

SL2: Die Simple Language mit Modulsystem

Benjamin Bisping, Rico Jasper,
Sebastian Lohmeier und Friedrich Psiorz

Compilerbauprojekt SoSe 2013
Technische Universität Berlin
20.09.2013

Einführung

Syntax und Parser

Semantische Analyse

Codegenerierung und Signaturen

Fehlermeldungen

Prelude und Bibliotheken

Beispielprogramme und Tests

Fazit

Beamer-Beispiele

Einführung

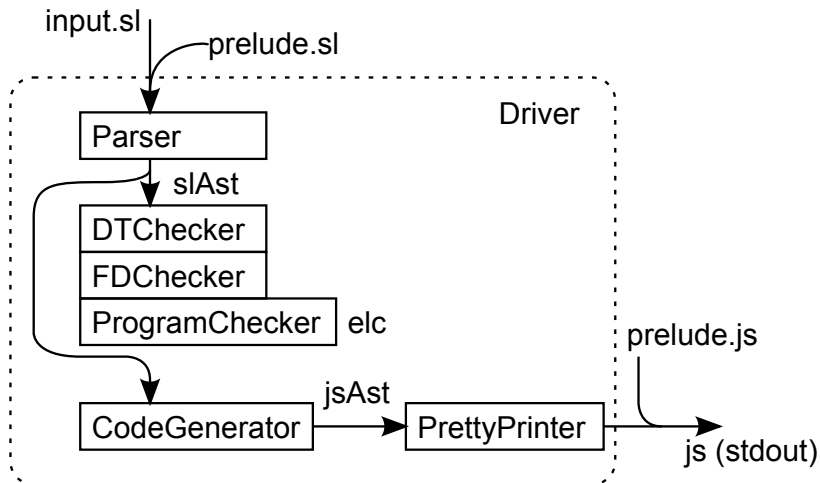
SL: typischer und funktional im Browser (JavaScript)



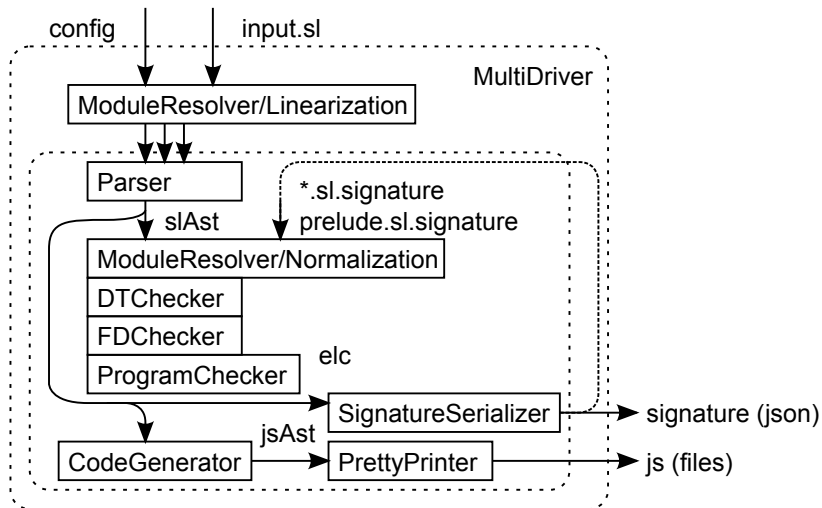
SL2: unabhängig kompilierbare Module

- ▶ Moduldefinition und -import (auch für das Prelude)
- ▶ Export und einfache Qualifizierung von Funktionen und Datentypen
- ▶ Einbindung von Funktionen und Datentypen aus JavaScript
- ▶ Anpassungen der Syntax und Semantik
- ▶ Fehlermeldungen verbessert
- ▶ Compilierung ins Dateisystem
- ▶ Bibliotheken, Beispielprogramme und Tests

Altes Framework



Neues Framework



Syntax und Parser

Fritz

Semantische Analyse

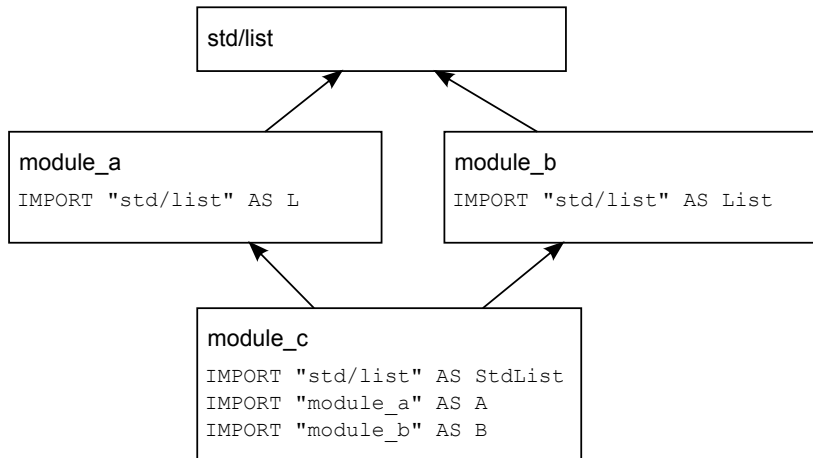
1. Auflösung von Importen
2. Modulnormalisierung
3. Datentypen und Funktionen überprüfen
4. Type-Checking

Import-Überprüfung

- ▶ Import-Anweisung: Paar aus Pfad und Bezeichner
`IMPORT "path/to/module" AS MyModule`
- ▶ eindeutige Modul-Bezeichner-Zuordnung
- ▶ Annahme: genau ein Pfad identifiziert ein Modul
- ▶ erlaubte Pfade:
 - ▶ Kleinbuchstaben
 - ▶ Zahlen
 - ▶ Minus (-) und Unterstrich (_)
 - ▶ relative Pfade

Modulnormalisierung

- ▶ keine Modulübergreifende Modul-Bezeichner-Zuordnung
- ▶ Normalisierung erforderlich



Kontextprüfung

- ▶ Berücksichtigung von importierten Datentypen und Funktionen
- ▶ initialer Kontext um Modulkontext erweitert
- ▶ Type-Checker weitestgehend unverändert

Codegenerierung und Signaturen

1. Modulsignatur
2. Compileraufruf und Pfade
3. Abhängigkeitsanalyse
4. require.js
5. Code-Generierung

Modulsignatur

- ▶ Signatur für semantische Analyse erforderlich
- ▶ Inhalt:
 - ▶ Importliste
 - ▶ Datendefinitionen
 - ▶ Funktionssignaturen
- ▶ Mögliche Signaturformate:
 - ▶ native Serialisierung
 - ▶ SL
 - ▶ JSON

Modulsignatur – JSON

```
IMPORT "some/module" AS M
```

JSON:

```
"imports" : [  
  {  
    "name" : "M",  
    "path" : "some\\module"  
  }  
]
```

Compileraufruf und Pfade

```
> run-main de.tuberlin.uebb.sl2.impl.Main  
[-d <output directory>]  
[-cp <classpath directory>]  
-sourcepath <source directory>  
<module files>
```

Abhängigkeitsanalyse I

Ein Modul ist zu kompilieren, wenn

1. Quell-Datei in `<module files>`, oder
2. importiert
und Quell-Datei im `<source directory>`
keine Signatur-Datei im `<classpath directory>`, oder
3. importiert
und Quell-Datei im `<source directory>`
und Signatur-Datei im `<classpath directory>`
und Quell-Datei jünger als Signatur-Datei.

Abhängigkeitsanalyse II

*A.sl → B.sl
A.sl.signature B.sl.signature

A.sl → *B.sl
A.sl.signature B.sl.signature

A.sl → B.sl → *C.sl
A.sl.signature B.sl.signature C.sl.signature

require.js

require.js statt Common.js

Installation in node.js (u.U. relativ zum akt. Verzeichnis)

```
> npm install requirejs
```

Code-Generierung I

```
> run-main de.tuberlin.uebb.sl2.impl.Main -sourcepath  
src/main/sl/examples/ boxsort.sl
```

```
boxsort.sl.signature
```

```
boxsort.sl.js
```

```
main.js
```

```
require.js
```

```
index.html
```

Code-Generierung II

```
IMPORT "std/debuglog" AS Dbg
```

```
...
```

```
PUBLIC FUN main : DOM Void
```

```
DEF main =
```

```
    Web.document &= \ doc .
```

```
    ...
```

```
DEF getNode (NodeWithNumber n1 i1) = n1
```

```
...
```

Code-Generierung III: boxsort.sl.js

```
define(function(require, exports, module) {  
  var $$std$prelude = require("std/prelude.sl");  
  var Dbg = require("std/debuglog.sl");  
  ...  
  function $getNode(_arg0) { ... };  
  ...  
  var $main = function () { ... }();  
  exports.$main = $main  
});
```

Code-Generierung IV: main.js

```
if (typeof window === 'undefined') {  
  /* in node.js */  
  var requirejs = require('requirejs');  
  
  requirejs.config({  
    //Pass the top-level main.js/index.js require  
    //function to requirejs so that node modules  
    //are loaded relative to the top-level JS file.  
    nodeRequire: require,  
    paths: {std : "C:/Users/monochromata/git/sl2/target/  
      scala-2.10/classes/lib" }  
  });  
  
  requirejs(["boxsort.sl"], function($$$boxsort) {  
    $$$boxsort.$main()  
  });  
...  
...
```

Code-Generierung V: main.js

```
...  
} else {  
  require.config({  
    paths: {std : "file:/C:/Users/monochromata/git/sl2/  
      target/scala-2.10/classes/lib/" }  
  });  
  
  /* in browsers*/  
  require(["boxsort.sl"], function($$$boxsort) {  
    $$$boxsort.$main()  
  });  
}
```

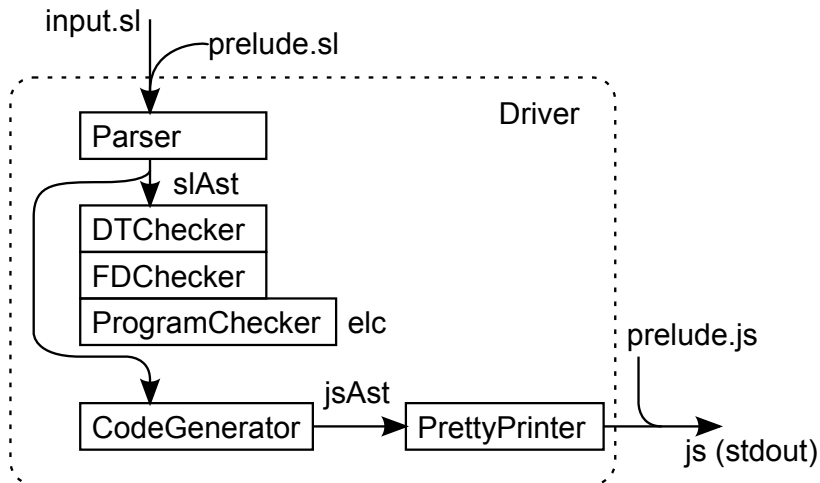
Fehlermeldungen

Fritz

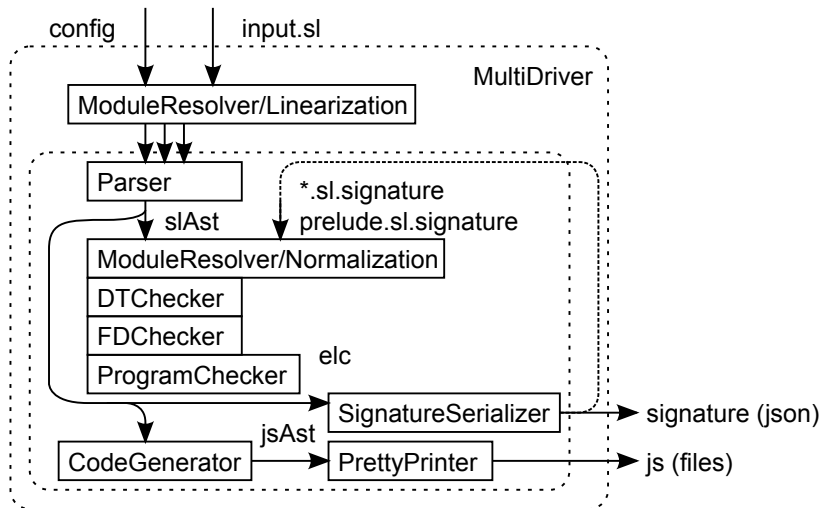
Prelude und Bibliotheken

- Ben

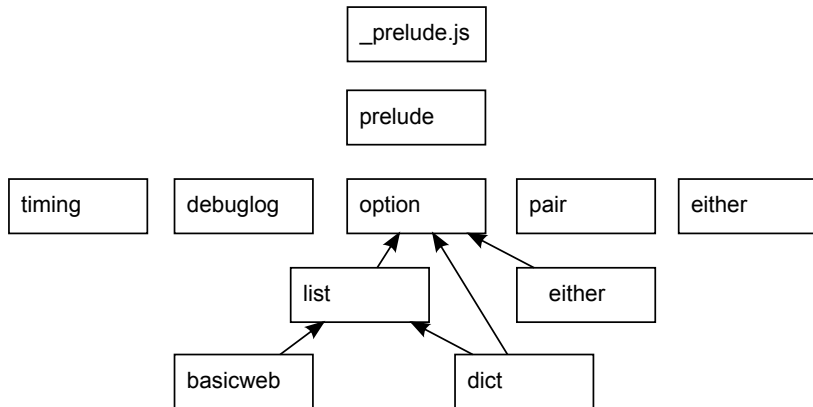
Prelude im alten Framework



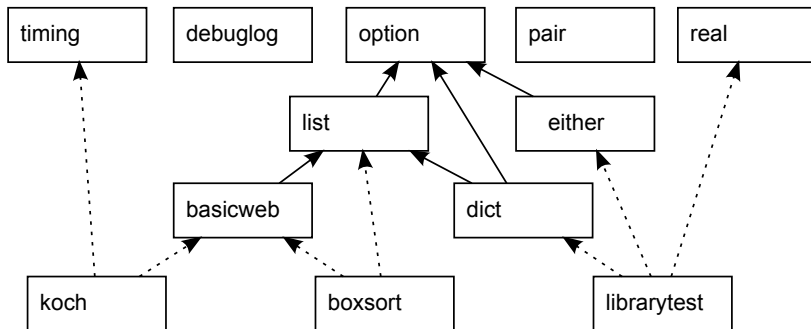
Prelude im neuen Framework



Bibliotheken



Bibliotheken



Beispielprogramme und Tests

- ... evtl Live-Programmierung - Ben

Fazit I

- ▶ Modulare typsichere Webanwendungen im Browser und node.js möglich
- ▶ Modulimporte, qualifizierte Bezeichner, Exporte
- ▶ Fehlermeldungen verbessert
- ▶ Prelude in Module überführt
- ▶ initiale Standard-Bibliothek erstellt

→ Pflichtenheft erfüllt

Fazit II

Mögliche Erweiterungen

- ▶ Flexiblerer Import
- ▶ Statische zyklische Abhängigkeiten
- ▶ Konfiguration der Codegenerierung für require.js
- ▶ Verbesserte Typchecker-Fehlermeldungen
- ▶ Erweiterte Bibliotheken

Itemize und enumerate

bullet points: itemize, Nummerierung: enumerate

- ▶ **EMMA** and *motor modules*
- ▶ Spreading activation with :bll 0.3 :mas 3 :rt 0 :ga 1
:retrieval-activation 4 :visual-activation 2
:imaginal-activation 8
- ▶ New TWM nodes created in imaginal buffer to keep parsing state and context in goal buffer
- ▶ Word frequencies from dlexdb.de for base levels and EMMA

Eine Tabelle

| Condition (2 items, 6 subjects) | DA | IA |
|---------------------------------|--------|--------|
| Experiment (unreliable) | 3824ms | 3946ms |
| Model 1 | 3242ms | 3323ms |
| Model 2 | 3527ms | 3293ms |

Quellcode anzeigen

[containsverbatim=true] nach frame-Beginn nicht vergessen

```
for c in range(k):  
    Mus[:,c]=X.T[np.where(rPrime==c)].T.mean(1)
```

vs.

```
for i in range(k):  
    mu[:,i] = np.mean(X[:,raux==i],axis=1)
```

Grafiken einbinden

Zur Skalierung einfach den Faktor ohne Multiplikationszeichen vor die Breitenangabe schreiben.

