *)

5.4 Externe Definitionen

TODO (* Das ist wohl eher für Ben... *)

6 Prelude und Bibliotheken

TODO (* Ben: Beispiele, Schwierigkeiten, Designentscheidungen *)

6.1 Prelude

6.2 Floats

7 Fehlermeldungen

TODO (* Fritz *)

8 Zusammenfassung

TODO (* ... *)