JavaScript Module System: Exploring the Design Space (Extended with Formalization)

Junhee Cho Sukyoung Ryu
KAIST
{ssaljalu, sryu.cs}@kaist.ac.kr

Abstract

While JavaScript is one of the most widely used programming languages not only for web applications but also for large projects, it does not provide a language-level module system. JavaScript developers have used the module pattern to avoid name conflicts by themselves, but the prevalent uses of multiple libraries and even multiple versions of a single library in one application complicate maintenance of namespace. The next release of the JavaScript language specification will support a module system, but the module proposal in prose does not clearly describe its semantics. Several tools already support the new features in the next release of JavaScript by translating the new features into the current JavaScript, but their module semantics are different from the proposal.

In this paper, we identify some of the design issues in the JavaScript module system. We describe ambiguous or undefined semantics of the module system with concrete examples, show how the existing tools support them incorrectly, and discuss reasonable choices for the design issues. We specify the formal semantics of the module system, which provides unambiguous description of the design choices. We provide an implementation of the module system as a source-to-source transformation from JavaScript with modules to the current JavaScript that the current JavaScript engines can evaluate. Our formalism and implementation techniques of the JavaScript module system are applicable to other scripting languages.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features

General Terms Languages

Keywords JavaScript, module system, source-to-source transformation

1. Introduction

JavaScript [4] is originally developed as a simple scripting language but now it is one of the most widely used programming languages. JavaScript enriches web documents by supporting dynamic interaction with users. When a user provides a request to a web document by clicking a link, for example, the web browser responds to the request by rendering another web document. If the request triggers a local change in the web document, the browser does not need to reload the entire web page. For instance, when a web page embeds a small messenger widget inside it, the browser may reload only the widget but the enclosing page for each message exchange. JavaScript enables such dynamic partial updates of web documents by communicating with servers through HTTP and updating web documents via browser APIs for Document Object Model (DOM) [30]. Thanks to the capability, 98 out of the 100 most visited websites according to Alexa [13] use JavaScript [9], and its use outside client-side web programming keeps growing. Large stand-alone projects such as node.js [14] for building scalable network applications use JavaScript and web applications on various platforms including Samsung Smart TV SDK [25] and Tizen SDK [6].

However, JavaScript does not provide any language-level module system. Simple scripts may not need to maintain namespace but a module system is necessary for programming in the large. Because JavaScript does not have the standard libraries, JavaScript developers use more than a dozen libraries: Dojo [5] focuses on building complete web applications, jQuery [15] focuses on improving the interaction between JavaScript and HTML, MooTools [29] emphasizes animation supports, Prototype [27] focuses on adding new features to JavaScript, YUI [12] focuses on improving the user interaction, and other libraries have their own strength. Due to the lack of a module system and standard libraries, JavaScript developers often use multiple libraries in one project to selectively use functionalities from them. Similarly, some JavaScript projects use even multiple versions of a single library to use both new API functions introduced in a newer version and old API functions deprecated in the newer version. While multiple versions of a single library obviously share symbols with the same names, different libraries often use the same symbol names too, which lead to name conflicts when a single project uses them. For example, because both jQuery and Prototype use the symbol `$`, if an HTML document includes both libraries by `<script>` tags, the library included later overrides the symbol `$` from the other library with its own `$`. Figure 1 shows an example from the jQuery API document [16]; it loads two versions of jQuery, `jquery-1.9.1.js` and `jquery-1.6.2.js`, and then restores jQuery's globally scoped variables to the first loaded jQuery, `jquery-1.9.1.js`.

To alleviate the problem, the JavaScript community has used the "module pattern" [20] and different libraries provide different ways to address name conflicts. The module pattern is to use functions to simulate modules by assigning an anonymous function to a namespace object and adding "private" members to the anonymous function. It provides some forms of namespace manipulation but such a programming pattern is verbose and error prone, and programmers are responsible for checking that they follow the pattern correctly. Libraries also provide mechanisms to handle name con-

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.9.1.js">
  </script>
</head>
<body>

<div id="log">
  <h3>Before $.noConflict(true)</h3>
</div>
<script src="http://code.jquery.com/jquery-1.6.2.js">
</script>

<script>
var $log = $( "#log" );

$log.append( "2nd loaded jQuery version ($): " +
             $.fn.jquery + "<br>" );

/*
Restore globally scoped jQuery variables to
the first version loaded (the newer version)
*/
jq162 = jQuery.noConflict(true);

$log.append( "<h3>After $.noConflict(true)</h3>" );
$log.append( "1st loaded jQuery version ($): " +
             $.fn.jquery + "<br>" );
$log.append( "2nd loaded jQuery version (jq162): " +
             jq162.fn.jquery + "<br>" );
</script>

</body>
</html>
```

Figure 1. Multiple versions of jQuery in one HTML document, an excerpt from the jQuery API document [16]

flicts. Because many JavaScript libraries use `$` as a variable or function name, jQuery provides the `noConflict` mechanism to give up its uses of `$` [16]. When a web page loads jQuery, it saves the values already bound to `$` and binds the jQuery object to `$` and `jQuery`. If `$.noConflict()` is called after loading jQuery, it restores the saved values to `$` and only `jQuery` refers to the jQuery object. While Prototype and MooTools do not provide such a mechanism to avoid name conflicts, they have different semantics to handle name conflicts. Prototype does not save the old values bound to `$` before loading the library; it is programmers' responsibility to save the old values before loading Prototype. On the contrary, MooTools binds the MooTools object to `$` only if nothing is already bound to `$`, and aliased to `document.id` [28]. Therefore, programmers should deal with each library with its own mechanism.

Given the growing popularity of JavaScript as a general purpose language for programming in the large and the need for a standard, language-level support for namespace manipulation, the next release of JavaScript (JavaScript.next) plans to include a module system. The ECMAScript Harmony proposals [2] are a collection of proposals for the next ECMA-262 language specification, and most of the proposals have already been incorporated into the ECMAScript 6 specification draft [3]. While the Harmony proposals include a module proposal [11], the proposal description is very brief and informal. Even in the ECMAScript 6 specification draft, the sections related to modules are empty. The Harmony module proposal describes the high-level ideas and the design principles of the module system succinctly, it does not specify the complete semantics of modules especially the various interactions with the existing JavaScript language features.

Despite of the current status of the module proposal, the long-awaited module system gets eagerly adopted in various ways. Kang *et al.* [17] present a formalization of the core Harmony module proposal that does not include dynamic module loading. They introduce a formal specification and implementation of the module system by desugaring JavaScript extended with modules to λ_{JS} [10], a core-calculus of JavaScript. Traceur [7] is a compiler to translate a JavaScript.next program to a JavaScript program, which enables developers to use the new features introduced in JavaScript.next in advance. It provides various new features including a module system and asks for user inputs to reflect the feedback in the ECMAScript standards process. Among many variant languages of JavaScript, TypeScript [19] is a typed superset of JavaScript that compiles to plain JavaScript. It supports classes, modules, and interfaces to improve static checking and verification of TypeScript programs.

Unfortunately, the immature status of the Harmony module proposal causes discrepancies between various module extensions of JavaScript. Kang *et al.*'s module system clearly describes the module semantics via desugaring to λ_{JS}, but because λ_{JS} is very different from JavaScript it is not trivial to understand the semantics of a JavaScript program using modules in terms of λ_{JS}. While λ_{JS} serves an important role to formally reason about JavaScript program behaviors utilizing the research results on other programming languages based on λ calculus, the semantics of λ_{JS} is very different from the original JavaScript and the execution of huge desugared λ_{JS} programs consumes a lot of memory for immutable object manipulations. Also, λ_{JS} does not support dynamic code generation functions such as `eval`. Many websites use the extremely dynamic features of JavaScript [22–24]; particular examples are creating JavaScript objects out of JSON strings by calling the `eval` function with the JSON strings. Even though many websites use dynamic code generation, Kang *et al.*'s module system does not support them because λ_{JS} does not support them. On the contrary, Traceur and TypeScript do not formally describe their module systems. Traceur partially implements the module proposal and TypeScript also provides some variant of the module proposal.

While several approaches address the JavaScript.next module system today, the lack of a formal specification of the module system results in different semantics by different approaches. Because the Harmony module proposal is brief and informal, it does not specify the semantics of interactions between the module system and the existing language features precisely, which makes different approaches implement ambiguous features differently. However, the proposal does not provide any ground to identify design issues, to evaluate different design choices, and to check the semantics equality.

In this paper, we investigate the design issues of the JavaScript module system with concrete code examples. While one of the goals of the module system is orthogonality from existing features, the informal proposal reveals some ambiguous semantics of the module system and interactions with existing features. First, because of the interactions between the new module system and existing features, both Traceur and TypeScript provide wrong semantics of import and export statements by source-to-source transformation. For example, when a Traceur module imports an exported variable from another module, the importing module assigns the current value of the variable to the imported variable without any connection to the exported variable, which does not satisfy the module proposal. Second, the proposal specifies the semantics when there are conflicts between import declarations, but it does not specify any semantics when there are conflicts between function, variable, module, and import declarations. Third, while every JavaScript object has its prototype, a module instance object that is a first-class object that reflects the exported bindings of an evalu-

ated module does not have any prototype. The unique difference of module instance objects from ordinary JavaScript objects leads to peculiar semantics of module instance objects. Finally, the module proposal does not specify the interactions between module declarations and dynamic code generation functions such as `eval`. We describe each of the design issues in more detail with examples, and we discuss and explain our design choices. We believe that such an exploration of the design space will help the ECMAScript committee design the module system more clearly and reasonably, and the third-party developers support the module system correctly.

To rigorously discuss and describe the JavaScript module system, we present a formal specification of the module system via source-to-source transformation rules. Because the ECMAScript language specification describes the semantics informally in prose, we developed a formal specification and implementation of a Scalable Analysis Framework for ECMAScript (SAFE) [18] and made both the formal specification and the implementation publicly available:

`http://plrg.kaist.ac.kr/redmine/projects/jsf/repository`

We provide a formal specification of the module system by extending the formal specification of SAFE. It describes the semantics of the module system precisely, and it also specifies rewriting rules from JavaScript with modules to plain JavaScript.

We provide an implementation of the module system by rewriting JavaScript programs with modules to plain JavaScript programs. We also do this by extending the implementation of SAFE. We extended the JavaScript parser and AST structures in SAFE to support the module system, and implemented an extra phase to compile the module syntax away. By supporting a JavaScript.next feature today via source-to-source translation precisely, JavaScript application developers can try future features in advance and contribute their feedback to the design of JavaScript.next, and JavaScript engine developers can apply the same technique to their existing engines without modifying the current implementation largely. Our implementation of the module system is also available to the public via the SAFE repository.

The contributions of this paper are as follows:

- We describe several design issues of the JavaScript module system with concrete code examples, and we discuss reasonable design choices.
- We present an unambiguous formal specification of the module system, which makes it possible to evaluate various design choices formally. Because our formal specification of the module semantics and the rewriting rules is not limited to JavaScript, we can apply the same techniques to other scripting languages.
- We provide an implementation of the JavaScript module system so that JavaScript developers can try a future JavaScript feature today and JavaScript engine developers can support the future feature today.

The remainder of this paper is organized thus. In Section 2, we introduce design issues in the JavaScript module system with the comparison of how Traceur and Prototype support the module system, and we also discuss the design issues and reasonable choices. Section 3 describes a formalization of the module system and it addresses some of alternative semantics depending on the design choices. It also proves two safety properties of the module system: isolation of namespaces and validity of module environments. In Section 4, we describe our implementation of the module system focusing on the key ideas such as the translation phase, and we discuss limitations of our approach. 5 discusses related work and we conclude in 6.

```
module Foo {
  export var foo = 42;
  export function inc() { foo++; }
}
import foo from Foo;
Foo.inc();
foo; // 43
```

Figure 2. Changing the value of an exported name

```
var Foo = (function() {
  "use strict";
  var foo = 42;
  function inc() {
    foo++;
  }
  return Object.preventExtensions(Object.create(
    null,
    { foo: { get: function() { return foo; },
             enumerable: true },
      inc: { get: function() { return inc; },
             enumerable: true }
    }));
}).call(this);
var foo = Foo.foo;
Foo.inc();
foo; // 42
```

Figure 3. Translation of Figure 2 by Traceur

```
var Foo;
(function (Foo) {
    Foo.foo = 42;
    function inc() {
        Foo.foo++;
    }
    Foo.inc = inc;
})(Foo || (Foo = {}));
var foo = Foo.foo;
Foo.inc();
foo; // 42
```

Figure 4. Translation of Figure 2 by TypeScript

2. Design Issues in JavaScript Module System

As we discussed in Section 1, a brief informal description of the Harmony module proposal leads to discrepancies between implementations of the module system. Both the source-to-source translators Traceur and TypeScript provide wrong semantics for import and export declarations, for example. In this section, we discuss four important semantic issues to make the module proposal more elaborate and precise, and to prevent the current module implementations from diverging from the proposal semantics.

2.1 Import and Export Declarations

The Harmony proposal does not describe the semantics of the import and export declarations completely. The proposal specifies their semantics as follows:

> "Export declarations declare that a top-level declaration in a module is visible externally to the module"

```
module Foo {
    var foo = "foo";
    module Bar {
        export var foo = "bar", bar = "bar";
        eval("delete foo; delete bar;");
    }
    export { foo, Bar };
}
Foo.Bar.foo; // "bar"? undefined? error?
Foo.Bar.bar; // "bar"? undefined? error?
```

Figure 5. Export declarations and the `eval` function

```
(function Foo() {
    var foo = "foo", bar = 42;
    (function Bar() {
        bar; // undefined;
        var bar = "bar";
        bar; // "bar";
    })();
})();
```

Figure 6. Static variable declaration in a function scope

> "Import declarations bind another module's exports as local variables."

Based on the proposal, both Traceur and TypeScript translate an import declaration to an assignment expression, assigning the current value bound to the exported name to the imported name, which does not reflect any future changes of the value of the exported name. Figure 2 presents a simple module declaration of `Foo` exporting a variable `foo` and a function `inc`. The top-level imports `foo` from the module `Foo`, increments the value of `foo` by calling the exported function `inc`, and gets the value of the imported `foo`. According to the semantics in the Harmony proposal, the value should be 43. However, both Traceur and TypeScript do not preserve the semantics. Figure 3 and Figure 4 present the translated version by Traceur and TypeScript, respectively. They are slightly different; Traceur extends `Object` with `preventExtensions` to faithfully simulate read-only module instance objects, but TypeScript simply supports the traditional module pattern with a different syntax for import statements. More importantly, both of them produce different results 42, because the translation is merely desugaring an import statement to an assignment expression without maintaining a reference to the original exported name.

Because the Harmony proposal does not specify the interactions between the import and export statements and other existing JavaScript features, there are several cases where the semantics is not clear. For example, as Figure 5 illustrates, what happens when evaluation of the `eval` function deletes variables that are exported to out of the module scope? Depending on the semantics, `Foo.Bar.foo` may evaluate to `"bar"` if the `eval` function call does not delete the variable `foo`, it may evaluate to `undefined` if the call deletes the variable, or it may throw an error if the semantics disallows such a delete operation. Interestingly, Traceur and TypeScript behave differently: Traceur throws a syntax error for trying to delete `foo` in strict mode because it translates a module body to a strict code, and TypeScript evaluates it to `"bar"`.

Another case of unclear semantics is the interactions between the import and export statements and evaluation of variable and function declarations. Before describing the interactions, let us review the JavaScript semantics for evaluation of variable and function declarations. When a JavaScript engine evaluates a program code or a function body code, it first binds all the variable and function declarations in the code and then evaluates the code. For a variable declaration with an initialization expression, the variable declaration is conceptually divided into a variable declaration without any initialization expression and an assignment expression to bind the initialization expression to the variable. Thus, variable and function declarations are bound before actually evaluating any other statements; in a single scope, variables or functions may be used before definition textually. For example, Figure 6 shows that two occurrences of `bar` in the body of the function `Bar` evaluate to different values. The first reference to `bar` evaluates to `undefined` because it is already declared but not yet initialized, and the second reference to `bar` evaluates to `"bar"` because it is initialized with the value. In effect, the body of `Bar` has the same semantics with the following:

```
var bar;      // variable declaration
bar;          // not yet initialized
bar = "bar";  // variable initialization
bar;          // after initialization
```

On the other hand, a variable declaration in a code string passed to `eval` as its argument is not evaluated as a variable declaration in the enclosing code; the variable declaration is bound after evaluating the `eval` function call, which behaves differently from static variable declaration. Consider the example in Figure 7. The first reference to `bar` evaluates to 42 instead of `undefined` because the variable `bar` is not yet declared because the `eval` function call is not yet evaluated.

As the module pattern shows, a module scope is analogous to a function scope, which leads to similar inconsistent behaviors with module scope. What happens when evaluation of the `eval` function declares a variable that is already declared in the module scope? It is not clear whether a variable declared by the `eval` call shadows the already declared variable with the same name. It is not clear either whether such a dynamic variable declaration should affect the values of the exported variables. For example, consider the example in Figure 8. The second reference to `bar` in the module `Bar` and the property access to `Foo.Bar.bar` outside the module may evaluate

```
(function Foo() {
    var foo = "foo", bar = 42;
    (function Bar() {
        bar; // 42;
        eval("var bar = 'bar';");
        bar; // "bar";
    })();
})();
```

Figure 7. Dynamic variable declaration in a function scope

```
module Foo {
    export var foo = "foo", bar = 42;
    module Bar {
        export bar;
        bar; // 42;
        eval("var bar = 'bar';");
        bar; // 42? "bar"?
    }
    export Bar;
}
Foo.Bar.bar; // 42? "bar"?
```

Figure 8. Dynamic variable declaration in a module scope

```
module Foo {
    export var foo = "foo";
}
module Bar {
    import foo from Foo;
    foo; // "foo";
    eval("module Foo { export var foo = 42; };"
        + "export Foo;");
    foo; // "foo"? 42?
}
Bar.foo; // "foo"? 42?
```

Figure 9. Dynamic variable declaration in module scope

to 42 if the dynamic variable declaration does not have any effects; both of them may evaluate to `"bar"` if the dynamic variable declaration affects even the exported name. Or, the second reference to `bar` may evaluate to `"bar"` and `Foo.Bar.bar` may evaluate to 42, if the dynamic variable declaration is in effect only in a module scope. Indeed, Traceur evaluates both of them to 42 but, interestingly, TypeScript evaluates them to `undefined` because their simple translation does not handle nested variable declarations correctly.

Unlike Traceur and TypeScript, our formal semantics correctly preserves the semantics of the module system. Before describing it in detail in Section 3, we provide a high-level explanation here. If we translate an export statement to a getter property in a module instance object rather than an assignment, the value of the exported name would be always in synch with the value of the original variable. However, maintaining only a module instance object may lose connections to variables declared by `eval` function calls; it will refer to the new variable declared by `eval` calls rather than the variable declared in an enclosing scope. Instead, if we translate an export statement to a pair of an exported name and its enclosing environment record, either a declarative environment record for a module scope or an object environment record for a module instance object, and we translate any references to the exported name to references to the name in the environment record, we always get the correct value.

Analogous problem arises with import statements and we translate the import statements similarly to the export statements. For example in Figure 9, if we translate an import statement to a property access in a module instance object, the second reference to `foo` in the body of the module `Bar` will refer to the new variable declared by the module `Foo` created by the `eval` function call. Instead, we translate an import statement to a pair of an imported name and the module instance object that includes the imported entity, and we translate the references to the imported name to references to the property name in the module instance object, which always results in the correct value.

2.2 Name Conflict Resolution

While one of the main goals of the module system is to avoid name conflicts, the Harmony proposal does not specify a name resolution mechanism precisely. A program may have various kinds of name conflicts between declarations in a program or a module: function declarations, variable declarations, module declarations, export declarations, and import declarations. The Harmony proposal describes name conflict resolution between export declarations and between import declarations, but it does not describe a name conflict resolution between other kinds of declarations. To make things more complicated, the binding and evaluation order of declarations in a JavaScript program is different from their textual order.

```
foo(); // "foo";
bar; // undefined;
function foo() { return "bar"; }
function foo() { return "foo"; }
var foo, bar = "foo", bar = "bar";
function bar() { return 42; }
bar; // "bar";
```

Figure 10. Name conflicts between functions and variables

```
function foo() { return "bar"; }
function foo() { return "foo"; }
function bar() { return 42; }
var foo, bar, bar;
foo(); // "foo";
bar; // undefined;
bar = "foo";
bar = "bar";
bar; // "bar";
```

Figure 11. Same semantics for Figure 10

Before describing the name conflict resolution including module declarations, let us review the JavaScript semantics for evaluation of variable and function declarations again in more detail. As we discussed in Section 2.1, regardless of the textual order of function and variable declarations in a single scope, all the function declarations are evaluated first, all the variable declarations are bound and initialized to `undefined` next, and the remaining statements including the assignments from the variable declarations with initialization expressions, if any, are evaluated in their textual order. When there are name conflicts between function declarations, the function declaration comes later in textual order overrides the other function declaration. Name conflicts between variable declarations have no effects in terms of name binding, but if both declarations have initialization expressions they are evaluated in order during the evaluation of the remaining statements. Similarly, when a function declaration and a variable declaration declare the same name and the variable declaration has an initialization expression, the name is bound when the function declaration is evaluated and the evaluation of the assignment expression that is the initialization expression of the variable declaration overrides the function declaration.

For example, Figure 10 shows a JavaScript code with name conflicts between various declarations. According to the JavaScript semantics that we described so far, the code example has the same semantics with the code example in Figure 11. The second function declaration overrides the first function declaration with the same name; variables are declared right after evaluating function declarations and initialized to `undefined`. Note that variable declarations do not override function declarations.

Based on the JavaScript name resolution semantics, we would want to have an analogous and consistent semantics for name resolution between declarations including module declarations. However, we found that several options are incomparably reasonable. As with function declarations, we may want to evaluate module declarations earlier than other declarations. We may also want to evaluate import and export declarations before other declarations; we may want to evaluate module, import and export declarations in any order, or we may want to evaluate module declarations before import and export declarations. The proposal does not specify any evaluation order between them to break the name conflicts, and multiple resolution semantics seem to be all reasonable.

```
module Foo {
    export function toString() { return 42; }
}
module Bar {}
"" + {}; // "[object Object]";
"" + Foo; // "42";
"" + Bar; // TypeError
```

Figure 12. Modules with and without `toString`

In our formalization of the module system, we chose to let one kind of declarations may override other kinds of declarations: module declarations override the other kinds of declarations, import declarations override function and variable declarations, and function declarations override variable declarations. We made this design choice because we consider that module and import declarations are more important than local declarations, and a module declaration is more important than an import declaration which is simply an alias of a name in a module instance object. We do not argue that this design choice is the best option; rather, we provide our formalism as a starting point for any other researchers and developers to extend and modify according to their preferred design decisions.

2.3 Prototypes of Module Instance Objects

While every JavaScript object has the `Object` prototype object as the top of its prototype chain, the Harmony proposal makes an exception for module instance objects. The proposal specifies that:

> "A module instance object is a prototype-less object that provides read-only access to the exports of the module."

However, the unique characteristic of module instance objects may cause confusion when they are used with ordinary JavaScript objects, which does not satisfy one of the design goals of the Harmony module proposal: orthogonality from existing features.

Because `Object` provides default implementations for the `toString` and `valueOf` properties, every ordinary JavaScript object uses them unless some object in its prototype chain or the object itself provides its own implementations. For example, as Figure 12 shows, when we concatenate a string with an object, the object is implicitly converted to a string by calling the function bound to the `toString` property and concatenated to the given string. JavaScript developers use such implicit type conversions heavily without worrying about the existence of the `toString` property, but module instance objects now break such programming. As in Figure 12, concatenation of a string to a module may result in a string or the `TypeError` exception depending on the existence of the `toString` property in the module instance object.

Indeed, this feature is one of the sources to make differences between the module semantics of Traceur and TypeScript. While Traceur throws the `TypeError` exception for concatenating an empty string to the `Bar` module, TypeScript produces the `"[object Object]"` string.

2.4 `eval` Function

Though we used the `eval` function in our code examples so far, the Harmony proposal does not specify the interactions between the module system and the `eval` function clearly. While the proposal states that:

> "Reflective evaluation, via `eval` or the module loading API starts a new compilation and linking phase for the dynamically evaluated code."

it does not specify the semantics of module declarations, export declarations, and import declarations when they are evaluated by the `eval` function, which is not straightforward. For example, whether the evaluation of an export declaration by the `eval` function affects the enclosing scope is debatable because it may require changes in the read-only module instance object that is already *sealed*. Since the `[[Extensible]]` internal property of sealed objects have the `false` value, we cannot add new properties to sealed objects. Also, since every property of sealed objects has the `false` value for its `[[Configurable]]` internal property, we cannot delete or modify the property. Therefore, prohibiting the evaluation of export and import declarations by the `eval` function from affecting the enclosing scope may be a reasonable option.

In our formalization of the module system, we chose to evaluate module, import, and export declarations by the `eval` function only when the function is called either in the global scope or in a module scope. Because module declarations can appear only at the top level of a program or a module body, we believe this design choice is consistent with the module proposal. If the `eval` function is called from a scope that is not the global scope or a module scope, evaluation of the function ignores any module, import, and export declarations in the input string.

3. Formalization of JavaScript Module System

In this section, we formally specify the syntax and semantics of the JavaScript module system. Due to space limitations, we describe only the central parts of the formal specification in this paper and we refer the interested readers to our companion report [1].

3.1 JavaScript Module System

Our formalization of the JavaScript module system is based on our previous work [17] which translates JavaScript programs extended with the module system to λ_{JS} programs. The formalization in this paper is more flexible and adaptable in that it can address several design choices as we discussed in Section 2, and because it describes the JavaScript module semantics in terms of plain JavaScript instead of a very different core calculus λ_{JS}, it is more easily understandable and practical. At the same time, because the formalization in this paper is built on top of the previous work, we could easily build and check the validity of the formalization.

Before describing the formalization of our JavaScript module system, let us review the Harmony module proposal. The Harmony proposal aims to provide a simple and usable module system to avoid the need for global names in JavaScript. Module declarations can appear only at the top level of a program or a module body. When a JavaScript engine evaluates a program with modules, it first statically constructs a module environment which holds all the names in the global object and modules, and all the import and export relations. Then, it evaluates module, import, function, and variable declarations, and statements in order. When it evaluates module declarations, it first instantiates every module by constructing a module scope and a module instance object. Each module declaration introduces a new scope called a *module scope*, and its module body is evaluated in the module scope. An evaluated module called a *module instance object* contains lexically encapsulated members and exported bindings. An evaluation of a module results in a JavaScript object that reflects the exported bindings of a module instance called a *module instance object*. After instantiating the modules, a JavaScript engine makes all the module instance objects read only and initializes all the module bindings. Initializing a module binding is to evaluate the statements in the module body in its module scope. For mutually recursive import statements, function, variable, and nested module declarations are evaluated in the module scope, and the getters for the exported names are set in the module instance object. Finally, it seals all the module instance ob-

Program	::=	*SourceElement* $*$
SourceElement	::=	*Statement*
	|	*VariableDeclaration*
	|	*FunctionDeclaration*
	|	*ImportDeclaration*
	|	*ModuleDeclaration*
ModuleDeclaration	::=	`module` *Identifier* { *ModuleBody* }
ModuleBody	::=	*ModuleElement* $*$
ModuleElement	::=	*SourceElement*
	|	*ExportDeclaration*
ExportDeclaration	::=	`export` *ExportSpecifierSet*
		(, *ExportSpecifierSet*) $*$;
	|	`export` *VariableDeclaration*
	|	`export` *FunctionDeclaration*
	|	`export get` *Identifier*()
		{ *FunctionBody* }
	|	`export set` *Identifier*(*Identifier*)
		{ *FunctionBody* }
ExportSpecifierSet	::=	{ *ExportSpecifier*
		(, *ExportSpecifier*) $*$ }
		(`from` *Path*)?
	|	*Identifier* (`from` *Path*)?
	|	$*$ (`from` *Path*)?
ExportSpecifier	::=	*Identifier* (: *Path*)?
Path	::=	*Identifier*(.*Identifier*) $*$
ImportDeclaration	::=	`import` *ImportClause*
		(, *ImportClause*) $*$;
ImportClause	::=	*Path* `as` *Identifier*
	|	*ImportSpecifierSet* `from` *Path*
ImportSpecifierSet	::=	{ *ImportSpecifier*
		(, *ImportSpecifier*) $*$ }
	|	*Identifier*
ImportSpecifier	::=	*Identifier* (: *Identifier*)?

Figure 13. Syntax of the module system

jects to make them read only and substitutes each imported name with its canonical name of which the imported name is an alias.

Our formalization builds on top of SAFE [18], which provides both a formal specification and implementation of JavaScript. We describe the module system using the Intermediate Representation (IR) syntax of SAFE; SAFE formally specifies compilation steps from plain JavaScript Abstract Syntax Tree (AST) to IR, and dynamic semantics of IR. We extend the syntax of JavaScript with module support as presented in Figure 13 and environments to look up names as in Figure 14.

The translation technique uses the main idea of the module pattern as shown in Figure 15. Because any function in JavaScript may be used as a constructor with the `new` statement, we translate a module declaration to a `new` statement with a function as a constructor. Then, we achieve the module scope by the function scope, and we achieve the module instance object by the newly created object bound to `this`. First, the translator instantiates modules as Figure 16 describes. To come back to the function scope for ini-

$x \in \mathit{Identifier}$	Identifier
$\phi ::= \epsilon \mid x(.\,x)*$	Path
$\varphi_i ::= x(.\,x) * .\,(x)$	Internal qualified name
$\varphi_e ::= x(.\,x)*$	External qualified name
$\varphi ::= \varphi_i \mid \varphi_e$	Qualified name
$\overline{\varphi} ::= \varphi.*$	Expanded qualified name
$\tau ::= \texttt{var} \mid \texttt{module} \mid \texttt{local}$	Type
$\Sigma ::= \varphi \to (\tau, \varphi)$	Environment

Figure 14. Module environment

$$\frac{\begin{array}{c} p = s_1 \cdots s_n \\ \{i_1, \cdots, i_k\} = \{i | 1 \le i \le n \wedge s_i \in \mathit{ModuleDeclaration}\} \\ (H_0, A_0, ct_0) = (H, A, \texttt{Normal}(\texttt{empty})) \\ (H_{j-1}, A_{j-1}, tb, \Sigma, \phi), s_{i_j} \to_{\mathit{inst}} (H_j, A_j), ct_j \quad \forall 1 \le j \le k \\ (H'_0, A'_0, ct'_0) = (H_k, A_k, ct_k) \\ (H'_{j-1}, A'_{j-1}, tb, \Sigma, \phi), s_{i_j} \to_{\mathit{seal}} (H'_j, A'_j), ct'_j \quad \forall 1 \le j \le k \\ (H''_0, A''_0, ct''_0) = (H'_k, A'_k, ct'_k) \\ (H''_{j-1}, A''_{j-1}, tb, \Sigma, \phi), s_{i_j} \to_{\mathit{init}} (H''_j, A''_j), ct''_j \quad \forall 1 \le j \le k \end{array}}{(H, A, tb, \Sigma, \phi), p \to_{\mathit{mod}} (H''_k, A''_k), ct''_k}$$

Figure 15. Module compilation

$$\frac{\begin{array}{c} \phi' = \begin{cases} M & \text{if } \phi = \epsilon \\ \phi.\,M & \text{otherwise} \end{cases} \\ \mathit{CreateBinding}(H, A, M, \texttt{eval}) = (H', A') \\ \mathit{SetBinding}(H', A', M, \texttt{undefined}, \texttt{strict}) = (H'', A'', v) \\ l = \mathit{NewLoc}() \qquad H_0 = H''[l \mapsto \mathit{NewModInstObj}()] \\ A_0 = \{\} :: A'' \\ (H_0, A_0, tb, \Sigma, \phi'), s_1 \cdots s_n \to_f (H_1, A_1), ct_1 \\ (H_1, A_1, tb, \Sigma, \phi'), s_1 \cdots s_n \to_v (H_2, A_2), ct_2 \\ (H_2, A_2, tb, \Sigma, \phi'), s_1 \cdots s_n \to_{\mathit{imp}} (H_3, A_3), ct_3 \\ (H_3, A_3, tb, \Sigma, \phi'), s_1 \cdots s_n \to_{\mathit{inst}} (H_4, A_4), ct_4 \\ (H_4, A_4, l, \Sigma, \phi'), s_1 \cdots s_n \to_{\mathit{exp}} (H_5, A_5), ct_5 \\ H_6 = H_5[l \mapsto H_5(l)[[\texttt{Scope}]] \mapsto A_5]] \\ \mathit{PutValue}(H_5, A'', M, H_5(l), \texttt{strict}) = (H^\circ, A^\circ, v) \end{array}}{\begin{array}{l} (H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \\ \quad \to_{\mathit{inst}} (H^\circ, A^\circ), \texttt{Normal}(v) \end{array}}$$

Figure 16. Module instantiation

tialization from outside after instantiation, the function also sets a temporary closure for initialization in the global context with a random fresh name. Then, as Figure 17 describes, `Object.seal` seals the module instance objects, and calling the closure initializes the module by evaluating the statements of the module body the function scope as Figure 18 describes. Finally, the translator deletes the closures from the global context.

The significant difference from Kang *et al.* is that JavaScript with module is translated to JavaScript itself instead of λ_{JS}. Additionally, the translation is source-to-source translation preserving the name space with limited support for `eval` function. Since the translation is source-to-source translation, it is able to utilize

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{Object.seal}(M); \rightarrow_s (H_0, A_0), ct_0 \quad (H, A, tb, \Sigma, \phi), M \rightarrow_e l \quad (H_{i-1}, H(l).\texttt{[[Scope]]}, tb, \Sigma, \phi), s_i \rightarrow_{seal} (H_i, A_i), ct_i}{(H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{seal} (H_n, A_n), ct_n}$$

Figure 17. Module instance object sealing

$$\frac{(H, A, tb, \Sigma, \phi), M \rightarrow_e l \quad (H, H(l).\texttt{[[Scope]]}, tb, \Sigma, \phi), s_1 \cdots s_n \rightarrow_l (H_0, A_0), ct_0 \quad (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), s_i \rightarrow_{init} (H_i, A_i), ct_i \quad \forall 1 \le i \le n}{(H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{init} (H_n, A_n), ct_n}$$

Figure 18. Module initialization

$$\frac{\mathit{CreateBinding}(H, A, x, \texttt{eval}) = (H', A') \quad \mathit{SetBinding}(H', A', x, \epsilon, \texttt{strict}) = (H'', A'', err)}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p; \rightarrow_{imp} (H'', A''), \texttt{Throw}(err)}$$

$$\frac{\mathit{CreateBinding}(H, A, x, \texttt{eval}) = (H', A') \quad \mathit{SetBinding}(H', A', x, \epsilon, \texttt{strict}) = (H'', A'', v)}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p; \rightarrow_{imp} (H'', A''), \texttt{Normal}(v)}$$

Figure 19. Import declaration

any current JavaScript engine with the module rewriter to interpret a program in JavaScript with module. For instance, programmers may develop their JavaScript program with module, and translate it to the current JavaScript program without module using the module rewriter so that any current JavaScript engine can run the program. JavaScript community tends to accept an advance of the specification slowly. There is a time gap of couple of years for them to accept new JavaScript specification. In the mean while, the module rewriter would be a substitution until JavaScript engines accept the module system. Besides, the module rewriter preserves the name space except during compiling modules. For the `eval` function calls after compiling modules, the semantics are preserved. However, the temporary helper functions have a chance of name conflict with new names that the `eval` function calls introduce during compiling modules. Though, by choosing long random strings of complicated characters as fresh names for the helper functions, for example, with non-Latin alphabets, we can reduce the possibility of name conflict.

3.2 Formal Specification of the Design Issues

Import and export declarations We defined an export to be an accessor property in module instance object of which getter evaluates the corresponding identifier expression or property access expression then returns the result and of which setter is undefined. By doing so, the binding which a property access to an module instance object had been resulted from might be shadowed by a new binding which created after `eval` function is called.

We defined an import to be a binding of an identifier and the corresponding identifier expression or property access expression, and its evaluation follows the evaluation of the expression. Similarly to the definition of export, the binding which evaluation of an import had been resulted from might be shadowed by a new binding which created after `eval` function is called.

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{get}\ x()\ \{\ \texttt{return}\ p;\ \} \rightarrow_m (H', y, ap) \quad \mathit{DefineOwnProperty}(H', A, tb, \mathit{ToString}(H', y), ap, \texttt{false}) = (H'', A'', err)}{(H, A, tb, \Sigma, \phi), \texttt{export}\ \{x\colon p\}; \rightarrow_{exp} (H'', A''), \texttt{Throw}(err)}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{get}\ x()\ \{\ \texttt{return}\ p;\ \} \rightarrow_m (H', y, ap) \quad \mathit{DefineOwnProperty}(H', A, tb, \mathit{ToString}(H', y), ap, \texttt{false}) = (H'', A'', v)}{(H, A, tb, \Sigma, \phi), \texttt{export}\ \{x\colon p\}; \rightarrow_{exp} (H'', A''), \texttt{Normal}(v)}$$

$$\frac{\begin{array}{c}\mathit{lookup}[\![p]\!](\Sigma; \phi) = (\texttt{module}, \varphi) \\ \phi' = \begin{cases} M & \text{if } \varphi = M \\ \phi''.M & \text{if } \varphi = \phi''.(M) \end{cases} \\ \{x_1, \cdots, x_n\} = \{x \mid \phi'.(x) \in \Sigma\} \\ (H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{get}\ x_i()\ \{\ \texttt{return}\ p.x_i;\ \} \\ \quad \rightarrow_m (H'_i, y_i, ap_i) \\ \mathit{DefineOwnProperty}(H'_i, A_{i-1}, tb, \mathit{ToString}(H'_i, y_i), ap_i, \\ \quad \texttt{false}) = (H_i, A_i, v_i)\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export}\ *\ \texttt{from}\ p; \rightarrow_{exp} (H_n, A_n), \texttt{Normal}(v_n)}$$

Figure 20. Export declaration

$$\frac{\mathit{PutValue}(H, A, x', p.x, \texttt{strict}) = (H', A', err)}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p; \rightarrow_l (H', A'), \texttt{Throw}(err)}$$

$$\frac{\mathit{PutValue}(H, A, x', p.x, \texttt{strict}) = (H', A', v)}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p; \rightarrow_l (H', A'), \texttt{Normal}(v)}$$

Figure 21. Linking

$$\frac{\mathit{Lookup}(H, A, x, \texttt{strict}) = (\sigma, p) \quad (H, A, tb, \Sigma, \phi), p \rightarrow_e ve}{(H, A, tb, \Sigma, \phi), x \rightarrow_e ve}$$

Figure 22. Evaluation of imported names

Handling name conflicts In Figure 16, top-level declarations are hoisted in the order of function declaration, variable declaration, import declaration, module declaration, and export declaration, as aforementioned our design choice. Note that export declaration is independent from the other top-level declarations. One may simply set the order of hoisting.

In Figures 19, 21, and 20, import declaration and export declaration override the current binding in the environment. If the last import declaration should override the others, we would evaluate the import declarations and the linking according to the order of the module body in Figure 16. If the first import declaration should override the others, we would evaluate them in the inverse order of the module body in Figure 16. Likewise, we can also set which export declaration should override the others.

Modules should be instantiated and initialized in the order of code, so we cannot simply invert the order. Rather, we should consider the two cases; whether the current binding has the value

$$\frac{\cdots \quad A = ER :: A' \quad M \notin Dom(M) \quad \cdots}{(H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{inst} \cdots}$$

$$\frac{\begin{array}{c}\cdots \quad A = ER :: A' \quad M \in Dom(M) \\ ER[M] \notin Object \quad \cdots\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{inst} \cdots}$$

$$\frac{\begin{array}{c}\cdots \quad A = ER :: A' \quad M \in Dom(M) \\ ER[M] \in Object \\ ER[M].[[\texttt{Class}]] \neq \texttt{"Module"} \quad \cdots\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{inst} \cdots}$$

$$\frac{\begin{array}{c}\cdots \quad A = ER :: A' \quad M \in Dom(M) \\ ER[M] \in Object \\ ER[M].[[\texttt{Class}]] = \texttt{"Module"} \quad \cdots\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{inst} \cdots}$$

Figure 23. Current binding checking

$$NewModInstObj() = \begin{array}{l}\{[[\texttt{Class}]] : \texttt{"Module"}, \\ \ [[\texttt{Extensible}]] : \texttt{true}, \\ \ [[\texttt{Prototype}]] : \#Null, \\ \ [[\texttt{Scope}]] : \{\}, \\ \ \texttt{@property} : \{\}\}\end{array}$$

Figure 24. Prototype-less module instance object

$$NewModInstObj() = \begin{array}{l}\{[[\texttt{Class}]] : \texttt{"Module"}, \\ \ [[\texttt{Extensible}]] : \texttt{true}, \\ \ [[\texttt{Prototype}]] : \#ObjectPrototype, \\ \ [[\texttt{Scope}]] : \{\}, \\ \ \texttt{@property} : \{\}\}\end{array}$$

Figure 25. Module instance object with a prototype

of a module instance object, or not. It could be checked as in Figure 23.

Prototype of module instance object $NewModInstObj$ is a helper function which creates new module instance object with empty scope and empty property as placeholders. The value of `[[Class]]` internal property is set to `"Module"`, and the value of `[[Extensible]]` internal property is set to `true` until the object is sealed. The value of `[[Prototype]]` internal property is set to the null location if module instance object should not have prototype like in Figure 24, or it is set to the location of the Object prototype object if module instance object should have the Object prototype object as prototype like in Figure 25.

eval function The semantics of `eval` function for modules are in the figure 26. If the current context is either in the global scope or in a module scope, it updates the module environment and evaluates the code. Otherwise, it removes module declarations, import declarations, and export declarations from the code, then evaluates the new code.

3.3 Validity Properties of JavaScript Module System

We want to show two properties: no free variable causes an error, and module scope is isolated. In this paper, we only provide proof sketch.

$$\frac{\begin{array}{c}(H, A, tb, \Sigma, \phi), y \rightarrow_e v \quad v = \#GlobalEval \\ (H, A, tb, \Sigma, \phi), z_1 \rightarrow_e v_1 \quad (H, A, tb, \Sigma, \phi), z_2 \rightarrow_e v_2 \\ v_1 \in Loc \quad v_3 = ToString(H, v_2) \\ A \neq \#Global \quad \nexists l.H(l) \in Object \wedge \\ H(l).[[\texttt{Class}]] = \texttt{"Module"} \wedge H(l).[[\texttt{Scope}]] = A \\ v_4 = RemoveModule(v_3) \\ (H, A, tb, \Sigma', \phi), x = \texttt{eval}(v_1, v_4) \rightarrow_{eval} (H', A'), ct\end{array}}{(H, A, tb, \Sigma, \phi), x = y(z_1, z_2) \rightarrow_s (H', A'), ct}$$

$$\frac{\begin{array}{c}(H, A, tb, \Sigma, \phi), y \rightarrow_e v \quad v = \#GlobalEval \\ (H, A, tb, \Sigma, \phi), z_1 \rightarrow_e v_1 \quad (H, A, tb, \Sigma, \phi), z_2 \rightarrow_e v_2 \\ v_1 \in Loc \quad v_3 = ToString(H, v_2) \\ A = \#Global \vee \exists l.H(l) \in Object \wedge \\ H(l).[[\texttt{Class}]] = \texttt{"Module"} \wedge H(l).[[\texttt{Scope}]] = A \\ \Sigma' = Env[\![v_3]\!](\Sigma, \phi) \\ (H, A, tb, \Sigma', \phi), x = \texttt{eval}(v_1, v_3) \rightarrow_{eval} (H', A'), ct\end{array}}{(H, A, tb, \Sigma, \phi), x = y(z_1, z_2) \rightarrow_s (H', A'), ct}$$

Figure 26. Evaluation of the `eval` function

Property 1 (Absent Binding)**.** *A valid program p successfully compiled does not cause an error due to an absent binding.*

$$\nexists x.\exists H, A, E.p \rightarrow^* (H, A), E\langle x\rangle$$

Proof sketch. Let $FV(s)$ denote the set of the free variables in the statement s. Show that evaluation of statement does not add new free variable, and show that application of execution context does not add new free variable:

$$\forall H, A, H', A'.(H, A), s \rightarrow (H', A'), s' \Rightarrow FV(s) \subseteq FV(s')$$

$$FV(s) \subseteq FV(E\langle s\rangle)$$

Now, assume that an absent binding causes an error:

$$\exists x.\exists H, A, E.p \rightarrow^* (H, A), E\langle x\rangle$$

Because p is a valid program, $FV(p) = \emptyset$. Then, with the lemmas

$$\emptyset = FV(p) \supseteq FV(E\langle x\rangle) \supseteq FV(x) \supseteq x$$

which is a contradiction. Therefore,

$$\nexists x.\exists H, A, E.p \rightarrow^* (H, A), E\langle x\rangle$$

□

Property 2 (Isolation of module scope)**.** *A variable declared in module scope does not affect outside the module.*

$$\begin{array}{c}M \notin \text{dom}(\text{Env}[\![p]\!]) \wedge \\ \text{valid}[\![\texttt{module}\ M\ \{\ \texttt{var}\ x{=}v_1;\ \}\ p]\!] \wedge \\ \text{valid}[\![\texttt{module}\ M\ \{\ \texttt{var}\ x{=}v_2;\ \}\ p]\!] \wedge \\ \texttt{module}\ M\ \{\ \texttt{var}\ x{=}v_1;\ \}\ p \rightarrow_p (H', A'), \text{ct}_1 \wedge \\ \texttt{module}\ M\ \{\ \texttt{var}\ x{=}v_2;\ \}\ p \rightarrow_p (H'', A''), \text{ct}_2 \\ \Rightarrow \text{ct}_1 = \text{ct}_2\end{array}$$

4. Implementation of JavaScript Module System

We implemented the JavaScript module system on top of SAFE [18] as a source-to-source translator and made it publicly available:

`http://plrg.kaist.ac.kr/redmine/projects/jsf/repository`

Figure 27 illustrates a high-level architecture of the important parts of SAFE. The architecture accepts a JavaScript program, translates it through Abstract Syntax Tree (AST), Intermediate Representation (IR), and Context Flow Graph (CFG), each of which is utilized

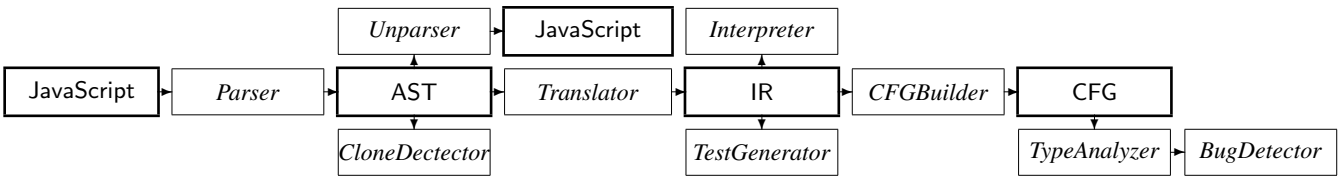

Figure 27. SAFE architecture

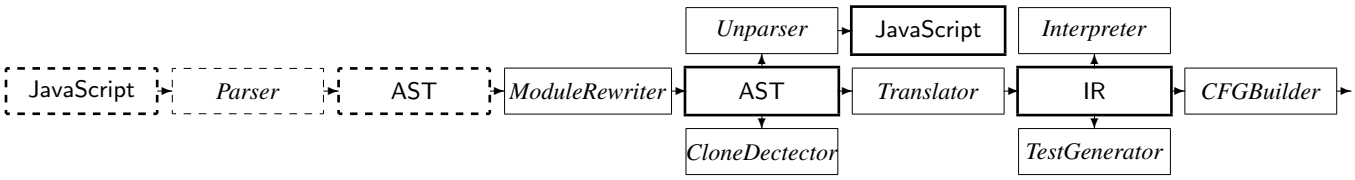

Figure 28. SAFE architecture with module support

by various analyses and tools. For example, AST suites the best for clone detection and pretty printing the original JavaScript program; IR works best for evaluation and test data generation; and CFG works best for type-based analysis and bug detection among others. For presentation brevity, we omit other features of SAFE such as Web IDL [31] supports. Figure 28 presents the extend architecture of SAFE with support for the module system. The dotted boxes denote extended parts to support the module system, and the *ModuleRewriter* phase translates an AST that may include module syntax into another AST that does not include any module syntax. Due to space limitations, we omit the details of *ModuleRewriter*, but discuss its properties and limitations.

Our implementation of the module system serves several roles. First, JavaScript developers can use the module system today before the official JavaScript.next is available. They can develop a large JavaScript program using modules, desugar the module syntax away and pretty print it to a plain JavaScript program, and deploy the plain JavaScript program that can be executed in any today's JavaScript engines. Second, JavaScript engine developers can evaluate their implementations and ours to articulate and discuss ambiguous semantics from the Harmony proposal. Because our implementation is open sourced, the developers can apply our technique to their engines. Or, they can even use our *ModuleRewriter* as a preprocessor to support the module system as a sort of macros.

The translation mechanism is based on the module pattern: it uses a function scope to represent a module scope and the function returns an object as a module instance object with exported names as its properties. Because translation of a module consists of instantiation and initialization, the control should reenter the function scope at initialization, which is not available with JavaScript objects. Therefore, our implementation creates a temporary object in the global scope which has temporary functions created from the module scope at instantiation time, and it calls the temporary functions to initialize the modules at initialization time. We implement import statements by replacing the references to imported names with the corresponding property accesses to the module instance objects. Figure 29 presents a translated code of the example in Figure 2 in plain JavaScript by *ModuleRewriter*.

The limitation of *ModuleRewriter* is due to the quirky dynamic semantics of JavaScript: the `eval` function and the `with` statement. First, *ModuleRewriter* does not handle the references to imported names in a string passed to the `eval` function as an argument. Because *ModuleRewriter* statically collects the references to imported names and replaces them to corresponding property accesses to module instance objects at compile time, it should be able to collect the references to imported names from a string at compile time. Secondly, the `with` statement takes an object, called a *with object*, and introduces the properties of the with object to the environment at run time, which basically supports dynamic scoping. The properties of the with object shadow any bindings with the same name in the enclosing context. Therefore, a reference to an identifier may evaluate to an entity defined in the enclosing context or to a property in the with object. Because it is not possible to determine whether an identifier refers to an imported name at compile time, it is not possible to replace identifiers in the `with` statement. Thanks to Park *et al.* [21], we can desugar most of the `with` statements at compile time; for example, we can desugar the following `with` statement:

```
with (o) { x = 42; }
```

to the following code without using the `with` statement

```
("x" in o ? o.x : x) = 42;
```

preserving the semantics. Indeed, the SAFE architecture includes the `with` rewriter implementation by default. Therefore, *ModuleRewriter* can handle the `with` statements unless they call the `eval` function.

5. Related Work

The ECMAScript Harmony proposal [2] describes the new features adopted in the next release of ECMAScript such as modules, classes, generators, and block scoped bindings. Most of the proposal have already been incorporated into the ECMAScript 6 language specification draft, but the module proposal [11] is not yet included. The module proposal provides a brief, high-level description of the JavaScript module system but the informal specification in prose does not explain the module semantics precisely.

Kang and Ryu [17] designed a formal specification and implementation of a JavaScript module system based on the Harmony proposal. They formalized and implemented their module system using λ_{JS} [10], which is a core-calculus of ECMAScript 3. While λ_{JS} provides a small set of conventional formal semantics and translation rules from JavaScript to λ_{JS}, its implementation is impractically slow. They described the formal semantics of the module system as a set of translation rules from JavaScript with modules to λ_{JS}, and they implemented the module system by extending the translation rules. While the formalization and implementa-

```
var __initarg = {},
    __initfun = {},
    __extmod = {},
    __intmod = {},
    __Object = Object,
    Foo, foo;
Foo = __extmod.Foo = new (function(arguments) {
  function inc() { foo ++; }
  var inc, foo;
  __intmod.Foo = {
      get inc() { return inc; },
      get foo() { return foo; }
  };
  (function (__this) {
    var __desc =
      __Object.getOwnPropertyDescriptor(__this, "foo");
    delete __this.foo;
    if(typeof __desc == "undefined")
      __desc = { configurable : true };
    __desc.get = (function foo() {
      return __intmod.Foo.foo;
    });
    __Object.defineProperty(__this, "foo", __desc);
  })(this);
  (function (__this) {
    var __desc =
      __Object.getOwnPropertyDescriptor(__this, "inc");
    delete __this.inc;
    if(typeof __desc == "undefined")
      __desc = { configurable : true };
    __desc.get = (function inc() {
      return __intmod.Foo.inc;
    });
    __Object.defineProperty(__this, "inc", __desc);
  })(this);
  __initfun.Foo = (function(arguments) {
      foo = 42;
  });
  __initarg.Foo = {
      get arguments() { return arguments; },
      set arguments(x) { arguments = x; }
  };
})({
  get arguments() { return arguments; },
  set arguments(x) { arguments = x; }
   });
__Object.seal(Foo);
__initfun.Foo.call(this, __initarg.Foo);
Foo.inc();
__intmod.Foo.foo;
```

Figure 29. Translation of Figure 2 by *ModuleRewriter*

tion are sold and rigorous, the practicality of their approach is not clear. Because the semantics of λ_{JS} is very much different from the original JavaScript, the formal semantics does not help understand the behavior of JavaScript programs much. Because the generated λ_{JS} programs are huge, executing λ_{JS} programs has been absurdly slow. On the contrary, our formalization is similarly rigorous but much easier to understand because it is a source-to-source translation, and our implementation is much more efficient and practical.

Traceur [7] is a compiler developed by Google that compiles JavaScript.next code into one in ECMAScript 5. While their implementation is in more than 20,000 lines of JavaScript, they do not provide any formal specification or a detailed description of the system. As we have seen in this paper, the module implementation in Traceur does not satisfy the semantics in the Harmony module proposal in many cases.

TypeScript is a superset of JavaScript that supports a type system for static checking and verification. It also provides some features of JavaScript.next including the module system, and it compiles a TypeScript program into a plain JavaScript program. Unlike Traceur, supporting JavaScript.next closely today is not one of the main goals of TypeScript. It focuses more on type inference and type annotation for better static checking and analysis. Thus, its module support is more premature than Traceur, often leads to different semantics from the Harmony module proposal. It supports somewhat different module syntax and it does not support mutually recursive imports, for example.

While both Traceur and TypeScript support very premature module systems with different semantics from the Harmony proposal without any detailed description of them, our module system is formally specified and implemented preserving the semantics of the Harmony proposal. Our implementation is built on top of SAFE, a scalable analysis framework for JavaScript, which provides various facilities for JavaScript program manipulation. Because the implementation of the SAFE architecture uses tools to automatically generate AST nodes by ASTGen [26] and parsers by Rats! [8], it is easily adaptable to any syntax changes and extensions. The various components of SAFE such as the unparser to pretty print plain JavaScript syntax, the interpreter to evaluate JavaScript programs, and the `with` rewriter to desugar the `with` statements away have greatly improved the quality and productivity of our implementation.

6. Conclusion

JavaScript is now one of the most widely used programming languages from small scripts embedded in web documents to large stand-along projects such as web applications and compilers. The lack of any language-level module system in JavaScript has caused many problems and an ad-hoc programming pattern, which finally introduce a language-level module system to JavaScript in the next release of the ECMAScript language specification. Even though the ECMAScript Harmony proposal for the next release includes a JavaScript module system, its current status is not yet ready to be incorporated into the language specification draft. Its brief, informal description of the module system does not specify the module semantics precisely, which often leads to ambiguous or unclear interactions with existing language features. Thus, even Traceur from Google and TypeScript from Microsoft do not support the module system semantics correctly yet.

In this paper, we investigate and describe design issues with the JavaScript module system with concrete code examples. We explain each issue using the results from Traceur and TypeScript and discuss reasonable design decisions for the issues. We present a formal specification of the module system as a source-to-source transformation from JavaScript with modules to plain JavaScript. We also provide an open-source implementation of the module system so that JavaScript developers can use the future module system in advance and JavaScript engine developers to take advantage of our implementation techniques. We believe that our discussion on the design issues of the module system and its formal specification and implementation will help the designers of the Harmony proposal and the developers of the proposal understand the module system more clearly. Because our module system is not particularly tied to JavaScript, we expect that it will be applicable to other scripting languages such as Python and Ruby.

References

[1] Junhee Cho and Sukyoung Ryu. Javascript module system: Exploring the design space (extended with proofs). `http://plrg.kaist.ac.kr/research/publications`.

[2] ECMA. Harmony proposals. http://wiki.ecmascript.org/doku.php?id=harmony:proposals.

[3] ECMA. Harmony proposals – draft specification for ES.next (Ecma-262 Edition 6). http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts.

[4] ECMA. *ECMA-262: ECMAScript Language Specification*. 5th edition, December 2009.

[5] Dojo Foundation. Dojo toolkit. http://dojotoolkit.org.

[6] Linux Foundation. Tizen: an open source, standards-based software platform for multiple device categories. https://www.tizen.org.

[7] Google. Traceur-compiler: Google's vehicle for JavaScript language design experimentation. https://code.google.com/p/traceur-compiler/.

[8] Robert Grimm. Rats! – An Easily Extensible Parser Generator. http://cs.nyu.edu/~rgrimm/xtc/rats.html.

[9] Salvatore Guarnieri, Marco Pistoia, Omer Tripp, Julian Dolby, Stephen Teilhet, and Ryan Berg. Saving the world wide web from vulnerable JavaScript. In *Proceedings of the 20th Internaional Symposium on Software Testing and Analysis*, 2011.

[10] Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. The essence of Javascript. In *Proceedings of the 24th European Conference on Object-Oriented Programming*, 2010.

[11] Dave Herman and Sam Tobin-Hochstadt. Harmony proposals – modules. http://wiki.ecmascript.org/doku.php?id=harmony:modules.

[12] Yahoo! Inc. YUI library. http://yuilibrary.com.

[13] Alexa Internet. Alexa. http://www.alexa.com.

[14] Joynet. Node.js. http://nodejs.org/.

[15] jQuery Foundation. jQuery. http://jquery.com/.

[16] jQuery Foundation. jQuery.noConflict(). http://api.jquery.com/jQuery.noConflict/.

[17] Seonghoon Kang and Sukyoung Ryu. Formal specification of a JavaScript module system. In *Proceedings of the 2012 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2012.

[18] Hongki Lee, Sooncheol Won, Joonho Jin, Junhee Cho, and Sukyoung Ryu. SAFE: Formal specification and implementation of a scalable analysis framework for ECMAScript. In *Proceedings of the 19th International Workshop on Foundations of Object-Oriented Languages*, 2012.

[19] Microsoft. Typescript. http://www.typescriptlang.org.

[20] Eric Miraglia. A javascript module pattern. http://www.yuiblog.com/blog/2007/06/12/module-pattern/.

[21] Changhee Park, Hongki Lee, and Sukyoung Ryu. All about the `with` statement in JavaScript: Removing `with` statements in JavaScript applications. In *Proceedings of the Dynamic Language Symposium 2013*, 2013.

[22] Paruj Ratanaworabhan, Benjamin Livshits, David Simmons, and Benjamin G. Zorn. JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications. In *Proceedings of the USENIX 2010 Conference on Web Application Development*, 2010.

[23] Gregor Richards, Christian Hammer, Brian Burg, and Jan Vitek. The eval that men do: a large-scale study of the use of eval in JavaScript applications. In *Proceedings of the 25th European Conference on Object-Oriented Programming*, 2011.

[24] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. An analysis of the dynamic behavior of JavaScript programs. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2010.

[25] Samsung Electronics. Samsung Smart TV. http://developer.samsung.com/smarttv.

[26] Brian R. Stoler, Eric Allen, and Dan Smith. ASTGen. http://sourceforge.net/projects/astgen.

[27] Prototype Core Team. Prototype. http://prototypejs.org/.

[28] The MooTools Dev Team. The dollar safe mode. http://mootools.net/blog/2009/06/22/the-dollar-safe-mode/.

[29] The MooTools Dev Team. MooTools: a compact JavaScript framework. http://mootools.net.

[30] W3C. Document object model (dom). http://www.w3.org/DOM/.

[31] W3C. Web IDL. http://www.w3.org/TR/WebIDL.

A. Formalization of JavaScript Module System

A.1 Preliminaries

Definition 1. *Given $x_0 \in X$ and $y_0 \in Y$, let $f = [x_0 \mapsto y_0]$ denote a partial function $f : X \to Y$ from X to Y defined as:*

$$f(x) = y_0 \quad \textit{if } x = x_0$$

Definition 2. *Let $f : X \to Y$ be a partial function from X to Y. Then, let $g = f[x_0 \mapsto y_0]$ denote a partial function $g : X \to Y$ from X to Y defined as:*

$$g(x) = \begin{cases} y_0 & \textit{if } x = x_0 \\ f(x) & \textit{if } x \neq x_0 \wedge x \in \mathrm{Dom}(f) \end{cases}$$

Definition 3. *Let $f : X \to Y$ be a partial function from X to Y. Then, let $g = f \setminus x_0$ denote a partial function $g : X \to Y$ from X to Y defined as:*

$$g(x) = f(x) \quad \textit{if } x \neq x_0 \wedge x \in \mathrm{Dom}(f)$$

A.2 Syntax

Program	::=	*SourceElement* $*$
SourceElement	::=	*Statement*
	|	*VariableDeclaration*
	|	*FunctionDeclaration*
	|	*ImportDeclaration*
	|	*ModuleDeclaration*
ModuleDeclaration	::=	`module` *[NoNewLine]* *Identifier* { *ModuleBody* }
ModuleBody	::=	*ModuleElement* $*$
ModuleElement	::=	*SourceElement*
	|	*ExportDeclaration*
ExportDeclaration	::=	`export` *ExportSpecifierSet*(, *ExportSpecifierSet*) $*$;
	|	`export` *VariableDeclaration*
	|	`export` *FunctionDeclaration*
	|	`export get` *Identifier*() { *FunctionBody* }
	|	`export set` *Identifier*(*Identifier*) { *FunctionBody* }
ExportSpecifierSet	::=	{ *ExportSpecifier*(, *ExportSpecifier*) $*$ } (`from` *Path*)?
	|	*Identifier* (`from` *Path*)?
	|	$*$ (`from` *Path*)?
ExportSpecifier	::=	*Identifier* (: *Path*)?
Path	::=	*Identifier*(.*Identifier*) $*$
ImportDeclaration	::=	`import` *ImportClause*(, *ImportClause*) $*$;
ImportClause	::=	*Path* `as` *Identifier* module renaming as *Identifier*
	|	*ImportSpecifierSet* `from` *Path* import from a module *Path*
ImportSpecifierSet	::=	{ *ImportSpecifier*(, *ImportSpecifier*) $*$ }
	|	*Identifier*
ImportSpecifier	::=	*Identifier* (: *Identifier*)?

A.3 Environment

A.3.1 Definition of module environment

Identifier x	$\in$	*Identifier*
Path ϕ	::=	$\epsilon \mid x(.\,x) *$
Internal qualified name φ_i	::=	$x(.\,x) * .\,(x)$
External qualified name φ_e	::=	$x(.\,x) *$
Qualified name φ	::=	$\varphi_i \mid \varphi_e$
Expanded qualified name $\overline{\varphi}$	::=	$\varphi.*$

An internal qualified name is visible only in the module scope and the nested module scopes. An external qualified name is visible everywhere. A path is either an empty path ϵ or a sequence of identifiers separated by dot.

$$\begin{array}{rcl} \text{Type } \tau & ::= & \texttt{var} \mid \texttt{module} \mid \texttt{local} \\ \text{Environment } \Sigma & ::= & \varphi \to (\tau, \varphi) \end{array}$$

An environment is a partial function from qualified names to pairs of a type and a qualified name. Given $\Sigma(\varphi_1) = (\tau, \varphi_2)$, φ_1 is an alias of φ_2. τ represents whether the name is of a funtion, a variable, or a module declared in the scope.

A.3.2 Static resolution of module environment

Module environment constructor

$$\mathit{Env}[\![\texttt{function } f(x_1, \cdots, x_n) \{s_1 \cdots s_m\}]\!](\Sigma; \phi) = \mathit{Env}[\![\texttt{var } f]\!](\Sigma; \phi)$$

$$\mathit{Env}[\![\texttt{var } x_1(= e_1), \cdots, x_n(= e_n);]\!](\Sigma; \phi) = \begin{cases} [x \mapsto (\texttt{var}, x) \mid x \in \{x_1, \cdots, x_n\}] & \text{if } \phi = \epsilon \\ [\phi.(x) \mapsto (\texttt{var}, \phi.(x)) \mid x \in \{x_1, \cdots, x_n\}] & \text{otherwise} \end{cases}$$

$$\mathit{Env}[\![\texttt{module } M \{s_1 \cdots s_n\}]\!](\Sigma; \phi) = \begin{cases} [M \mapsto (\texttt{module}, M)] \cup \mathit{Env}[\![s_1 \cdots s_n]\!](\Sigma; M) & \text{if } \phi = \epsilon \\ [\phi.(M) \mapsto (\texttt{module}, \phi.(M))] \cup \mathit{Env}[\![s_1 \cdots s_n]\!](\Sigma; \phi.M) & \text{otherwise} \end{cases}$$

$$\mathit{Env}[\![\texttt{export } \mathit{ess}_1, \cdots, \mathit{ess}_n;]\!](\Sigma; \phi) = \bigcup_{1 \le i \le n} \mathit{Env}[\![\texttt{export } \mathit{ess}_i]\!](\Sigma; \phi)$$

$$\mathit{Env}[\![\texttt{export } \{es_1, \cdots, es_n\} (\texttt{from } p)]\!](\Sigma; \phi) = \bigcup_{1 \le i \le n} \mathit{Env}[\![\texttt{export } \{es_i\} (\texttt{from } p)]\!](\Sigma; \phi)$$

$$\mathit{Env}[\![\texttt{export } \{x\} (\texttt{from } p)]\!](\Sigma; \phi) = \mathit{Env}[\![\texttt{export } \{x\colon x\} (\texttt{from } p)]\!](\Sigma; \phi)$$

$$\mathit{Env}[\![\texttt{export } \{x\colon p\} \texttt{ from } p_0]\!](\Sigma; \phi) = \mathit{Env}[\![\texttt{export } \{x\colon p_0.p\}]\!](\Sigma; \phi)$$

$$\mathit{Env}[\![\texttt{export } \{x\colon x_0\}]\!](\Sigma; \phi) = \begin{cases} [\phi.x \mapsto (\tau, \phi.(x_0))] & \text{if } \Sigma(\phi.(x_0)) = (\tau, \phi.(x_0)) \quad \text{unimported} \\ [\phi.x \mapsto (\tau, \phi'.x')] & \text{if } \Sigma(\phi.(x_0)) = (\tau, \phi'.x') \quad \text{imported} \end{cases}$$

$$\mathit{Env}[\![\texttt{export } \{x\colon x_0\}]\!](\Sigma; \phi) = [\phi.x \mapsto \Sigma(\phi.(x_0))] \quad \text{if } \phi.(x_0) \in \mathrm{Dom}(\Sigma)$$

$$\mathit{Env}[\![\texttt{export } \{x\colon p.x_0\}]\!](\Sigma; \phi) = [\phi.x \mapsto (\tau, \varphi)] \quad \text{if } \mathit{lookup}[\![p.x_0]\!](\Sigma; \phi) = (\tau, \varphi)$$

$$\mathit{Env}[\![\texttt{export } x (\texttt{from } p)]\!](\Sigma; \phi) = \mathit{Env}[\![\texttt{export } \{x\} (\texttt{from } p)]\!](\Sigma; \phi)$$

$$\mathit{Env}[\![\texttt{export var } x_1(= e_1), \cdots, x_n(= e_n);]\!](\Sigma; \phi) = [\phi.(x) \mapsto (\texttt{var}, \phi.(x)), \phi.x \mapsto (\texttt{var}, \phi.(x)) \mid x \in \{x_1, \cdots, x_n\}]$$

$$\mathit{Env}[\![\texttt{export function } f(x_1, \cdots, x_n) \{s_1 \cdots s_m\}]\!](\Sigma; \phi) = \mathit{Env}[\![\texttt{export var } f]\!](\Sigma; \phi)$$

$$\mathit{Env}[\![\texttt{export get } f() \{s_1 \cdots s_m\}]\!](\Sigma; \phi) = [\phi.f \mapsto \phi.f]$$

$$\mathit{Env}[\![\texttt{export set } f(x)\ \{s_1 \cdots s_m\}]\!](\Sigma;\phi) = [\phi.f \mapsto \phi.f]$$

$$\mathit{Env}[\![\texttt{import } ic_1, \cdots, ic_n]\!](\Sigma;\phi) = \bigcup_{1\le i\le n} \mathit{Env}[\![\texttt{import } ic_i]\!](\Sigma;\phi)$$

$$\mathit{Env}[\![\texttt{import } p \texttt{ as } M]\!](\Sigma;\phi) = [\phi.(M) \mapsto (\texttt{module}, \varphi)] \quad \text{if } \mathit{lookup}[\![p]\!](\Sigma;\phi) = (\texttt{module}, \varphi)$$

$$\mathit{Env}[\![\texttt{import } \{is_1, \cdots, is_n\} \texttt{ from } p]\!](\Sigma;\phi) = \bigcup_{1\le i\le n} \mathit{Env}[\![\texttt{import } \{is_i\} \texttt{ from } p]\!](\Sigma;\phi)$$

$$\mathit{Env}[\![\texttt{import } \{x\} \texttt{ from } p]\!](\Sigma;\phi) = \mathit{Env}[\![\texttt{import } \{x\colon x\} \texttt{ from } p]\!](\Sigma;\phi)$$

$$\mathit{Env}[\![\texttt{import } \{x\colon x'\} \texttt{ from } p]\!](\Sigma;\phi) = [\phi.(x') \mapsto (\tau,\varphi)] \quad \text{if } \mathit{lookup}[\![p.x]\!](\Sigma;\phi) = (\tau,\varphi)$$

$$\mathit{Env}[\![\texttt{import } x \texttt{ from } p]\!](\Sigma;\phi) = \mathit{Env}[\![\texttt{import } \{x\} \texttt{ from } p]\!](\Sigma;\phi)$$

Lookup function

$$\mathit{lookup}[\![M.\phi]\!](\Sigma;\phi_0) = \begin{cases} \mathit{lookup}'[\![\phi]\!](\Sigma;\phi'.M') & \text{if } \Sigma(\phi_0.(M)) = (\texttt{module}, \phi'.(M')) \\ \mathit{lookup}_\perp[\![M.\phi]\!](\Sigma;\phi_0) & \text{otherwise} \end{cases}$$

$$\mathit{lookup}[\![x]\!](\Sigma;\phi_0) = \begin{cases} (\tau,\varphi) & \text{if } \Sigma(\phi_0.(x)) = (\tau,\varphi) \\ \mathit{lookup}_\perp[\![x]\!](\Sigma;\phi_0) & \text{otherwise} \end{cases}$$

$$\mathit{lookup}_\perp[\![\phi]\!] = \begin{cases} \mathit{lookup}[\![\phi]\!](\Sigma;\phi_0') & \text{if } \phi_0 = \phi_0'.M \\ \perp & \text{otherwise} \end{cases}$$

$$\mathit{lookup}'[\![M.\phi]\!](\Sigma;\phi_0) = \begin{cases} \mathit{lookup}'[\![\phi]\!](\Sigma;\phi'.M') & \text{if } \Sigma(\phi_0.M) = (\texttt{module}, \phi'.(M')) \\ \perp & \text{otherwise} \end{cases}$$

$$\mathit{lookup}'[\![x]\!](\Sigma;\phi_0) = \begin{cases} (\tau,\varsigma) & \text{if } \Sigma(\phi_0.x) = (\tau,\varphi) \\ \perp & \text{otherwise} \end{cases}$$

Fixed point of module environment

$$\mathit{Env}^*[\![p]\!] = \bigcup_{i\ge 0} \mathit{Env}^i[\![p]\!](\emptyset;\epsilon)$$

where $\mathit{Env}^0[\![p]\!](\Sigma;\phi) = \Sigma$

$\mathit{Env}^i[\![p]\!](\Sigma;\phi) = \mathit{Env}[\![p]\!](\mathit{Env}^{i-1}[\![p]\!](\Sigma;\phi);\phi)$ for $i > 0$

A.4 Semantics

A.4.1 Domains

$$\mathit{StoreValue} ::= \{\texttt{[[Value]]} : \mathit{Val} \cup \mathit{Path} \cup \{\perp\}, \texttt{[[Mutable]]} : \mathit{Bool}, \texttt{[[Configurable]]} : \mathit{Bool}\}$$

A.4.2 Program

$$\frac{\begin{array}{r}(H_0,A_0,tb,\Sigma,\phi) = (\mathit{InitHeap}, \#\mathit{Global}, \#\mathit{Global}, \mathit{Env}^*[\![p]\!]) \\ (H_0,A_0,tb,\Sigma,\phi), p \to_f (H_1,A_1), ct_1 \quad \text{function declaration} \\ (H_1,A_1,tb,\Sigma,\phi), p \to_v (H_2,A_2), ct_2 \quad \text{variable declaration} \\ (H_2,A_2,tb,\Sigma,\phi), p \to_{mod} (H_3,A_3), ct_3 \quad \text{module declaration} \\ (H_3,A_3,tb,\Sigma,\phi), p \to_{imp} (H_4,A_4), ct_4 \quad \text{import declaration} \\ (H_4,A_4,tb,\Sigma,\phi), p \to_s (H_5,A_5), ct_5 \quad \text{statement}\end{array}}{p \to_p (H_5,A_5), ct_5}$$

A.4.3 Function declaration

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{function}\ f(x_1, \cdots, x_n)\ \{s_1 \cdots s_n\} \rightarrow_f (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{export function}\ f(x_1, \cdots, x_n)\ \{s_1 \cdots s_n\} \rightarrow_f (H', A'), ct}$$

A.4.4 Variable declaration

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{var}\ x_1(= e_1), \cdots, x_n(= e_n); \rightarrow_v (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{export var}\ x_1(= e_1), \cdots, x_n(= e_n); \rightarrow_v (H', A'), ct}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{var}\ M; \rightarrow_v (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_v (H', A'), ct}$$

$$\frac{\begin{array}{c}(H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{import}\ e_i; \rightarrow_v (H_i, A_i), ct_i \qquad \forall 1 \leq i \leq n\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{import}\ e_1, \cdots, e_n; \rightarrow_v (H_n, A_n), ct_n}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{var}\ x; \rightarrow_v (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{import}\ p\ \texttt{as}\ x; \rightarrow_v (H', A'), ct}$$

$$\frac{\begin{array}{c}(H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{import}\ \{e_i\}\ \texttt{from}\ p; \rightarrow_v (H_i, A_i), ct_i \qquad \forall 1 \leq i \leq n\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{e_1, \cdots, e_n\}\ \texttt{from}\ p; \rightarrow_v (H_n, A_n), ct_n}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x\}\ \texttt{from}\ p; \rightarrow_v (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\}\ \texttt{from}\ p; \rightarrow_v (H', A'), ct}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{var}\ x'; \rightarrow_v (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p; \rightarrow_v (H', A'), ct}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\}\ \texttt{from}\ p; \rightarrow_v (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{import}\ x\ \texttt{from}\ p; \rightarrow_v (H', A'), ct}$$

A.4.5 Module declaration

$$\frac{\begin{array}{c}p = s_1 \cdots s_n \qquad \{i_1, \cdots, i_k\} = \{i | 1 \leq i \leq n \wedge s_i = \texttt{module}\ M_i\ \{\cdots\}\} \\ (H_0, A_0, ct_0) = (H, A, \texttt{Normal(empty)}) \\ (H_{j-1}, A_{j-1}, tb, \Sigma, \phi), s_{i_j} \rightarrow_{inst} (H_j, A_j), ct_j \quad \forall 1 \leq j \leq k \\ (H'_0, A'_0, ct'_0) = (H_k, A_k, ct_k) \\ (H'_{j-1}, A'_{j-1}, tb, \Sigma, \phi), s_{i_j} \rightarrow_{seal} (H'_j, A'_j), ct'_j \quad \forall 1 \leq j \leq k \\ (H''_0, A''_0, ct''_0) = (H'_k, A'_k, ct'_k) \\ (H''_{j-1}, A''_{j-1}, tb, \Sigma, \phi), s_{i_j} \rightarrow_{init} (H''_j, A''_j), ct''_j \quad \forall 1 \leq j \leq k\end{array}}{(H, A, tb, \Sigma, \phi), p \rightarrow_m (H''_k, A''_k), ct''_k}$$

A.4.6 Module instantiation

$$\begin{array}{rl} \mathit{NewModInstObj}(A) = & \{\texttt{[[Class]]} : \texttt{"Module"}, \\ & \texttt{[[Extensible]]} : \texttt{true}, \\ & \texttt{[[Prototype]]} : \#\mathit{Null}, \\ & \texttt{[[Scope]]} : \{\}, \\ & \texttt{@property} : \{\}\} \end{array}$$

$$\phi' = \begin{cases} M & \text{if } \phi = \epsilon \\ \phi.M & \text{otherwise} \end{cases}$$

$$\frac{\begin{array}{c}l = NewLoc() \qquad H_0 = H[l \mapsto \mathit{NewModInstObj}()] \\ A_0 = \{\} :: A \\ (H_0, A_0, tb, \Sigma, \phi'), s_1 \cdots s_n \rightarrow_f (H_1, A_1), ct_1 \\ (H_1, A_1, tb, \Sigma, \phi'), s_1 \cdots s_n \rightarrow_v (H_2, A_2), ct_2 \\ (H_2, A_2, l, \Sigma, \phi'), s_1 \cdots s_n \rightarrow_{inst} (H_3, A_3), ct_3 \\ H_4 = H_3[l \mapsto H_3(l)[\texttt{[[Scope]]} \mapsto A_3]] \\ \mathit{PutValue}(H_4, A, M, H_4(l), \texttt{strict}) = (H', A', v)\end{array}}{(H, A, tb, \Sigma), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{inst} (H', A'), v}$$

$$\frac{\begin{array}{c}(H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{export}\ e_i; \rightarrow_{inst} (H_i, A_i), ct_i\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export}\ e_1, \cdots, e_n; \rightarrow_{inst} (H_n, A_n), ct_n}$$

$$\frac{\begin{array}{c}(H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{export}\ \{e_i\}\ (\texttt{from}\ p); \rightarrow_{inst} (H_i, A_i), ct_i\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export}\ \{e_1, \cdots, e_n\}\ (\texttt{from}\ p); \rightarrow_{inst} (H_n, A_n), ct_n}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{export } \{x\texttt{: } x\} \texttt{ (from } p\texttt{)}; \rightarrow_{inst} (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{export } \{x\} \texttt{ (from } p\texttt{)}; \rightarrow_{inst} (H', A'), ct}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{export } \{x\texttt{: } p.\, p'\}; \rightarrow_{inst} (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{export } \{x\texttt{: } p'\} \texttt{ (from } p\texttt{)}; \rightarrow_{inst} (H', A'), ct}$$

$$\frac{\begin{array}{c}(H, A, tb, \Sigma, \phi), \texttt{get } x() \texttt{ \{ return } p\texttt{; \} } \rightarrow_m (H', y, ap) \\ \mathit{DefineOwnProperty}(H', A, tb, \mathit{ToString}(H', y), ap, \texttt{false}) = (H'', A'', v)\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export } \{x\texttt{: } p\}; \rightarrow_{inst} (H'', A''), \texttt{Normal}(v)}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{export } \{x\} \texttt{ (from } p\texttt{)}; \rightarrow_{inst} (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{export } x \texttt{ (from } p\texttt{)}; \rightarrow_{inst} (H', A'), ct}$$

$$\frac{\begin{array}{c}\{x_1, \cdots, x_n\} = \{x \mid \phi.\,(x) \in \Sigma\} \\ (H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{get } x_i() \texttt{ \{ return } x_i\texttt{; \} } \rightarrow_m (H_i', y_i, ap_i) \\ \mathit{DefineOwnProperty}(H_i', A_{i-1}, tb, \mathit{ToString}(H_i', y_i), ap_i, \texttt{false}) = (H_i, A_i, v_i)\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export } *\texttt{; } \rightarrow_{inst} (H_n, A_n), \texttt{Normal}(v_n)}$$

$$\frac{\begin{array}{c}\mathit{lookup}[\![p]\!](\Sigma; \phi) = (\texttt{module}, \varphi) \\ \phi' = \begin{cases} M & \text{if } \varphi = M \\ \phi''.M & \text{if } \varphi = \phi''.(M) \end{cases} \\ \{x_1, \cdots, x_n\} = \{x \mid \phi'.\,(x) \in \Sigma\} \\ (H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{get } x_i() \texttt{ \{ return } p.x_i\texttt{; \} } \rightarrow_m (H_i', y_i, ap_i) \\ \mathit{DefineOwnProperty}(H_i', A_{i-1}, tb, \mathit{ToString}(H_i', y_i), ap_i, \texttt{false}) = (H_i, A_i, v_i)\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export } * \texttt{ from } p\texttt{; } \rightarrow_{inst} (H_n, A_n), \texttt{Normal}(v_n)}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{export } f\texttt{; } \rightarrow_{inst} (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{export function } f(x_1, \cdots, x_n) \texttt{ \{} s_1 \cdots s_m\texttt{\} } \rightarrow_{inst} (H', A'), ct}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{export } \{x_1, \cdots, x_n\}\texttt{; } \rightarrow_{inst} (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{export var } x_1(= e_1), \cdots, x_n(= e_n) \rightarrow_{inst} (H', A'), ct}$$

$$\frac{\begin{array}{c}(H, A, tb, \Sigma, \phi), \texttt{get } f() \texttt{ \{} s_1 \cdots s_n\texttt{\} } \rightarrow_m (H', y, ap) \\ \mathit{DefineOwnProperty}(H', A, tb, \mathit{ToString}(H', y), ap, \texttt{false}) = (H'', A'', v)\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export get } f() \texttt{ \{} s_1 \cdots s_n\texttt{\} } \rightarrow_{inst} (H'', A''), \texttt{Normal}(v)}$$

$$\frac{\begin{array}{c}(H, A, tb, \Sigma, \phi), \texttt{set } f(x) \texttt{ \{} s_1 \cdots s_n\texttt{\} } \rightarrow_m (H', y, ap) \\ \mathit{DefineOwnProperty}(H', A, tb, \mathit{ToString}(H', y), ap, \texttt{false}) = (H'', A'', v)\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{export set } f(x) \texttt{ \{} s_1 \cdots s_n\texttt{\} } \rightarrow_{inst} (H'', A''), \texttt{Normal}(v)}$$

A.4.7 Sealing

$$\frac{\begin{array}{c}(H, A, tb, \Sigma, \phi), \texttt{Object.seal}(M)\texttt{; } \rightarrow_s (H_0, A_0), ct_0 \\ (H, A, tb, \Sigma, \phi), M \rightarrow_e l \\ (H_{i-1}, H(l).\texttt{[[Scope]]}, tb, \Sigma, \phi), s_i \rightarrow_{seal} (H_i, A_i), ct_i\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{module } M \texttt{ \{} s_1 \cdots s_n\texttt{\} } \rightarrow_{seal} (H_n, A_0), ct_n}$$

A.4.8 Module initialization

$$\frac{\begin{array}{c}(H, A, tb, \Sigma, \phi), M \rightarrow_e l \\ (H, H(l).\texttt{[[Scope]]}, tb, \Sigma, \phi), s_1 \cdots s_n \rightarrow_{imp} (H_0', A_0'), ct_0' \\ (H_0', A_0', tb, \Sigma, \phi), s_1 \cdots s_n \rightarrow_s (H_0, A_0), ct_0 \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), s_i \rightarrow_{init} (H_i, A_i), ct_i\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{module } M \texttt{ \{} s_1 \cdots s_n\texttt{\} } \rightarrow_{init} (H_n, A_0), ct_n}$$

A.4.9 Import declaration

$$\frac{\begin{array}{c}(H_0, A_0) = (H, A) \\ (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{import } e_i\texttt{; } \rightarrow_{imp} (H_i, A_i), ct_i \qquad \forall 1 \leq i \leq n\end{array}}{(H, A, tb, \Sigma, \phi), \texttt{import } e_1, \cdots, e_n\texttt{; } \rightarrow_{imp} (H_n, A_n), ct_n}$$

$$\frac{\mathit{PutValue}(H, A, x, p, \texttt{strict}) = (H', A', v)}{(H, A, tb, \Sigma, \phi), \texttt{import } p \texttt{ as } x\texttt{; } \rightarrow_{imp} (H', A'), \texttt{Normal}(v)}$$

$$\frac{\mathit{PutValue}(H, A, x, p, \texttt{strict}) = (H', A', err)}{(H, A, tb, \Sigma, \phi), \texttt{import}\ p\ \texttt{as}\ x; \rightarrow_{imp} (H', A'), \texttt{Throw}(err)}$$

$$\frac{(H_0, A_0) = (H, A) \quad (H_{i-1}, A_{i-1}, tb, \Sigma, \phi), \texttt{import}\ \{e_i\}\ \texttt{from}\ p; \rightarrow_{imp} (H_i, A_i), ct_i \qquad \forall 1 \leq i \leq n}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{e_1, \cdots, e_n\}\ \texttt{from}\ p; \rightarrow_{imp} (H_n, A_n), ct_n}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x\}\ \texttt{from}\ p; \rightarrow_{imp} (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\}\ \texttt{from}\ p; \rightarrow_{imp} (H', A'), ct}$$

$$\frac{\mathit{PutValue}(H, A, x', p.x, \texttt{strict}) = (H', A', v)}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p; \rightarrow_{imp} (H', A'), \texttt{Normal}(v)}$$

$$\frac{\mathit{PutValue}(H, A, x', p.x, \texttt{strict}) = (H', A', err)}{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p; \rightarrow_{imp} (H', A'), \texttt{Throw}(err)}$$

$$\frac{(H, A, tb, \Sigma, \phi), \texttt{import}\ \{x\}\ \texttt{from}\ p; \rightarrow_{imp} (H', A'), ct}{(H, A, tb, \Sigma, \phi), \texttt{import}\ x\ \texttt{from}\ p; \rightarrow_{imp} (H', A'), ct}$$

A.4.10 Proxy

8.7.2

PutValue :

$\mathit{Heap} \times \mathit{Env} \times \mathit{Var} \times (\mathit{Val} \cup \mathit{Path}) \times \texttt{strict} \rightarrow \mathit{Heap} \times \mathit{Env} \times \mathit{ValError}$

$\mathit{PutValue}(H, A, x, v, b) =$

$$\begin{cases} \mathit{InterpreterError} & \text{if } \mathit{Lookup}(H, A, x, \texttt{strict}) = (l, y) \wedge v \in \mathit{Path} \\ (H, A, \texttt{ReferenceError}) & \text{if } \mathit{Lookup}(H, A, x, \texttt{strict}) = (l, y) \wedge v \in \mathit{Val} \wedge l = \texttt{\#Null} \wedge b \\ \mathit{Put}(H, A, \texttt{\#Global}, y, v, \texttt{false}) & \text{if } \mathit{Lookup}(H, A, x, \texttt{strict}) = (l, y) \wedge v \in \mathit{Val} \wedge l = \texttt{\#Null} \wedge \neg b \\ \mathit{Put}(H, A, l, y, v, b) & \text{if } \mathit{Lookup}(H, A, x, \texttt{strict}) = (l, y) \wedge v \in \mathit{Val} \wedge l \neq \texttt{\#Null} \\ \mathit{SetBindingDER}(H, A, y, v, b) & \text{if } \mathit{Lookup}(H, A, x, \texttt{strict}) = (\sigma, y) \end{cases}$$

8.12.5

SetBindingDER :

$\mathit{Heap} \times \mathit{Env} \times \mathit{Var} \times (\mathit{Val} \cup \mathit{Path}) \times \mathit{strict} \rightarrow \mathit{Heap} \times \mathit{Env} \times \mathit{ValError}$

10.2.2.1

$\mathit{Lookup} : \mathit{Heap} \times \mathit{Env} \times \mathit{PName} \times \texttt{strict} \rightarrow (\mathit{EnvRec}, \mathit{PName} \cup \mathit{Path})$

$\mathit{Lookup}(H, A, x, \texttt{strict}) =$

$$\begin{cases} (\texttt{\#Null}, x) & \text{if } A = \texttt{\#Global} \wedge \neg \mathit{HasProperty}(H, \texttt{\#Global}, x) \\ (l, x) & \text{if } A = \texttt{\#Global} \wedge \mathit{HasProperty}(H, \texttt{\#Global}, x) \wedge \mathit{GetProperty}(H, \texttt{\#Global}, x) = (\_, l) \\ (\sigma, x) & \text{if } A = \sigma :: A' \wedge x \in \mathit{Dom}(\sigma) \\ \mathit{Lookup}(H, A', x, \texttt{strict}) & \text{if } A = \sigma :: A' \wedge x \notin \mathit{Dom}(\sigma) \\ (l', x) & \text{if } A = l :: A' \wedge \mathit{HasProperty}(H, l, x) \wedge \mathit{GetProperty}(H, l, x) = (\_, l') \\ \mathit{Lookup}(H, A', x, \texttt{strict}) & \text{if } A = l :: A' \wedge \neg \mathit{HasProperty}(H, l, x) \end{cases}$$

11.1.2

10.3.1

10.2.2.1

$$\frac{\mathit{Lookup}(H, A, x, \texttt{strict}) = (\sigma, p) \quad (H, A, tb, \Sigma, \phi), p \rightarrow_{path} ve}{(H, A, tb, \Sigma, \phi), x \rightarrow_{e} ve}$$

$$\frac{\mathit{Lookup}(H, A, x, \texttt{strict}) = (\sigma, y)}{(H, A, tb, \Sigma, \phi), x \rightarrow_{e} \mathit{GetBindingValue}(H, \sigma, y, \texttt{strict})}$$

$$\frac{\mathit{Lookup}(H, A, x, \texttt{strict}) = (l, y)}{(H, A, tb, \Sigma, \phi), x \rightarrow_{e} \mathit{GetBindingValue}(H, l, y, \texttt{strict})}$$

$$\frac{(H, A, tb, \Sigma, \phi), x \rightarrow_{e} ve}{(H, A, tb, \Sigma, \phi), x \rightarrow_{path} ve}$$

$$\frac{(H, A, tb, \Sigma, \phi), x \rightarrow_e v_0 \quad H_0 = H \quad \left\{ \begin{array}{c} CheckObjectCoercible(v_{i-1}) \neq err \\ ToObject(H_{i-1}, v_{i-1}) = (H_i, l_i) \\ v_i = Get(H_i, l_i, "x_i") \end{array} \right\} \quad \forall 1 \leq i \leq n}{(H, A, tb, \Sigma, \phi), x.\, x_1.\, \cdots.\, x_n \rightarrow_{path} v_n}$$

A.5 Rewriting Rule

A.5.1 Helper functions

$$Arguments[\![s_1 \cdots s_n]\!](x) = s'_1 \cdots s'_n$$
where $\Sigma = [\texttt{arguments} \mapsto (\texttt{var}, x.\, \texttt{arguments}, \texttt{local})]$
and $(\Sigma, \epsilon), s_1 \cdots s_n \rightarrow_{imp} s'_1 \cdots s'_n$

$$Arguments[\![s_1 \cdots s_n]\!] = Arguments[\![s_1 \cdots s_n]\!](\texttt{arguments})$$

```
Bypass = {
    get arguments(){return arguments;},
    set arguments(x){arguments = x;},
}
```

Global

$$Append(\phi, x) = \left\{ \begin{array}{ll} x & \text{if } \phi = \epsilon \\ \phi.\, x & \text{if } \phi \neq \epsilon \end{array} \right.$$

$$Prefix(\phi) = \left\{ \begin{array}{ll} \_x & \text{if } \phi = x \\ \_x.\, Prefix(\phi') & \text{if } \phi = x.\, \phi' \end{array} \right.$$

$$IntMod(\phi) = \texttt{<> intmod.}\ \phi$$

$$ExtMod(\phi) = \texttt{<> extmod/}x_1\texttt{/}\cdots\texttt{/}x_n \qquad \text{where } \phi = x_1.\cdots.x_n$$

A.5.2 Program

Rewriting of a program is to collect module declarations, import declarations, and the others in the order, then to rewrite each of them. Hoisting function declarations and variable declarations are omitted because it will be done by every JavaScript interpreter. Rewriting module declarations consists of two parts; to initialize all the module instance objects with function declarations and variable declarations in the modules, then to interpret the module bodies and freezing the module instance objects.

$$\frac{(\Sigma, \phi), p \rightarrow_f p_f \qquad (\Sigma, \phi), p \rightarrow_v p_v \qquad (\Sigma, \phi), p \rightarrow_{inst} p_{inst} \qquad (\Sigma, \phi), p \rightarrow_{init} p_{init} \qquad (\Sigma, \phi), p \rightarrow_s p_s}{(\Sigma, \phi), p \rightarrow p_f\ p_v\ p_{inst}\ p_{init}\ p_s}$$

A.5.3 Function declaration

$$\frac{(\Sigma, \phi), s_i \rightarrow_f s'_i \qquad \forall 1 \leq i \leq n}{(\Sigma, \phi), s_1 \cdots s_n \rightarrow_f s'_1 \cdots s'_n}$$

$$\frac{(\Sigma, \phi), \texttt{function}\ f(x_1, \cdots, x_n)\ \{s_1 \cdots s_m\} \rightarrow_{imp} s'}{(\Sigma, \phi), (\texttt{export})\ \texttt{function}\ f(x_1, \cdots, x_n)\ \{s_1 \cdots s_m\} \rightarrow_f s'}$$

A.5.4 Variable declaration

$$\frac{\begin{array}{c} \phi = \epsilon \qquad \{x_1, \cdots, x_m\} = \{x | (x) \in Dom(\Sigma)\} \\ v_1 = \texttt{<> Object=Object} \qquad v_2 = \texttt{<> intmod=\{\}} \qquad v_3 = \texttt{<> extmod=\{\}} \\ v_4 = \texttt{<> initfun=\{\}} \qquad v_5 = \texttt{<> initarg=\{\}} \end{array}}{(\Sigma, \phi), s_1 \cdots s_n \rightarrow_v \texttt{var}\ x_1, \cdots, x_m, v_1, v_2, v_3, v_4, v_5;}$$

$$\frac{\begin{array}{c} \phi \neq \epsilon \qquad \{x_1, \cdots, x_m\} = \{x | \phi.\,(x) \in Dom(\Sigma)\} \\ e_i = \left\{ \begin{array}{ll} \texttt{get } x_i\texttt{()}\ \texttt{\{return } x_i\texttt{;\}} & \text{if } \Sigma(\phi.(x_i)) = (\texttt{var}, \phi.(x_i)) \\ \texttt{get } x_i\texttt{()}\ \texttt{\{return <> intmod.}\ \phi'.\, x'\texttt{;\}} & \text{if } \Sigma(\phi.(x_i)) = (\tau, \phi'.(x')) \quad \phi \neq \phi' \\ \texttt{get } x_i\texttt{()}\ \texttt{\{return <> extmod.}\ Sub(\phi').\, x'\texttt{;\}} & \text{if } \Sigma(\phi.(x_i)) = (\tau, \phi'.x') \end{array} \right. \\ 1 \leq i \leq m \end{array}}{(\Sigma, \phi), s_1 \cdots s_n \rightarrow_v \texttt{var } x_1, \cdots, x_m\texttt{; <> intmod.}\ \phi\texttt{=\{}e_1, \cdots, e_m\texttt{\};}}$$

A.5.5 Module instantiation

$$\frac{(\Sigma,\phi), s_i \rightarrow_{inst} s_i' \qquad \forall 1 \le i \le n}{(\Sigma,\phi), s_1 \cdots s_n \rightarrow_{inst} s_1' \cdots s_n'}$$

$$\frac{\begin{array}{c}\phi' = \mathit{Append}(\phi, M) \\ (\Sigma,\phi'), s_1 \cdots s_n \rightarrow_f s_f \qquad (\Sigma,\phi'), s_1 \cdots s_n \rightarrow_v s_v \\ (\Sigma,\phi'), s_1 \cdots s_n \rightarrow_{inst} s_{inst} \qquad (\Sigma,\phi'), s_1 \cdots s_n \rightarrow_{exp} s_{exp} \\ (\Sigma,\phi'), s_1 \cdots s_n \rightarrow_{up} s_{up} \\ e = \texttt{function(arguments)}\ \{\mathit{Arguments}[\![s_f\ s_v\ s_{inst}\ s_{exp}\ s_{up}]\!]\} \\ s_1 = M\texttt{=<> extmod.}\mathit{Sub}(\phi')\texttt{=new}\ (e)(\mathit{Bypass}); \\ s_2 = \texttt{<> Object.seal}(M); \end{array}}{(\Sigma,\phi), \texttt{module}\ M\ \{s_1 \cdots s_n\} \rightarrow_{inst} s_1\ s_2}$$

A.5.6 Export declaration

$$\frac{(\Sigma,\phi), s_i \rightarrow_{exp} s_i' \qquad \forall 1 \le i \le n}{(\Sigma,\phi), s_1 \cdots s_n \rightarrow_{exp} s_1' \cdots s_n'}$$

$$\frac{(\Sigma,\phi), \texttt{export}\ e_i \rightarrow_{exp} s_i \qquad \forall 1 \le i \le n}{(\Sigma,\phi), \texttt{export}\ e_1, \cdots, e_n \rightarrow_{exp} s_1 \cdots s_n}$$

$$\frac{(\Sigma,\phi), \texttt{export}\ \{e_i\}\ (\texttt{from}\ p) \rightarrow_{exp} s_i \qquad \forall 1 \le i \le n}{(\Sigma,\phi), \texttt{export}\ \{e_1, \cdots, e_n\}\ (\texttt{from}\ p) \rightarrow_{exp} s_1 \cdots s_n}$$

$$\frac{(\Sigma,\phi), \texttt{export}\ \{x\texttt{: } x\}\ (\texttt{from}\ p) \rightarrow_{exp} s}{(\Sigma,\phi), \texttt{export}\ \{x\}\ (\texttt{from}\ p) \rightarrow_{exp} s}$$

$$\frac{(\Sigma,\phi), \texttt{export}\ \{x\texttt{: } p.\, p'\} \rightarrow_{exp} s}{(\Sigma,\phi), \texttt{export}\ \{x\texttt{: } p'\}\ (\texttt{from}\ p) \rightarrow_{exp} s}$$

$$\frac{\begin{array}{c}\mathit{lookup}[\![p]\!](\Sigma;\phi) = (\tau, \phi'.(x')) \\ s = \texttt{(function() \{} \\ \quad \texttt{var <> desc =<> Object.getOwnPropertyDescriptor(this,"}x\texttt{");} \\ \quad \texttt{delete this.}x\texttt{;} \\ \quad \texttt{if (typeof <> desc == "undefined")} \\ \quad\quad \texttt{<> desc = \{configurable : true\};} \\ \quad \texttt{d.get = function() \{return } \mathit{IntMod}(\phi'.\, x')\texttt{;\};} \\ \quad \texttt{<> Object.defineOwnProperty(this,"}x\texttt{", <> desc);} \\ \texttt{\})();} \end{array}}{(\Sigma,\phi), \texttt{export}\ \{x\texttt{: } p\} \rightarrow_{exp} s}$$

$$\frac{\begin{array}{c}\mathit{lookup}[\![p]\!](\Sigma;\phi) = (\tau, \phi'.x') \\ s = \texttt{(function() \{} \\ \quad \texttt{var <> desc =<> Object.getOwnPropertyDescriptor(this,"}x\texttt{");} \\ \quad \texttt{delete this.}x\texttt{;} \\ \quad \texttt{if (typeof <> desc == "undefined")} \\ \quad\quad \texttt{<> desc = \{configurable : true\};} \\ \quad \texttt{d.get = function() \{return } \mathit{ExtMod}(\phi').\, x'\texttt{;\};} \\ \quad \texttt{<> Object.defineOwnProperty(this,"}x\texttt{", <> desc);} \\ \texttt{\})();} \end{array}}{(\Sigma,\phi), \texttt{export}\ \{x\texttt{: } p\} \rightarrow_{exp} s}$$

$$\frac{(\Sigma,\phi), \texttt{export}\ \{x\}\ (\texttt{from}\ p) \rightarrow_{exp} s}{(\Sigma,\phi), \texttt{export}\ x\ (\texttt{from}\ p) \rightarrow_{exp} s}$$

$$\frac{\begin{array}{c}\{x_1, \cdots, x_n\} = \{x \mid \phi.\,(x) \in \mathit{Dom}(\Sigma)\} \\ (\Sigma,\phi), \texttt{export}\ x_i \rightarrow_{exp} s_i \qquad \forall 1 \le i \le n \end{array}}{(\Sigma,\phi), \texttt{export}\ * \rightarrow_{exp} s_1 \cdots s_n}$$

$$\frac{\begin{array}{c}\mathit{lookup}[\![p]\!](\Sigma,\phi) = (\texttt{module}, \phi'.(M)) \\ \phi'' = \mathit{Append}(\phi', M) \\ \{x_1, \cdots, x_n\} = \{x \mid \phi''.\,(x) \in \mathit{Dom}(\Sigma)\} \\ (\Sigma,\phi), \texttt{export}\ x_i\ \texttt{from}\ p \rightarrow_{exp} s_i \qquad \forall 1 \le i \le n \end{array}}{(\Sigma,\phi), \texttt{export}\ *\ \texttt{from}\ p \rightarrow_{exp} s_1 \cdots s_n}$$

$$\frac{(\Sigma,\phi), \texttt{export}\ f \rightarrow_{exp} s}{(\Sigma,\phi), \texttt{export function}\ f(x_1, \cdots, x_n)\ \{s_1 \cdots s_m\} \rightarrow_{exp} s}$$

$$\frac{(\Sigma,\phi), \texttt{export}\ \{x_1,\cdots,x_n\} \rightarrow_{exp} s}{(\Sigma,\phi), \texttt{export var}\ x_1(=e_1),\cdots,x_n(=e_n) \rightarrow_{exp} s}$$

$$\frac{\begin{array}{c}(\Sigma,\phi), s_1 \cdots s_n \rightarrow_{imp} s'_1 \cdots s'_n \\ s = \texttt{(function() \{ var <> desc =<> Object.getOwnPropertyDescriptor(this,"}f\texttt{"); delete this.}f\texttt{; if (typeof <> desc == "undefined") <> desc = \{configurable : true\}; <> desc.get = function() \{}s'_1 \cdots s'_n\texttt{\}; <> Object.defineOwnProperty(this,"}f\texttt{", <> desc); \})();}\end{array}}{(\Sigma,\phi), \texttt{export get}\ f()\ \{s_1 \cdots s_n\} \rightarrow_{exp} s}$$

$$\frac{\begin{array}{c}(\Sigma,\phi), s_1 \cdots s_n \rightarrow_{imp} s'_1 \cdots s'_n \\ s = \texttt{(function() \{ var <> desc =<> Object.getOwnPropertyDescriptor(this,"}f\texttt{"); delete this.}f\texttt{; if (typeof <> desc == "undefined") <> desc = \{configurable : true\}; <> desc.set = function(}x\texttt{) \{}s'_1 \cdots s'_n\texttt{\}; <> Object.defineOwnProperty(this,"}f\texttt{", <> desc); \})();}\end{array}}{(\Sigma,\phi), \texttt{export set}\ f(x)\ \{s_1 \cdots s_n\} \rightarrow_{exp} s}$$

A.5.7 Update function

$$\frac{\begin{array}{c}(\Sigma,\phi), s_i \rightarrow_{up} s'_i \qquad \forall 1 \le i \le n \\ (\Sigma,\phi), s'_1 \cdots s'_n \rightarrow_{imp} s''_1 \cdots s''_n \\ f = \texttt{function(arguments)}\ \{\textit{Arguments}[\![s''_1 \cdots s''_n]\!]\} \\ s_1 = \texttt{<> initfun.}\ \phi\texttt{=}f\texttt{;} \qquad s_2 = \texttt{<> initarg.}\ \phi\texttt{=}\textit{Bypass}\texttt{;}\end{array}}{(\Sigma,\phi), s_1 \cdots s_n \rightarrow_{up} s_1\ s_2}$$

$$\frac{\phi' = \textit{Append}(\phi, M) \qquad f = \texttt{<> initfun.}\ \phi' \qquad o = \texttt{<> initarg.}\ \phi'}{(\Sigma,\phi), \texttt{module}\ M\{s_1 \cdots s_n\} \rightarrow_{up} f\texttt{.call(this,}o\texttt{);}}$$

$$\frac{s \in \textit{Statement}}{(\Sigma,\phi), s \rightarrow_{up} s}$$

A.5.8 Module initialization

$$\frac{(\Sigma,\phi), s_i \rightarrow_{init} s'_i \qquad \forall 1 \le i \le n}{(\Sigma,\phi), s_1 \cdots s_n \rightarrow_{init} s'_1 \cdots s'_n}$$

$$\frac{\phi' = \textit{Append}(\phi, M) \qquad f = \texttt{<> initfun.}\ \phi' \qquad o = \texttt{<> initarg.}\ \phi'}{(\Sigma,\phi), \texttt{module}\ M\{s_1 \cdots s_n\} \rightarrow_{init} f\texttt{.call(this,}o\texttt{);}}$$

A.5.9 Alias

11.1

$$(\Sigma,\phi), \texttt{this} \rightarrow_{imp} \texttt{this}$$

$$\frac{\textit{lookup}[\![\phi.(x)]\!](\Sigma;\phi) = (\tau, \varphi_e, \texttt{module})}{(\Sigma,\phi), x \rightarrow_{imp} \textit{Global}.\ \varphi_e}$$

$$\frac{\textit{lookup}[\![\phi.(x)]\!](\Sigma;\phi) = (\tau, \varphi_e, \texttt{local})}{(\Sigma,\phi), x \rightarrow_{imp} \varphi_e}$$

$$\frac{\textit{lookup}[\![\phi.(x)]\!](\Sigma;\phi) \neq (\tau, \varphi_e, \varsigma)}{(\Sigma,\phi), x \rightarrow_{imp} x}$$

$$\frac{e \in \textit{Literal}}{(\Sigma,\phi), e \rightarrow_{imp} e}$$

$$\frac{(\Sigma,\phi), e_i \rightarrow_{imp} e'_i \qquad \forall 1 \le i \le n \wedge \exists e_i}{(\Sigma,\phi), [(e_1),\cdots,(e_n)] \rightarrow_{imp} [(e'_1),\cdots,(e'_n)]}$$

$$\frac{(\Sigma,\phi),e_i \rightarrow_{imp} e'_i \qquad \forall 1 \le i \le n}{(\Sigma,\phi),\{e_1,\cdots,e_n(,)\} \rightarrow_{imp} \{e'_1,\cdots,e'_n(,)\}}$$

$$\frac{(\Sigma,\phi),e \rightarrow_{imp} e'}{(\Sigma,\phi),(e) \rightarrow_{imp} (e')}$$

11.1.5

$$\frac{(\Sigma,\phi),e \rightarrow_{imp} e'}{(\Sigma,\phi),x{:}e \rightarrow_{imp} x{:}e'}$$

$$\frac{\begin{array}{c}(\Sigma,\phi),s_1\cdots s_n \rightarrow_{hoist} A \\ \{x_1,\cdots,x_n\} = \{f,\texttt{arguments}\} \cup A \\ \Sigma' = \Sigma[\phi.(x_1) \mapsto (\texttt{var},x_1,\texttt{local}),\cdots,\phi.(x_n) \mapsto (\texttt{var},x_n,\texttt{local})] \\ (\Sigma',\phi),s_1\cdots s_n \rightarrow_{imp} s'_1\cdots s'_n\end{array}}{(\Sigma,\phi),\texttt{get}\ f()\ \{s_1\cdots s_n\} \rightarrow_{imp} \texttt{get}\ f()\ \{s'_1\cdots s'_n\}}$$

$$\frac{\begin{array}{c}(\Sigma,\phi),s_1\cdots s_n \rightarrow_{hoist} A \\ \{x_1,\cdots,x_n\} = \{f,\texttt{arguments}\} \cup A \\ \Sigma' = \Sigma[\phi.(x_1) \mapsto (\texttt{var},x_1,\texttt{local}),\cdots,\phi.(x_n) \mapsto (\texttt{var},x_n,\texttt{local})] \\ (\Sigma',\phi),s_1\cdots s_n \rightarrow_{imp} s'_1\cdots s'_n\end{array}}{(\Sigma,\phi),\texttt{set}\ f()\ \{s_1\cdots s_n\} \rightarrow_{imp} \texttt{set}\ f()\ \{s'_1\cdots s'_n\}}$$

11.2

$$\frac{(\Sigma,\phi),e_1 \rightarrow_{imp} e'_1 \qquad (\Sigma,\phi),e_2 \rightarrow_{imp} e'_2}{(\Sigma,\phi),e_1[e_2] \rightarrow_{imp} e'_1[e'_2]}$$

$$\frac{(\Sigma,\phi),e_1 \rightarrow_{imp} e'_1 \qquad (\Sigma,\phi),e_2 \rightarrow_{imp} e'_2}{(\Sigma,\phi),e_1.e_2 \rightarrow_{imp} e'_1.e'_2}$$

$$\frac{(\Sigma,\phi),e \rightarrow_{imp} e' \qquad (\Sigma,\phi),e_i \rightarrow_{imp} e'_i \qquad \forall 1 \le i \le n}{(\Sigma,\phi),\texttt{new}\ e(e_1,\cdots,e_n) \rightarrow_{imp} \texttt{new}\ e'(e'_1,\cdots,e'_n)}$$

$$\frac{(\Sigma,\phi),e \rightarrow_{imp} e'}{(\Sigma,\phi),\texttt{new}\ e \rightarrow_{imp} \texttt{new}\ e'}$$

$$\frac{(\Sigma,\phi),e \rightarrow_{imp} e' \qquad (\Sigma,\phi),e_i \rightarrow_{imp} e'_i \qquad \forall 1 \le i \le n}{(\Sigma,\phi),e(e_1,\cdots,e_n) \rightarrow_{imp} e'(e'_1,\cdots,e'_n)}$$

$$\frac{\begin{array}{c}(\Sigma,\phi),s_1\cdots s_m \rightarrow_{hoist} A \\ \{y_1,\cdots,y_k\} = \{f,\texttt{arguments},x_1,\cdots,x_n\} \cup A \\ \Sigma' = \Sigma[\phi.(y_1) \mapsto (\texttt{var},y_1,\texttt{local}),\cdots,\phi.(y_k) \mapsto (\texttt{var},y_k,\texttt{local})] \\ (\Sigma',\phi),s_1\cdots s_m \rightarrow_{imp} s'_1\cdots s'_m\end{array}}{\begin{array}{c}(\Sigma,\phi),\texttt{function}\ f(x_1,\cdots,x_n)\ \{s_1\cdots s_m\} \rightarrow_{imp} \\ \texttt{function}\ f(x_1,\cdots,x_n)\ \{s'_1\cdots s'_m\}\end{array}}$$

$$\frac{\begin{array}{c}(\Sigma,\phi),s_1\cdots s_m \rightarrow_{hoist} A \\ \{y_1,\cdots,y_k\} = \{\texttt{arguments},x_1,\cdots,x_n\} \cup A \\ \Sigma' = \Sigma[\phi.(y_1) \mapsto (\texttt{var},y_1,\texttt{local}),\cdots,\phi.(y_k) \mapsto (\texttt{var},y_k,\texttt{local})] \\ (\Sigma',\phi),s_1\cdots s_m \rightarrow_{imp} s'_1\cdots s'_m\end{array}}{\begin{array}{c}(\Sigma,\phi),\texttt{function}\ (x_1,\cdots,x_n)\ \{s_1\cdots s_m\} \rightarrow_{imp} \\ \texttt{function}\ (x_1,\cdots,x_n)\ \{s'_1\cdots s'_m\}\end{array}}$$

11.3

$$\frac{op \in \{++,--\} \qquad (\Sigma,\phi),e \rightarrow_{imp} e'}{(\Sigma,\phi),eop \rightarrow_{imp} e'op}$$

11.4

$$\frac{op \in \{\texttt{delete},\texttt{void},\texttt{typeof},++,--,+,-,\sim,!\} \qquad (\Sigma,\phi),e \rightarrow_{imp} e'}{(\Sigma,\phi),ope \rightarrow_{imp} ope'}$$

11.5 - 11.14

$$\frac{\begin{array}{c}op \in \left\{\begin{array}{c}*,/,\%,+,-,<<,>>,>>>,<,>,<=,>=,\texttt{instanceof},\texttt{in}, \\ ==,!=,===,!==,\&\&,||,\&,\hat{},|,=,*=,/=,\%=,+=,-=, \\ <<=,>>=,>>>=,\&=,\hat{}=,!=,,\end{array}\right\} \\ (\Sigma,\phi),e_1 \rightarrow_{imp} e'_1 \qquad (\Sigma,\phi),e_2 \rightarrow_{imp} e'_2\end{array}}{(\Sigma,\phi),e_1 ope_2 \rightarrow_{imp} e'_1 ope'_2}$$

12

$$\frac{(\Sigma,\phi), s_i \rightarrow_{imp} s'_i \qquad \forall 1 \leq i \leq n}{(\Sigma,\phi), s_1 \cdots s_n \rightarrow_{imp} s'_1 \cdots s'_n}$$

$$\frac{(\Sigma,\phi), s_i \rightarrow_{imp} s'_i \qquad \forall 1 \leq i \leq n}{(\Sigma,\phi), \{s_1 \cdots s_n\} \rightarrow_{imp} \{s'_1 \cdots s'_n\}}$$

$$\frac{(\Sigma,\phi), x_i{=}e_i \rightarrow_{imp} s'_i \qquad \forall 1 \leq i \leq n \wedge \exists e_i}{(\Sigma,\phi), \texttt{var}\ x_1(=e_1), \cdots, x_n(=e_n) \rightarrow_{imp} \{s'_1 \cdots s'_n\}}$$

$$(\Sigma,\phi), ; \rightarrow_{imp} ;$$

$$\frac{(\Sigma,\phi), e_1 \rightarrow_{imp} e'_1 \qquad (\Sigma,\phi), s_1 \rightarrow_{imp} s'_1 \qquad (\Sigma,\phi), s_2 \rightarrow_{imp} s'_2}{(\Sigma,\phi), \texttt{if}\ (e_1)\ s_1\ (\texttt{else}\ s_2) \rightarrow_{imp} \texttt{if}\ (e'_1)\ s'_1\ (\texttt{else}\ s'_2)}$$

$$\frac{(\Sigma,\phi), s \rightarrow_{imp} s' \qquad (\Sigma,\phi), e \rightarrow_{imp} e'}{(\Sigma,\phi), \texttt{do}\ s\ \texttt{while}\ (e); \rightarrow_{imp} \texttt{do}\ s\ \texttt{while}\ (e);}$$

$$\frac{(\Sigma,\phi), e \rightarrow_{imp} e' \qquad (\Sigma,\phi), s \rightarrow_{imp} s'}{(\Sigma,\phi), \texttt{while}\ (e)\ s \rightarrow_{imp} \texttt{while}\ (e)\ s'}$$

$$\frac{\begin{array}{c}(\Sigma,\phi), e_1 \rightarrow_{imp} e'_1 \qquad (\Sigma,\phi), e_2 \rightarrow_{imp} e'_2 \qquad (\Sigma,\phi), e_3 \rightarrow_{imp} e'_3 \\ (\Sigma,\phi), s \rightarrow_{imp} s'\end{array}}{(\Sigma,\phi), \texttt{for}\ (e_1;e_2;e_3)\ s \rightarrow_{imp} \texttt{for}\ (e'_1;e'_2;e'_3)\ s'}$$

$$\frac{\begin{array}{c}(\Sigma,\phi), \texttt{var}\ x_1(=e_1), \cdots, x_n(=e_n) \rightarrow_{imp} s'_0 \\ (\Sigma,\phi), e_{n+1} \rightarrow_{imp} e'_{n+1} \qquad (\Sigma,\phi), e_{n+2} \rightarrow_{imp} e'_{n+2} \\ (\Sigma,\phi), s \rightarrow_{imp} s'\end{array}}{\begin{array}{l}(\Sigma,\phi), \texttt{for}\ (\texttt{var}\ x_1(=e_1), \cdots, x_n(=e_n);e_{n+1};e_{n+2})\ s \rightarrow_{imp} \\ \quad \{s'_0\ \texttt{for}\ (;e'_{n+1};e'_{n+2})\ s'\}\end{array}}$$

$$\frac{(\Sigma,\phi), e_1 \rightarrow_{imp} e'_1 \qquad (\Sigma,\phi), e_2 \rightarrow_{imp} e'_2 \qquad (\Sigma,\phi), s \rightarrow_{imp} s'}{(\Sigma,\phi), \texttt{for}\ (e_1\ \texttt{in}\ e_2)\ s \rightarrow_{imp} \texttt{for}\ (e'_1\ \texttt{in}\ e'_2)\ s'}$$

$$\frac{\begin{array}{c}(\Sigma,\phi), \texttt{var}\ x_1(=e_1) \rightarrow_{imp} s'_0 \\ (\Sigma,\phi), x_1 \rightarrow_{imp} e'_0 \qquad (\Sigma,\phi), e_2 \rightarrow_{imp} e'_2 \qquad (\Sigma,\phi), s \rightarrow_{imp} s'\end{array}}{(\Sigma,\phi), \texttt{for}\ (\texttt{var}\ x_1(=e_1)\ \texttt{in}\ e_2)\ s \rightarrow_{imp} \{s'_0\ \texttt{for}\ (e'_0\ \texttt{in}\ e'_2)\ s'\}}$$

$$(\Sigma,\phi), \texttt{continue}\ (x); \rightarrow_{imp} \texttt{continue}\ (x);$$

$$(\Sigma,\phi), \texttt{break}\ (x); \rightarrow_{imp} \texttt{break}\ (x);$$

$$\frac{(\Sigma,\phi), e \rightarrow_{imp} e'}{(\Sigma,\phi), \texttt{return}\ (e); \rightarrow_{imp} \texttt{return}\ (e');}$$

$$\frac{(\Sigma,\phi), e \rightarrow_{imp} e' \qquad (\Sigma,\phi), s \rightarrow_{imp} s'}{(\Sigma,\phi), \texttt{with}\ (e)\ s \rightarrow_{imp} \texttt{with}\ (e')\ s'}$$

$$\frac{(\Sigma,\phi), e \rightarrow_{imp} e' \qquad (\Sigma,\phi), c_i \rightarrow_{imp} c'_i \qquad \forall 1 \leq i \leq n}{(\Sigma,\phi), \texttt{switch}\ (e)\ \{c_1 \cdots c_n\} \rightarrow_{imp} \texttt{switch}\ (e')\ \{c'_1 \cdots c'_n\}}$$

$$\frac{(\Sigma,\phi), e \rightarrow_{imp} e' \qquad (\Sigma,\phi), s_i \rightarrow_{imp} s'_i \qquad \forall 1 \leq i \leq n}{(\Sigma,\phi), \texttt{case}\ e{:}s_1 \cdots s_n \rightarrow_{imp} \texttt{case}\ e'{:}s'_1 \cdots s'_n}$$

$$\frac{(\Sigma,\phi), e \rightarrow_{imp} e' \qquad (\Sigma,\phi), s_i \rightarrow_{imp} s'_i \qquad \forall 1 \leq i \leq n}{(\Sigma,\phi), \texttt{default}\ e{:}s_1 \cdots s_n \rightarrow_{imp} \texttt{default}\ e'{:}s'_1 \cdots s'_n}$$

$$\frac{\Sigma' = \Sigma[x \mapsto (\texttt{var}, x, \texttt{local})] \qquad (\Sigma',\phi), s \rightarrow_{imp} s'}{(\Sigma,\phi), x{:}s \rightarrow_{imp} x{:}s'}$$

$$\frac{(\Sigma,\phi), e \rightarrow_{imp} e'}{(\Sigma,\phi), \texttt{throw}\ e; \rightarrow_{imp} \texttt{throw}\ e';}$$

$$\frac{\begin{array}{c}\Sigma' = \Sigma[x \mapsto (\texttt{var}, x, \texttt{local})] \\ (\Sigma,\phi), s_1 \rightarrow_{imp} s'_1 \qquad (\Sigma',\phi), s_2 \rightarrow_{imp} s'_2 \qquad (\Sigma,\phi), s_3 \rightarrow_{imp} s'_3\end{array}}{\begin{array}{l}(\Sigma,\phi), \texttt{try}\ s_1\ (\texttt{catch}\ (x)\ s_2)\ (\texttt{finally}\ s_3) \rightarrow_{imp} \\ \quad \texttt{try}\ s'_1\ (\texttt{catch}\ (x)\ s'_2)\ (\texttt{finally}\ s'_3)\end{array}}$$

$$(\Sigma,\phi), \texttt{debugger}; \rightarrow_{imp} \texttt{debugger};$$

B. Rewriting of JavaScript Module System

B.1 Environment

The construction of environments for `export *` is omitted for now.

$$
\begin{array}{rcl}
x & \in & \textit{Identifier} \\
\phi & ::= & x(.\,x)* \\
\varphi_i & ::= & \phi.\,(x) \\
\varphi_e & ::= & \phi.\,x \\
\varphi & ::= & \varphi_i \mid \varphi_e \\
\overline{\varphi} & ::= & \phi.* \\
\tau & = & \texttt{var} \mid \texttt{module} \\
\varsigma & = & \epsilon \mid \texttt{local} \mid \texttt{export}\ \varphi_e \\
\rho & = & \bot \mid \tau\varphi \\
\overline{\rho} & = & \bot \mid \top \\
\Sigma & = & \{(\varphi, \rho\varsigma), \cdots\} \cup \{(\overline{\varphi}, \overline{\rho}), \cdots\} \\
\Sigma^* & = & \epsilon \mid \Sigma^*\Sigma \mid \Sigma^* x
\end{array}
$$

$$
\textit{Env}_m[\![\texttt{var}\ x_1(=e_1)\ ,\ \cdots\ ,\ x_n(=e_n)\ ;]\!](\Sigma;\phi) =
\begin{cases}
\{(\phi.(x), \texttt{var}\ \phi.(x)\ \texttt{export}\ \phi.x), (\phi.x, \texttt{var}\ \phi.x\ \epsilon) \mid x \in \{x_1, \cdots, x_n\}\} & \text{if } \phi = \epsilon \\
\{(\phi.(x), \texttt{var}\ \phi.(x)\ \epsilon) \mid x \in \{x_1, \cdots, x_n\}\} & \text{otherwise}
\end{cases}
$$

$$
\textit{Env}_m[\![\texttt{module}\ M\ \{s_1 \cdots s_n\}]\!](\Sigma;\phi) =
\{(\phi.(M), \texttt{module}\ \phi.(M)\ \epsilon)\} \cup \textit{Env}_m[\![s_1 \cdots s_n]\!](\Sigma;\phi.M)
$$

$$
\textit{Env}_m[\![\texttt{export}\ \textit{ess}_1,\ \cdots, \textit{ess}_n;]\!](\Sigma;\phi) =
\bigcup_{1 \le i \le n} \textit{Env}_m[\![\texttt{export}\ \textit{ess}_i]\!](\Sigma;\phi)
$$

$$
\textit{Env}_m[\![\texttt{export}\ \{es_1,\ \cdots, es_n\}\ (\texttt{from}\ p)]\!](\Sigma;\phi) =
\bigcup_{1 \le i \le n} \textit{Env}_m[\![\texttt{export}\ \{es_i\}\ (\texttt{from}\ p)]\!](\Sigma;\phi)
$$

$$
\textit{Env}_m[\![\texttt{export}\ \{x\}\ (\texttt{from}\ p)]\!](\Sigma;\phi) =
\textit{Env}_m[\![\texttt{export}\ \{x\colon x\}\ (\texttt{from}\ p)]\!](\Sigma;\phi)
$$

$$
\textit{Env}_m[\![\texttt{export}\ \{x\colon p\}\ \texttt{from}\ p_0]\!](\Sigma;\phi) =
\textit{Env}_m[\![\texttt{export}\ \{x\colon p_0.p\}]\!](\Sigma;\phi)
$$

$$
\textit{Env}_m[\![\texttt{export}\ \{x\colon x_0\}]\!](\Sigma;\phi) =
\begin{cases}
\{(\phi.(x_0), \bot\ \texttt{export}\ \phi.x), (\phi.x, \bot\ \epsilon), (\phi.x, \texttt{var}\ \phi.x\ \epsilon)\} & \text{if } (\phi.(x_0), \texttt{var}\ \phi'.(x')\ \varsigma) \in \Sigma \\
\{(\phi.(x_0), \bot\ \texttt{export}\ \phi.x), (\phi.x, \bot\ \epsilon), (\phi.x, \texttt{var}\ \phi'.x'\ \epsilon)\} & \text{if } (\phi.(x_0), \texttt{var}\ \phi'.x'\ \varsigma) \in \Sigma \\
\{(\phi.(x_0), \bot\ \texttt{export}\ \phi.x), (\phi.x, \bot\ \epsilon)\} & \text{otherwise}
\end{cases}
$$

$$
\textit{Env}_m[\![\texttt{export}\ \{x\colon p.x_0\}]\!](\Sigma;\phi) =
\begin{cases}
\{(\phi.x, \bot), (\phi.x, \texttt{var}\ \varphi\ \epsilon)\} & \text{if } \textit{lookup}[\![p.x_0]\!](\Sigma;\phi) = \texttt{var}\ \varphi \\
\{(\phi.x, \bot)\} & \text{otherwise}
\end{cases}
$$

$$
\textit{Env}_m[\![\texttt{export}\ x\ (\texttt{from}\ p)]\!](\Sigma;\phi) =
\textit{Env}_m[\![\texttt{export}\ \{x\}\ (\texttt{from}\ p)]\!](\Sigma;\phi) =
$$

$$
\textit{Env}_m[\![\texttt{export var}\ x_1(=e_1)\ ,\ \cdots\ ,\ x_n(=e_n)\ ;]\!](\Sigma;\phi) =
\{(\phi.(x), \texttt{var}\ \phi.(x)\ \texttt{export}\ \phi.x), (\phi.x, \texttt{var}\ \phi.x\ \epsilon) \mid x \in \{x_1, \cdots, x_n\}\}
$$

$$
\textit{Env}_m[\![\texttt{export function}\ f(x_1,\ \cdots,\ x_n)\ \{\ s_1,\ \cdots,\ s_m\ \}]\!](\Sigma;\phi) =
\textit{Env}_m[\![\texttt{export var}\ f]\!](\Sigma;\phi)
$$

$$
\textit{Env}_m[\![\texttt{export get}\ f()\ \{\ s_1,\ \cdots,\ s_m\ \}]\!](\Sigma;\phi) =
\textit{Env}_m[\![\texttt{export var}\ f]\!](\Sigma;\phi)
$$

$$
\textit{Env}_m[\![\texttt{export set}\ f(x_1,\ \cdots,\ x_n)\ \{\ s_1,\ \cdots,\ s_m\ \}]\!](\Sigma;\phi) =
\textit{Env}_m[\![\texttt{export var}\ f]\!](\Sigma;\phi)
$$

$$
\begin{aligned}
&\mathit{Env}_m[\![\texttt{import}\ ic_1,\ \cdots,\ ic_n]\!](\Sigma;\phi) = \\
&\quad \bigcup_{1\le i\le n} \mathit{Env}_m[\![\texttt{import}\ ic_i]\!](\Sigma;\phi) \\
&\mathit{Env}_m[\![\texttt{import}\ p\ \texttt{as}\ M]\!](\Sigma;\phi) = \\
&\quad \begin{cases} \{(\phi.(M), \bot\ \epsilon), (\phi.(M), \texttt{module}\ \varphi\ \epsilon)\} & \text{if } \mathit{lookup}[\![p]\!](\Sigma;\phi) = \texttt{module}\ \varphi \\ \{(\phi.(M), \bot\ \epsilon)\} & \text{otherwise} \end{cases} \\
&\mathit{Env}_m[\![\texttt{import}\ \{is_1,\ \cdots,\ is_n\}\ \texttt{from}\ p]\!](\Sigma;\phi) = \\
&\quad \bigcup_{1\le i\le n} \mathit{Env}_m[\![\texttt{import}\ \{is_1\}\ \texttt{from}\ p]\!](\Sigma;\phi) \\
&\mathit{Env}_m[\![\texttt{import}\ \{x\}\ \texttt{from}\ p]\!](\Sigma;\phi) = \\
&\quad \mathit{Env}_m[\![\texttt{import}\ \{x\colon x\}\ \texttt{from}\ p]\!](\Sigma;\phi) \\
&\mathit{Env}_m[\![\texttt{import}\ \{x\colon x'\}\ \texttt{from}\ p]\!](\Sigma;\phi) = \\
&\quad \begin{cases} \{(\phi.(x'), \bot\ \epsilon), (\phi.(x'), \texttt{var}\ \varphi\ \epsilon)\} & \text{if } \mathit{lookup}[\![p.x]\!](\Sigma;\phi) = \tau\ \varphi \\ \{(\phi.(x'), \bot\ \epsilon)\} & \text{otherwise} \end{cases} \\
&\mathit{Env}_m[\![\texttt{import}\ x\ \texttt{from}\ p]\!](\Sigma;\phi) = \\
&\quad \mathit{Env}_m[\![\texttt{import}\ \{x\}\ \texttt{from}\ p]\!](\Sigma;\phi)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{lookup}[\![M.\phi]\!](\Sigma;\phi_0) &= \begin{cases} \mathit{lookup}'[\![\phi]\!](\Sigma;\phi'.M') & \text{if } (\phi_0.(M), \texttt{module}\ \phi'.(M')\ \varsigma) \in \Sigma \\ \mathit{lookup}_\bot[\![M.\phi]\!](\Sigma;\phi_0) & \text{otherwise} \end{cases} \\
\mathit{lookup}[\![x]\!](\Sigma;\phi_0) &= \begin{cases} \tau\ \varphi & \text{if } (\phi_0.(x), \tau\ \varphi\ \varsigma) \in \Sigma \\ \mathit{lookup}_\bot[\![x]\!](\Sigma;\phi_0) & \text{otherwise} \end{cases} \\
\mathit{lookup}_\bot[\![\phi]\!] &= \begin{cases} \mathit{lookup}[\![\phi]\!](\Sigma;\phi_0') & \text{if } \phi_0 = \phi_0'.M \\ \bot & \text{otherwise} \end{cases} \\
\mathit{lookup}'[\![M.\phi]\!](\Sigma;\phi_0) &= \begin{cases} \mathit{lookup}'[\![\phi]\!](\Sigma;\phi'.M') & \text{if } (\phi_0.M, \texttt{module}\ \phi'.(M')\ \varsigma) \in \Sigma \\ \bot & \text{otherwise} \end{cases} \\
\mathit{lookup}'[\![x]\!](\Sigma;\phi_0) &= \begin{cases} \tau\ \varsigma & \text{if } (\phi_0.x, \tau\ \varphi\ \varsigma) \in \Sigma \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Env}[\![p]\!] &= \bigcup_{i\ge 0} \mathit{Env}_m^i[\![p]\!](\emptyset;\epsilon) \\
\text{where } \mathit{Env}_m^0[\![p]\!](\Sigma;\phi) &= \Sigma \\
\mathit{Env}_m^i[\![p]\!](\Sigma;\phi) &= \mathit{Env}_m[\![p]\!](\mathit{Env}_m^{i-1}[\![p]\!](\Sigma;\phi);\phi) \text{ for } i \ge 0
\end{aligned}
$$

B.2 Rewriting Rule

B.2.1 Program

Rewriting of a program is to collect module declarations, import declarations, and the others in the order, then to rewrite each of them. Hoisting function declarations and variable declarations are omitted because it will be done by every JavaScript interpreter. Rewriting module declarations consists of two parts; to initialize all the module instance objects with function declarations and variable declarations in the modules, then to interpret the module bodies and freezing the module instance objects.

$$
\frac{(\Sigma,\phi), p \to_m p_m \qquad (\Sigma,\phi), p \to_u p_u \qquad (\Sigma,\phi), p \to_s p_s}{(\Sigma,\phi), p \to p_m\ p_u\ p_s}
$$

B.2.2 Module Declaration

Names of Function Declarations *FunName* is a helper function that collects the names of functions declared in a module.

$$
\mathit{FunName}[\![s_1 \cdots s_n]\!](\Sigma,\phi) = \bigcup_{1\le i\le n} \mathit{FunName}[\![s_i]\!](\Sigma,\phi)
$$

$$
\mathit{FunName}[\![(\texttt{export})\ \texttt{function}\ f(x_1,\ \cdots,\ x_n)\ \{\ s_1\ \cdots\ s_m\ \}]\!](\Sigma,\phi) = \{f\}
$$

Names of Variables in Module *VarName*, *PriVarName*, *PubVarName* are helper functions that collect the names of variables, unexported variables, and exported variables in a module, respectively.

$$
\mathit{VarName}[\![s_1 \cdots s_n]\!](\Sigma,\phi) = \{x \mid (\phi.(x), \texttt{var}\ \varphi\ \varsigma) \in \Sigma\} \setminus \mathit{FunName}[\![s_1 \cdots s_n]\!](\Sigma,\phi)
$$

$$
\mathit{PriVarName}[\![s_1 \cdots s_n]\!](\Sigma,\phi) = \{x \mid (\phi.(x), \texttt{var}\ \varphi\ \varsigma) \in \Sigma \wedge (\phi.x, \texttt{var}\ \varphi\ \varsigma) \notin \Sigma\} \setminus \mathit{FunName}[\![s_1 \cdots s_n]\!](\Sigma,\phi)
$$

$$
\mathit{PubVarName}[\![s_1 \cdots s_n]\!](\Sigma,\phi) = \{x \mid (\phi.x, \texttt{var}\ \varphi\ \varsigma) \in \Sigma\} \setminus \mathit{FunName}[\![s_1 \cdots s_n]\!](\Sigma,\phi)
$$

Names of Submodules in Module *ModName* is a helper function that collects the names of submodule declared in a module.

$$\mathit{ModName}(\Sigma, \phi) = \{M \mid (\phi.(M), \texttt{module}\ \varphi\ \varsigma) \in \Sigma\}$$

Fresh Name of Update Function for Module *UpdateName* is a helper function that finds a unused name in a module for temporary helper function for the corresponding module instance object.

$$\begin{aligned}\mathit{UpdateName}(\Sigma, \phi) = {} & \texttt{update}k \\ & \text{where } k = \min\{n \mid n \in \mathbb{N} \wedge (\phi.(\texttt{update}n), \tau\ \varphi\ \varsigma) \in \Sigma\}\end{aligned}$$

Function Declaration *FunDecl* is a helper function that collects the function declarations declared in a module.

$$\mathit{FunDecl}[\![s_1 \cdots s_n]\!](\Sigma, \phi) = \bigcup_{1 \le i \le n} \mathit{FunDecl}[\![s_1]\!](\Sigma, \phi)$$

$$\begin{aligned}&\mathit{FunDecl}[\![(\texttt{export})\ \texttt{function}\ f(x_1, \cdots, x_n)\ \{\ s_1\ \cdots\ s_n\ \}]\!](\Sigma, \phi) = \\ &\quad \texttt{function}\ f(x_1, \cdots, x_n)\ \{\ s_1 \cdots s_n\ \}\end{aligned}$$

Variable Declaration *VarDecl* is a helper function that declares all the variables in a module.

$$\mathit{VarDecl}[\![s_1 \cdots s_n]\!](\Sigma, \phi) = \{\texttt{var}\ x \mid x \in \mathit{VarName}[\![s_1 \cdots s_n]\!](\Sigma, \phi)\}$$

Exports *Exports* is a helper function that sets the getters for exported names in a module.

$$\begin{aligned}&\mathit{Exports}(\Sigma, \phi) = \\ &\quad \{\texttt{Object.defineProperty(this, ``}x\texttt{'', \{get : function() \{ return } x\texttt{; \}\})} \mid (\phi.x, \tau\ \varphi\ \varsigma) \in \Sigma\}\end{aligned}$$

Update Function *UpdateFun* is a helper function that generates a helper function in module instance object, of which body is the module body except the function, variable, and submodule declarations.

$$\mathit{UpdateFun}[\![s_1\ \cdots\ s_n]\!](\Sigma, \phi) = \bigcup_{1 \le i \le n} \mathit{UpdateFun}[\![s_i]\!](\Sigma, \phi)$$

$$\mathit{UpdateFun}[\![(\texttt{export})\ \texttt{var}\ x_1(= e_1),\ \cdots,\ x_n(= e_n)]\!](\Sigma, \phi) = \{x_i = e_i \mid 1 \le i \le n \wedge \exists e_i\}$$

$$\begin{aligned}&\mathit{UpdateFun}[\![\texttt{export get}\ f()\ \{\ s_1 \cdots s_n\ \}]\!](\Sigma, \phi) = \\ &\quad \texttt{if (typeof Object.getOwnPropertyDescriptor(this, ``}f\texttt{'') === ``undefined'')} \\ &\qquad \texttt{Object.defineProperty(this, ``}f\texttt{'', \{} \\ &\qquad\quad \texttt{get : function() \{ } s_1 \cdots s_n \texttt{ \} \})} \\ &\quad \texttt{else} \\ &\qquad \texttt{Object.defineProperty(this, ``}f\texttt{'', \{} \\ &\qquad\quad \texttt{get : function() \{ } s_1 \cdots s_n \texttt{ \},} \\ &\qquad\quad \texttt{set : Object.getOwnPropertyDescriptor(this, ``}f\texttt{'').set \})}\end{aligned}$$

$$\begin{aligned}&\mathit{UpdateFun}[\![\texttt{export set}\ f()\ \{\ s_1 \cdots s_n\ \}]\!](\Sigma, \phi) = \\ &\quad \texttt{if (typeof Object.getOwnPropertyDescriptor(this, ``}f\texttt{'') === ``undefined'')} \\ &\qquad \texttt{Object.defineProperty(this, ``}f\texttt{'', \{} \\ &\qquad\quad \texttt{set : function() \{ } s_1 \cdots s_n \texttt{ \} \})} \\ &\quad \texttt{else} \\ &\qquad \texttt{Object.defineProperty(this, ``}f\texttt{'', \{} \\ &\qquad\quad \texttt{get : Object.getOwnPropertyDescriptor(this, ``}f\texttt{'').get,} \\ &\qquad\quad \texttt{set : function() \{ } s_1 \cdots s_n \texttt{ \} \})}\end{aligned}$$

$$\mathit{UpdateFun}[\![s]\!](\Sigma, \phi) = s \qquad \text{if } s \in \mathit{Statement}$$

Substituting Qualified Name in Expression $\mathit{QualName}_e$ is a helper function that substitutes imported identifiers in an expression to the corresponding exported qualified names.

$$\begin{aligned}\mathit{QualName}_e[\![\texttt{this}]\!](\Gamma) &= \texttt{this} \\ \mathit{QualName}_e[\![x]\!](\Gamma) &= \begin{cases} \Gamma[x] & \text{if } x \in \Gamma \\ x & \text{otherwise} \end{cases} \\ \mathit{QualName}_e[\![\mathit{literal}]\!](\Gamma) &= \mathit{literal} \\ \mathit{QualName}_e[\![[(e_1),\ \cdots,\ (e_n)]]\!](\Gamma) &= [(\mathit{QualName}[\![e_1]\!](\Gamma)),\ \cdots,\ (\mathit{QualName}[\![e_n]\!](\Gamma))] \\ \mathit{QualName}_e[\![\{e_1,\ \cdots,\ e_n(,)\}]\!](\Gamma) &= \{\mathit{QualName}[\![e_1]\!](\Gamma),\ \cdots,\ \mathit{QualName}[\![e_n]\!](\Gamma)(,)\} \\ \mathit{QualName}_e[\![(e)]\!](\Gamma) &= (\mathit{QualName}[\![e]\!](\Gamma))\end{aligned}$$

$$\mathit{QualName}_e[\![x\colon e]\!](\Gamma) = x\colon \mathit{QualName}[\![e]\!](\Gamma)$$

$$\mathit{QualName}_e[\![\texttt{get}\ f()\ \{\ s_1\ \cdots\ s_n\ \}]\!](\Gamma) = \texttt{get}\ f()\ \{\ \mathit{QualName}_s[\![s_1]\!](\Gamma')\ \cdots\ \mathit{QualName}_s[\![s_n]\!](\Gamma')\ \}$$
where $\Gamma' = \Gamma \setminus \{f, \texttt{arguments}\} \setminus \mathit{HoistedName}[\![s_1 \cdots s_n]\!]$

$$\mathit{QualName}_e[\![\texttt{set}\ f(x)\ \{\ s_1\ \cdots\ s_n\ \}]\!](\Gamma) = \texttt{set}\ f(x)\ \{\ \mathit{QualName}_s[\![s_1]\!](\Gamma')\ \cdots\ \mathit{QualName}_s[\![s_n]\!](\Gamma')\ \}$$
where $\Gamma' = \Gamma \setminus \{f, x, \texttt{arguments}\} \setminus \mathit{HoistedName}[\![s_1 \cdots s_n]\!]$

$$\mathit{QualName}_e[\![\texttt{function}\ f(x_1,\ \cdots,\ x_n)\ \{\ s_1\ \cdots\ s_n\ \}]\!](\Gamma) =$$
$$\texttt{function}\ f(x_1,\ \cdots,\ x_n)\ \{\ \mathit{QualName}_s[\![s_1]\!](\Gamma \setminus \{f, x_1, \cdots, x_n\})\ \cdots\ \mathit{QualName}_s[\![s_n]\!](\Gamma')\ \}$$
where $\Gamma' = \Gamma \setminus \{f, x_1, \cdots, x_n, \texttt{arguments}\} \setminus \mathit{HoistedName}[\![s_1 \cdots s_n]\!]$

$$\mathit{QualName}_e[\![\texttt{function}\ (x_1,\ \cdots,\ x_n)\ \{\ s_1\ \cdots\ s_n\ \}]\!](\Gamma) =$$
$$\texttt{function}\ (x_1,\ \cdots,\ x_n)\ \{\ \mathit{QualName}_s[\![s_1]\!](\Gamma \setminus \{x_1, \cdots, x_n\})\ \cdots\ \mathit{QualName}_s[\![s_n]\!](\Gamma')\ \}$$
where $\Gamma' = \Gamma \setminus \{x_1, \cdots, x_n, \texttt{arguments}\} \setminus \mathit{HoistedName}[\![s_1 \cdots s_n]\!]$

$$\begin{aligned}
\mathit{QualName}_e[\![e_1[e_2]]\!](\Gamma) &= \mathit{QualName}_e[\![e_1]\!](\Gamma)[\mathit{QualName}_e[\![e_2]\!](\Gamma)] \\
\mathit{QualName}_e[\![e.x]\!](\Gamma) &= \mathit{QualName}_e[\![e_1]\!](\Gamma).x \\
\mathit{QualName}_e[\![\texttt{new}\ e(e_1,\ \cdots,\ e_n)]\!] &= \texttt{new}\ \mathit{QualName}_e[\![e]\!](\Gamma)(\mathit{QualName}_e[\![e_1]\!](\Gamma),\ \cdots,\ \mathit{QualName}_e[\![e_n]\!](\Gamma)) \\
\mathit{QualName}_e[\![\texttt{new}\ e]\!](\Gamma) &= \texttt{new}\ \mathit{QualName}_e[\![e]\!](\Gamma) \\
\mathit{QualName}_e[\![e(e_1,\ \cdots,\ e_n)]\!] &= \mathit{QualName}_e[\![e]\!](\Gamma)(\mathit{QualName}_e[\![e_1]\!](\Gamma),\ \cdots,\ \mathit{QualName}_e[\![e_n]\!](\Gamma))
\end{aligned}$$

$$\mathit{QualName}_e[\![e[++\mid --]]\!](\Gamma) = \mathit{QualName}_e[\![e]\!](\Gamma)[++\mid --]$$
$$\mathit{QualName}_e[\![op\ e]\!](\Gamma) = op\ \mathit{QualName}_e[\![e]\!](\Gamma)$$
where $op \in \{\texttt{delete}, \texttt{void}, \texttt{typeof}, ++, --, +, -, \sim, !\}$

$$\mathit{QualName}_e[\![e_1\ op\ e_2]\!](\Gamma) = \mathit{QualName}[\![e_1]\!](\Gamma)\ op\ \mathit{QualName}_e[\![e_2]\!](\Gamma) \qquad \text{where}$$
$$op \in \left\{ \begin{array}{l} *, /, \%, +, -, <<, >>, >>>, <, >, <=, >=, \texttt{instanceof}, \texttt{in}, ==, !=, ===, !==, \\ \&\&, ||, \&, \hat{}, |, =, *=, /=, \%=, +=, -=, <<=, >>=, >>>=, \&=, \hat{}=, !=, , \end{array} \right\}$$

Substituting Qualified Name in Statement $\mathit{QualName}_s$ is a helper function that substitutes imported identifiers in a statement to the corresponding exported qualified names.

$$\mathit{QualName}_s[\![\{\ s_1\ \cdots\ s_n\ \}]\!](\Gamma) = \{\ \mathit{QualName}_s[\![s_1]\!](\Gamma)\ \cdots\ \mathit{QualName}_s[\![s_n]\!](\Gamma)\ \}$$

$$\mathit{QualName}_s[\![\texttt{var}\ x_1\ (= e_1),\ \cdots,\ x_n\ (= e_n)]\!](\Gamma) = \texttt{var}\ x_1(= \mathit{QualName}_e[\![e_1]\!](\Gamma)),\ \cdots,\ x_n(= \mathit{QualName}_e[\![e_n]\!](\Gamma))$$

$$\mathit{QualName}_s[\![;]\!](\Gamma) = ;$$

$$\mathit{QualName}_s[\![e]\!](\Gamma) = \mathit{QualName}_e[\![e]\!](\Gamma)$$

$$\mathit{QualName}_s[\![\texttt{if}\ (e)\ s_1\ (\texttt{else}\ s_2)]\!](\Gamma) = \texttt{if}\ (\mathit{QualName}_e[\![e]\!](\Gamma))\ \mathit{QualName}_s[\![s_1]\!](\Gamma)\ (\texttt{else}\ \mathit{QualName}_s[\![s_2]\!](\Gamma))$$

$$\mathit{QualName}_s[\![\texttt{do}\ s\ \texttt{while}\ (e);]\!](\Gamma) = \texttt{do}\ \mathit{QualName}_s[\![s]\!](\Gamma)\ \texttt{while}\ (\mathit{QualName}_e[\![e]\!](\Gamma));$$

$$\mathit{QualName}_s[\![\texttt{while}\ (e)\ s]\!](\Gamma) = \texttt{while}\ (\mathit{QualName}_e[\![e]\!](\Gamma))\ \mathit{QualName}_s[\![s]\!](\Gamma)$$

$$\mathit{QualName}_s[\![\texttt{for}\ (e_1; e_2; e_3)\ s]\!](\Gamma) =$$
$$\texttt{for}\ (\mathit{QualName}_e[\![e_1]\!](\Gamma); \mathit{QualName}_e[\![e_2]\!](\Gamma); \mathit{QualName}_e[\![e_3]\!](\Gamma))\ \mathit{QualName}_s[\![s]\!](\Gamma)$$

$$\mathit{QualName}_s[\![\texttt{for}\ (\texttt{var}\ x_1(= e_1),\ \cdots,\ x_n(= e_n); e_{n+1}; e_{n+2})\ s]\!](\Gamma)$$
$$\texttt{for}\ (\texttt{var}\ x_1(= \mathit{QualName}_e[\![e_1]\!](\Gamma)),\ \cdots,\ x_n(= \mathit{QualName}_e[\![e_n]\!](\Gamma));$$
$$\mathit{QualName}_e[\![e_{n+1}]\!](\Gamma);$$
$$\mathit{QualName}_e[\![e_{n+2}]\!](\Gamma))\ \mathit{QualName}_s[\![s]\!](\Gamma)$$

$$\mathit{QualName}_s[\![\texttt{for}\ (e_1\ \texttt{in}\ e_2)\ s]\!](\Gamma) =$$
$$\texttt{for}\ (\mathit{QualName}_e[\![e_1]\!](\Gamma)\ \texttt{in}\ \mathit{QualName}_e[\![e_2]\!](\Gamma))\ \mathit{QualName}_s[\![s]\!](\Gamma)$$

$$\mathit{QualName}_s[\![\texttt{for}\ (\texttt{var}\ x_1(= e_1),\ \cdots,\ x_n(= e_n)\ \texttt{in}\ e_{n+1})\ s]\!](\Gamma)$$
$$\texttt{for}\ (\texttt{var}\ x_1(= \mathit{QualName}_e[\![e_1]\!](\Gamma)),\ \cdots,\ x_n(= \mathit{QualName}_e[\![e_n]\!](\Gamma))\ \texttt{in}\ \mathit{QualName}_e[\![e_{n+1}]\!](\Gamma))$$
$$\mathit{QualName}_s[\![s]\!](\Gamma)$$

$$\mathit{QualName}_s[\![\texttt{continue}\ (x);]\!](\Gamma) = \texttt{continue}\ (x);$$

$$\mathit{QualName}_s[\![\texttt{break}\ (x);]\!](\Gamma) = \texttt{break}\ (x);$$
$$\mathit{QualName}_s[\![\texttt{return}\ (e);]\!](\Gamma) = \texttt{return}\ (\mathit{QualName}_e[\![e]\!](\Gamma));$$
$$\mathit{QualName}_s[\![\texttt{with}\ (e)\ s]\!](\Gamma) = \texttt{with}\ (\mathit{QualName}_e[\![e]\!](\Gamma))\ \mathit{QualName}_s[\![s]\!](\Gamma)$$
$$\mathit{QualName}_s[\![\texttt{switch}\ (e)\ \{\ c_1\ \cdots\ c_n\ \}]\!](\Gamma) = \texttt{switch}\ (\mathit{QualName}_e[\![e]\!](\Gamma))\ \{\ \mathit{QualName}_s[\![c_1]\!](\Gamma)\ \cdots\ \mathit{QualName}_s[\![c_n]\!](\Gamma)\ \}$$
$$\mathit{QualName}_s[\![\texttt{case}\ e\texttt{:}\ s_1\ \cdots\ s_n]\!](\Sigma) = \texttt{case}\ e\texttt{:}\ \mathit{QualName}_s[\![s_1]\!](\Sigma)\ \cdots\ \mathit{QualName}_s[\![s_n]\!](\Sigma)$$
$$\mathit{QualName}_s[\![\texttt{default}\ \texttt{:}\ s_1\ \cdots\ s_n]\!](\Sigma) = \texttt{default}\ \texttt{:}\ \mathit{QualName}_s[\![s_1]\!](\Sigma)\ \cdots\ \mathit{QualName}_s[\![s_n]\!](\Sigma)$$
$$\mathit{QualName}_s[\![x\texttt{:}\ s]\!](\Sigma) = x\texttt{:}\ \mathit{QualName}_s[\![s]\!](\Sigma \setminus \{x\})$$
$$\mathit{QualName}_s[\![\texttt{throw}\ e]\!](\Sigma) = \texttt{throw}\ \mathit{QualName}_e[\![e]\!](\Sigma)$$
$$\begin{aligned}\mathit{QualName}_s[\![\texttt{try}\ s_1\ (\texttt{catch}\ (x)\ s_2)\ (\texttt{finally}\ s_3)]\!](\Sigma) = \\ \texttt{try}\ \mathit{QualName}_s[\![s_1]\!](\Sigma)\ (\texttt{catch}\ (x)\ \mathit{QualName}_s[\![s_2]\!](\Sigma \setminus \{x\}))\ (\texttt{finally}\ \mathit{QualName}_s[\![s_3]\!](\Sigma))\end{aligned}$$
$$\mathit{QualName}_s[\![\texttt{debugger;}]\!] = \texttt{debugger;}$$

Mapping to Qualified Name *QualNameEnv* is a helper function that mapping from imported identifiers to the corresponding exported qualified names.

$$\mathit{QualNameEnv}(\Sigma, \phi) = \begin{cases} \{x \mapsto \phi'.x' \mid (\phi.(x), \tau\ \phi'.x'\ \varsigma) \in \Sigma\} & \text{if } \phi = \epsilon \\ \{x \mapsto \phi'.x' \mid (\phi.(x), \tau\ \phi'.x'\ \varsigma) \in \Sigma\} \cup \mathit{QualNameEnv}(\Sigma, \phi') & \text{if } \phi = \phi'.M \end{cases}$$

Module Declaration For each module declaration, a function is introduced as a constructor that constructs the module instance object. The function scope of the constructor is used as the module scope. For each exported name, the module instance object has a getter for the property. For mutually recursive imports, the constructor declares the functions and the variables in the module and the remaining parts are stored in a temporary helper function.

$$\frac{(\Sigma, \phi), s_i \rightarrow_m s_i' \qquad \forall 1 \le i \le n}{(\Sigma, \phi), s_1 \cdots s_n \rightarrow_m s_1' \cdots s_n'}$$

$$(\Sigma, \phi), \texttt{module}\ M\ \{\ s_1 \cdots s_n\ \} \rightarrow_m \mathit{ModDecl}[\![\texttt{module}\ M\ \{\ s_1 \cdots s_n\ \}]\!](\Sigma, \phi)$$

$$\begin{aligned}&\mathit{ModDecl}[\![\texttt{module}\ M\ \{\ s_1 \cdots s_n\ \}]\!](\Sigma, \phi) = \\ &\quad \texttt{var}\ M = \texttt{new}\ \mathit{SetPrototype}(f, p)\ () \\ &\text{where } f = \texttt{function}()\ \{ \\ &\qquad \mathit{QualName}_e[\![\mathit{FunDecl}[\![s_1 \cdots s_n]\!](\Sigma, \phi)]\!](\mathit{QualNameEnv}(\Sigma, \phi)) \\ &\qquad \mathit{QualName}_e[\![\mathit{VarDecl}[\![s_1 \cdots s_n]\!](\Sigma, \phi)]\!](\mathit{QualNameEnv}(\Sigma, \phi)) \\ &\qquad \mathit{QualName}_e[\![\mathit{ModDecl}[\![s_1 \cdots s_n]\!](\Sigma, \phi)]\!](\mathit{QualNameEnv}(\Sigma, \phi)) \\ &\qquad \mathit{QualName}_e[\![\mathit{Exports}(\Sigma, \phi)]\!](\mathit{QualNameEnv}(\Sigma, \phi)) \\ &\qquad \texttt{this}.\mathit{UpdateName}(\Sigma, \phi) = \texttt{function(arguments)}\ \{ \\ &\qquad\quad \mathit{QualName}_s[\![\mathit{UpdateFun}[\![s_1 \cdots s_n]\!](\Sigma, \phi)]\!](\mathit{QualNameEnv}(\Sigma, \phi)) \\ &\qquad \} \\ &\quad \} \\ &\text{and } p = \begin{cases} \texttt{Object.prototype} & \text{if } \phi = \epsilon \\ \phi & \text{otherwise} \end{cases}\end{aligned}$$

$$\mathit{SetPrototype}(f, p) = \texttt{function(f, p)}\ \{\ \texttt{f.prototype} = \texttt{p; return f;}\ \}\ (f, p)$$

B.2.3 Module Update

Now, all the function and the variables in the module instance objects are declared. In the order, the temporary helper functions are called, initiating the module instance objects. Then, the helper functions are deleted and the module instance objects are freezed.

$$\frac{\{i_1, \cdots, i_k\} = \{i \mid s_i \in \mathit{ModuleDeclaration}\} \qquad (\Sigma, \phi), s_1 \cdots s_n \rightarrow_u s_{i_1} \cdots s_{i_k} \qquad (\Sigma, \phi), s_1 \cdots s_n \rightarrow_u s' \qquad u = \{\ \phi.M.\mathit{UpdateFun}(\Sigma, \phi)() \ \ \texttt{delete}\ \phi.M.\mathit{UpdateFun}(\Sigma, \phi) \ \ \texttt{Object.freeze}(\phi.M) \ \ s'\ \}}{(\Sigma, \phi), \texttt{module}\ M\ \{\ s_1\ \cdots\ s_n\ \} \rightarrow_u u}$$

B.2.4 Source Element

In the rest of program, the imported names are renamed with the corresponding exported names.

$$\frac{\{i_1, \cdots, i_k\} = \{i \mid 1 \leq i \leq n \wedge s_i \notin \textit{ModuleDeclaration} \cup \textit{ImportDeclaration}\}}{(\Sigma, \phi), s_1 \cdots s_n \rightarrow \textit{QualName}_s[\![s_{i_1} \cdots s_{i_k}]\!](\textit{QualNameEnv}(\Sigma, \phi))}$$