Formal Specification of a JavaScript Module System

Seonghoon Kang and Sukyoung Ryu

PLRG @ KAIST

January 17, 2012

Table of Contents

1. JavaScript and Module System

2. Brief Introduction to λ_{JS}

3. JavaScript with Modules

4. Properties and Implementation

5. Conclusion

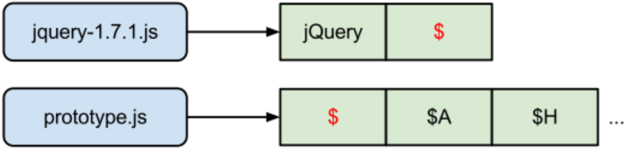
JavaScript

JavaScript was first envisioned as a scripting language for simple tasks, but now being used on a much larger scale than intended.

However, JavaScript is not yet ready for programming in the large: it does not support a *module system*.

JavaScript and (Missing) Module System

All names share the same namespace: a huge problem when a commonly-used name is shared by multiple libraries.



JavaScript and Module System

There are two directions towards a module system in JavaScript:

Simulating modules via function scopes ("module pattern")
Widespread throughout the JavaScript community, but very limited without any static verification.

Extending the JavaScript language itself
Proposed as the informal ECMAScript.next proposal in prose. But given the strangeness of JavaScript, who can be sure about its correctness?

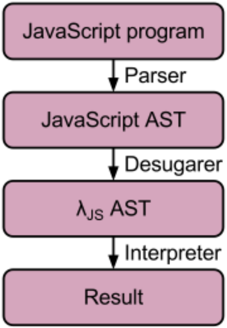
Our Contributions

Our contributions are as follows:

1. Developed the formal specification of the module system described by the ECMAScript.next proposal,
2. Proved its essential properties, and
3. Implemented and successfully tested it.

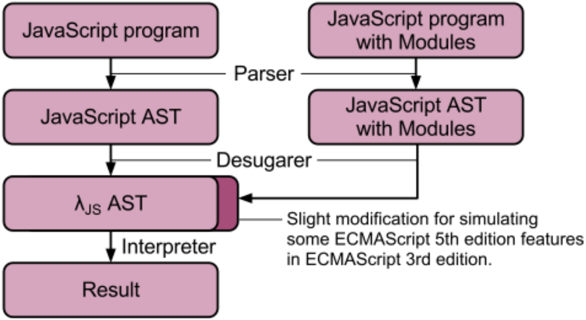
λ_{JS}

λ_{JS} is a core language for JavaScript, which also implements a complete JavaScript implementation.



λ_{JS} (Modified)

To implement a module system, we have to modify some portions of λ_{JS}.



Slight modification for simulating some ECMAScript 5th edition features in ECMAScript 3rd edition.

JavaScript with Modules

Our module system supports the following extensions:

$\texttt{module}\ M\ \{s\cdots\}$	Module definition
$\texttt{module}\ M\ \texttt{=}\ M\cdots\texttt{.}M\texttt{;}$	Module alias
$\texttt{import}\ M\cdots\texttt{.}x\texttt{;}$	Qualified name import
$\texttt{import}\ M\cdots\texttt{.}x\texttt{:}\ x\texttt{;}$	Aliased import
$\texttt{import}\ M\cdots\texttt{.*;}$	Import all
$\texttt{export var}\ x\ [\texttt{=}\ e]\texttt{;}$	Exported variable declaration
$\texttt{export function}\ x\texttt{(}x\cdots\texttt{)}\ \{s\cdots\}$	Exported function definition
$\texttt{export module}\ M\ \{s\cdots\}$	Exported module definition
$\texttt{export module}\ M\ \texttt{=}\ M\cdots\texttt{.}M\texttt{;}$	Exported module alias
$\texttt{export}\ x\texttt{;}$	Exported local name
$\texttt{export}\ x\texttt{:}\ x\texttt{;}$	Aliased export of local name
$\texttt{export}\ x\texttt{:}\ M\cdots\texttt{.}x\texttt{;}$	Aliased export of qualified name

Module System Example

```
module Browser {
  export module DOM {
    var document = ...;
    export document;
  }
}
module DesignMode {
  import Browser.DOM.document: DOMdocument;
  export function initialize() {
    document = DOMdocument.createElement(‘iframe’); ...
  }
  export var document;
}
module DOM = Browser.DOM;
import DOM.*;
DesignMode.initialize();
document.body.appendChild(DesignMode.document);
```

Module System Example

```
module Browser {
  export module DOM {
    var document = ...;
    export document;
  }
}
module DesignMode {
  import Browser.DOM.document: DOMdocument;
  export function initialize() {
    document = DOMdocument.createElement(‘iframe’); ...
  }
  export var document;
}
module DOM = Browser.DOM;
import DOM.*;
DesignMode.initialize();
document.body.appendChild(DesignMode.document);
```

Module System Example

```
module Browser {
  export module DOM {
    var document = ...;
    export document;
  }
}
module DesignMode {
  import Browser.DOM.document: DOMdocument;
  export function initialize() {
    document = DOMdocument.createElement(‘iframe’); ...
  }
  export var document;
}
module DOM = Browser.DOM;
import DOM.*;
DesignMode.initialize();
document.body.appendChild(DesignMode.document);
```

Module System Example

```
module Browser {
  export module DOM {
    var document = ...;
    export document;
  }
}
module DesignMode {
  import Browser.DOM.document: DOMdocument;
  export function initialize() {
    document = DOMdocument.createElement(‘iframe’); ...
  }
  export var document;
}
module DOM = Browser.DOM;
import DOM.*;
DesignMode.initialize();
document.body.appendChild(DesignMode.document);
```

Module System Example

```
module Browser {
  export module DOM {
    var document = ...;
    export document;
  }
}
module DesignMode {
  import Browser.DOM.document: DOMdocument;
  export function initialize() {
    document = DOMdocument.createElement('iframe'); ...
  }
  export var document;
}
module DOM = Browser.DOM;
import DOM.*;
DesignMode.initialize();
document.body.appendChild(DesignMode.document);
```

Modification of the Core Language

The *module instance object* requires a modification in λ_{JS}.

To implement the module instance object, we add an *unmodifiable JavaScript object*.

Desugaring Environment

To simplify the desugaring process of a given program, we construct a *desugaring environment* before the actual desugaring.

The desugaring environment maps qualified names to their corresponding identities (information about their creation and name resolution).

Module Desugaring

Desugaring of modules is very simple: correct desugaring of names. For example,

- For a module name, we should return a module instance object.
- For an exported variable name, we should return a value of the variable from (the mutable version of) the module instance object directly.

Properties of Module System: Validity

The desugaring process does not produce an invalid λ_{JS} program:

Theorem (Validity of Desugaring Rules)

The desugared program of a valid program p should not cause an error condition due to an absent (λ_{JS}) binding.

$$\neg\exists x.\, \exists\sigma.\, \epsilon \; \mathit{desugar}[\![p]\!] \rightarrow^{*} \sigma E\langle x\rangle$$

Properties of Module System: Isolation

The module system does not allow outside code to inspect internal variables that are not exported:

Theorem (Isolation of Module Scopes)

A module-scope variable that is not exported is not visible outside its enclosing module.

$$
\begin{aligned}
&\forall M.\, \forall x.\, (\neg\exists\rho.\, \exists\varsigma.\, ((M), \rho\, \varsigma) \in \mathrm{Env}[\![p]\!] \implies \\
&\qquad\qquad \neg\forall v.\, \exists\sigma.\, \epsilon\ \mathit{desugar}[\![\mathtt{module}\ M\ \{\mathtt{var}\ x\ \mathtt{=}\ v\mathtt{;}\}\ p]\!] \to^{*} \sigma\, v)
\end{aligned}
$$

Implementation and Testing

We have also implemented the module system on top of λ_{JS}. The resulting implementation is successfully tested with:

- The Mozilla JavaScript test suite for existing JavaScript features, and
- Our own harness tests for module features.

Our implementation is available at:

`http://plrg.kaist.ac.kr/research/software`

Conclusion

We developed a formal specification of a module system for JavaScript, based on the informal, work-in-progress proposal, ECMAScript.next:

- We define the desugaring process from JavaScript to the λ_{JS} core language,
- Proved the essential properties of our desugaring process, and
- Implemented and tested it with the real-world JavaScript test suite and our own harness tests.

Our future work has two directions: the conversion from λ_{JS} to module-free JavaScript, or the conversion from JavaScript with modules to module-free JavaScript.